Mathematics Without Set Theory

Or: How I Learned to Stop Worrying and Love Martin-Lof Type Theory

Alex Grabanski

12/1/2017

Your Waifu (ZFC) is Shit-Tier

Constructive Type Theory is God-Tier

Alex Grabanski

12/1/2017

What is ZFC, Anyhow?

- System of axioms on top of first-order logic (FOL) given by:
- Citation: (`http://www.mtnmath.com/whatrh/node57.html`)

1. Axiom of extensionality (See Section 5.5.1).

$$\forall x \forall y \;\; (\forall z \;\; z \in x \equiv z \in y) \equiv (x = y)$$

2. Axiom of the empty set (See Section 5.5.2).

$$\exists x \forall y \;\; y \notin x$$

3. Axiom of unordered pairs (See Section 5.5.3).

$$\forall x \forall y \; \exists z \; \forall w \;\; w \in z \equiv (w = x \vee w = y)$$

4. Axiom of union (See Section 5.5.4).

$$\forall x \exists y \, \forall z \;\; z \in y \equiv (\exists t \; z \in t \wedge t \in x)$$

5. Axiom of infinity (See Section 5.5.5).

$$\exists x \, \emptyset \in x \wedge [\forall y \, (y \in x) \rightarrow (y \cup \{y\} \in x)]$$

6. Axiom schema of replacement (See Section 6.3).

$$[\forall x \exists ! \, y A_n(x, y)] \rightarrow \forall u \exists v (B(u, v))$$
$$B(u, v) \equiv [\forall r (r \in v \equiv \exists s [s \in u \wedge A_n(s, r)])]$$

7. Axiom of the power set (See Section 6.6).

$$\forall x \exists y \forall z [z \in y \equiv z \subseteq x]$$

8. Axiom of choice (See Section 6.7).

$$\forall C \exists f \forall e [(e \in C \wedge e \neq \emptyset) \rightarrow f(e) \in e]$$

Fun :D Activity Time :D :D :D

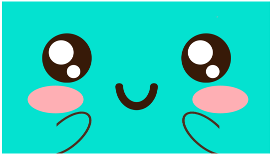
- Define natural numbers using sets :D

Fun :D Activity Time :D :D :D

- Define natural numbers using sets :D
- Define pairs using sets :D ($S \times S$)

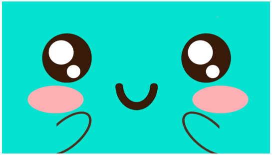
Fun :D Activity Time :D :D :D

- Define natural numbers using sets :D
- Define pairs using sets :D
- U Got This
- I Believe in U



Fun :D Activity Time :D :D :D

- Define natural numbers using sets :D
- Define pairs using sets :D
- U Got This
- I Believe in U
- Define functions using sets :D



One Problem

One Problem

Answers to Earlier Questions: Natural Numbers

- The natural numbers are defined by the *Axiom of Infinity* as the collection of sets
  - $\{\}$ (zero)
  - $\{\{\{\}\}\{\}\}$ (one)
  - $\{\{\{\{\}\}\{\}\}\{\{\{\{\}\}\{\}\}\}\}$ (two)
  - And so on, where if S is the set representing some number, $S \cup \{S\}$ gives the next one
- This representation is absolute garbage.

Answers to Earlier Questions: Pairs

- Pairs are defined by the Axiom of Pairing :D

Answers to Earlier Questions: Pairs

- ~~Pairs are defined by the Axiom of Pairing :D~~
- Actually no, that just says that for any two elements, we can make a set containing both of them. :'(
- How do we do Cartesian Products?
- (x, y) for $x \in X$, $y \in Y$ translates to
  - $\{\{x\}, \{x, y\}\}$
  - and a whole bunch of garbage proving from the axioms that the collection of all things of this form is a valid set

Figure: Look at it!

Answers to Earlier Questions: Functions

- A function from set A to set B is a subset $R \subseteq A \times B$ such that given an $a \in A$, if (a, b_1) is in R, and (a, b_2) is in R, then b_1 and b_2 must be the same.
- Intuitively: Only one output per input
- Not-so-intuitively: this condition:

$$\forall a \in A \; \forall b_1 \in B \; \forall b_2 \in B \quad (a, b_1) \in R \wedge (a, b_2) \in R \rightarrow b_1 = b_2$$

- But wait! $\forall a \in A$ is actually syntactic sugar, and so is (a, b_1).
- $$\forall a (a \in A \rightarrow \forall b_1 (b_1 \in B \rightarrow \forall b_2 (b_2 \in B \\ \rightarrow \{a, \{a, b_1\}\} \in R \wedge \{a, \{a, b_2\}\} \in R \rightarrow b_1 = b_2)))$$

Answers to Earlier Questions: Functions

But wait, we're not finished!

- We actually defined *partial functions* – we need to ensure that every input has a corresponding output!
- So we also need:

$$\forall a \, (a \in A \to \exists b \quad b \in B \land (a, b) \in R)$$

- Okay great, we did it, but...

I WANT YOU TO LOOK AT IT

$$\forall a\,(a \in A \to \exists b \quad b \in B \wedge (a,b) \in R)$$

$$\wedge \forall a\,(a \in A \to \forall b_1\,(b_1 \in B \to \forall b_2\,(b_2 \in B$$

$$\to \{a, \{a, b_1\}\} \in R \wedge \{a, \{a, b_2\}\} \in R \to b_1 = b_2)))$$

Why ZFC sucks

Your Waifu is Shit Because It:

- Overcomplicates $\mathbb{N}$
- Overcomplicates pairs
- Overcomplicates functions

Why ZFC sucks II, Electric Boogaloo

Your Waifu is Shit Because It (Continued):

- Overuses Deus Ex Machina (Non-constructivity)
- Forgets proof contents
- Is literally just FOL duct-taped to rules for manipulating weird curly brackets

The last three are not just problems with ZFC, they're problems with *any* axiomatic system built on top of classical FOL.

The Trouble With Non-Constructivity, In A Nutshell

- If we can build things just by knowing that it's *impossible* that something *doesn't* exist

The Trouble With Non-Constructivity, In A Nutshell

- If we can build things just by knowing that it's *impossible* that something *doesn't* exist
- Then we don't know crap about how it got there in the first place

Ex: Proofs by Contradiction

- Consider the statement ”There exists an irrational real number.”
- We can prove it using contradiction: Suppose that all real numbers were rational. Rationals are countable. Reals are not. Contradiction. BAM.

Ex: Proofs by Contradiction

- Consider the statement ”There exists an irrational real number.”
- We can prove it using contradiction: Suppose that all real numbers were rational. Rationals are countable. Reals are not. Contradiction. BAM.
- This tells us absolutely nothing about the answer to this slightly different question:

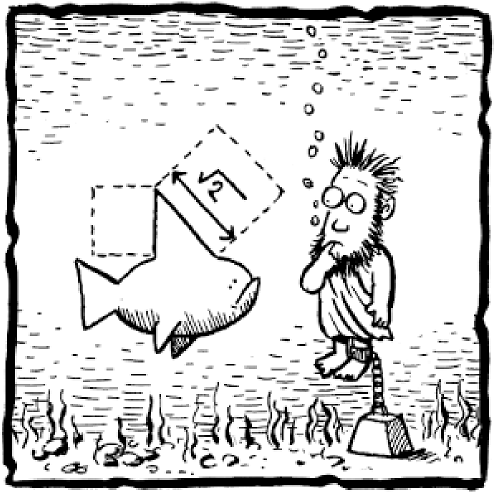
Ex: Proofs by Contradiction

- Consider the statement ”There exists an irrational real number.”
- We can prove it using contradiction: Suppose that all real numbers were rational. Rationals are countable. Reals are not. Contradiction. BAM.
- This tells us absolutely nothing about the answer to this slightly different question:
- ”Can you give me an example of an irrational number?”

Ex: Proofs by Contradiction

- Consider the statement ”There exists an irrational real number.”
- We can prove it using contradiction: Suppose that all real numbers were rational. Rationals are countable. Reals are not. Contradiction. BAM.
- This tells us absolutely nothing about the answer to this slightly different question:
- ”Can you give me an example of an irrational number?”
- ”No, and we’ll drown anybody who tries to.”

Pythagoras Time :D



Source: *Alex's Adventures in Numberland* by Alex Bellos

Ex: Proofs using the Law of The Excluded Middle

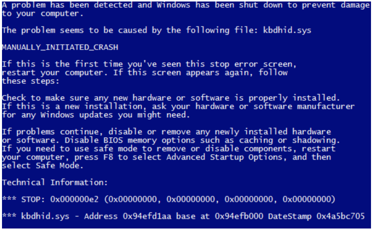
- Claim: Every computer program either halts, or it doesn't.
- Proof: smash that MF "$P \vee \neg P$"
- Different question: "Can you tell me whether or not Windows will ever *start* responding?"

Ex: Proofs using the Law of The Excluded Middle

- Claim: Every computer program either halts, or it doesn't.
- Proof: smash that MF "$P \vee \neg P$"
- Different question: "Can you tell me whether or not Windows will ever *start* responding?"
- For general programs, this is the *Halting Problem* – no algorithm exists to determine if arbitrary programs halt!

Ex: Proofs using the Law of The Excluded Middle

- Claim: Every computer program either halts, or it doesn't.
- Proof: smash that MF "$P \vee \neg P$"
- Different question: "Can you tell me whether or not Windows will ever *start* responding?"
- For general programs, this is the *Halting Problem* – no algorithm exists to determine if arbitrary programs halt!
- For Windows, at least we know that if it does, you'll be seeing one of these:



The Trouble With Forgetting Proofs, In a Nutshell

- Just because we proved something doesn't mean that it doesn't matter how we've proved it.
- If we remember how we proved things, we might be able to use them as algorithms, so long as we proved them constructively.
- General flow of the proof $\simeq$ General flow of the algorithm

The Spaghetti Parable

- I went into Little Italy and bought some uncooked spaghetti
- But I have a compulsive need to sort my spaghetti by height before I cook it.
- Ohhhhh noooo

Figure: Spaghett

Spaghetti Sort To The Rescue!

Here's the Algorithm:

- Step 1: Take all the spaghetti in one hand
- Step 2: Put your other palm at the ends of the spaghetti
- Step 3: Push to create a level surface
- Step 4: Hold the spaghetti above the surface of a table, orthogonal to it
- Step 5: Slowly lower the spaghetti down onto the table
- Step 6: Remove the first noodle which hits the table. That's the longest one, so put it to the left of your spaghetti line.
- Repeat Step 5 and 6 until no spaghetti remains

Every List of Naturals May Be Sorted: Proof

- Suppose we have a list $L = [x_1, ...x_n] \in \mathbb{N}^n$.
- L may be *sorted* if there exists a permutation $\sigma \in S_n$ such that $L_\sigma = [x_{\sigma(1)}, ...x_{\sigma(n)}]$ and for every i, $x_{\sigma(i)} \leq x_{\sigma(i+1)}$.

Every List of Naturals May Be Sorted: Proof

- Spaghetti Sort Proof
- We proceed by induction on the size of the smallest element:
  - Base: The maximum element is 0. Then the list is already sorted, dummy.
  - Induction: Suppose that we can sort all lists whose maximum element is m or smaller. Suppose we have a list with a maximum of $m + 1$. Take all of the elements which are zero and permute them to the beginning of the list. Then, consider the sublist after the zeroes. If we subtract 1 from every element, we can sort that sublist. When we're done, add 1 back to every element in that sublist. Since (+1) and (-1) are inverses and $0 \leq x$ for any $x \in \mathbb{N}$, the list is now sorted. $\square$

Another Sorting Proof

There's another, less elegant proof that "every list may be sorted", but we first need a lemma:

Lemma (Combining Two Sorted Lists)

Suppose that we have two sorted lists $L_1 = [x_1, ...x_n]$ *and* $L_2 = [y_1, ...y_m]$*. Then we can build a sorted version of* L_1 *concatenated with* L_2*.*

- Boring Proof: We know how to sort lists □
- More interesting proof: Repeatedly pull out the minimum of the two lists to build the result. (This may be viewed a double structural induction.)

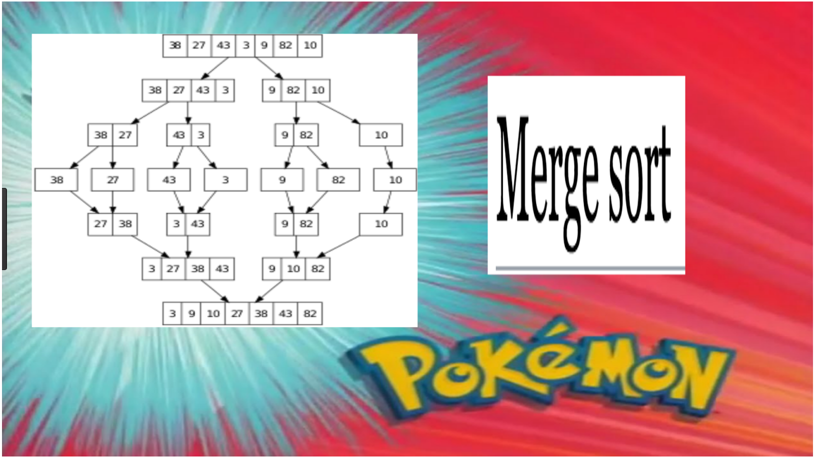
Another Sorting Proof

Proof (By Strong Induction on Length):

- Base case (length 0, 1): Lists of length zero and one are already sorted, dummy.
- Inductive Step: Suppose that we can sort any list of length less than n. Split into two halves, of sizes $floor(n)$ and $n - floor(n)$. Both of these are less than n. Sort them, recombine with the lemma. $\square$

WHO'S THAT SORTING ALGORITHM?

IT'S... Mergesort?



Sure, both Spaghetti Sort and Mergesort give ways to prove that we can sort lists. But the details of the proof *matter* if we wanna sort things. If the length of the list is denoted by n...

- Mergesort takes $O(nlog(n))$ operations
- Spaghetti Sort takes... wait, what even is its runtime?

Sure, both Spaghetti Sort and Mergesort give ways to prove that we can sort lists. But the details of the proof *matter* if we wanna sort things. If the length of the list is denoted by n...

- Mergesort takes $O(nlog(n))$ operations
- Spaghetti Sort takes... wait, what even is its runtime?
- Let m be the size of the largest element in the list.
- In the worst case, spaghetti sort decrements all n elements m times
- $O(nm)$

Sure, both Spaghetti Sort and Mergesort give ways to prove that we can sort lists. But the details of the proof *matter* if we wanna sort things. If the length of the list is denoted by n...

- Mergesort takes $O(nlog(n))$ operations
- Spaghetti Sort takes... wait, what even is its runtime?
- Let m be the size of the largest element in the list.
- In the worst case, spaghetti sort decrements all n elements m times
- $O(nm)$
- For 64-bit unsigned integers, $m = 18,446,744,073,709,551,615$.

Sure, both Spaghetti Sort and Mergesort give ways to prove that we can sort lists. But the details of the proof *matter* if we wanna sort things. If the length of the list is denoted by n...

- Mergesort takes $O(nlog(n))$ operations
- Spaghetti Sort takes... wait, what even is its runtime?
- Let m be the size of the largest element in the list.
- In the worst case, spaghetti sort decrements all n elements m times
- $O(nm)$
- For 64-bit unsigned integers, $m = 18,446,744,073,709,551,615$.
- :D That's a constant factor (!!)
- Spaghetti sort is $O(n)$ for sorting uint64's (!!!)

Sorting Out the Moral of the Story

The Bigger Picture

- ZFC is hopelessly non-constructive garbage.

The Bigger Picture

- ZFC is hopelessly non-constructive garbage.
- Who else was hopelessly non-constructive garbage?

The Bigger Picture

- ZFC is hopelessly non-constructive garbage.
- Who else was hopelessly non-constructive garbage?

ZFC =

-

The Bigger Picture

- ZFC is hopelessly non-constructive garbage.
- Who else was hopelessly non-constructive garbage?

$$\mathrm{ZFC} =$$

-
- (He got rejected from art school)

Why Care About Constructive Type Theory?

- Many modern-day proof assistants (Coq, Agda) have MLTT as a sub-language
- Proofs in constructive MLTT *always* are actually *runnable computer programs*
- We can corrupt the mathematical youth by turning them over to the dark side (Computer Science) :D

A Sampling of MLTT

- Instead of talking about *sets*, we talk about *types*.
- We have exactly two *judgments*:
  - $x \equiv y$, which means "x and y may be rewritten to each other" (for definitions, we write $:\equiv$)
  - $x : A$, which means "x belongs to the type A"
- Judgments are NOT propositions!
- If we write one down, that means it's a FACT.
- Example:

$$3 : \mathbb{N}$$

$$f : \mathbb{N} \to \mathbb{N}$$

$$f :\equiv x \mapsto x * 2$$

$$f(3) \equiv (x \mapsto x * 2)(3) \equiv 3 * 2 \equiv 6$$

(after rewriting using the definition of multiplication)

A Sampling of MLTT : Natural Numbers

Types are defined by how to build them – their *constructors* For example, the type of natural numbers

$$\mathbb{N}$$

is defined by postulating the existence of the constructors:

$$0 : \mathbb{N}$$

$$S : \mathbb{N} \rightarrow \mathbb{N}$$

Examples:

- S(0) - one
- S(S(0)) - two
- S(S(S(0))) - three

... and so on. That is, the natural numbers actually look like goddamned counting numbers in this system.

A Sampling of MLTT: Good Riddance, Curly Braces

- Functions are *the* primitive concept here. We don't have to define them with something else.
- Pairs? $A \times B$ is defined by a constructor which takes two arguments and gives you something of type $A \times B$.
- Disjoint union? $A \sqcup B$ is defined by two constructors, $inL : A \to A \sqcup B$ and $inR : B \to A \sqcup B$.

A Sampling of MLTT: Actually Doing Stuff With Types

- With constructors, we have prescribed functions *into* types.
- How do we get stuff out?
- *Recursion* and *Induction* principles.
- Basically, these say that to define a function out of a type, you only need to define what it does to the constructors.

A Sampling of MLTT: Actually Doing Stuff With Types

- With constructors, we have prescribed functions *into* types.
- How do we get stuff out?
- *Recursion* and *Induction* principles.
- Basically, these say that to define a function out of a type, you only need to define what it does to the constructors.
- Ex 1: To define a function $f : A \times B \to C$, you only need to define $f((a, b))$ for $a : A$ and $b : B$.

A Sampling of MLTT: Actually Doing Stuff With Types

- With constructors, we have prescribed functions *into* types.
- How do we get stuff out?
- *Recursion* and *Induction* principles.
- Basically, these say that to define a function out of a type, you only need to define what it does to the constructors.
- Ex 1: To define a function $f : A \times B \to C$, you only need to define $f((a, b))$ for $a : A$ and $b : B$.
- Ex 2: To define a function $f : \mathbb{N} \to A$, you only need to define $f(0)$ and $f(S(n))$, assuming that we already know $f(n)$.

A Sampling of MLTT: No More Goddamned Duct Tape

Let **0** be the type with no constructors (so there's always a function of type $\mathbf{0} \to A$ for any type A), and let **1** be the type with a single constructor $\star : \mathbf{1}$.

If we squint and read $\to$ as logical implication, **0** as "false", and **1** as "true"...

- $A \times B$ behaves like $A \wedge B$, e.g $A \wedge B \to A$:

$$pr_1 : A \times B \to A$$

$$pr_1((a, b)) :\equiv a$$

- $A \sqcup B$ behaves like $A \vee B$ e.g proof of C by cases:

$$f : (A \sqcup B) \times ((A \to C) \times (B \to C)) \to C$$

$$f((inL(a), (g, h))) :\equiv g(a)$$

$$f((inR(b), (g, h))) :\equiv h(b)$$

A Sampling of MLTT: Doing Logic

- With that, we recover logic

A Sampling of MLTT: Doing Logic

- With that, we recover logic
- Err, constructive logic. Defining the negation as $\neg A$ as $A \to \mathbf{0}$, which we can roughly read as ”to have an element of A is absurd”, it turns out that we can prove stuff like DeMorgan’s laws:
- $$\neg(A \sqcup B) \to \neg A \times \neg B$$

A Sampling of MLTT: Doing Logic

- With that, we recover logic
- Err, constructive logic. Defining the negation as $\neg A$ as $A \to \mathbf{0}$, which we can roughly read as ”to have an element of A is absurd”, it turns out that we can prove stuff like DeMorgan’s laws:
- $$\neg(A \sqcup B) \to \neg A \times \neg B$$
- ...

A Sampling of MLTT: Doing Logic

- With that, we recover logic
- Err, constructive logic. Defining the negation as $\neg A$ as $A \to \mathbf{0}$, which we can roughly read as ”to have an element of A is absurd”, it turns out that we can prove stuff like DeMorgan’s laws:
- $$\neg(A \sqcup B) \to \neg A \times \neg B$$
- ...
- oWo, what is this? We can’t actually prove

  $$\neg(A \times B) \to \neg A \sqcup \neg B$$

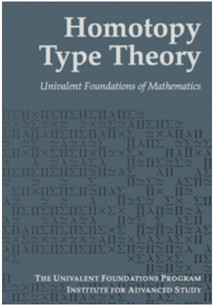
  Intuitively, ”If having A and B together is absurd, it’s not necessarily the case that having A is absurd (by itself) or having B is absurd (by itself)”

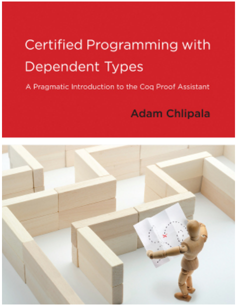
MLTT: The Big Picture

Benefits of MLTT:

- Defining types is *always* as simple as defining how we can build them
- Defining functions is *always* as simple as defining how they act on generic elements or constructors of the domain's type.
- We don't need to duct-tape logic onto MLTT – we get logic *for free* (including FOL, but this requires introducing *dependent types*, a topic for another day)
- Everything is constructive. We can't prove the law of the excluded middle, the law of double negation, etc.
- If we use constructive MLTT in e.g. Agda or Coq, we can compile it to Haskell, C, Common Lisp, ... etc. and get *runnable code* from our proofs.

Further References





Questions?