Learn You Some Algebras for Glorious Good!

Peter Harpending <peter@harpending.org>

March 7, 2015

Contents

Chapter 1

Introduction

Before I bore you with a bunch of crap you don't care about, let's do some math, shall we?

There are basically three notions with which you need to be familiar in order to do anything interesting in math. Those three things are *sets*, *functions*, and *proofs*. Unfortunately, to be familiar with one, you have to be familiar with the other two.[1]

So, what are each of those things?

- A *set* is an unordered collection of things. There is also no repetition. For instance, $\{2,5\}$ is the same as $\{5,2\}$ (because order doesn't matter). $\{2,5,5\}$ would be the same set, because there's no notion of multiplicity.
- A *function* is a mathematical construct (well, obviously, else I wouldn't be talking about it). Basically, it takes some input, does something to it, and spits out some output. If you give the function the same input a bunch of times, you should get the same result each time. This concept is called "referential transparency." If the function is not referentially transparent, then it's not a function. It's something else.

[1]You'll learn as we go along, when math people use a common term like *set*, *function*, *proof*, *group*, *continuous* or *closed*, they usually mean something similar in concept to the colloquial term, but there are some strings attached. This is usually the case in the sciences too (e.g. *theory*, *hypothesis*, *experiment*).

- A *proof* is basically where you take a bunch of simple facts, called *axioms*, and chain them together to make *theorem*s. It's sort of like sticking puzzle pieces together to form a picture.

  The puzzle pieces (in this case, the axioms) aren't usually very interesting on their own. However, the picture they form (in this case, the theorem) can be really cool and enlightening. The proof would be analogous to an explicit set of instructions explaining how to put the pieces together.

Once you are familiar with each of those concepts, we can do all sorts of cool stuff. Throughout the book, we will prove all of the following:

- If you tap your finger against a bridge at exactly the right frequency, the bridge will collapse. (Resonance)
- The formula used to calculate the interest rate on your mortgage is actually just a fancy form of the ratios of angles in a triangle. (Euler's formula)

1.1 How to read the book

The best way to read this book is to just read it. Don't skip sections, or look ahead, or anything like that. Just read it straight through. It's also pretty important that you read the rest of this chapter. I promise it's not too boring.

Do all of the exercises. There aren't that many. However, they are pretty difficult. The exercises all have solutions, which are in § F.

The exercises are designed to make you think, and widen your perspective on the topic at hand. They are not designed to be tedious. They are difficult, but the good kind of difficult.

It would be perfectly okay to just do the exercises (all of them), and then go back and read the text when you don't understand something.

§ D is a reference section. It contains every single theorem, definition, identity, and property in this book. Unlike the contents of this book, § D is not meant to be read straight through. However, if you don't remember the name of something, or want to know if some property is true, § D is the place to look.

Please note that this book is far from finished. I've estimated that it will take me 2000 git commits to finish the book, and I'm currently at 747 git commits.

My writing strategy involves writing the bare minimum information, with criminally few examples or exercises, so I can get the structure right, then to go back and fill in the blanks. So, until this book is finished, it's going to be horrifyingly fast paced.

1.2 Introduction (for real this time)

This is a math book. Well, duh. Why did I write it?

Most math (and science) books nowadays seem to value keeping an academic tone over ensuring that the reader understands the material, and — more importantly — enjoys reading the book.

I take the opposite approach. I want to create a book that is fun to read and easy to understand, while eschewing the practice of making myself look good.

The inspiration for this book is *Learn You a Haskell for Great Good!*, by Miran Lipovača. Haskell is a programming language, and LYAH is a great book for learning Haskell. If you are interested in a print copy of LYAH, see [12].

There is also an incomplete and unofficial Russian translation (`https://github.com/gazay/lysa`), courtesy of Alexey Gaziev.

1.3 The community

Despite the fact that I used "I" in the first part of the book, LYSA is actually a community project, and many people participate in the writing of this book.

If you want to talk to us, or to other math people, come see us in `#lysa` on Freenode. If you don't know what IRC is, or you don't have a client set up, you can connect through Freenode's webchat (`http://webchat.freenode.net/?channels=lysa`).

If you have any questions about LYSA (or math), feel free to ask in the IRC channel (`#lysa` on FreeNode in case you forgot).

If you want to submit a correction, or have some issue, or want to add some content, really anything having to do with the content of the book, you can visit our GitLab page (`https://gitlab.com/lysa/lysa`). We also have a woefully incomplete website (`http://learnyou.org`) and a community on Reddit (`https://lysa.reddit.com/`).

1.4 Haskell

In this book, I cover a lot of hard stuff.[2] Sometimes, it's useful to program your way through a problem. Every programmer will tell you that programming teaches a manner of thinking.

Many programmers will cite Steve Jobs[3] famous quote, regarding the use of programming in his job,

> *[sic] ... much more importantly, it had nothing to do with using [the programs we wrote] for anything practical. It had to do with using them to be a mirror of your thought process; to actually learn how to think. I think everybody in this country should learn how to program a computer — should learn a computer language — because it teaches you how to think.*

That first sentence or two is actually a pretty good description of mathematics (and programming). Both are incredibly useful, and have endless practical applications. That's not the point, though. The whole usefulness thing is a side gig. It's about learning how to think, and having a rigorous language through which to express your thoughts. Furthermore, the rigor of the language helps you build

[2]This isn't actually true. Math isn't hard, stupid!

[3]For you youngsters, Steve Jobs is the former CEO of Apple. He's dead now.

upon your current thoughts to find out even cooler things. That's what math is about.

Programming and math go hand-in-hand. Programmers and mathematicians will attest to this; I certainly can. For that reason, throughout this book, there will be coding exercises in the programming language Haskell.[4]

Instructions for installing Haskell can be found on their website (`https://www.haskell.org/platform/`).

1.4.1 Install a text editor

In order to edit Idris code, you need a plain-text editor (as opposed to a word processor).

Some popular plain-text editors are:

1. Gedit (`https://wiki.gnome.org/Apps/Gedit`) - very easy to use. I recommend either Gedit or Kate for beginners.

2. Kate (`http://kate-editor.org/get-it/`) - marginally harder than Gedit, but it has more features.

   Linux/BSD users: If you are not a KDE user, then don't use Kate. It brings in a ton of KDE dependencies. Here's the result of trying to install it on my machine:

[4] I was originally going to use another language called Idris, but Idris is, at the time of this writing, so buggy that it is unusable.

```
% sudo pacman -S kate
[sudo] password for pete:
resolving dependencies...
:: There are 2 providers available for phonon-qt5-backend:
:: Repository extra
   1) phonon-qt5-gstreamer  2) phonon-qt5-vlc

Enter a number (default=1): 2
looking for conflicting packages...

Packages (54) attica-qt5-5.6.0-1  gamin-0.1.10-8  karchive-5.6.0-1  kauth-5.6.0-1  k
              kcodecs-5.6.0-1  kcompletion-5.6.0-1  kconfig-5.6.0-1  kconfigwidgets-
              kcoreaddons-5.6.0-1  kcrash-5.6.0-2  kdbusaddons-5.6.0-1  kded-5.6.0-1
              kguiaddons-5.6.0-1  ki18n-5.6.0-1  kiconthemes-5.6.0-1  kinit-5.6.0-1
              kitemmodels-5.6.0-1  kitemviews-5.6.0-1  kjobwidgets-5.6.0-1  knewstuf
              knotifications-5.6.0-1  kparts-5.6.0-1  kservice-5.6.0-1  ktexteditor-
              ktextwidgets-5.6.0-1  kwallet-5.6.0-1  kwidgetsaddons-5.6.0-1  kwindow
              kxmlgui-5.6.0-1  libdbusmenu-qt5-0.9.3+14.10.20140619-1  libgit2-1:0.2
              libimobiledevice-1.1.7-1  libplist-1.11-1  libusbmuxd-1.0.9-1  libxkbc
              media-player-info-19-1  phonon-qt5-4.8.3-1  phonon-qt5-vlc-0.8.2-1  po
              qt5-base-5.4.0-3  qt5-declarative-5.4.0-3  qt5-script-5.4.0-3  qt5-svg
              qt5-x11extras-5.4.0-3  qt5-xmlpatterns-5.4.0-3  qtchooser-48-1  solid-
              threadweaver-5.6.0-1  upower-0.99.2-1  kate-14.12.2-2

Total Download Size:    33.42 MiB
Total Installed Size:  178.32 MiB

:: Proceed with installation? [Y/n] n
```

3. Vim (`http://www.vim.org/`) - It has a sharp, but not steep learning curve.

4. GNU Emacs (`https://www.gnu.org/software/emacs/`) has an absolutely insane learning curve, but is a wonderful editor once you spend 3 years learning how to use it.

1.5 Sage

Back when I was in high school, we had these crappy little calculators from a company called Texas Instruments. We weren't allowed to use anything beyond a crappy 1980's-era calculator, because our overlords feared that we would become too reliant on the machines to do our work.

They also tried to convince us that people in real life use these crappy TI calculators. Well, they're full of crap. No professional limits themselves to crappy 80's technology. Most people nowadays use computer algebra systems to do fancy calculations. These can generate graphs, solve equations, the whole 9 yards.

Most of these computer algebra systems are proprietary and very expensive (hundreds of dollars a year). However, there is one which is gratis and libre: Sage. It's just as good as any of the others. I won't provide an in-depth tutorial for using Sage; it's far too complicated. However, I will occasionally use Sage to generate graphs, or something.

The way you generate graphs in Sage is by inputting a number of commands to do so. You can put these commands into a file, and make programs with them. Any non-trivial use of Sage requires that you know a programming language called Python. I am not so cruel as to force you to learn two programming languages.

You don't need to have Sage installed, unlike Haskell, although I do recommend it. You can find instructions for installing Sage on their website (`http://wiki.sagemath.org/DownloadAndInstallationGuide`).

With every graph, I'll include the Sage code to generate the graph. To run the code, copy the code into a file (the file name will be at the bottom of the code listing), and then run `sage filename` in a terminal. (You'll need to be in the same working directory).

```
#!/usr/bin/env sage

print "Hello, LYSA readers!"
```

SageHelloWorld.sage

To run this (on Linux/BSD/OS X), you would save the file to `SageHelloWorld.sage`, then run `sage SageHelloWorld.sage` in a terminal to run it.

```
% sage SageHelloWorld.sage
Hello, LYSA readers!
```

To be more specific, I have that file saved to

`/home/pete/prj/lysa/en/book/graphs/SageHelloWorld.sage`

To run the file, I use the `cd` command (cd = change directory) to change my working directory to `/home/pete/prj/lysa/en/book/graphs`, then run the above command.

If you are on Windows, good luck!

1.6 Target audience

The explanation of why programming is useful is a good segue into discussing the target audience.

When I was first writing the book, I wrote it in an effort to strengthen my own understanding. So, the target audience was me. The very first versions of this book were about a abstractish branch of math called commutative algebra. Later on, it seemed more fitting to abstractly go over the basics of math. That's what the current version of the book does.

That doesn't answer the question: who is the target audience? Well, people who want to learn basic and intermediate algebra, and to learn why it's so interesting.

Most books treat math as a tool you can use for calculations. I treat math as a language you can use to express your ideas. That's the core difference. This book will hopefully give you an interest in math itself, rather than just a cursory knowledge of it.

With that in mind, my book is going to approach the topics much differently than other books on the same topic. I rely very much on abstraction and intuition.

1.7 Licensing

This book is libre[5]. You can copy this book and give it to your friend. You can even print it out and sell it to people.[6] If, for instance, you are a schoolteacher and want to use this for your class, you are free to edit it to your liking and give the modified copy to your students. The only string I attach is, you have to allow anyone to whom you give the book do the same thing (i.e. they have to be free to copy/modify/change your version). The details of this can be found in § A.

LYSA is licensed under the GNU Free Documentation License. § A contains the license. Please read the license; it's actually pretty comprehensible.

The source for this book can be downloaded at `https://gitlab.com/lysa/lysa/repository/archive.tar.gz`. If you are looking to contribute, it's probably best to clone the git repository. You can clone the git repository by running `git clone https://gitlab.com/lysa/lysa.git` in a terminal.

1.8 Conventions used throughout

You don't actually have to read this section, but it would be useful.

1. `Things in monospace` are either code snippets or commands to be run in a terminal. I have separate stylings for `terminal commands` and `inline code snippets`. That said, they are separate but equal, at least for the time being.

2. The § symbol refers to a section. So § 3.2 means "chapter 3, section 2".

3. Even though most of the writers are American, I still use the British convention of putting periods after quotation marks: "like this". The British

[5]*Libre* is a French word, which, translated to English, means *free* in the sense of liberty, as opposed to price. Think *free speech*, not *free beer*.

[6]There are some restrictions though, see § A.

convention is less ambiguous. If you see the American convention anywhere in this book, please report it.

4. I will often recommend software. However, I will not recommend any non-libre software, or any software that costs money.

5. "I" refers to me. "We" refers to both me and you, the reader. "You" refers to, you guessed it, the reader. It's the convention in academia to use the so-called "royal we", such as "we subtract 2 from both sides of the equation to obtain the result . . . ".

   Sometimes, we will accidentally use the royal we, out of habit. Crap, I just did it there! See? It's very difficult to avoid. Like any of the other conventions herein, if you see it broken, please report the error to the authors. You can use the bug tracker (`https://gitlab.com/lysa/lysa/issues/new`), or, if you don't want to make a (free and libre) GitLab account, you can email me at `peter@harpending.org`.

6. Oh yeah, sometimes I'll use `monospace` in things like URLs or emails for the sake of disambiguity.

7. Most of the authors use some version of Linux. Hence, when there are instructions for computer things (such as installing Haskell), I'll write instructions for Linux, because that's what I know. There are two solutions here:

   (a) You could try out Linux (it's gratis, and it's easy).

   (b) If you know how to do the thing on your OS, and there aren't instructions for your OS, you could write up instructions and add them to the book. If you don't know how to do that, you could bring it up in the bug tracker (`https://gitlab.com/lysa/lysa/issues/new`) or email me at `peter@harpending.org`.

8. If you see some number as a superscript in the middle of text: like this[7], then the number refers to a footnote. If the superscript number is in the middle of math, it's probably just math.

9. If there's some number in brackets, like this: [12], then it's a citation. If you're reading this as a PDF, you can actually click on the number, and

[7]Hey, you found me!

your PDF reader will take you to the relevant bibliography entry. Go ahead, check it out! I'll wait. You can do the same thing for footnotes and URLs.[8]

10. If you see something *in italics*, it's usually a vocabulary word. Often there will be a term with a section number next to it: *like this* (§ D.1), usually somewhere in § D. § D is a reference section, which has theorems, vocabulary, identities, stuff like that. So, the reference to a section in § D next to a term points to the relevant section in the appendix. Like all of the other references — citations, footnotes, URLs — you can click on the section title, and your PDF reader will take you there.

[8]Well, clicking the URL will open up your web browser, but you get the point

Chapter 2

Booleans, simple logic, and simple operators

Before we get into interesting content, you have to understand some stuff. This stuff is pretty easy. This will likely be the shortest and easiest chapter in the book.

You might think math is about dealing with numbers and pumping out formulas. Well, that's not what math is about. As said in § 1.4, it's about using math as a language to express your thoughts. Most people don't think about numbers all day; thus, we deal with things in math that aren't numbers.

In this next section, we're going to outline some basic rules for reasoning about things. You need to know these rules to do really cool stuff. Although, as you will (hopefully) see, these rules can be fun to toy around with on their own.

It's okay if you don't remember all of these rules. You can always find a list in § D.3.

2.1 Implications

The first thing you need to understand is the notion that "if x is true, then y is also true. But, if y is true, it's not necessarily true that x is false." As always, mathematicians are too lazy to write this stuff out by hand, so they have notation for it.

1. $a \implies b$ means that "a implies b". It doesn't necessarily mean that b implies a. It means that if a is true, then b is also true.

   If someone is decapitated, then they will die. So,

   $$\text{Decapitated} \implies \text{Dead}$$

   However, if someone is dead, it doesn't necessarily mean that they were decapitated. They could have been shot, or stabbed, or had a heart attack. There are endless possibilities.

2. $a \impliedby b$ is the same as writing $b \implies a$. It's sometimes convenient to use $a \impliedby b$ instead. $a \impliedby b$ should be read "a is implied by b".

3. When I strike through some mathematical operator, like this: $\not\Longrightarrow$, it means that you can semantically but "not" in front of whatever the operator says. So, $\not\Longrightarrow$ means "not implies". That doesn't make much grammatical sense in English, so "does not imply" might be better. Nonetheless, you get the point.

   Moving on from the example above:

   $$\text{Decapitated} \implies \text{Dead}$$

   If someone is decapitated, then they're also dead (at least within a few seconds). However, if someone is dead, it's not necessarily true that they were decapitated.

   $$\text{Decapitated} \not\Longleftarrow \text{Dead}$$

4. If something is not true, then I'll put a $\neg$ in front of it. So, if I want to say that a is false, then I'll write $\neg a$.

5. If I want to pose a question, I could just ask the question. For instance, "Is $\neg$Decapitated $\impliedby$ $\neg$Dead true?".

   However, that quickly becomes difficult, usually when there are multiple assertions in a mathematical expression, and you don't know which one

I'm asking about. Moreover, since I use the same font for text and math, if I have both, it might be hard to tell which is math and which is text. So, to help with ambiguity, I'll put a ? over the operator I'm asking about:

$$\neg\text{Decapitated} \overset{?}{\impliedby} \neg\text{Dead}$$

See, that's much easier.

6. Now, on to that question - Is "not decapitated" implied by "not dead". Well, let's think about it. If someone is not dead, then they couldn't have been decapitated, because if they were decapitated, then they would be dead. Therefore, if someone is not dead, then they weren't decapitated.

   That word jumble was probably confusing. Mathematicians don't like to be confused. I'll make it symbolic for you.

$$\begin{aligned} \text{Decapitated} &\implies \text{Dead} \\ &\Downarrow \\ \neg\text{Decapitated} &\impliedby \neg\text{Dead} \end{aligned}$$

   Okay, I just used a vertical arrow. I'm sure you can figure out what it means.

7. So, hopefully you agree that

$$\begin{aligned} \text{Decapitated} &\implies \text{Dead} \\ &\Downarrow \\ \neg\text{Decapitated} &\impliedby \neg\text{Dead} \end{aligned}$$

   However,

$$\begin{aligned} \text{Decapitated} &\implies \text{Dead} \\ &\overset{?}{\Uparrow} \\ \neg\text{Decapitated} &\impliedby \neg\text{Dead} \end{aligned}$$

   Hm, the question mark doesn't work so well there. Oh well! Anyway, the answer is actually yes. We can figure this out by learning a rule about $\neg$. Namely, that

$$\neg\neg a = a$$

In this case, we know

$$\neg\text{Decapitated} \impliedby \neg\text{Dead}$$

So, if we just “not” both sides, and flip $\implies$ to $\impliedby$,

$$\begin{gathered}\neg\neg\text{Decapitated} \implies \neg\neg\text{Dead}\\ \text{Decapitated} \implies \text{Dead}\end{gathered}$$

This is basically just the rule mentioned in #3. Yay, we learned something!

2.1.1 Exercises

Ex. 1 — $A \not\Longrightarrow B$

$$A \overset{?}{\Longrightarrow} \neg B$$

2.1.2 And and or

So, sometimes we need to combine two pieces of logic together. There are two ways we can do this - logical-or and logical-and.

I put logical- in front of them, because the mathematical meaning is slightly different than the colloquial meaning.

Mathematicians are lazy, so we don’t like to write “logical-and” whenever we want to say it, so instead we use the symbol $\wedge$.

$A \wedge B$ is true if (and only if) both A and B are true. If one of them is false, then the entire thing is false.

On the other side, we have logical-or. The symbol for logical-or is $\vee$. $A \vee B$ is true if either A or B is true, or if both of them are true. You could think of logical-or as being equivalent to the colloquial "and/or".

A	**B**	$\mathbf{A \wedge B}$	$\mathbf{A \vee B}$
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

This is pretty simple. If you're having trouble remembering which symbol is logical-and and which one is logical-or, remember that the logical-and symbol — $\wedge$ — looks vaguely like an A.

I'm going to introduce some new notation: the $\iff$ symbol.

$$(A \iff B) = (A \implies B) \wedge (A \impliedby B)$$

$\iff$ should be read as "if (and only if)". Sometimes I'll write the word iff — with two 'f's — that's the same as "if (and only if)".

In order to do the exercises (yes, there are exercises), you need to know these properties of $\implies$.

1. For all a, $a \implies a$.
2. For all a and b, and c, $((a \implies b) \wedge (b \implies c)) \implies (a \implies c)$. For this reason, we can write $a \implies b \implies c$ without any ambiguity.

I'm too lazy to type "For all" each time, so I'm going to use the $\forall$ symbol; it's a common symbol in math that means "for all". $\forall$ looks like an upside-down A, so it should be easy to remember.

Here are some properties you need to remember:

Cancellative property $\neg\neg a = a;\ \forall a$

> You know what, that's too hard. Instead from now on, instead of saying $a = b;\ \forall a, b$, I'm just going to write $a \equiv b$. Good?

Reflexive property $a \wedge a \equiv a$

Associative property $a \wedge (b \wedge c) \equiv (a \wedge b) \wedge c$

Commutative property $a \wedge b \equiv b \wedge a$

Distributive property $a \wedge (b \vee c) \equiv (a \wedge b) \vee (b \wedge c)$

De Morgan's first law $\neg (a \wedge b) \equiv \neg a \vee \neg b$

Reflexive property $a \vee a \equiv a$

Associative property $a \vee (b \vee c) \equiv (a \vee b) \vee c$

Commutative property $a \vee b \equiv b \vee a$

These values, true and false, are called *Booleans*. They are named after a mathematician named George Boole who studied them to extent.[3]

2.1.3 Exercises

Ex. 2 — $\neg (A \vee B) \overset{?}{\equiv} \neg A \wedge \neg B$
I'll spoil it for you. This is called *De Morgan's second law*, and it's true. You can prove it using the cancellative property and the first law.

Ex. 3 — Here's another one for you.

$$a \vee (b \wedge c) \overset{?}{\equiv} (a \vee b) \wedge (b \vee c)$$

You need the distributive property, as well as the proof from ex. 2.
While we are at it, these proofs can be found in § D.3, as well as the solutions to these problems.

Ex. 4 — $\neg A \implies B \implies \neg C$

$$C \overset{?}{\implies} A$$

Ex. 5 — $A \;\not\Longrightarrow\; B \;\Longrightarrow\; \neg C$

$$A \overset{?}{\Longrightarrow} C$$

Ex. 6 — $A \;\Longrightarrow\; B \;\Longleftarrow\; C$

$$A \overset{?}{\Longleftrightarrow} C$$

Ex. 7 — $A \wedge \neg(B \wedge (C \vee D)) \overset{?}{\equiv} A \wedge (B \vee C) \wedge (B \vee D)$

Chapter 3

Sets

In math, it's often useful to consider *collections* of objects. There are basically two types of collections: *sets* and *vectors*. Sets are unordered, and multiplicity doesn't matter. Vectors are ordered, and multiplicity does matter.

For instance, $\{3,4\}$ and $\{4,3\}$ are the same *set*, but $(3,4)$ and $(4,3)$ are different *vectors*.

Likewise, $\{20,38\}$ and $\{38,20,38,20,20,20,20,20\}$ are the same *set*. You guessed it, $(20,38)$ and $(38,20,38,20,20,20,20,20)$ are different *vectors*. Sets are — at least ostensibly — much more important. More importantly, they are much easier to understand.

Sets were first studied to extent by Georg Cantor, a German mathematician, in the second half of the nineteenth century. Back in his own day, the results Cantor found by studying sets were considered so thoroughly bizarre that many of his colleagues simply refused to believe that Cantor could be right. In the end, Cantor turned out to be right all along. His ideas can be found in any introductory text on mathematics—including this one.

You probably figured it out from above: the notation is $\{$Braces$\}$ for sets, and $($Parentheses$)$ for vectors.

If you can't remember whether to use braces {the curly things}, or parentheses (the round things), remember: *a **brace** is used to **set** a broken bone.* I don't have a horrible pun having to do with parentheses and vectors, and for that, I apologize.

3.1 Elements

Let's invent a set.

$$Q = \{7,7,9,5,10,1,6,6,2,10\}$$

There we go. Remember, order and multiplicity don't matter. But, for the sake of clarity, let's put the elements in order, and deduplicate them.

$$Q = \{1,2,5,6,7,9,10\}$$

Yay! You may have noticed that I slipped in the word *element* into the previous sentence. Objects in the set are called *elements*. Yay, we figured out what that word means!

It's too strenuous on our weak mathematical hands to write "10 is an element of Q", so instead we have the symbol $\in$. $\in$ is a very terrible attempt at drawing an E. If you can't remember what $\in$ is, think "$\in$lement of".[1]

So, I'm going to ask you a question:

$$11 \overset{?}{\in} Q$$

(See, I used that thing from earlier with the question mark. I told you it would help.) Well, the answer is no, 11 is not an element of Q. As always, mathematicians are too weak to write "11 is not an element of Q" every time they want to say it, so instead they write

$$11 \notin Q$$

By contrast,

[1] You better think this, because it took me 30 minutes to get the alignment right. So, you know, remember $\in$ this way, or else...

$$6 \in Q$$

What if we want to say "both 6 and 2 are elements of Q"? Well, again, we could write it out like:

$$(6 \in Q) \land (2 \in Q)$$

But that's too cumbersome, so instead we'll write

$$2,6 \in Q$$

But won't that get confusing? Only if we put parentheses or braces around 2 or 6.

$$\{2,6\} \in Q$$

That's confusing, don't do that (yet).

There's one more thing I need to go over, which is the null set - it's the simplest set, as it contains no elements. "Null set" takes too long to write, so we use $\varnothing$ instead.

3.2 Subsets and Supersets

Remember when I said $\{2,6\} \in Q$, and we were really confused? In case you don't remember, $Q = \{1,2,5,6,7,9,10\}$. $\{2,6\}$ is obviously not an element of Q. However, $\{2,6\}$ is *in* Q but it's not an element. It's weird. How do we express this notion?

The answer is with *subsets*. "sub" means "smaller", so a "subset" would be a "smaller set". A is a subset of B is all of the elements in A are also in B. The notation is $A \subseteq B$. Some people will read that as "A is contained in B".

Referring to the previous example,

$$\{2,6\} \subseteq Q$$

Wait, what is with the little line under the round half circley thing? So, actually, there are two types of subsets - *proper* and *improper*. $A \subseteq B$ is for improper subsets. $A \subset B$ is for proper subsets. What's the difference, then?

$A \subseteq B$ allows for the possibility that $A = B$. $A \subset B$ means that B is *strictly larger* than A; there are some elements in B that are not in A.

I've already defined $\forall$ in § 2.1.2. I'm now going to add another symbol, $\exists$, which means "exists". I mention $\forall$, because $\exists$ is used in the same context.

Anyway, back to business. I use $\subseteq$ for improper subsets, and $\subset$ for proper subsets. However, some people will use $\subset$ for improper subsets, and something like $\subsetneq$ or $\subsetneqq$ for proper subsets. You have to look out.

Here's something cool: $\varnothing$ has no elements, so it's a subset of every set.

$$\varnothing \subseteq A;\ \forall A$$

I'm sure you can figure this out, two sets are equal iff they have the same elements. $A \subseteq B$ means that B has all of the elements that A has. $A \supseteq B$ would mean that A has all of the elements of B. So if both of those are true, then $A = B$. That is:

$$A = B \iff A \subseteq B \wedge A \supseteq B$$

So, what did we learn?

1. There are unordered collections with no multiplicity, called *sets*.
2. There are ordered collections with multiplicity, called *vectors*.
3. Given an object o and a set A, you can ask $o \overset{?}{\in} A$.

4. You can pull smaller sets out of a set (I'll show you the mechanics below). The way to express that a set is "embedded" in another set, without being an element is with *subsets*. $A \overset{?}{\subseteq} B$ would be asking "are all of the elements in A also in B?". (The converse doesn't necessarily have to be true).

5. There's a pretty easy set which has no elements, called $\varnothing$. An interesting property of $\varnothing$ is that it is a subset of all other sets.

6. Two sets are equal iff they have the same elements.

3.3 Combining sets together

Next, we're going to talk about ways to combine two sets together. There are any number of ways to do this. However, the two most common ways are through *unions* and *intersections*.

The union symbol is pretty easy to memorize — it looks like a giant U: $\cup$. Think "$\cup$nion".[2].

If A and B are sets, then $A \cup B$ is the set of elements that are in either A or in B. That is:

$$A \cup B = \{\, x \in \mathcal{A};\ (x \in A) \vee (x \in B) \,\}$$

You might also remember this by the fact that the union symbol $\cup$ looks vaguely like the or symbol $\vee$.

What the hell is that notation? That's called a *class comprehension*. It's a hacky way to describe a set. Technically, it describes a *class*, which is different than a set. That said, at least until chapter 5, they only describe sets!

Remember earlier when I would show you the mechanics for defining arbitrary subsets of a set? Well, this is a common way to do it.

[2]You don't have to remember it this way, because $\cup$ is already aligned reasonably well with the letters. Thus I didn't have to spend 30 minutes getting the alignment correct. (See footnote 1.)

You should look at the expression in two pieces: before the semicolon and after the semicolon. The term before the semicolon explains what each element looks like, and defines it to be a member of an ambient set. In this case — and in most cases — the element will just be x, or a, or something of the like.

$$A \cup B = \{\, x \in \mathcal{A};\ (x \in A) \vee (x \in B) \,\}$$

Okay, cool. The right side of the semicolon lists conditions that must be true about the thing on the left. In this case, x must be in A, logical-or it must be in B.

The $\mathcal{A}$ is shorthand for "ambient set". It's a set such that $A, B \subseteq \mathcal{A}$. For reasons I can't quite explain yet, you need to specify that the element variable (x in this case) is an element of another set. The consequence of this is, a set comprehension can only be used to make a *subset* of another set. $\mathcal{A}$ is the shorthand for "there's another set out there, but I don't care what it is (and you shouldn't either)".

Can you think of another consequence?

Well, since we are defining the union using a notation which can only be used to describe subsets, we can only take the union of two sets if they are both subsets of some larger set! That's annoying. Oh well.

3.3.1 Back to business

Let's look at that definition of the union again:

$$A \cup B = \{\, x \in \mathcal{A};\ (x \in A) \vee (x \in B) \,\}$$

You know already that $\vee$ is commutative (the order doesn't matter), so you can write this

$$A \cup B = \{\, x \in \mathcal{A};\ (x \in B) \,\} \vee (x \in A)$$

You've probably figured out, any property of $\vee$ is also true of $\cup$

Reflexive property $a \cup a \equiv a$

Associative property $a \cup (b \cup c) \equiv (a \cup b) \cup c$

Commutative property $a \cup b \equiv b \cup a$

3.3.2 The intersection

The intersection is, you guessed it - what happens when we use $\wedge$ in the above definition instead of $\vee$. Can you guess what the symbol for "intersection" is? If you guessed $\cap$, then you are right!

$$A \cap B = \{\, x \in \mathcal{A};\ (x \in A) \wedge (x \in B) \,\}$$

Since $\wedge$ is also reflexive, associative, and commutative, the same properties exist for $\cap$.

Reflexive property $a \cap a \equiv a$

Associative property $a \cap (b \cap c) \equiv (a \cap b) \cap c$

Commutative property $a \cap b \equiv b \cap a$

You can actually reduce the definition of $\cap$ to

$$A \cap B = \{\, x \in A;\ (x \in B) \,\}$$

(I do this in the proofs a lot)

The standard example of unions and intersections is to use "Venn diagrams". I drew some myself in fig. 3.1.[3]

[3]It's a tradition from Learn You a Haskell that each book include poorly-drawn explanatory graphics by the author.

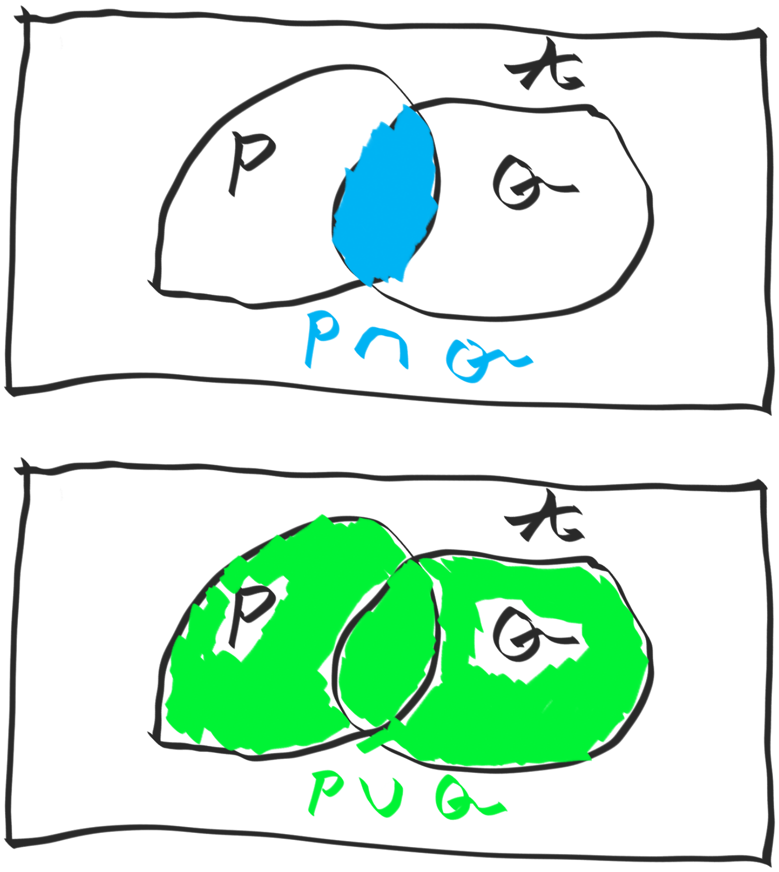

Figure 3.1: The Venn diagrams. That symbol in the left circle is supposed to be a Q. My handwriting sucks, deal with it.

3.3.3 More on the comprehension

I use the comprehension quite a lot (other books do as well), so I should at least take a subsubsection to explain it.

I'm going to list some *axioms* about comprehensions, which you would do well to remember:

Let p be a unary predicate. Basically, it takes the x, and determines whether or not that x is to be included in the comprehended set.

1. $\{x \in A;\ x \in \{y \in B; p(y)\}\} \equiv \{x \in A;\ x \in B \wedge p(x)\}$
2. $\{x \in A;\ x \notin \{y \in B; p(y)\}\} \equiv \{x \in A;\ x \notin B \vee \neg p(x)\}$

3.3.4 Back to unions and intersects

The final piece of the puzzle is this:

$\mathbf{A} \cap (\mathbf{B} \cup \mathbf{C}) \equiv (\mathbf{A} \cap \mathbf{B}) \cup (\mathbf{A} \cap \mathbf{C})$

Proof.

$$\begin{aligned}
A \cap (B \cup C) &:= \{x \in A;\ x \in B \vee x \in C\} \\
(A \cap B) \cup (A \cap C) &:= \{x \in \mathcal{A};\ (x \in A \wedge x \in B) \vee (x \in A \wedge x \in C)\} \\
&:= \{x \in \mathcal{A};\ x \in A \wedge (x \in B \vee x \in C)\} \\
&:= \{x \in A;\ x \in B \vee x \in C\}
\end{aligned}$$

□

You're probably wondering what that white box is. It's mathematical shorthand for "I'm done proving stuff".

I can't faithfully discuss sets any more without talking about functions. So, you get a lucky break! No exercises (because the fun exercises require that you know about functions.)

I would tell you to look in § D.4 if you don't remember stuff. However, § D.4 assumes you have read both § 4 and § 5. So, in conclusion, don't look at § D.4 quite yet.

Chapter 4

Functions

As promised, this chapter discusses functions.

So, what is a function?

So far, we've been dealing with *values* - like 2, $\{3,2,5\}$, and 90. They are static. Static things are fine, but they aren't very interesting. It's much more interesting to examine *changing things* — more specifically, things that change *predictably* and *transparently*.

Enter the *function* (§ D.5). It's a mathematical construct. A function takes some input, and maps it to an output. Functions are sometimes referred to as *mappings* or *morphisms*.

Let's look at a simple function, which takes a number and adds 2 to it

$$\begin{aligned} &f : \mathbb{Z} \to \mathbb{Z} \\ &f = \lambda\,(x) \to x + 2 \end{aligned}$$

Pretty simple, right? Okay, so what happens when we send 28 to f?

$$\begin{aligned} f\,(x = 28) &= \lambda\,(x = 28) \to 28 + 2 \\ &= 30 \end{aligned}$$

Alternatively, since it's obvious we're working with x:

$$\begin{aligned} f(28) &= 28+2 \\ &= 30 \end{aligned}$$

This highlights an important property of functions: *referential transparency* (§ D.5). If you send a function the same input twice, you should get the same output both times. That is,

$$a = b \implies f(a) = f(b)$$

Note that the opposite is not always true:

$$a = b \not\Longleftarrow f(a) = f(b)$$

(If that is true, then the function is *injective*).

Using the lambda — λ — is common when I am using a function without giving it a name. However, usually I will use this notation:

$$\begin{aligned} &f : \mathbb{Z} \to \mathbb{Z} \\ &f(x) = x+2 \end{aligned}$$

The whole $f : \mathbb{Z} \to \mathbb{Z}$ thing should be pretty obvious. If not, it means that f is a function that takes a member of $\mathbb{Z}$ (the whole numbers, both negative and positive), and takes it to another member of $\mathbb{Z}$. Other people might use the notation

$$\mathbb{Z} \xrightarrow{f} \mathbb{Z}$$

That notation is undoubtedly easier to understand. However, as we'll see, that notation quickly becomes unfeasible.

With this in mind, if $f : A \to B$, then A is the *domain* of f, written $\mathbf{dom}(f)$, and B is the *codomain* of f, written $\mathbf{codom}(f)$.

With regard to the function we were just discussing

$$\begin{aligned} &f : \mathbb{Z} \to \mathbb{Z} \\ &f(x) = x + 2 \end{aligned}$$

$\mathbb{Z}$ is both the domain and the codomain. If this is the case, then we say that f is a *closure*.[1] f is "closed under $\mathbb{Z}$", meaning that things can't use f to escape from $\mathbb{Z}$. f is closed.

If you can't remember all of these terms, don't worry, they are all listed in § D.5.

4.0.5 Functions with multiple arguments

Remember my explanation of vectors earlier? If not, vectors are like sets, but order and repetition matter.

Here's a function that takes two arguments, and adds them to each other.

$$\begin{aligned} &f : (\mathbb{Z}, \mathbb{Z}) \to \mathbb{Z} \\ &f(x, y) = x + y \end{aligned}$$

Pretty easy to understand, right?

If you haven't figured it out from the context, the inputs to the function are called the *arguments*.

Here's a similar function that takes three arguments and adds them to each other

$$\begin{aligned} &f : (\mathbb{Z}, \mathbb{Z}, \mathbb{Z}) \to \mathbb{Z} \\ &f(x, y, z) = x + y + z \end{aligned}$$

[1] There's a programming language called Clojure, whose name is a pun on this concept.

You can name your function anything you want, same with the arguments (it doesn't have to be f). It's just a common convention, which you don't have to follow.

What if I want to add a bunch of things together?

Good idea!

$$\begin{array}{l} f : (\mathbb{Z}, \mathbb{Z}, \ldots, \mathbb{Z}) \to \mathbb{Z} \\ f(x_1, x_2, x_3, \ldots, x_n) = x_1 + x_2 + x_3 + \cdots + x_n \end{array}$$

That however isn't ideal, because we have no guarantee that the arguments in the ... are actually integers. How about we have a *set* of integers, and we just take the sum? This has the added benefit of less typing

$$\begin{array}{l} f : \mathbf{Set}(\mathbb{Z}) \to \mathbb{Z} \\ f(s) = \sum s \end{array}$$

So,

$$\begin{aligned} f(\{1,2,3,4,5\}) &= \sum\{1,2,3,4,5\} \\ &= 1+2+3+4+5 \\ &= 15 \end{aligned}$$

4.0.6 Eta-reductions

Mathematicians like to make themselves look smart. One such way is to invent fancy terms for simple things. One such term is the η-reduction.

Let's look at that function we just had

$$\begin{array}{l} f : \mathbf{Set}(\mathbb{Z}) \to \mathbb{Z} \\ f(s) = \sum s \end{array}$$

Notice that we are repeating s on both sides of the equation. It would seem much simpler, and just as clear, to write:

$$\begin{aligned} &f : \mathbf{Set}(\mathbb{Z}) \to \mathbb{Z} \\ &f = \Sigma \end{aligned}$$

That's all an η-reduction is: if you see an extraneous argument, you remove it to make things simpler. As long as we have the signature — the $f : \mathbf{Set}(\mathbb{Z}) \to \mathbb{Z}$ thing – it's pretty clear what f does. This is a prime example of mathematicians being both lazy and pretentious at the same time: a practice designed to allow us to be lazier, to which mathematicians have assigned a ridiculous name to make it sound hard.

What the hell is η?

η is the Greek letter eta; it's pronounced "eight-uh".

The ancient Greeks were too dumb to comprehend the concept of "eight". Every time someone brought it up, they said "uh" immediately thereafter. The sound "eight-uh" became so common that they decided to make it a letter.

The Greeks' poor comprehension of simple mathematics remains to this day, and is largely the reason for their current financial crisis.[8]

If you ever take a physics course, you will undoubtedly notice that Greek letters are used frequently in physics. This is the physicists way of subtly hinting that they actually have no idea what they are talking about, and pleading for help from the mathematicians.

4.0.7 Other lambda calculi

This entire idea where you take simple concepts and make them sound really fancy is called λ *calculus* (§ D.6). If you hear people talk about "calculus", they are talking about something else, not this. Nobody is pretentious enough to actually talk about λ calculus.

Anyway, here's a brief summary of λ calculus. You can find this in § D.6, too.

You might want to brush up on your Greek alphabet. I have a nice table of Greek letters in § D.7.

λ **abstraction** A way to write a function: $\lambda\,(x,y) \to x+y$

α **conversion** Changing the names of the arguments. For instance, you can write the above function as

$$\lambda\,(a,b) \to a+b$$

β **reduction** Partially calculating a result. For instance

$$\lambda\,(2,y) \to 2+y$$

Can be β reduced to

$$\lambda\,(y) \to 2+y$$

η **conversion** Removing or adding extraneous free arguments. The last function

$$\lambda\,(2,y) \to 2+y$$

Can be η *reduced* to

$$2+$$

Which could then be η *abstracted* to

$$\lambda\,(2,\kappa) \to 2+\kappa$$

4.1 Currying

We sort of got side-tracked by toying around with sets and making fun of physicists. Hopefully that introduction introduced you to the basic concept of a function, and let you know that they can take multiple arguments

Let's look at that function again:

$$\begin{aligned} &f : (\mathbb{Z},\mathbb{Z},\mathbb{Z}) \to \mathbb{Z} \\ &f(x,y,z) = x+y+z \end{aligned}$$

What if you wanted to bind $x = 3$, but leave the rest "free"?

$$\begin{aligned} &f : (\mathbb{Z},\mathbb{Z},\mathbb{Z}) \to \mathbb{Z} \\ &f(x=3,y,z) = 3+y+z \end{aligned}$$

Okay, cool. We now have another function:

$$\begin{aligned} &f(3) : (\mathbb{Z},\mathbb{Z}) \to \mathbb{Z} \\ &f(3,y,z) = 3+y+z \end{aligned}$$

So, actually, instead of needing 3 integers to do its job, f only needed one. However, instead of spitting out another integer, it spit out a function. So, we could write f's signature as:

$$\begin{aligned} &f : \mathbb{Z} \to ((\mathbb{Z},\mathbb{Z}) \to \mathbb{Z}) \\ &f(x,y,z) = x+y+z \end{aligned}$$

Okay, that's sort of weird and unintuitive. Let's try writing f differently:

$$\begin{aligned} &f : \mathbb{Z} \to ((\mathbb{Z},\mathbb{Z}) \to \mathbb{Z}) \\ &f = \lambda(x) \to (\lambda(y,z) \to x+y+z) \end{aligned}$$

Let's look at the second half of that:

$$\lambda(y,z) \to x+y+z : (\mathbb{Z},\mathbb{Z}) \to \mathbb{Z}$$

(This assumes that we know what x is)

Let's try splitting this up again:

$$\lambda\,(y) \rightarrow (\lambda\,(z) \rightarrow x+y+z) : \mathbb{Z} \rightarrow (\mathbb{Z} \rightarrow \mathbb{Z})$$

You give this function a value for y, and instead of giving you a value, it gives you another function, hence the signature $\mathbb{Z} \rightarrow (\mathbb{Z} \rightarrow \mathbb{Z})$.

Let's plug this back into f:

$$f : \mathbb{Z} \rightarrow (\mathbb{Z} \rightarrow (\mathbb{Z} \rightarrow \mathbb{Z}))$$
$$f = \lambda\,(x) \rightarrow (\lambda\,(y) \rightarrow (\lambda\,(z) \rightarrow x+y+z))$$

So, instead of f taking three integers, it now only takes one, but spits out a function, which in turn spits out a function, which spits out an integer.

This idea of making a function into a chain of functions is called "Currying".[4] It's named after a dead mathematician named Haskell Curry (ca. 1900-1982), who developed the technique. The programming language Haskell is also named after Mr. Curry.

Getting back to that function, those parentheses are somewhat burdensome, let's get rid of them

$$\begin{aligned} f &: \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{Z} \rightarrow \mathbb{Z} \\ f &= \lambda\,(x) \rightarrow \lambda\,(y) \rightarrow \lambda\,(z) \rightarrow x+y+z \\ f\,(x,y,z) &= x+y+z \end{aligned}$$

That's much easier to read. It should be understood that the parentheses are right-associative: the parentheses "associate" rightward — i.e. it's $a \rightarrow (b \rightarrow (c \rightarrow d))$, not $((a \rightarrow b) \rightarrow c) \rightarrow d$.[13]

That's Currying for you.

4.1.1 Piecewise functions

As a random aside, I'm going to introduce you to the *piecewise function*. It's a function whose definition changes based on the input.

$$\begin{aligned} q &: \mathbb{N} \to \mathbb{Z} \\ q(x) &:= \begin{cases} x \text{ is even} & \to & \frac{x}{2} \\ x \text{ is odd} & \to & `\left(\frac{x+1}{2}\right) \end{cases} \end{aligned}$$

Let's look at $q(0)$: 0 is even, so $q(0) = \frac{0}{2} = 0$.

Let's make a table:

x	$q(x)$	$q(x)$ reduced
0	$0 \div 2$	0
1	$`((1+1) \div 2)$	$`1$
2	$2 \div 2$	1
3	$`((3+1) \div 2)$	$`2$
4	$4 \div 2$	2
5	$`((5+1) \div 2)$	$`3$
6	$6 \div 2$	3
7	$`((7+1) \div 2)$	$`4$
8	$8 \div 2$	4

Hopefully you get this. It's pretty simple.

4.1.2 Vocabulary

I've been sort of dropping these vocabulary terms throughout the beginning of the chapter. That said, I'll list them here, so you know where they are. (They're also in § D.5).

1. All functions are *transparent* — $a = b \implies f(a) = f(b)$
2. If $f : A \to B$, then A is the *domain* of f and B is the *codomain* of f.

3. If $f : A \to B$, and there are no two distinct elements of A that map to the same thing in B, then f is *injective*.

$$\begin{array}{l} f : A \to B \\ \nexists\,(a,b)\,;\ a,b \in A \wedge a \neq b \wedge f(a) = f(b) \iff f \text{ is injective} \end{array}$$

4. If $f : A \to B$, then the elements in B that can be expressed as $f(x)\,;\ x \in A$ form the *image*.

$$\begin{array}{l} f : A \to B \\ \operatorname{im}(f) = \{\, f(x) \in B;\ x \in A \,\} \end{array}$$

5. If the image of a function is equal to its codomain, then the function is *surjective*.

$$\begin{array}{l} f : A \to B \\ B = \{\, f(x) \in B;\ x \in A \,\} \iff f \text{ is surjective} \end{array}$$

6. If a function is both injective and surjective, then it is *bijective*.

7. Some functions have *inverses*. That is, if

$$f : A \to B$$

$$\begin{array}{l} \operatorname{arc}(f) : B \to A \\ \operatorname{arc}(f,x) = x;\ \forall x \in A \end{array}$$

Remember that, because of currying, $\operatorname{arc}(f,x) = \operatorname{arc}(f)(x)$. That is:

$$\begin{array}{rcl} f & : & A \to B \\ \operatorname{arc} & : & (A \to B) \to B \to A \\ \operatorname{arc}(f) & : & B \to A \end{array}$$

If a function has an inverse, it is said to be *invertible*.

8. If a function is invertible, then the image of the inverse is called the *preimage*.

4.1.3 Exercises

Ex. 8 — I knew you were going to just gloss over those, so I made a really hard (i.e. fun) problem: prove that a function is invertible if (and only if) it is bijective. This is a very difficult proof, but you really need to understand it.

Chapter 5

More stuff about sets (and functions)

By now, you hopefully have some idea into the basic intuition behind sets and functions. Moreover, you've proven some cool stuff about them – for instance, you proved that a function is invertible iff it is bijective.

That's kind of cool, right? It's much easier to verify that a function is bijective than it is to find its inverse. So, right off the bat, you can see if a function is invertible without trying to invert it. Despite what you may think, a common problem in math is to find inverse functions.

Speaking of functions, I'm going to define a really simple function:

$$\begin{aligned} &\mathrm{id} : a \to a \\ &\mathrm{id}(x) = x \end{aligned}$$

That function is about as simple as functions get. It's not a very interesting function, but it's handy when defining things.

This is a slightly less simple function, but nonetheless important

$$\begin{aligned} &\mathrm{flip} : (a \to b \to c) \to b \to a \to c \\ &\mathrm{flip}(f, x, y) = f(y, x) \end{aligned}$$

It takes a function, f, which takes an a and a b, and then returns another function with the arguments flipped. You won't usually see $\text{flip}\,(\lambda\,(x,y) \to x-y, 3, 5)$ floating around. Instead

$$\begin{array}{rcl} f & : & p \to q \to r \\ \text{flip}\,(f) & : & q \to p \to r \end{array}$$

Here's an infix operator:

$$\begin{array}{l} \circ : (b \to c) \to (a \to b) \to a \to c \\ (f \circ g)\,(x) = f\,(g\,(x)) \end{array}$$

We're going to look at some more stuff with sets.

5.1 Set subtraction

First off, we have "set subtraction". This is sometimes called the "relative complement".

$$P \setminus Q = \{x \in P;\; x \notin Q\}$$

$P \setminus Q$ is all of the elements in P that are not in Q.

What would $P \setminus P$ be, then? Well, $\varnothing$, of course, right!

$$P \setminus P = \{x \in P;\; x \notin P\} = \varnothing$$

Well, that looks like a contradiction, doesn't it? Well, sort of. It's a contradiction if you assume that P is nonempty — if you assume that there is some element in P — if an element is both in P and not in P, that would be madness! But, if you realize that the comprehension is the "set of objects satisfying the condition",

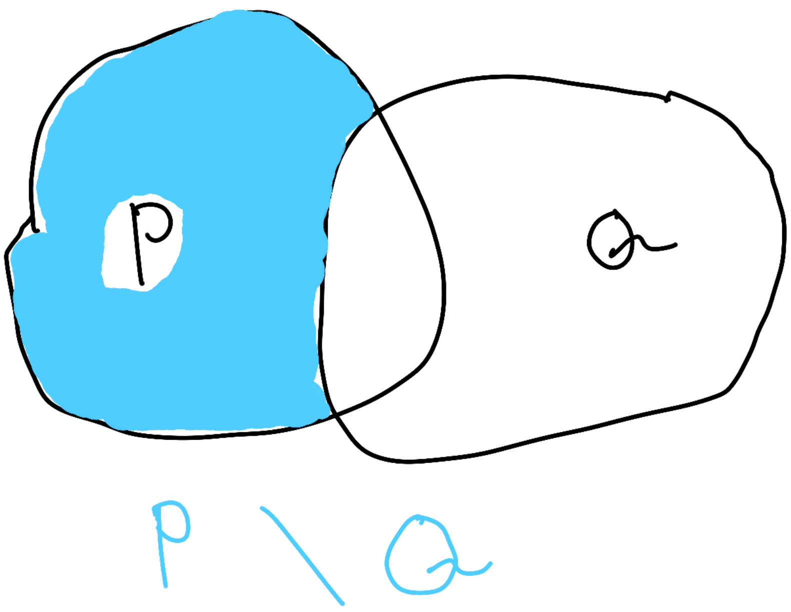

Figure 5.1: Set subtraction

then you don't encounter a contradiction. $P \setminus P$ is the set of all objects in P that are not in P: there are no elements satisfying this condition, thus $P \setminus P \equiv \varnothing$.

Figure 5.1 explains this idea graphically.

5.1.1 Complement

If $P \subset \mathcal{A}$, then $\mathcal{A} \setminus P$ is called the "*complement* of P with respect to $\mathcal{A}$". Yes, that's compl*e*ment with an 'e', not compl*i*ment, with an 'i'.

I drew another diagram to illustrate the complement in fig. 5.2.

In these exercises, you are going to prove every identity there is about sets.

5.1.2 Exercises

Ex. 9 — $A \setminus (B \cap C) \equiv (A \setminus B) \cup (A \setminus C)$

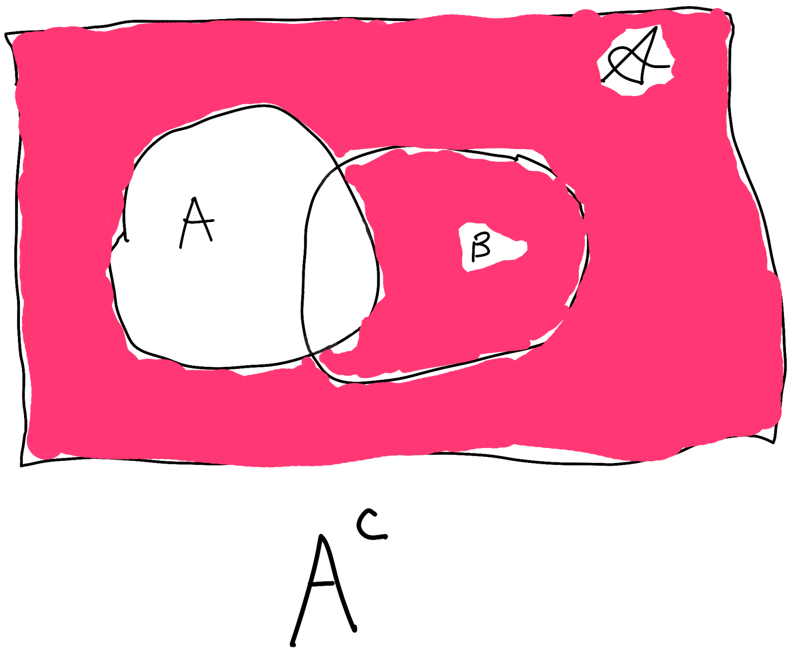

Figure 5.2: The complement, illustrated graphically. I tried to draw an $\mathcal{A}$ in the upper right corner. It turned out terribly.

Ex. 10 — $A \setminus (B \cup C) \equiv (A \setminus B) \cap (A \setminus C)$

Ex. 11 — $A \setminus (B \setminus C) \equiv (A \setminus B) \cup (A \cap C)$

Ex. 12 — $(A \setminus B) \cap C \equiv (A \cap C) \setminus B \equiv A \cap (C \setminus B)$

Ex. 13 — $(A \setminus B) \cup C \equiv (A \cup C) \setminus (B \setminus C)$

Ex. 14 — $A \setminus A \equiv \varnothing$

Ex. 15 — $A \setminus \varnothing \equiv A$

Ex. 16 — $\varnothing \setminus A \equiv \varnothing$

Ex. 17 — $A \cap (B \cup C) \equiv (A \cap B) \cup (A \cap C)$

Ex. 18 — $A \cup (B \cap C) \equiv (A \cup B) \cap (A \cup C)$

Ex. 19 — $(A^c)^c \equiv A$

Ex. 20 — $(A \cap B)^c \equiv A^c \cup B^c$

Ex. 21 — $(A \cup B)^c \equiv A^c \cap B^c$

5.2 Cartesian products

The next thing we need to go over is a *Cartesian product* - it's basically a way to double up on a set. So, say we have a set A, and another set B, the Cartesian product is the set of all 2-vectors, where the first element is from A, and the second element is from B.

$$\begin{array}{l} \times : \mathbf{Set}\,(a) \to \mathbf{Set}\,(b) \to \mathbf{Class}\,(a,b) \\ A \times B := \{\,(x,y)\,;\; x \in A \wedge y \in B\,\} \end{array}$$

It's the class of all pairs of the elements in either set. It is actually a set, too, but I haven't taught you enough to prove that. I also haven't taught you what a class is. So, for the time being, know that the Cartesian product produces a set, but at the same time you don't know that.

Let's see some examples!

$$\begin{array}{rcll} \{1,2,3\} \times \{4,5,6\} & = & \{ & (1,4) \\ & & , & (1,5) \\ & & , & (1,6) \\ & & , & (2,4) \\ & & , & (2,5) \\ & & , & (2,6) \\ & & , & (3,4) \\ & & , & (3,5) \\ & & , & (3,6) \\ & & \} & \end{array}$$

Alright, remember earlier when I told you to install Haskell? Well, if you didn't, do it now: § 1.4.

```
% ghci
GHCi, version 7.8.4: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
Prelude>
```

GHCi is the Glasgow Haskell Compiler, interactive. It's an interactive interpreter for Haskell.

If you want to change your prompt to something other than `Prelude>`, then write something like this:

```
Prelude> :set prompt "ghci: "
ghci:
```

If you want to do this permanently, add `:set prompt "ghci:  "` to `~/.ghc/ghci.conf`.

I didn't just have you open up GHCi for no good reason. It's really easy to play with these Cartesian products in GHCi. This way, you can experiment.

Remember the definition of the Cartesian product:

$$A \times B := \{\,(x,y)\,;\, x \in A \wedge y \in B\,\}$$

Let's try that example out. Math doesn't directly translate into Haskell. Some of the syntax is a bit different.

```
ghci: [(a,b) | a <- [1,2,3], b <- [4,5,6]]
[(1,4),(1,5),(1,6),(2,4),(2,5),(2,6),(3,4),(3,5),(3,6)]
it :: (Num t1, Num t) => [(t, t1)]
```

That last line probably doesn't show up for you. It's just telling us the type of our expression. To have it show up automatically for you, run `:set +t`, or add it to `~/.ghc/ghci.conf`.

There are a number of ways to actually work out the mechanics of the product. The simplest, and easiest way, is through *recursion*. So, let's go over the mechanics of $\{1,2,3\} \times \{4,5,6\}$. You take the first element of the first set, in this case, 1, and take the Cartesian product of $\{1\}$ and $\{4,5,6\}$:

$$\{1\} \times \{4,5,6\} = \{(1,4),(1,5),(1,6)\}$$

That's unreadable

$$\begin{array}{rcll} \{1\} \times \{4,5,6\} & = & \{ & (1,4) \\ & & , & (1,5) \\ & & , & (1,6) \\ & & \} & \end{array}$$

Pretty easy. Then you do the same thing with the second element.

$$\begin{array}{rcll} \{2\} \times \{4,5,6\} & = & \{ & (2,4) \\ & & , & (2,5) \\ & & , & (2,6) \\ & & \} & \end{array}$$

You guessed it!

$$\begin{array}{rcll} \{3\}\times\{4,5,6\} & = & \{ & (3,4) \\ & & , & (3,5) \\ & & , & (3,6) \\ & & \} & \end{array}$$

Then you take the union:

$$\begin{array}{rccll} \{1,2,3\}\times\{4,5,6\} & = & \bigcup & \{ & \{1\}\times\{4,5,6\}, \\ & & & , & \{2\}\times\{4,5,6\}, \\ & & & , & \{3\}\times\{4,5,6\}, \\ & & & \} & \end{array}$$

I'm going to have to display the intermediate result in a separate thing, because it's just awful if I try to smush it in with the rest:

$$\begin{array}{ll} \{ & (1,4) \\ , & (1,5) \\ , & (1,6) \\ \} & \end{array} \bigcup \begin{array}{ll} \{ & (2,4) \\ , & (2,5) \\ , & (2,6) \\ \} & \end{array} \bigcup \begin{array}{ll} \{ & (3,4) \\ , & (3,5) \\ , & (3,6) \\ \} & \end{array}$$

Which evaluates to

$$\begin{array}{rcll} \{1,2,3\}\times\{4,5,6\} & = & \{ & (1,4) \\ & & , & (1,5) \\ & & , & (1,6) \\ & & , & (2,4) \\ & & , & (2,5) \\ & & , & (2,6) \\ & & , & (3,4) \\ & & , & (3,5) \\ & & , & (3,6) \\ & & \} & \end{array}$$

Okay, great. I hope you understand this so far. What happens when we take the product of a set with itself?

$$\begin{array}{rcll}
\{1,2,3\} \times \{1,2,3\} & = & \{ & (1,1) \\
 & & , & (1,2) \\
 & & , & (1,3) \\
 & & , & (2,1) \\
 & & , & (2,2) \\
 & & , & (2,3) \\
 & & , & (3,1) \\
 & & , & (3,2) \\
 & & , & (3,3) \\
 & & \} &
\end{array}$$

The Cartesian product of a set A with itself is usually denoted A^2 instead of $A \times A$. Like I said, we like to be lazy.

Appendices

Appendix A

GNU Free Documentation License

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this

License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with . . . Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Appendix B

How to learn math

Now that that's all out of the way, let's talk a little about math. When this chapter is over, we're going to dive right in to proving a bunch of things you already know to be true. We feel that without a little explanation, these proofs may leave you a little lost or confused. We'll save the explanation of the proofs for later, but right now, we're going to talk about how to actually learn math, and the proofs are a great example.

Most people's experience with math is through their primary and possibly secondary education, which is or was a dreary affair in general, and math probably even moreso, unless you're one of the lucky few. By lucky few, we don't mean those wizards with a sort of inherent ability to do math–the first thing you need to know about learning math is that math is for everyone with a brain–that's you, right? You see, your brain is a pattern recognition engine, and that's all math is: the study of patterns. Unlike reading or history, your body comes with a biological imperative to know math. There's some really great brain studies on the topic, but that's boring, and I said we're already done with the boring part, so let's move on.

In that last paragraph, we presented what we hold to be the proper answer to 'what is math': the study of patterns. This is completely different from most people's interaction with math: in primary school, we are taught how to apply four operations to solve math problems. You're given something about two trains leaving a station and going different speeds and different directions and yadda yadda yadda and before you know it your teacher turned everything into a math problem and it all seemed so forced–a layer on top of what was intuitive, and made everything complicated. We agree–this is a counterintuitive approach to math, and it makes math very confusing and disconnected. Math is just the study of patterns. That is, math is not so much a way to solve a set of problems that exist in a sphere apart from what is natural, but a way to understand what's going on in the world around us. When you learn math, you should think of it as a science–another level of detail in the amazing world we live in.

That's how this book is written. It's written to reflect that math is a single unified study. While you're reading it, try to think of how what you're learning clarifies or refines early material. This is a big deal to us, because one thing we dislike most about the standard way of learning math is that at some point in everyone's math career, they learn they were taught something that wasn't actually true. We want to avoid that.

Appendix C

Philosophy and/or FAQ

by Peter Harpending <peter@harpending.org>

This book is written with a certain philosophy in mind. Explaining my philosophy will answer a number of questions I am often asked.

First, I'll start with the license. he license I chose for this book is the GNU Free Documentation License (FDL). Again, "free" refers to freedom, not price. The FDL is similar in spirit to another license, the Creative Commons Attribution-ShareAlike License (CC-SA). CC-SA is much more popular than the FDL, mostly because it is much more general (e.g. you could distribute a painting under CC-SA, but not the FDL). The CC-SA license and the FDL are both "copyleft" licenses, in that they require that derivative works be licensed under the same license.

So, why did you go with the FDL instead of CC-SA?

Simply put, the CC-SA license is too general to fit our purposes. The FDL is specifically designed for reference texts, so it has a clause requiring that the work be made available in source form. The CC-SA has no such requirement.

To go on about this, I need to define "freedom" in academic works. This is a modified version of the GNU Project's definition of free software (`https://gnu.org/philosophy/free-sw.html`). In my view, an academic work is free if you have:

- The freedom to use the digital document as you wish, for any purpose (freedom 0).
- The freedom to reproduce exact copies of the document in any medium. (e.g. print the book). (freedom 1)
- The freedom to distribute and/or sell exact copies of the document to whomever you choose, in any medium. (freedom 2).
- The freedom to modify your own copy of the book. Access to the source is a precondition for this. (freedom 3)
- The freedom to reproduce modified copies of the document in any medium. (e.g. print the book). (freedom 4)
- The freedom to distribute and/or sell exact copies of your modified version to others, in any medium (freedom 5).

To guarantee freedom for everyone, we unfortunately have to restrict freedom 3 a little bit. You can't modify the book in such a way that would restrict others' freedom. Such ways would include

- Implementing digital restrictions management, or DRM.
- Removing the license.
- Releasing your modified version under a different license.

I suppose someone who ran in a different clique would wonder why people put up so much fuss about something as silly as a license. The license explains exactly what the reader can and can't do with my work. When my objective is freedom, allowing everyone the maximum quantity of freedom requires some copyright trickery, hence the 10-page-long license.

You would think that this freedom would be implicit in any academic work. Unfortunately, that's not the case.

An extreme instance of this trope of freedom restriction was a wonderful company named Myriad Genetics. Myriad Genetics attempted to patent human genes. Fortunately, the United States Supreme Court struck down this class of patents in a unanimous decision (`http://www.supremecourt.gov/opinions/12pdf/12-398_1b7d.pdf`). For those of you who aren't American, unanimous US Supreme Court decisions are incredibly rare. Myriad Genetics is headquartered within walking distance from my house, so this was big news in my area.

Anyway, the point of all this is, freedom is important, especially in academic works. Part of the reason I wrote this book is that there are very few free textbooks.

To put it another way, it is of no benefit for the work to be nonfree. If the work is free, I, as a writer, benefit from people giving me feedback, and improving upon my work. You, as a reader, benefit from the freedom. The only people who don't benefit are distributors (e.g. a publisher). However, in the age of the internet, the need for a for-profit publisher isn't exactly clear.

To be clear, it is perfectly okay for a publisher to publish this book, and to attempt to profit off of it. However, the publisher wouldn't have the traditional nonfree monopoly over the book, which might discourage a publisher.

Appendix D

Identities, theorems, and the like

This appendix just lists identities, theorems, and stuff like that. It's for reference, not for reading.

D.1 Equality

D.1.1 Properties

Reflexive property $a \equiv a$

Commutative property $(a = b) \iff (b = a)\,;\ \forall a, b$

Transitive property $(a = b) \wedge (b = c) \implies (a = c)\,;\ \forall a, b, c$

D.1.2 Notation

$\mathbf{a} = \mathbf{b}$ means that a and b are the same thing.

$\mathbf{a} \equiv \mathbf{b}$ means that $a = b$, for all a and b. $a \equiv b$ should be read "a is identically equivalent to b".

a := b means that a is defined to be equal to b. In practice, this is the same as $\equiv$, but is semantically different.

D.2 Implications

Reflexive property $a \implies a; \forall a$

Transitive property $(a \implies b) \wedge (b \implies c) \implies (a \implies c); \forall a, b, c$

Negation $(a \implies b) \iff (\neg a \impliedby \neg b); \forall a, b$

D.3 Booleans

Definition A *Boolean* is a value of either true or false. The study of Booleans is called *Boolean algebra*. The rules for Booleans also work for propositions. The set of Booleans is often referred to as $\mathbb{B} = \{\text{True}, \text{False}\}$

Logical-and $a \wedge b$ is pronounced "a logical-and b". It is true iff a and b are both true.

$$\wedge : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$$

Logical-or $a \vee b$ is pronounced "a logical-or b". It is true if one or more of a and b are true.

$$\vee : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$$

Logical-not $\neg a$ is pronounced "logical-not a". $\neg$ takes true to false, and false to true.

$$\neg : \mathbb{B} \to \mathbb{B}$$

Cancellative property $\neg \circ \neg \equiv \text{id}$

Nomenclature Booleans are named after George Boole, who was the first to study them to any extent.

D.3.1 Logical-and

Reflexive property $a \wedge a \equiv a$

Associative property $a \wedge (b \wedge c) \equiv (a \wedge b) \wedge c$

Commutative property $a \wedge b \equiv b \wedge a$

Distributive property $a \wedge (b \vee c) \equiv (a \wedge b) \vee (b \wedge c)$

D.3.2 Logical-or

Reflexive property $a \vee a \equiv a$

Associative property $a \vee (b \vee c) \equiv (a \vee b) \vee c$

Commutative property $a \vee b \equiv b \vee a$

Distributive property $a \vee (b \wedge c) \equiv (a \vee b) \wedge (b \vee c)$

This is a consequence of the distributive property mentioned in § D.3.1, De Morgan's first law, and the cancellative property.

Proof. Start with the first property

$$a \wedge (b \vee c) \equiv (a \wedge b) \vee (b \wedge c)$$

Apply $\neg$ to both sides

$$\neg (a \wedge (b \vee c)) \equiv \neg ((a \wedge b) \vee (b \wedge c))$$

Apply De Morgan's laws

$$\neg a \vee \neg (b \vee c) \equiv \neg (a \wedge b) \wedge \neg (b \wedge c)$$

Do it again

$$\neg a \vee (\neg b \wedge \neg c) \equiv (\neg a \vee \neg b) \wedge (\neg b \vee \neg c)$$

Let $p, q, r = \neg a, \neg b, \neg c$, respectively.

$$p \vee (q \wedge r) \equiv (p \vee q) \wedge (q \vee r)$$

□

D.3.3 De Morgan's Laws

De Morgan's first law $\neg(a \wedge b) \equiv \neg a \vee \neg b$

Derived law $\neg(a \vee b) \equiv \neg a \wedge \neg b$

Proof. Start with the first law

$$\neg(a \wedge b) \equiv \neg a \vee \neg b$$

Let $p = \neg a$, $q = \neg b$

$$p \vee q \equiv \neg(\neg p \wedge \neg q)$$

Apply $\neg$ to both sides of $\equiv$

$$\neg(p \vee q) \equiv \neg p \wedge \neg q$$

□

D.4 Sets

D.4.1 Definitions

Unions $a \cup b := \{x \in \mathcal{A};\ x \in a \vee x \in b\}$

Intersects $a \cap b := \{x \in a;\ x \in b\}$

Set subtraction (or relative complement) $a \setminus b := \{x \in a;\ x \notin b\}$

Complement (sometimes absolute complement) $a^c := \{x \in \mathcal{A};\ x \notin a\}$

D.4.2 Identities

Unions

Reflexive property $a \cup a \equiv a$

Associative property $a \cup (b \cup c) \equiv (a \cup b) \cup c$

Commutative property $a \cup b \equiv b \cup a$

Intersects

Reflexive property $a \cap a \equiv a$

Associative property $a \cap (b \cap c) \equiv (a \cap b) \cap c$

Commutative property $a \cap b \equiv b \cap a$

Set subtraction identities

1. $\mathbf{A} \setminus (\mathbf{B} \cap \mathbf{C}) \equiv (\mathbf{A} \setminus \mathbf{B}) \cup (\mathbf{A} \setminus \mathbf{C})$

   Proof. Let $A, B, C \subseteq \mathcal{A}$.

$$\begin{aligned} A \setminus (B \cap C) &:= \{x \in A;\ x \notin \{y \in B;\ y \in C\}\} \\ &:= \{x \in A;\ x \notin B \vee y \notin C\} \end{aligned}$$

$$\begin{aligned} (A \setminus B) \cup (A \setminus C) &:= \{x \in \mathcal{A};\ x \in \{y \in A;\ y \notin B\} \vee x \in \{z \in A;\ z \notin C\}\} \\ &:= \{x \in \mathcal{A};\ (x \in A \wedge x \notin B) \vee (x \in A \wedge x \notin C)\} \end{aligned}$$

$$x \in A \implies x \in \mathcal{A}$$

Therefore

$$\begin{array}{rcl} (A\setminus B)\cup(A\setminus C) & := & \{x\in A;\ x\notin B\vee x\notin C\} \\ A\setminus(B\cap C) & := & \{x\in A;\ x\notin B\vee y\notin C\} \\ A\setminus(B\cap C) & \equiv & (A\setminus B)\cup(A\setminus C) \end{array}$$

□

2. $\mathbf{A}\setminus(\mathbf{B}\cup\mathbf{C})\equiv(\mathbf{A}\setminus\mathbf{B})\cap(\mathbf{A}\setminus\mathbf{C})$

   Proof.

$$\begin{array}{rcl} A\setminus(B\cup C) & := & \{x\in A;\ x\notin B\wedge x\notin C\} \\ (A\setminus B)\cap(A\setminus C) & := & \{x\in \mathcal{A};\ x\in\{y\in A;\ y\notin B\}\wedge x\in\{z\in A;\ z\notin C\}\} \\ & := & \{x\in A;\ x\notin B\wedge x\notin C\} \end{array}$$

□

3. $\mathbf{A}\setminus(\mathbf{B}\setminus\mathbf{C})\equiv(\mathbf{A}\setminus\mathbf{B})\cup(\mathbf{A}\cap\mathbf{C})$

   Proof.

$$\begin{array}{rcl} A\setminus(B\setminus C) & := & \{x\in A;\ x\notin\{y\in B;\ y\notin C\}\} \\ & := & \{x\in A;\ x\notin B\vee x\in C\} \\ (A\setminus B)\cup(A\cap C) & := & \{x\in \mathcal{A};\ x\in\{y\in A;\ y\notin B\}\vee x\in\{z\in A;\ z\in C\}\} \\ & := & \{x\in A;\ x\notin B\vee x\in C\} \end{array}$$

□

4. $(\mathbf{A}\setminus\mathbf{B})\cap\mathbf{C}\equiv(\mathbf{A}\cap\mathbf{C})\setminus\mathbf{B}\equiv\mathbf{A}\cap(\mathbf{C}\setminus\mathbf{B})$

   Proof.

$$\begin{array}{rcl} (A\setminus B)\cap C & := & \{x\in \mathcal{A};\ x\in\{y\in A;\ y\notin B\}\wedge x\in C\} \\ & := & \{x\in A;\ x\notin B\wedge x\in C\} \\ (A\cap C)\setminus B & := & \{x\in \mathcal{A};\ x\in\{y\in A;\ y\in C\}\wedge x\notin B\} \\ & := & \{x\in A;\ x\notin B\wedge x\in C\} \\ A\cap(C\setminus B) & := & \{x\in A;\ x\notin B\wedge x\in C\} \end{array}$$

□

5. $(\mathbf{A} \setminus \mathbf{B}) \cup \mathbf{C} \equiv (\mathbf{A} \cup \mathbf{C}) \setminus (\mathbf{B} \setminus \mathbf{C})$

   Proof.

$$\begin{array}{rcl} (A \setminus B) \cup C & := & \{x \in \mathcal{A};\ x \in \{y \in A;\ y \notin B\} \vee x \in C\} \\ & := & \{x \in \mathcal{A};\ (x \in A \wedge x \notin B) \vee x \in C\} \\ (A \cup C) \setminus (B \setminus C) & := & \{x \in \mathcal{A};\ x \in \{y \in \mathcal{A};\ y \in A \vee y \in C\} \wedge x \notin \{z \in B;\ z \notin C\}\} \\ & := & \{x \in \mathcal{A};\ (x \in A \vee x \in C) \wedge \neg(x \in B \wedge x \notin C)\} \\ & := & \{x \in \mathcal{A};\ (x \in A \vee x \in C) \wedge (x \notin B \vee x \in C)\} \\ & := & \{x \in \mathcal{A};\ x \in C \vee (x \in A \wedge x \notin B)\} \end{array}$$

   □

6. $\mathbf{A} \setminus \mathbf{A} \equiv \varnothing$

   Proof.

$$A \setminus A := \{x \in A;\ x \notin A\}$$

   There are no elements in A that are also not in A, and the set with no elements is $\varnothing$. □

7. $\mathbf{A} \setminus \varnothing \equiv \mathbf{A}$

   Proof.

$$A \setminus \varnothing := \{x \in A \setminus x \notin \varnothing\}$$

   $\varnothing$, by definition has no elements, so all elements in A satisfy the condition $x \notin \varnothing$. Thus,

$$A \setminus \varnothing \equiv A$$

   □

8. $\varnothing \setminus \mathbf{A} \equiv \varnothing$

   Proof.

$$\varnothing \setminus A := \{x \in \varnothing;\ x \notin A\}$$

   There are no elements in $\varnothing$, so everything fails the condition on the left-hand-side of the ; , hence $\varnothing \setminus A \equiv \varnothing$. □

Distributive properties

$\mathbf{A} \cap (\mathbf{B} \cup \mathbf{C}) \equiv (\mathbf{A} \cap \mathbf{B}) \cup (\mathbf{A} \cap \mathbf{C})$

Proof.

$$\begin{aligned}
A \cap (B \cup C) &:= \{x \in A;\ x \in B \vee x \in C\} \\
(A \cap B) \cup (A \cap C) &:= \{x \in \mathcal{A};\ (x \in A \wedge x \in B) \vee (x \in A \wedge x \in C)\} \\
&:= \{x \in \mathcal{A};\ x \in A \wedge (x \in B \vee x \in C)\} \\
&:= \{x \in A;\ x \in B \vee x \in C\}
\end{aligned}$$

□

$\mathbf{A} \cup (\mathbf{B} \cap \mathbf{C}) \equiv (\mathbf{A} \cup \mathbf{B}) \cap (\mathbf{A} \cup \mathbf{C})$

Proof.

$$\begin{aligned}
A \cup (B \cap C) &:= \{x \in \mathcal{A};\ x \in A \vee (x \in B \wedge x \in C)\} \\
(A \cup B) \cap (A \cup C) &:= \{x \in \mathcal{A};\ (x \in A \vee x \in C) \wedge (x \in A \vee x \in C)\} \\
&:= \{x \in \mathcal{A};\ x \in A \vee (x \in B \wedge x \in C)\}
\end{aligned}$$

□

Complements

$(\mathbf{A^c})^\mathbf{c} \equiv \mathbf{A}$

Proof.

$$\begin{aligned}
A \setminus (A \setminus B) &\equiv (A \setminus A) \cup (A \cap B) \\
&\equiv \varnothing \cup (A \cap B) \\
&\equiv A \cap B \\
(A^c)^c &:= \mathcal{A} \setminus (\mathcal{A} \setminus A) \\
&:= \mathcal{A} \cap A \\
&:= A
\end{aligned}$$

□

De Morgan's law $(\mathbf{A} \cap \mathbf{B})^\mathbf{c} \equiv \mathbf{A^c} \cup \mathbf{B^c}$

Proof.

$$\begin{aligned} A \setminus (B \cap C) &\equiv (A \setminus B) \cup (A \setminus C) \\ \mathcal{A} \setminus (A \cup B) &\equiv (\mathcal{A} \setminus A) \cup (\mathcal{A} \setminus B) \\ &\equiv A^c \cup B^c \end{aligned}$$

□

De Morgan's derived law $\mathbf{(A \cup B)^c \equiv A^c \cap B^c}$

Proof.

$$\begin{aligned} A \setminus (B \cup C) &\equiv (A \setminus B) \cap (A \setminus C) \\ \mathcal{A} \setminus (A \cup B) &\equiv (\mathcal{A} \setminus A) \cap (\mathcal{A} \setminus B) \\ &\equiv A^c \cap B^c \end{aligned}$$

□

D.4.3 ZFC

ZFC Short for Zermelo-Fraenkel-Choice: a set of axioms rigorously describing set theory.

Nomenclature Named after Ernst Zermelo, who formulated the axioms, and Abraham Fraenkel, who greatly improved them.

Russell's paradox A paradox proposed by Bertrand Russell in the early 20th century regarding unrestricted set comprehensions

$$\begin{aligned} &A = \{x;\, x \notin x\} \\ &A \overset{?}{\in} A \end{aligned}$$

ZF ZFC without the axiom of choice

D.5 Functions

D.5.1 Vocabulary

Function A mathematical construct mapping an input to an output.

Referential transparency $a = b \implies f(a) = f(b)$. All functions are referentially transparent.

Domain If $f : A \to B$, then A is the *domain* of f.

Codomain If $f : A \to B$, then B is the *codomain* of f.

Image $\mathrm{im}(f) := \{ f(x) \in B;\ x \in A \}$

Injectivity $\nexists\,(a,b);\ a,b \in A \wedge a \neq b \wedge f(a) = f(b)$

Surjectivity $\mathbf{codom}(f) = \mathrm{im}(f)$

Bijectivity A function is *bijective* if it is both injective and surjective.

Invertibility A function is invertible iff it is bijective. The inverse of f is $\mathrm{arc}(f)$

Preimage $\mathrm{preim} := \mathbf{codom} \circ \mathrm{arc}$

Argument The specific input values to a function.

Signature If $f : A \to B$ is a function, then $A \to B$ is its signature.

D.5.2 Notation

: notation $f : A \to B$ means that f takes an item from A, and outputs an item to B, where A and B are types.

Currying Taking a function of multiple arguments, and transforming it into a chain of functions each taking one argument.

Normal signature

$+ : (\mathbb{C}, \mathbb{C}) \to \mathbb{C}$

Curried signature:

$+ : \mathbb{C} \to \mathbb{C} \to \mathbb{C}$

This doesn't change the behavior of the function, only the semantics.

Likewise, *uncurrying* is to undo the currying.

Composition We can smush two functions together with $\circ$:

$$\circ : (b \to c) \to (a \to b) \to a \to c$$
$$(f \circ g)(x) := f(g(x))$$

D.6 Lambda calculus

λ abstraction A way to write a function: $\lambda\,(x,y) \to x+y$

α conversion Changing the names of the arguments. For instance, you can write the above function as

$$\lambda\,(a,b) \to a+b$$

β reduction Partially calculating a result. For instance

$$\lambda\,(2,y) \to 2+y$$

Can be β reduced to

$$\lambda\,(y) \to 2+y$$

η conversion Removing or adding extraneous free arguments. The last function

$$\lambda\,(2,y) \to 2+y$$

Can be η *reduced* to

$$2+$$

Which could then be η *abstracted* to

$$\lambda\,(2,\kappa) \to 2+\kappa$$

D.7 Greek alphabet

Letter	**Pronounciation**	**Rough latin equivalent**
A, α	Alpha	A
B, β	Beta	B
Γ, γ	Gamma	G
Δ, δ	Delta	D
E, ε	Epsilon	E, j**e**t, phl**eg**m
Z, ζ	Zeta	Z
H, η	Eta	Eh, r**ai**n, **eigh**t
Θ, θ	Theta	Th, **th**eater, **th**under
I, ι	Iota	Ee, f**ee**t, j**ee**p
K, κ	Kappa	K
Λ, λ	Lambda	L
M, μ	Mu	M
N, ν	Nu	N
Ξ, ξ	Xi	Ks, du**cks**
O, o	Omicron	Oh, **oa**t
Π, π	Pi	P
P, ρ	Rho	R
Σ, σ	Sigma	S
T, τ	Tau	T
Υ, υ	Upsilon	U
Φ, ϕ	Phi	F
X, χ	Chi	Sh, **sh**opping
Ψ, ψ	Psi	Ps, cu**ps**
Ω, ω	Omega	O, b**o**ss

D.8 Special sets

Despite my informal notation, these are all sets

$$\mathbb{N} = \{0, 1, 2, 3, 4, 5, \ldots\}$$

$$\mathbb{Z} = \{\ldots, ‘5, ‘4, ‘3, ‘2, ‘1, 0, 1, 2, 3, 4, 5, \ldots\}$$

$\mathbb{R}$ any given number on the number line.

$\mathbb{Q} = \left\{ \frac{x}{y} \in \mathbb{R};\ x, y \in \mathbb{Z} \wedge y \neq 0 \right\}$

$\mathbb{C} = \left\{ a + bi;\ (a, b) \in \mathbb{R} \times \mathbb{R}, i = \sqrt{‘1} \right\}$

$\mathbb{I} = \mathbb{R} \setminus \mathbb{Q}$

D.8.1 Properties and identities

Addition

Additive identity $a + 0 \equiv 0$

Associative property $a + (b + c) \equiv (a + b) + c$

Commutative property $a + b \equiv b + a$

Cancellative property $a + b = a + c \implies b = c$

Negative property $\forall a \in \mathbb{C};\ \exists ‘a \in \mathbb{C};\ a + ‘a = 0$

Distribution $‘(a + b) \equiv ‘a + ‘b$

Subtraction

Definition $a - b := a + ‘b$

Associative property $a - (b - c) \equiv (a - b) - c$

Subtractive identity $a - 0 \equiv a$

Negative property $a - b \equiv a + ‘b$

Distribution theorem $a - (b + c) \equiv a - b - c$

Multiplication

Multiplicative identity $a \cdot 1 \equiv a$

Associative property $a \cdot (b \cdot c) \equiv (a \cdot b) \cdot c$

Commutative property $a \cdot b \equiv b \cdot a$

Cancellative property $a \cdot b = a \cdot c \implies b = c$

Divisive property $\forall a \in \mathbb{C};\ \exists \mathrm{arc}\,(a) \in \mathbb{C};\ a \cdot \mathrm{arc}\,(a) = 1$

$$\begin{aligned} \mathrm{arc}\,(0) &:= 1 \\ \mathrm{arc}\,(n) &:= \tfrac{1}{n} \end{aligned}$$

Distribution $\frac{a}{b \cdot c} \equiv \frac{a}{b} \cdot \frac{a}{c}$

Distribution over + $a \cdot (b + c) \equiv (a \cdot b) + (a \cdot c)$

Notation $ab = a \cdot b$ if a and b are two separate things.

Division

Divisive identity $a \div 1 \equiv a$

Separative property $a \div b \equiv a \cdot \frac{1}{b}$

Antiassociative property $a \div (b \div c) \equiv a \cdot (c \div b)$

Appendix E

Graph source code

This appendix contains all of the source code for the various graphs throughout the book.

```
#!/usr/bin/env sage

# the points
points = [(2,4), (-3,4), (-5,-1), (5,-3)]

# This next little bit constructs the labels for the points
labels = []                     # This is the list of labels.

# Loop through the points
for point in points:
    # The label needs to be slightly to the right of the point, as to
    # not overwrite it.
    newpoint = (point[0] + 0.1, point[1])
    # This label will just have the coordinate listed, with some
    # styling.
    this_label = text("$"+str(point)+"$", newpoint, fontsize=25,
                      rgbcolor=(0,0,0), horizontal_alignment="left")

    # Add this label to the list.
    labels.append(this_label)
labels = sum(labels)

# A plot of the points
myticks = [range(-100,100)] * 2
pts = list_plot(points,
                ticks=myticks,
                tick_formatter="latex",
                pointsize=25
               ) + labels
pts.set_axes_range(-6,6,-4,5)
pts.fontsize(20)
# Save it to a file
pts.save("VectorGraph2.png")
```

Listing E.1: This program puts a few points on a Cartesian coordinate plane.

```
#!/usr/bin/env sage

from numpy import arange

squarelist = []
textlist = []

for x in arange(-10,11):
    t = text('$' + str((x, x**2)) + '$',
             (x+0.3, x**2),
             rgbcolor='black',
             horizontal_alignment='left'
            )
    squarelist.append((x,x**2))
    textlist.append(t)

pts = list_plot(squarelist,
                ticks=2,
                tick_formatter=1,
                pointsize=15,
                color='red',
                axes_labels=['$x$', '$f(x)$']
               )
pts.set_axes_range(xmin=-11, xmax=11, ymin=-10, ymax=110)
pts.fontsize(15)
pts.save("x-squared-nolabels.png")

# Add labels
pts_with_labels = pts + sum(textlist)
pts_with_labels.save("x-squared-labels.png")

# Add connecting lines
pts_with_conn = pts_with_labels + list_plot(squarelist, plotjoined=True)
pts_with_conn.save("x-squared-joined.png")

# Add Curve
x=var('x')
pts_with_curve = pts + plot(x^2, (x, -11, 11), color='green')
pts_with_curve.save("x-squared-withcurve-nolabels.png")
```

Listing E.2: Produces a number of graphs corresponding to $\lambda(x) \to x^2$

Appendix F

Answers to the exercises

Answer (Ex. 1) — No. $A \not\Longrightarrow B$ means that A doesn't imply B. It doesn't necessarily mean that $\neg B$ is false — although that could very well be the case.

Answer (Ex. 2) — Start with the first law

$$\neg(a \wedge b) \equiv \neg a \vee \neg b$$

Let $p = \neg a$, $q = \neg b$. One of the rules of algebra is, if $x = y$, then you can substitute one in for the other.

$$p \vee q \equiv \neg(\neg p \wedge \neg q)$$

Another rule is, if you have an equation, and you do something to one side of the =, you have to do the same thing to the other side. Here, we're going to apply $\neg$ to both sides of $\equiv$

$$\neg(p \vee q) \equiv \neg p \wedge \neg q$$

Answer (Ex. 3) — Start with the first property

$$a \wedge (b \vee c) \equiv (a \wedge b) \vee (b \wedge c)$$

Apply $\neg$ to both sides

$$\neg(a \wedge (b \vee c)) \equiv \neg((a \wedge b) \vee (b \wedge c))$$

Apply DeMorgan's laws

$$\neg a \vee \neg(b \vee c) \equiv \neg(a \wedge b) \wedge \neg(b \wedge c)$$

Do it again (inside the parentheses).

$$\neg a \vee (\neg b \wedge \neg c) \equiv (\neg a \vee \neg b) \wedge (\neg b \vee \neg c)$$

Let $p, q, r = \neg a, \neg b, \neg c$, respectively.

$$p \vee (q \wedge r) \equiv (p \vee q) \wedge (q \vee r)$$

Answer (Ex. 4) — Yes.

*Proof.*By transition,

$$\begin{array}{ccccc} \neg A & \Longrightarrow & B & \Longrightarrow & \neg C \\ \neg A & & & \Longrightarrow & \neg C \end{array}$$

By the reversal property:

$$A \Longleftarrow C$$

□

Answer (Ex. 5) — Not by necessity.

*Proof.*By transition:

$$\begin{array}{ccccc} A & \not\Longrightarrow & B & \Longrightarrow & \neg C \\ A & \not\Longrightarrow & & & \neg C \end{array}$$

This does not imply

$$A \Longrightarrow C$$

To put it another way

$$\begin{tikzcd} A \arrow[r, Rightarrow, "/" description] & \neg C \arrow[d, Rightarrow, "/" description] \\ A \arrow[r, Rightarrow] & C \end{tikzcd}$$

□

Answer (Ex. 6) — No, there's nothing to indicate that A and C have any relation whatsoever.

Answer (Ex. 7) — No

*Proof.*Let's assume that it's true.

$$A \wedge \neg [B \wedge (C \vee D)] \equiv A \wedge (B \vee C) \wedge (B \vee D)$$

Let B, C, D all be true. Then:

$$\begin{aligned} A \wedge \neg [\text{True} \wedge (\text{True} \vee \text{True})] &\equiv A \wedge (\text{True} \vee \text{True}) \wedge (\text{True} \vee \text{True}) \\ A \wedge \neg [\text{True} \wedge (\text{True} \vee \text{True})] &\equiv A \wedge \text{True} \wedge \text{True} \\ A \wedge \neg [\text{True} \wedge (\text{True} \vee \text{True})] &\equiv A \wedge \text{True} \\ A \wedge \neg [\text{True} \wedge \text{True}] &\equiv A \wedge \text{True} \\ A \wedge \neg \text{True} &\equiv A \wedge \text{True} \\ A \wedge \neg \text{True} &\equiv A \wedge \text{True} \\ \text{False} &\equiv \text{True} \end{aligned}$$

Which is obviously false. □

Answer (Ex. 8) — Let's look at $f : A \to B$. If f is surjective, then $B = \text{im}(f)$, so we can write

$$f : A \to \text{im}(f)$$

In other words

$$f : \mathbf{dom}(f) \to \text{im}(f)$$

It must be true that f is a surjection for f to be invertible. Else there would be elements in the codomain of f that were not in the domain of $\text{arc}(f)$.

We've established

$$\mathrm{arc}\,(f) : \mathrm{im}\,(f) \to \mathbf{dom}\,(f)$$

Let's assume f is invertible. Then $\mathbf{dom}\,(f) = \mathrm{preim}\,(f)$. Thus

$$\mathrm{arc}\,(f) : \mathrm{im}\,(f) \to \mathbf{dom}\,(f)$$

For $\mathrm{arc}\,(f)$ to be a function — i.e. for f to be invertible, then it must be true that

$$\nexists\, a, b \in \mathrm{im}\,(f)\,;\ a = b;\ \mathrm{arc}\,(f, a) \neq \mathrm{arc}\,(f, b)$$

If we flip this around

$$\nexists\, a, b \in \mathrm{preim}\,(f)\,;\ a \neq b;\ f\,(a) = f\,(b)$$

That is, the definition of injectivity. Thus we have proven

$$f \text{ is invertible } \iff (f \text{ is an injection}) \wedge (f \text{ is a surjection}) \iff f \text{ is a bijection}$$

Bibliography

[1] Alpha conversion. URL: https://wiki.haskell.org/Alpha_conversion (visited on 03/01/2015).

[2] Beta reduction. URL: https://wiki.haskell.org/Beta_reduction (visited on 03/01/2015).

[3] Boolean algebra. URL: https://en.wikipedia.org/wiki/Boolean_algebra (visited on 03/01/2015).

[4] Currying. URL: https://en.wikipedia.org/wiki/Currying (visited on 02/23/2015).

[5] Eta conversion. URL: https://wiki.haskell.org/Eta_conversion (visited on 02/23/2015).

[6] Eta conversion. URL: https://wiki.haskell.org/Eta_conversion (visited on 03/01/2015).

[7] Greek alphabet. URL: https://en.wikipedia.org/wiki/Greek_alphabet (visited on 03/01/2015).

[8] Greek government-debt crisis. URL: https://en.wikipedia.org/wiki/Greek_financial_crisis (visited on 02/23/2015).

[9] Thomas Jech. Set Theory. New York, NY: Springer, 2003. ISBN: 3-540-44085-2.

[10] Lambda abstaction. URL: https://wiki.haskell.org/Lambda_abstraction (visited on 03/01/2015).

[11] Lambda calculus. URL: https://wiki.haskell.org/Lambda_calculus (visited on 03/01/2015).

[12] Miran Lipovača. Learn You a Haskell for Great Good! San Francisco, CA: No Starch Press, 2011.

[13] Operator associativity. URL: `https://en.wikipedia.org/wiki/Operatorassociativity` (visited on 02/23/2015).

[14] steve jobs on programming. URL: `https://www.youtube.com/watch?v=5Z1gfgM7kzo` (visited on 01/01/2015).

[15] Zermelo-Fraenkel set theory. URL: `https://en.wikipedia.org/wiki/Zermelo%E2%80%93Fraenkel_set_theory` (visited on 02/23/2015).