Learn You Some Algebras for Glorious Good!

Peter Harpending <`peter@harpending.org`>

February 21, 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.

  I'm going to use some notation for sets later in the book. The explanation for the notation doesn't really fit anywhere, so here it is:

$$A = \{\, x \mid y \,\}$$

[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*).

This should be read as "A is the set of all values x, such that y is True. This is called "set comprehension notation." It's not difficult, and you'll pick it up as we go along.

- 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.
- 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 on the page after the exercises.

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.

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 [2].

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 Idris

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

[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.

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 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 Idris. Idris is an interesting programming language for many reasons. The chief of which is that it can be used to prove things mathematically. Most programming languages can't do this. Idris can, which is why it is special.[4]

1.4.1 Installing Idris

This is something that is actually rather difficult to summarize, because it varies from operating system to operating system. I will put down the instructions for the operating systems I use. If you come upon this and don't see your operating system, please report this on the issue tracker (`https://gitlab.com/lysa/lysa/issues/new`). Better yet, you could add the instructions yourself, and ask me to merge your changes.

Linux

Arch You need the Haskell platform and the GNU Multiple-Precision (GMP) library.

```
# pacman -S ghc cabal-install gmp
% \cabal update
% \cabal install -j cabal-install
```

At this point, you'll want to add `~/.cabal/bin` to your `$PATH` variable.

[4]There are other programming languages that can prove things, namely Coq and Agda. However, I'm most familiar with Idris, and Idris is probably the most useful, so I'm using Idris. Deal with it.

```
  % cabal install -j alex happy haddock hscolour idris
```

`alex` and `happy` are dependencies for a number of Haskell packages. However, due to a long-standing bug in `cabal` (`https://github.com/haskell/cabal/issues/220`), they don't get pulled in when packages depend on them.

If the installation doesn't work, please report the bug to the Idris people (`https://github.com/idris-lang/Idris-dev/issues/new`).

Gentoo You will need the Haskell platform, along with the GNU Multiple Precision (GMP) library. As of 5 January 2015, the Haskell platform is only available on `~ARCH`, where `ARCH` is your processor architecture (e.g. `amd64`, `x86`). If you use `ARCH`, you can enable these by adding the following to `/etc/portage/package.keywords`:[5]

```
dev-lang/ghc
dev-haskell/cabal-install
```

Regardless of your `ACCEPT_KEYWORDS` variable, you should add the following to `/etc/portage/package.use`:

```
dev-lang/ghc binary
```

Otherwise, you have to compile GHC (the Haskell compiler) from scratch, and that takes forever.

Once you have that all out of the way, you'll want to run the following command as root:

```
# emerge -jav dev-lang/ghc dev-haskell/cabal-install
```

[5]If you already use `~ARCH`, you can ignore this

Warning: `-j` will make the installation a lot faster, but is more resource-intensive. If your power usage is precious, omit it (i.e. use `-av` instead).

Once GHC and cabal-install are installed, you'll want to run the following as a normal user:

```
% cabal update
% cabal install alex happy haddock hscolour
% cabal install idris
```

You can then get at the Idris shell by running `idris`.

1.4.2 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:

   ```
   % 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  kbookmarks-5.6.0-1
              kcodecs-5.6.0-1  kcompletion-5.6.0-1  kconfig
    -5.6.0-1  kconfigwidgets-5.6.0-1
              kcoreaddons-5.6.0-1  kcrash-5.6.0-2
    kdbusaddons-5.6.0-1  kded-5.6.0-1  kglobalaccel-5.6.0-1
              kguiaddons-5.6.0-1  ki18n-5.6.0-1  kiconthemes
    -5.6.0-1  kinit-5.6.0-1  kio-5.6.0-1
              kitemmodels-5.6.0-1  kitemviews-5.6.0-1
    kjobwidgets-5.6.0-1  knewstuff-5.6.0-1
              knotifications-5.6.0-1  kparts-5.6.0-1
    kservice-5.6.0-1  ktexteditor-5.6.0-1
              ktextwidgets-5.6.0-1  kwallet-5.6.0-1
    kwidgetsaddons-5.6.0-1  kwindowsystem-5.6.0-3
              kxmlgui-5.6.0-1  libdbusmenu-qt5
    -0.9.3+14.10.20140619-1  libgit2-1:0.21.5-1
              libimobiledevice-1.1.7-1  libplist-1.11-1
    libusbmuxd-1.0.9-1  libxkbcommon-x11-0.5.0-1
              media-player-info-19-1  phonon-qt5-4.8.3-1
    phonon-qt5-vlc-0.8.2-1  polkit-qt5-0.112-2
              qt5-base-5.4.0-3  qt5-declarative-5.4.0-3  qt5
    -script-5.4.0-3  qt5-svg-5.4.0-3
              qt5-x11extras-5.4.0-3  qt5-xmlpatterns-5.4.0-3
      qtchooser-48-1  solid-5.6.0-1  sonnet-5.6.0-1
              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.

1.4.3 Hello World in Idris

Pretty much every programmer in the world is familiar with the "Hello, World!" program. It's a program that exists in every language that just prints out `hello, world`.[6]

Access the Idris shell by running `idris` in a terminal. Here's what mine looks like:

```
% idris
     ____    __     _
    /  _/___/ /____(_)____
    / // __  / ___/ / ___/     Version 0.9.16-git:fca8309
  _/ // /_/ / /  / (__  )      http://www.idris-lang.org/
 /___/\__,_/_/  /_/____/       Type :? for help

Idris is free software with ABSOLUTELY NO WARRANTY.
For details type :warranty.
Idris>
```

Note that the `%` is there to indicate that the rest of the line should typed in a terminal. You shouldn't type the `%`.

There will be a flashing cursor after `Idris>`. That's where you type stuff.[7] Here's hello, world!

```
Idris> "hello, world"
"hello, world" : String
```

That was pretty simple. Note that you only have to type the thing following `Idris>` , then hit `Return`. You shouldn't type `Idris>` The stuff below it is what `idris` prints. By the way, it's incredibly important that

[6]Prints it in the terminal, not on a printer. Seriously, who uses paper these days?

[7]Note that, during the writing of this book, I've encountered a number of bugs in Idris. Trying to fix these bugs requires that I use the same version of Idris as the Idris developers, hence my using version `0.9.16-git:fca8309`. Your version will probably be different.

you type this stuff out yourself, rather than just reading it (or copying & pasting).

If things are preceded by `Idris>`, that means that it's run in the interactive environment.[8] If you see a bunch of Idris code, without `Idris>` in front of each line, that means you should put the code in a file.

Speaking of files, let's make an actual program that you can run on the command line. Write this in an editor, and save it to a file called `helloworld.idr`:

```
module Main

main : IO ()
main = putStrLn "hello, world"
```

helloworld.idr

The computer can't understand the code; the code exists for the benefit of humans. With that in mind, we need to turn the code into binary language, which the computer can understand. Doing this by hand would be a nightmare. Luckily, we have a program to do it for us. To compile the program we wrote, run `idris helloworld.idr -o helloworld`. To run it, run `./helloworld`.

```
% idris helloworld.idr -o helloworld
% ./helloworld
hello, world
```

1.5 Target audience

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

[8] In case you forgot, the way you access the interactive environment is to run `idris` in a terminal.

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.

1.6 Licensing

This book is libre[9]. You can copy this book and give it to your friend. You can even print it out and sell it to people.[10] 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,

[9] *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*.

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

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.

1.7 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 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. God dammit, 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 Idris), 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[11], 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: [2], then it's a citation. If you're reading this as a PDF, you can actually click on the number, and 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.[12]

[11]Hey, you found me!

[12]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.

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: $\centernot\implies$, it means that you can semantically but "not" in front of whatever the operator says. So, $\centernot\implies$ 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.

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

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:

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

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.

$$\begin{tikzcd}\text{Decapitated} \arrow[r, Rightarrow] \arrow[d, Rightarrow] & \text{Dead} \\ \neg\text{Decapitated} & \neg\text{Dead} \arrow[l, Rightarrow]\end{tikzcd}$$

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

7. So, hopefully you agree that

$$\begin{tikzcd}\text{Decapitated} \arrow[r, Rightarrow] \arrow[d, Rightarrow] & \text{Dead} \\ \neg\text{Decapitated} & \neg\text{Dead} \arrow[l, Rightarrow]\end{tikzcd}$$

   However,

$$\begin{tikzcd}\text{Decapitated} \arrow[r, Rightarrow] & \text{Dead} \\ \neg\text{Decapitated} \arrow[u, Rightarrow, "?"] & \neg\text{Dead} \arrow[l, Rightarrow]\end{tikzcd}$$

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 ¬. 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 $\impliedby$,

$$\begin{array}{l}\neg\neg\text{Decapitated} \implies \neg\neg\text{Dead}\\ \text{Decapitated} \implies \text{Dead}\end{array}$$

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

2.2 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".

This monstrosity is called a "truth table".

A	**B**	$\mathbf{A \land B}$	$\mathbf{A \lor B}$
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 — $\land$ — looks vaguely like an A.

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

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

In practice, $\iff$ is logically equivalent to $=$. That is, all of the properties of $=$ hold.[1]

1. For all a, $a = a$.
2. For all a and b, $(a = b) \implies (b = a)$.
3. For all a and b, and c, $((a = b) \land (b = c)) \implies (a = c)$.

You should go over those properties, and verify that they are true for $=$, and then verify that they are true for $\iff$.

Alright, cool, this was a pretty easy chapter. Only 240 lines of LaTeX! More importantly, I'm confident that this stuff is so easy, I won't even give you exercises![1]

[1]The real reason is, I can't come up with any exercises that aren't tedious.

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, for basic stuff, much more important. Also, 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

[1]You better think this, because it took me 5 minutes to get the alignment right. So, you know, remember $\in$ this way, or else...

$$11 \notin Q$$

By contrast,

$$6 \in Q$$

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.

Before I go on, I'm going to define a couple of other symbols. We're constantly saying "this thing exists" and "for all", so I'm going to add a couple of symbols. $\exists$ is for "there exists" — notice the backwards E, as in "exists". $\forall$ is for "for all" — notice the upside-down A, as in "all".

So, now that you've got all of this down (hopefully), I'll give you some exercises.

3.2.1 Exercises

Ex. 1 — $\overset{?}{\exists} A;\ A \subset \varnothing$

Ex. 2 — $\forall A;\ \varnothing \overset{?}{\subseteq} A$

Answers

Answer (Ex. 1) — $\varnothing$, by definition, has no elements. For there to exist a proper subset of $\varnothing$, there would need to be a set with fewer elements. There cannot be fewer than zero elements, hence there is no proper subset of $\varnothing$.

Answer (Ex. 2) — Yes. It's obvious that $\varnothing \subset A$ if $A \neq \varnothing$. The only thing left is, $\varnothing \overset{?}{\subseteq} \varnothing$. $\varnothing \subseteq \varnothing$ allows for $\varnothing = \varnothing$, which is obviously true.

Chapter 4

Programming

You probably thought I was just being mean when I made you install Idris at the beginning of the book. Well, I wasn't. Unfortunately, the three preceding chapters were needed to get here. We're going to program our way through the last few chapters!

Alright, let's pop open an `idris` shell, and try this out!

First things first, let's see what "type" of thing True and False are.

```
Idris> :type True
Prelude.Bool.True : Bool
Idris> :type False
Prelude.Bool.False : Bool
```

Since we are mathematicians, and we are lazy, instead of typing `:type True`, or `:type False`, we can instead type `:t True` or `:t False`.

```
Idris> :t True
Prelude.Bool.True : Bool
Idris> :t False
Prelude.Bool.False : Bool
```

Okay, let's look at $\wedge$ and $\vee$. As stated above, $A \wedge B$ is True if and only if both A and B are True. $\wedge$ isn't on many standard keyboards, so, in most programming languages, $\wedge$ is replaced with `&&`.

```
Idris> True && False
False : Bool
Idris> False && True
False : Bool
Idris> False && False
False : Bool
Idris> True && True
True : Bool
```

Idris's syntax So, before going much further, let's go over some of Idris's weird notation. `Bool` is short for "Boolean", as stated above. You should have already figured that out.

Okay, then what the hell is `Prelude.Bool.True`? Why not just `True`? This is actually reasonably complicated. Idris allows for *context-based overloading*, which basically means that `True` could (in theory) mean something different if you used it in some other context. With that in mind, Idris feels the need to tell you the *fully qualified* name of `True` so you know that `True` refers to `Prelude.Bool.True`, and not some other version of `True`.

However, when we type something like `True && True`, there's enough context to unambiguously determine which version of `True` we are talking about, so Idris doesn't feel the need to tell us that we're using `Prelude.Bool.True`.

I should note that use of commands preceded by a `:` are only to be used in the shell (the interactive environment). That is, you can't use them in actual code. You can do most of the `:` things in the code through other means, but the `:` commands are limited to the interactive environment.

Back to Business If you saw up there, we had to type out every single instance of which we could concieve. That's annoying. Let's figure out how to do this on our own. This will be a good segue to introduce *comprehension notation*

```
Idris> :let bools = [True, False]
Idris> [(x, y) | x <- bools, y <- bools]
[(True, True),
 (True, False),
 (False, True),
 (False, False)] : List (Bool, Bool)
```

Okay, what the hell is that? Let's go over this step-by-step

1. I made the list `[True, False]`, and bound it to the name `bools` using the `:let` command.

2. The next line is a *list comprehension.* Basically, it's a way of making a list of items that satisfy some conditions. The pipe in the middle is the key thing. The stuff to the right of the pipe is the conditions, and the stuff to the left is what each item looks like. Let's look at that comprehension again.

   ```
   [(x, y) | x <- bools, y <- bools]
   ```

   So, the stuff on the left says that each item will look something like `(x,y)`. `x` and `y` are each described on the right side of the pipe.

   We'll encounter the $\in$ operator a lot. Basically, when you have a set A, $x \in A$ means that "x is an element of A". You should read the $\in$ as a really crappy E, for "element of". `<-` is a silly way of trying to replicate a $\in$.

   So, on the right-hand side of the pipe, `x <- bools` just means "`x` is an element of `bools`". Ditto for `y`. Note that currently we are dealing with lists, which are slightly different than sets.

   So, we have a set of items that look like `(x,y)`, where `x` and `y` are both elements of `bools`. Indeed, that's the result:

   ```
   [(True, True),
    (True, False),
   ```

```
(False, True),
(False, False)] : List (Bool, Bool)
```

Alright, so we've listed all pairs of booleans. This is kind of boring. Also, pairs suck, and are hard to deal with.

We're going to make a *truth table*. Basically, we are going to have two booleans, `a` and `b`. We are then going to show the result of `a && b` and `a || b`.

We're going to have to move over to a file, call it `truthtable.idr`:

```
module Main

main : IO ()
main = return ()
```

`main = return ()` just says that the function `main` does nothing. If you compile and run this program, nothing interesting is going to happen.

Let's put our list of Booleans in:

```
bools : List Bool
bools = [True, False]
```

It's important to periodically compile your program to make sure there aren't any errors. Run `idris truthtable.idr` (don't include `-o truthtable`).

```
% idris truthtable.idr
     ____    __     _
    /  _/___/ /____(_)____
    / // __  / ___/ / ___/     Version 0.9.16
  _/ // /_/ / /  / (__  )      http://www.idris-lang.org/
 /___/\__,_/_/  /_/____/       Type :? for help
```

```

Idris is free software with ABSOLUTELY NO WARRANTY.
For details type :warranty.
*truthtable>
```

We've loaded the file `truthtable` into the interactive environment, and we can examine it interactively.

```
*truthtable> bools
[True, False] : List Bool
```

Cool!

If you've programmed in other languages, you're probably used to dealing mostly with primitive types (integers, strings, booleans). In a language like Idris, abstraction is lexically very cheap, and is thus encouraged. We're going to make a new data type for our row:

```
record TruthRow : Bool -> Bool -> Bool -> Bool -> Type where
  MkRow : (a : Bool) -> (b : Bool) -> TruthTable a b (a && b) (a
    || b)
```

Appendices

Appendix A

GNU Free Documentation License

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as

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

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled “History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

5. COMBINING DOCUMENTS

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 finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may

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.

Bibliography

[1] Edmund Landau. Foundations of Analysis. Providence, RI: AMS Chelsea Publishing, 1966.

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

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