Formalising SOTA corner-free set construction in Lean

Gareth Ma

April 24, 2024

Abstract

Placeholder. [**GreenTao12**] Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Contents

[1] Mac Lane would be proud.

1 Introduction

Important to remember that the essay should still be mathematical, so it should contain e.g. in depth explanation of dependent type theory. Remember to summarise the sections of the paper (In section X we ...)

2 Mathematical Background

2.1 3AP Sets

✓Mathematical background of corner-free sets, 3AP sets

Extremal combinatorics is the study of the following question: given a set of objects, find the smallest/largest (subset of) object which has a certain combinatorial property. Possibly the most famous example is the Ramsey numbers $R(m,n)$, which are the numbers N such that every N-vertex graph contains either a clique of size m or an independent set of order n [**SamJacques2024**]. The numbers' existence is proven by Ramsey in 1947 [**Erdös47**]. Another classical example is Waring's problem, which is a natural extension of Lagrange's four square theorem. It asks for $g(k)$, the smallest integer N such that every integer is the sum of N k^{th} powers. Lagrange's result trivially implies $g(2)=4$. Both of these problems are still wide open; the interested readers can refer to [**Ma2024**], a talk given by the author, for a brief survey on the second problem.

In order to develop tools to attack these classical problems, researchers turn to even simpler problems. One such problem is the 3-AP (arithmetic progression) problem, which for a given integer N asks for the size $v(N) = v_3(N)$ of the largest set $S \subseteq \mathbb{N} \cap [1,N]$ such that any three distinct integers $a,b,c \in S$, we have $a + c \neq 2b$. For example, for $N = 10$, we can choose $S = \{1,2,4,5,10\}$, and it is easy to check that all $\binom{5}{3} = 10$ subsets of 3 terms do not contain 3-APs. In 1942, Salem and Spencer proved that the size of such sets can be quite close to linear, providing a construction with size $N^{1-(\log 2+\varepsilon)/\log\log N}$ for all $\epsilon > 0$ [**SalemSpencer1942**]. In particular, this means that for any $\epsilon > 0$, we have $v(N) \notin O(N^{1-\epsilon})$. In 1946, Behrend gave an improved construction which achieves $N^{1-(2\sqrt{2\log 2}+\varepsilon)/\sqrt{\log N}}$ [**Behrend1946**]. This result is remarkable for three reasons: (1) The paper's title is the same as Salem and Spencer's paper, which confused many mathematicians. (2) The construction is very simple, with the main construction taking only a page. (3) To the knowledge of the author, this is the current best known lower bound (asymptotically).

One may also ask for upper bounds on $v(N)$. The first nontrivial upper bound on $v(N)$ was given by Roth [**Roth1953**], which showed that $v(N) = o(N)$. Subsequently, the result was refined by Heath-Brown [**HeathBrown1987**], Szemerédi [**Szemerédi1990**] and others. To the knowledge of the author, the current best known result in this direction is $v(N) < N/2^{O((\log N)^{\epsilon})}$ for all $\epsilon > 0$, achieved by Kelley and Meka [**KelleyMeka2023**] in 2023. These results are interesting as most of them require advanced analytic methods such as higher Fourier analysis on finite groups. This leads to developments such as the higher Gowers norms, for which Gowers received the Fields medal for in 1998 [**LLMM1999**]. It is an open problem to close the massive gap between the lower and upper bounds.

Finally, there are many generalisations of the problem. For example, Szemerédi extended Roth's Theorem to longer arithmetic progressions, proving that $v_k(N) = o(N)$ [**Szemerédi1975**]. In fact, Szemerédi proved a stronger theorem: every subset of natural numbers $A \subseteq \mathbb{N}$ with positive upper density contains arithmetic progressions of arbitrary length. In 2008, Green and Tao extended the result to the prime numbers, proving that the primes (which have zero upper density) contain arbitrarily long arithmetic progressions [**GreenTao2008**].

2.2 Corner-free Sets

Link the above to corner-free sets

Behrend's construction and now Ben Green's construction

3 Lean for the Working Mathematician [2]

Formally, Lean is an interactive theorem prover based on a Martin-Löf (dependent) Type Theory (MLTT) [**MartinLöf1984**]. This section aims to give a short introduction to the theory and how it works as the theory underlying a theorem prover. For an in-depth exposition of the theory, the reader is advised to consult [**Rijke2022**].

3.1 Dependent Type Theory

✓Introduce basics of dependent type theory, and compare it with set theory (e.g. instead of $x \in X$, we have $x : X$).

This is a summary of type theory, from the perspective of a practitioner. [3]

Type theory aims to be an alternative foundation for Mathematics, in place of the traditional set theory. It consists of *elements* and *types*, along with a set of *inference rules* which corresponds to axioms from logic and set theory.

Examples of *elements* include the integer 3, the propositions "$P := 1 = 2$", "$Q := \forall x \in \mathbb{Z}, 2x \in \text{Even}$", the sets $\mathbb{Q}$ and $\{2,3,5\}$. These each have a type. For example, 3 belongs to the type **Nat**, the type of natural numbers, and we denote this by a *judgement* $\vdash 3 : \textbf{Nat}$ (the $\vdash$ indicates the start of a judgement, and I will omit it when it is clear). There is also a type for all nice[4] propositions called **Prop**, and we may write $P, Q : \textbf{Prop}$. The types of $\mathbb{Q}$ and $\{2,3,5\}$ can be **NumberField** and **Set** $\mathbb{Z}$, which are the types for number fields and sets of integers respectively.

Everything in type theory has a type. In particular, there is a type of all "normal types"[5] (e.g. **Nat**, **Set** $\mathbb{Z}$ and **Prop**), which we denote by **Type** or $\textbf{Type}_1$. For example, the judgements $\textbf{Nat} : \textbf{Type}$ and $\textbf{Prop} : \textbf{Type}$ are valid. From this, we see that there is an infinite number of judgements $\textbf{Type}_i : \textbf{Type}_{i+1}$, for all $i \geq 1$. For us and for most cases, higher types ($\textbf{Type}_i$ for $i \geq 2$) are not required, so we will be ignoring them.

Note that the "colon relation" $x : X$ is not transitive. For example, $2 : \mathbb{N}$ and $\mathbb{N} : \text{Type}$ are valid judgements, but not $2 : \text{Type}$.

An important class of elements is the functions. **add some stuff here. I want the notation $T_1 \to T_2$.**

As the reader might have noticed, we have not done anything truly innovative. In fact, all concepts above naturally correspond to concepts from set theory. Types can be thought of as a collection of things, just like sets, and $x : X$ can be thought of as alternative notation for $x \in X$.

[2] Mac Lane would be proud.

[3] I am omitting many details, such as universes, contexts, equality types etc. for brevity.

[4] First-order logical propositions should suffice.

[5] This is not standard terminology, but rather to distinguish the types above from $\textbf{Type}_1$ or further types.

We now turn to the *inference rules*, which are axioms within the type theory that determine how elements and types interact. Here is an inference rule that represents type substitution:

$$\frac{\vdash t : T_1 \quad \vdash h : T_1 = T_2}{\vdash t : T_2}$$

The inference rule is expressed in Gentzen's notation [**Gentzen1935a**], [**Gentzen1935b**]. The "input" judgements (also called *hypotheses*) are above the line and the "output" judgement is below the line, and the rule as a whole states that given the hypotheses (in a context), one can create the output judgement. In informal English, this is saying is that "given an element t of type T_1 and an element h of type $T_1 = T_2$, we can produce an element of type T_2". In set theory, this translates to the tautology "if $x \in X$ and $X = Y$, then $x \in Y$", which is true as sets are determined by their elements.

Using this notation, we can express function applications above by simple inference rules:

$$\frac{\vdash f : \alpha \to \beta \quad \vdash a : \alpha}{\vdash f.a : \beta} \qquad \frac{\vdash T : \mathbb{N} \to \text{Type} \quad \vdash g : \prod_{n:\mathbb{N}} T(n) \quad \vdash n : \mathbb{N}}{\vdash g.n : Tn}$$

Another example of inference rules would be that of an *inductive type*, such as the type of natural numbers `Nat` or $\mathbb{N}$. We can define the type inductively, analogous to the Peano axioms, via two introduction rules: one for the zero elements 0, and one for constructing successors. We can express the two rules in Gentzen's notations simply as:

$$\frac{}{\vdash 0_{\mathbb{N}} : \mathbb{N}} \qquad \frac{}{\vdash \text{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}}$$

The first rule says that (with no hypothesis, that is, out of "thin air") an element $0_{\mathbb{N}}$ can always be constructed, while the second rule says that there is a function $\text{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}$ that constructs new $\mathbb{N}$. The type $\mathbb{N}$ is *defined* to be all elements constructable via these two methods.

We shall not continue in this path of type theory, as it quickly ventures into details of type theory that we will not need to understand Lean. The interested reader can refer to [**Rijke2022**] for a detailed resource on the topic.

3.2 DTT and Maths

Mention the relation of DTT with Math, that formalising a proof corresponds to creating a term with the correct type.

3.3 Curry-Howard Correspondence

Connect Lean with Mathlib: as demonstrated above, there is a strong relation betwen Mathematical proofs and typed expressions.

We have seen how the constructors of $\mathbb{N}$ are expressed in formal type theory language, and also how our intuition on equalities translate to type theoretical language. This suggests there is a strong relation between Mathematical proofs and typed terms, and indeed there is, via the *Curry-Howard correspondence*.

Recall that a type T can vaguely be thought of as a set X_T containing all elements of that type. For example, when $T = \mathbb{N}$, the set $X_{\mathbb{N}}$ is, well, just $\mathbb{N} = \{0,1,2,\cdots\}$. What about when $T = (2+2=4)$? The set X_T will be the set of all elements of type T, i.e. $X_T = \{x : T\}$. One interpretation of such elements $x \in X_T$ is that they are *proofs* of the proposition T.

To make this interpretation meaningful, further suppose that we have a term $f : T \to T'$, where $T' = (2+2)+1 = 4+1$. Informally, the term f simply adds 1 to both sides of an equality. By the inference rule for function applications, we can use $x : T$ and $f : T \to T'$ to construct $f.x : T'$. In particular, if we interpret terms $x : T$ and $f.x : T'$ as proofs of the propositions T and T' respectively, then $f : T \to T'$ serves not just as a function, but also a "proof step" in a Mathematical proof; for "proof steps" I mean theorems, lemmas, claims or algebraic steps that appear in a normal Mathematical proof on paper.

4 Limits in Lean

We follow the presentations of [**HIH2013**] and [**BCM2020**].

4.1 Filters

Introduce the mathematical background for filters, how they replace the traditional limits, and give some examples.

In first year of undergraduate, we have all seen different flavours of limits. For sequences we have $\lim_{n\to\infty} a_n$, while for functions we have $\lim_{x\to x_0} f(x)$. Beyond these two, there are a lot more variants: $\lim_{x\to x_0^+} f(x)$, $\lim_{x\to x_0 x\neq x_0} f(x)$, $\lim_{x\to +\infty} f(x)$ and several other negative counterparts. Intuitively they are all the same concept, but rigorously they have slightly different definitions: for a regular limit one writes $\forall \varepsilon > 0, \exists \delta > 0, (\forall x, |x - x_0| < \delta \implies \cdots)$, while for limits at infinity one writes $\forall \varepsilon > 0, \exists X > 0, (\forall x \geq X, \cdots)$. The problem is even worse when one takes into account what the limiting value is. For example, even the definitions of $\lim_{x\to x_0} f(x) = 3$ and $\lim_{x\to x_0} f(x) = +\infty$ differ.

Filters are introduced by Henri Cartan [**Cartan1937a**], [**Cartan1937b**] in order to unify the different notions of limits in topology.

4.2 Lean 101: Proving a Limit

Walkthrough the formalisation of the proof of $\lim_{x\to\infty} \frac{1}{1+x} = 0$, since this basically comes up later on.

5 Behrend’s 3AP-free Construction

Detailed mathematical proof of Behrend’s construction, including the interpretation using Freiman isomorphism. This way I can claim *original work*.

Before we dive into Ben Green’s result for corner-free sets, let us look at Behrend’s 3AP-free construction from 1946, which is simple to describe and serves to motivate Ben Green’s result.

6 Implementation: Construction

Describe the implementation of the construction (`construction.lean`) in detail.

7 Implementation: Asymptotics

Describe the implementation of the asymptotics proof (`cp.lean` and `cp2.lean`) in detail.

8 Implementation: Connecting the Dots

Words

9 Correctness via `eval`

9.1 Computability typeclasses

9.2 ?

10 Conclusion

11 Acknowledgement

I would like to thank my supervisor, Dr. Damiano Testa, for his tremendous support, encouragement, and the insightful discussions about the project during our weekly meetings. I would also like to thank Julian Berman, who is the creator of the `lean.nvim` plugin. This project will be drastically slowed down without the full power of my NeoVim setup. Finally, I would like to thank the Mathlib community as a whole, who answered my questions on the Leanprover Zulip server.