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 [13]. 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 [11], 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 $\epsilon > 0$ [16]. 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 $N^{1-(2\sqrt{2\log 2}+\varepsilon)/\sqrt{\log N}}$ [1]. 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 [15], which showed that $v(N) = o(N)$. Subsequently, the result was refined by Heath-Brown [8], Szemerédi [17] 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 [10] 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 $v_k(N) = o(N)$ [18]. 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 [7].

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) [12]. 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 [14].

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* $\vdash 3 : \mathbf{Nat}$ (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 $P, Q : \mathbf{Prop}$. 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 $\mathbf{Type}_1$. For example, the judgements $\mathbf{Nat} : \mathbf{Type}$ and $\mathbf{Prop} : \mathbf{Type}$ are valid. From this, we see that there is an infinite number of judgements $\mathbf{Type}_i : \mathbf{Type}_{i+1}$, for all $i \geq 1$. For us and for most cases, higher types ($\mathbf{Type}_i$ 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 $\mathbf{Type}_1$ 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 [5], [6]. 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.

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:

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

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 $\mathrm{succ}_{\mathbb{N}} : \mathbb{N} \to \mathbb{N}$ that constructs new $\mathbb{N}$. The type $\mathbb{N}$ is *defined* to be all elements constructable via these two methods.

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

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

And function *constructors* by the following inference rules:

$$\frac{x : \alpha \vdash b(x) : \beta}{\vdash \lambda x.b(x) : \alpha \to \beta} \qquad \frac{\vdash T : \alpha \to \mathrm{Type} \quad a : \alpha \vdash b(a) : T(a)}{\vdash \lambda x.b(x) : \prod_{a:\alpha} T(a)}$$

To give one final example, here is a computation rule for functions:

$$\frac{\vdash p : \alpha \quad \vdash h : \alpha \to \beta}{\vdash h(p) : \beta}$$

As we will see in the next sections, it plays a fundamental role in how theorem provers work.

We shall not continue in this direction 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 [14] for a detailed resource on the topic.

3.2 The 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 $T' = (2+2)+1 = 4+1$, where 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.

To give another example, let us prove the transitivity of implications from classical logic, i.e. that for propositions P,Q,R, if P implies Q and Q implies R, then P implies R. Through the Curry-Howard correspondence, it suffices to construct a term of type $P \to R$, given terms $h_1 : P \to Q$ and $h_2 : Q \to R$. The term desired can be constructed by $h_2 \circ h_1$, or more explicitly, by the term $f : P \to R$ given by $f(p) = h_2(h_1(p))$. In the language of inference rules, it can be written as follows :

$$\dfrac{\dfrac{\dfrac{\vdash h_1 : P \to Q}{p : P \vdash h_1(p) : Q} \quad \vdash h_2 : Q \to R}{p : P \vdash h_2(h_1(p)) : R}}{\vdash \lambda p.h_2(h_1(p)) : P \to R}$$

This is precisely how theorem provers such as Lean work via the Curry-Howard correspondence: by treating Mathematical statements as types and Mathematical proofs as terms of that type, a *valid* DTT term of the type would imply that the corresponding Mathematical statement is correct. Moreover, the validity of a DTT term is relatively easy to check by a computer: it boils down to checking if the types match up and everything is "intuitively correct", and definitely easier than checking a Mathematical proof filled with informal lingual.

3.3 DTT and Lean

✓Given examples of Lean proofs, relating to the CH correspondence.

Now that we have established the connection between DTT and Mathematics, as well as DTT and theorem provers, it is time to connect Mathematics with theorem provers, of which of course we focus on Lean.

Lean is a theorem prover built on top of a Dependent Type Theory, more specifically the Martin-Löf Type Theory (with several extra "add-ons"). Of course, users do not type in typed lambda expressions or inference trees directly into Lean, or else no Mathematicians would use it. Instead, users are able to type commands that manipulate the context (goal state + hypotheses) that mimicks paper proofs - see below for examples [6]. Lean has several different parts, such as the *elaborator* and *type inferencer*, which translate such commands into DTT terms (which may look like $\mathrm{Eq}((\lambda x.fx)y)(fy)$, for example). Finally, these terms are passed on to the Lean 4 kernel, which verifies the term is indeed correct. An extremely important detail is that the Lean 4 kernel is quite small, around 18200 lines of code (via `find ~/git/lean4/src -path "*/library/*.cpp" -or -path "*/kernel/*.cpp" | xargs cloc`).

As an example, let us formalise our proof for $(P \implies Q) \wedge (Q \implies R) \implies (P \implies R)$. From 3.2, our task is reduced to constructing a term of type $P \to R$, given two terms $h_1 : P \to Q$ and $h_2 : Q \to R$. In Lean, it is written as follows:

```
example {P Q R : Prop} (h₁ : P → Q) (h₂ : Q → R) : P → R := by
  intro p
  have q := h₁ p
  exact h₂ q
```

Let us unpack this slowly. The first line begins with the `example` keyword, which is used to indicate the start of a(n unnamed) proof. Next, the `{P Q R : Prop}` and `(h₁ : P → Q) (h₂ : Q → R)` tokens are the hypotheses of our claim. Here, in the curly braces we are stating the judgements $\vdash P,Q,R : \mathrm{Prop}$, while the latter ones in parentheses are the judgements $\vdash h_1 : P \to Q$ and $\vdash h_2 : Q \to R$. After the hypotheses, we have the goal state, that is, the type of which we would like to construct a term for. Here, it is the type $P \to R$. Simple!

From the second line onwards, we begin to write the proof of the statement. Here, I followed the proof given by the inference tree [**ch-tree**]. On the second line, we have `intro p`. Notice that at this point, our goal (type) is of the form $P \to R$, and from the inference rules of the function constructors, we know that it suffices to "give" an element $r : R$, for every element $p : P$. The `intro` keyword (called a *tactic*) effectively introduces the $p : P$ into the context. On the inference tree, it is turning the goal state from the final layer to the second last layer:

$$\frac{\cdots}{\vdash P \to R} \quad \longrightarrow \quad \cfrac{\cfrac{\cdots}{p : P \vdash R}}{\vdash P \to R}$$

Now that we have $p : P$ in context, we can proceed by applying h_1 at it. More specifically, we

[6]We will only use tactic mode in this essay.

introduce a term $q := h_1(p)$. This is done by the `have` keyword on the third line. The `q` on the left is the new variable name, while the `h₁ p` on the right is the definition.

$$\cfrac{\vdash h_1 : P \to Q}{\cdots} \quad \longrightarrow \quad \cfrac{\cfrac{\vdash p : P \quad \vdash h_1 : P \to Q}{\color{red}{\vdash h_1(p) : Q}}}{\cdots}$$

Finally, we can combine this term with h_2 to get $h_2(q) = h_2(h_1(p))$, which is a term of our goal, the type R. To conclude the proof with a term, we use the `exact` keyword, followed by the term. This completes the inference tree from 3.2.

For those who are curious, it is possible to print the full expression of type R constructed via the `#print` command within Lean. For the code above, here is the output:

```
theorem f : ∀ (P Q R : Prop), (P → Q) → (Q → R) → P → R :=
  fun P Q R h₁ h₂ p ⇒ let_fun q := h₁ p; h₂ q
```

We see that there is a `let_fun` statement which can be inlined (or substituted), but otherwise it is a single expression $\lambda P.\lambda Q.\lambda R.\lambda h_1.\lambda h_2.\lambda p.h_2 h_1 p)$.

3.4 Necessity of Mathlib

Give an example of a Mathematical proof to motivate the existence of Mathlib, introduce it, and give an example on how to use it.

4 Limits in Lean

We follow the presentations of [9] and [2].

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 [4], [3] 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.

References

[1] Felix A Behrend. “On sets of integers which contain no three terms in arithmetical progression”. In: *Proceedings of the National Academy of Sciences* 32.12 (1946), pp. 331–332 (cit. on p. 4).

[2] Kevin Buzzard, Johan Commelin, and Patrick Massot. “Formalising perfectoid spaces”. In: *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. POPL ’20. ACM, Jan. 2020. DOI: 10.1145/3372885.3373830. URL: http://dx.doi.org/10.1145/3372885.3373830 (cit. on p. 11).

[3] Henri Cartan. “Filtres et ultrafiltres”. In: *CR Acad. Sci. Paris* 205 (1937), pp. 777–779 (cit. on p. 11).

[4] Henri Cartan. “Théorie des filtres”. In: *CR Acad. Sci. Paris* 205 (1937), pp. 595–598 (cit. on p. 11).

[5] Gerhard Gentzen. “Untersuchungen über das logische Schließen. I”. In: *Mathematische Zeitschrift* 39.1 (Dec. 1935), pp. 176–210. ISSN: 1432-1823. DOI: 10.1007/bf01201353. URL: http://dx.doi.org/10.1007/BF01201353 (cit. on p. 7).

[6] Gerhard Gentzen. “Untersuchungen über das logische Schließen. II”. In: *Mathematische Zeitschrift* 39.1 (Dec. 1935), pp. 405–431. ISSN: 1432-1823. DOI: 10.1007/bf01201363. URL: http://dx.doi.org/10.1007/BF01201363 (cit. on p. 7).

[7] Benjamin Green and Terence Tao. “The primes contain arbitrarily long arithmetic progressions”. In: *Annals of Mathematics* 167.2 (Mar. 2008), pp. 481–547. ISSN: 0003-486X. DOI: 10.4007/annals.2008.167.481. URL: http://dx.doi.org/10.4007/annals.2008.167.481 (cit. on p. 5).

[8] D. R. Heath-Brown. “Integer Sets Containing No Arithmetic Progressions”. In: *Journal of the London Mathematical Society* s2-35.3 (June 1987), pp. 385–394. ISSN: 0024-6107. DOI: 10.1112/jlms/s2-35.3.385. URL: http://dx.doi.org/10.1112/jlms/s2-35.3.385 (cit. on p. 4).

[9] Johannes Hölzl, Fabian Immler, and Brian Huffman. “Type Classes and Filters for Mathematical Analysis in Isabelle/HOL”. In: *Interactive Theorem Proving*. Ed. by Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 279–294. ISBN: 978-3-642-39634-2 (cit. on p. 11).

[10] Zander Kelley and Raghu Meka. *Strong Bounds for 3-Progressions*. 2023. arXiv: 2302.05537 [math.NT] (cit. on p. 4).

[11] Gareth Ma. *Yet Another Talk about Prime Numbers*. 2024 (cit. on p. 4).

[12] Per Martin-Löf and Giovanni Sambin. *Intuitionistic type theory*. Vol. 9. Bibliopolis Naples, 1984 (cit. on p. 6).

[13] Sam Mattheus and Jacques Verstraete. *The asymptotics of $r(4,t)$*. 2024. arXiv: 2306.04007 [math.CO] (cit. on p. 4).

[14] Egbert Rijke. "Introduction to homotopy type theory". In: (2022). arXiv: 2212.11082 [math.LO] (cit. on pp. 6, 7).

[15] K. F. Roth. "On Certain Sets of Integers". In: *Journal of the London Mathematical Society* s1-28.1 (Jan. 1953), pp. 104–109. ISSN: 0024-6107. DOI: 10.1112/jlms/s1-28.1.104. URL: http://dx.doi.org/10.1112/jlms/s1-28.1.104 (cit. on p. 4).

[16] Raphaël Salem and Donald C Spencer. "On sets of integers which contain no three terms in arithmetical progression". In: *Proceedings of the National Academy of Sciences* 28.12 (1942), pp. 561–563 (cit. on p. 4).

[17] E. Szemerédi. "Integer sets containing no arithmetic progressions". In: *Acta Mathematica Hungarica* 56.1–2 (Mar. 1990), pp. 155–158. ISSN: 1588-2632. DOI: 10.1007/bf01903717. URL: http://dx.doi.org/10.1007/BF01903717 (cit. on p. 4).

[18] E. Szemerédi. "On sets of integers containing k elements in arithmetic progression". In: *Acta Arithmetica* 27 (1975), pp. 199–245. ISSN: 1730-6264. DOI: 10.4064/aa-27-1-199-245. URL: http://dx.doi.org/10.4064/aa-27-1-199-245 (cit. on p. 4).