Formalising large corner-free sets

Gareth Ma

April 25, 2024

Abstract

For my third year project, I formalised Ben Green's recent construction of large corner-free sets [9] in the Lean 4 Theorem Prover. My formalisation can be found at . This essay complements the project by explaining what mathematical formalisation actually is, and my contribution throughout the project.

Contents

[1] Mac Lane would be proud.

1 Mathematical Background

1.1 3-AP-free Sets

Extremal combinatorics studies questions of the following form: 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 [19]. The numbers' existence is proven by Ramsey in 1947 [6]. 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 [16], 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-free (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 $\epsilon > 0$ [23]. 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 [22], which showed that $v(N) = o(N)$. Subsequently, the result was refined by Heath-Brown [11], Szemerédi [24] 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 [13] 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 [14]. 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 $v_k(N) = o(N)$ [25]. 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. There are also various interesting results by replacing $\mathbb{N}$ with a vector space $\mathbb{F}_q^n$ or other abelian groups. In 2016, Ellenberg and Gijswijt, based on the *polynomial method* by Croot, Lev and Pach, obtained a groundbreaking (exponential improvement) result with a two page proof, for an upper bound on the size of

3-AP-free sets in $\mathbb{F}_q^n$ [5]. 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 [10].

1.2 Corner-free Sets

By increasing dimensions, we can obtain more patterns than just arithmetic progressions. This motivates the study of *corner-free* sets in $\mathbb{Z}^2$, which are sets $S \subseteq \mathbb{Z}^2$ such that for three distinct elements $(x, y), (x', y), (x, y') \in S$, we have $x' - x \neq y' - y$. Unlike 3-AP-free sets, the maximal corner-free sets are less studied. One of the first large constructions of corner-free sets can be obtained by slightly modifying Behrend's 3-AP-free set construction from 1946, achieving $2^{-c\sqrt{\log_2 N}+o(\sqrt{\log_2 N}}$, where $c = 2\sqrt{2} \approx 2.828$; see the discussion in Section 4 and Section 5. The next improvement would have to wait until [15] in 2021, where they used algorithmic ideas from communication complexity theory and 3-party protocols to construct a large corner-free set with $c = 2\sqrt{\log_2 e} \approx 2.404$. Shortly after, Green was able to push the construction further, obtaining $c = 2\sqrt{2\log_2\left(\frac{4}{3}\right)}$ [9]. It is precisely Green's result that we formalise in this project.

2 Lean for the Working mathematician [2]

Formally, Lean is an interactive theorem prover based on a Martin-Löf (dependent) Type Theory (MLTT) [17]. 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 [21].

2.1 Dependent Type Theory

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 we shall 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 class of (independent) functions. We are all familiar with such subjects: for A and B, suppose that for every $a : A$ we have a corresponding term $b(a) : B$. Then, we can construct a *function* $f : A \to B$, which is an element with a "computation rule" or "application rule": given an element $a : A$, we can apply f on a to get $f(a) := b(a)$.

However, this is often too restrictive for a type theory. For example, consider the *type function* (a function of types) Vec that sends natural numbers $n : \mathbb{N}$ to the $\textbf{Vec}_n$, the type of all length-n vectors. The type of this element would be of the form $\mathbb{N} \to *$, but we cannot describe $*$ in a satisfactory way; sure, we can say $\text{Vec} : \mathbb{N} \to \textbf{Vec}$, the type of all vectors of any length, but that

[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.

loses information about the type. For example, we cannot compose Vec with $\text{Dup}_n : \mathbf{Vec}_n \to \mathbf{Vec}_{2n}$.

This motivates the notion of *dependent function types*. As its name suggests, it is a function type where the codomain type depends on the argument from the domain. More rigorously, consider a type family B over A, i.e. a collection of types $B(x) : \mathbf{Type}$ for every $a : A$, as well as elements $b(a) : B(a)$ for every $a : A$. We can define a **dependent type** $\prod_{(x:A)} B(x)$, and a **dependent function** of such type $f : \prod_{(x:A)} B(x)$, which satisfies the computation rule $f(a) :\equiv b(a)$.

It is important to note that we have not done anything 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 $x \in X$. Independent functions even have the same notation as their set-theoretic counterparts!

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 [7], [8]. 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:

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

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.

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

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

And function *constructors* by the following inference rules:

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

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 [21] for a detailed resource on the topic.

2.2 The Curry-Howard Correspondence

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" we 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 :

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

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.

2.3 DTT and Lean

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, computed 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 2.2, 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, we again follow the proof given by the inference tree 2.2. 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

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

`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:

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

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

$$\frac{\vdash h_1 : P \to Q}{\cdots} \quad \longrightarrow \quad \frac{\dfrac{\vdash p : P \quad \vdash h_1 : P \to Q}{\vdash h_1(p) : Q}}{\cdots}$$

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 2.2.

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 $\lambda P.\lambda Q.\lambda R.\lambda h_1.\lambda h_2.\lambda p.h_2 h_1 p)$.

2.4 Necessity of Mathlib

Apart from several technical details (such as subtypes), we have described how to formalise set-theoretic statements in Lean, meaning that theoretically it is possible to construct most of mathematics in Lean now. Of course, it would be undesirable for everyone to maintain their own implementation of a mathematical library / definitions. Therefore, a community project called Mathlib was formed. To quote its GitHub README,

Quotation

> Mathlib is a user maintained library for the Lean theorem prover. It contains both programming infrastructure and mathematics, as well as tactics that use the former and allow to develop the latter.

For example, it contains a large amount of definitions and lemmas on mathematical objects from various fields, ranging from real analysis, algebraic geometry, number theory and linear algebra. No more defining $\mathbb{R}$ via Dedekind cuts from scratch for each project, as it is defined as `ℝ` or `Real` type in the `MATHLIB_ROOT/Mathlib/Data/Real/Basic.lean` file. To use it in a project file, simply include the line `import Mathlib.Data.Real.Basic` at the top of the file.

Apart from mathematical definitions, Mathlib also contains a large number of *tactics*, which are user commands that speeds up the process of proving statements. For example, suppose that one has to prove $a + b + c - a - b - c = 0$ for a commutative additive group G and $a, b, c : G$. Without automation, the best way to approach this would be to swap pairs of elements until terms of opposite signs are adjacent to each other. Here is one possible solution:

```
example {G : Type*} [AddCommGroup G] {a b c : G} :
    a + b + c - a - b - c = 0 := by
  rw [add_sub_right_comm, add_sub_right_comm, add_sub_cancel_left, add_sub_cancel_right, sub_self]
```

This is tedious and does not scale well as the number of terms increases. In Mathlib, there is a tactic called `abel` which is specifically designed to handle such goals. Instead of the long `rw` chain above, one can instead write

```
example {G : Type*} [AddCommGroup G] {a b c : G} :
    a + b + c - a - b - c = 0 := by
  abel
```

2.5 Pitfalls of Lean & Mathlib

Lean functions are always *total*. In other words, all functions must have a definition for every value in its domain. This is naturally satisfied already. For example, the addition operartor $+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is total. However, the subtraction operator is not, as $3 - 5$ does not have a value in $\mathbb{N}$. The way Lean handles these cases is by assigning them *junk values*. In the case of subtracting natural numbers, Lean defines $a - b = 0$ whenever $a < b$. A more interesting example is integer division, where $13/2$ again does not have a value in $\mathbb{N}$. This time, Lean assigns the value 6 to it, which is $\lfloor \frac{13}{2} \rfloor$. These are design choices that are made somewhat arbitrarily, and must be handled separately. An example of this can be found in Section 6.2, where we intentionally write `q ≤ 2 * b` instead of `q / 2 ≤ b`.

3 Limits in Lean

We follow the presentations of [12] and [2].

3.1 Filters

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. They are important for us[7] as limits (and hence asymptotics) are defined using the notion of filters, rather than the conventional topological / $\varepsilon - \delta$ deifnition. It is simplest to give the definition, then see how typical examples of filters appear in the context of limits.

Definition 1 (Filters [18])**.** *A **filter** on X is a collection $\mathcal{F}$ of subsets of X such that*

1. $X \in \mathcal{F}$
2. $U \in \mathcal{F} \land U \subseteq V \implies V \in \mathcal{F}$
3. $U \subseteq \mathcal{F} \land V \subseteq \mathcal{F} \implies U \cap V \in F$

By definition, a filter $\mathcal{F}$ on X is a subset of $\mathcal{P}(X)$, so it also corresponds to a function $\{0,1\}^X$, which has the usual poset structure of $S \leq T \iff S \subseteq T$ and a unique maximal element X. Then, requirement 2 translates to that if $U \in \mathcal{F}$, then for all elements V such that $U \subseteq V$, $V \in \mathcal{F}$. If we orient $\{0,1\}^X$ such that for $U \leq V$, the edge $U \to V$ is going "up", then requirement 2 says that for all $U \in \mathcal{F}$, the subgraph "above" U is also included in $\mathcal{F}$. The third requirement is a natural closeness property: $\mathcal{F}$ should be closed / stable under finite intersections.

Example 1. *Every topological space X and every point $x_0 \in X$ gives rise to a filter $\mathcal{F} = \mathcal{N}_{x_0}$ of neighbourhoods of x_0: $\mathcal{N}_{x_0} := \{V \subseteq \mathcal{X} : \exists U \subseteq V \text{ s.t. } U \text{ open} \land x_0 \in U\}$. As we will see, this corresponds to limits $\lim_{x\to x_0}$.*

Example 2. *The cofinite sets of the space $X = \mathbb{N}$ also form a filter $\mathcal{F}_{\mathbb{N}} = \{\mathcal{A} \subseteq \mathbb{N} : |\mathcal{A}^c| < \infty\}$. This naturally corresponds to limits $\lim_{n\to\infty}$, or generally statements happening "eventually". This also makes sense intuitively, as an "eventual" event satisfies the three requirements for a filter.*

[7]It can be argued that one does not have to *understand* filters to use filters, but of course it is better to.

Example 3. *Let $X = \mathbb{R}$. We can consider the filter $\mathcal{N}_{+\infty}$ generated by segments $[A, +\infty) \subset X$ for all $A \in \mathbb{R}$. That is, $\mathcal{N}_{+\infty}$ is the smallest filter / intersection of all filters containing all the segments, which is simply the collection of all subsets of $\mathcal{N}$ containing $[A, +\infty)$ for some $A \in \mathbb{R}$. This unifies the limits $\lim_{x\to+\infty}$. The filter $\mathcal{N}_{-\infty}$ is defined analogously. Looking ahead, the filters $\mathcal{N}_{+\infty}$ and $\mathcal{N}_{-\infty}$ are called* `atTop` *and* `atBot` *respectively, and will come up very often.*

Example 4. *Let X be any space and $S \subseteq X$. The collection of supersets of X forms a filter, called the principal filter of S in X.*

Given two filters $\mathcal{F}, \mathcal{G}$ of X, if $\mathcal{F} \subseteq \mathcal{G}$, then $\mathcal{G}$ is a *refinement* of *cF*, and we write $\mathcal{G} \leq \mathcal{F}$. By translating between the set-theoretic and filter-theoretic perspectives, it is easy to see that $\leq$ defines a poset structure on the space of filters of X.

Now that we have a handful of examples of filters, let us consider how the language of limits translate to language of filters, before giving the proper definition. For example, suppose that $\lim_{x\to 2} f(x) = 3$ for some $f : \mathbb{R} \to \mathbb{R}$. The most basic definition we can give is that

$$\forall \varepsilon > 0, \exists \delta > 0, 0 < |x - 2| < \delta \implies |f(x) - 3| < \varepsilon$$

Thinking more topologically, we may say that (where $\mathcal{B}_\delta(x_0)$ is the *punctured* open ball of radius $\delta > 0$ around x_0)

$$\forall \varepsilon > 0, \exists \delta > 0, f(\mathcal{B}_\delta(2)) \subseteq \mathcal{B}_\varepsilon(3)$$

To unify this even further, we would like to get rid of the quantifiers on ε and δ, of course by using filters. Let us consider the filter $\mathcal{G} = \mathcal{N}_3$ which consists of neighbourhoods around 3. For every $S \in \mathcal{N}_3$, there exists an open ball $\mathcal{B}_\varepsilon(3) \subset S$ for some $\varepsilon > 0$, and we want $f(\mathcal{B}_\delta(2)) \subseteq S$ for some $\delta > 0$. Somewhat surprisingly, it can be proven that this is equivalent to $f(\mathcal{N}_2) \subseteq S$. There is a catch though: $f(\mathcal{N}_2)$ is not necessarily a filter on $\mathbb{R}$! A possible fix is to take the filter closure of $f(\mathcal{N}_2)$, but a simpler fix is to observe that $f^{-1}(\mathcal{N}_3)$ is a filter. This also gives us the first mathematical proof of this essay:

Definition 2. *Let X, Y be spaces, $f : X \to Y$ be a function, and $\mathcal{F} \subseteq \mathcal{P}(X)$ be a filter on Y. The* ***pushforward filter*** *of $\mathcal{F}$ along f, denoted $f_*\mathcal{F}$, is defined as $f_*\mathcal{F} := \{A \subseteq Y : f^{-1}(A) \in \mathcal{F}\} \subseteq \mathcal{P}(\mathcal{F})$.*

Lemma 1. *The pushforward filter $f_*\mathcal{F}$ is indeed a filter.*

Proof. Firstly, $Y \in f_*\mathcal{F}$ as $f^{-1}(Y) = X \in \mathcal{F}$. Next, let $U \in f_*\mathcal{F}$ and $V \supset U$. Then, $f^{-1}(V) \supset f^{-1}(U)$. Since $f^{-1}(U) \in \mathcal{F}$, by superset rule we have $f^{-1}(V) \in \mathcal{F}$, yielding $V \in f_*\mathcal{F}$. Finally, suppose that $U, V \in f_*\mathcal{F}$. Then, $f^{-1}(U \cap V) = f^{-1}(U) \cap f^{-1}(V)$, as $a \in f^{-1}(U \cap V) \iff f(a) \in U \cap V \iff f(a) \in U \wedge f(a) \in V \iff a \in f^{-1}(U) \wedge a \in f^{-1}(V)$. Hence, $f^{-1}(U \cap V) \in \mathcal{F}$ by intersection rule of $\mathcal{F}$, meaning $U \cap V \in f_*\mathcal{F}$. □

The discussion above then motivates the following definition:

Definition 3 (Limits along filters 1)**.** *Let X, Y be spaces, $f : X \to Y$ be a function, $y_0 \in Y$ be a point, and $\mathcal{F} \subseteq \mathcal{P}(X)$ be a filter on X. Then, we say that f **tends to** y_0 **along the filter** $\mathcal{F}$ iff $f_* \mathcal{F} \leq \mathcal{N}_{y_0}$.*

However, sometimes convergence to a point is not sufficient, such as when describing $\lim_{x \to x_0} f(x) = +\infty$. Hence, the correct generality of limits should also have any filter instead of $\mathcal{N}_{y_0}$, cumulating in the following general form of limits along filters:

Definition 4 (Limits along filters 2)**.** *Let X, Y be spaces, $f : X \to Y$ be a function, and $\mathcal{F}, \mathcal{G}$ be filters of X, Y respectively. Then, we say that f **tends to** $\mathcal{G}$ **along the filter** $\mathcal{F}$ iff $f_* \mathcal{F} \leq \mathcal{G}$.*

Of course, we should verify that these are equivalent to the traditional definition of limits. We will verify it for the first definition in the following case:

Theorem 1. *Let $f : \mathbb{R} \to \mathbb{R}$ be a function where $\lim_{x \to x_0} f(x)$ exists. Then, $\lim_{x \to x_0} f(x) = c$ if and only if $f_* \mathcal{N}_{x_0} \leq \mathcal{N}_c$.*

Proof. ($\implies$): Suppose that $\lim_{x \to x_0} f(x) = c$. Then, $\forall \varepsilon > 0, \exists \delta > 0, f(\mathcal{B}_\delta(x_0)) \subseteq \mathcal{B}_\varepsilon(c)$. Let $S \in \mathcal{N}_c$. Then, $\exists \varepsilon > 0, \mathcal{B}_\varepsilon(c) \subseteq S$. For such ε, there exists $\delta > 0$ such that $f(\mathcal{B}_\delta(x_0)) \subseteq \mathcal{B}_\varepsilon(c) \subseteq S$. Hence, $\mathcal{B}_\delta(x_0) \subseteq f^{-1}(S)$. Since $\mathcal{B}_\delta(x_0) \in \mathcal{N}_{x_0}$, by the superset property of filters, $f^{-1}(S) \in \mathcal{N}_{x_0}$, meaning that $S \in f_* \mathcal{N}_{x_0}$.

($\impliedby$): Suppose that $f_* \mathcal{N}_{x_0} \leq \mathcal{N}_c$. By definition, for every neighbourhood S of c, $f^{-1}(S) \in \mathcal{N}_{x_0}$. Fix $\varepsilon > 0$. Then, $\mathcal{B}_\varepsilon(c)$ is a neighbourhood of c, meaning that $f^{-1}(\mathcal{B}_\varepsilon(c)) \in \mathcal{N}_{x_0}$. In particular, it implies that $f^{-1}(\mathcal{B}_\varepsilon(c)$ is open and contains x_0, meaning there exists $\delta > 0$ such that $\mathcal{B}_\delta(x_0) \subseteq f^{-1}(\mathcal{B}_\varepsilon(c)$. Hence, $f(\mathcal{B}_\delta(x_0) \subseteq \mathcal{B}_\varepsilon(c)$, which is equivalent to $\lim_{x \to x_0} f(x) = c$. □

3.2 Mathlib 101: Proving a Limit

In this section, we show what the formalisation of $\lim_{x \to \infty} \frac{1}{x} = 0$ looks like in Lean (with Mathlib), and how a proof of it is constructed. To begin we have to import definitions from several files within Mathlib. We also have to `open` *namespaces* so that we only have to type `inv_tendsto_atTop` instead of `Filter.Tendsto.inv_tendsto_atTop`, for example.

```
import Mathlib.Order.Filter.Basic
import Mathlib.Topology.Instances.Real

open Filter Tendsto Topology
```

To write down the statement $\lim_{x \to \infty} \frac{1}{x} = 0$ in Lean, we first have to translate it into the language of filters. Writing $f : \mathbb{R} \to \mathbb{R}$ for $f(x) = \frac{1}{x}$, the filter definition of limits give $f_* \mathcal{N}_{+\infty} \leq \mathcal{N}_0$. In Lean, there is a slightly higher level definition called `Tendsto`, which expresses the statement that "f tends to $\mathcal{G}$ along $\mathcal{F}$". For our statement, we can write[8]

[8]There is a notation for `nhds`, but my LaTeXfonts don't support it, so we will stick with `nhds` instead.

```
example : Tendsto (fun x : ℝ ↦ 1 / x) atTop (nhds 0) := by
```

To break this line down:

- The starting `example :` is an indicator for the start of a new statement;
- `Tendsto` takes three arguments $f, \mathcal{F}, \mathcal{G}$ in that order, and express $f \xrightarrow{\mathcal{F}} \mathcal{G}$;
- `fun ... ↦ ...` is Lean 4's syntax for the lambda functions $\lambda x.fx$;
- `atTop` is the filter $\mathcal{N}_{+\infty}$; and
- `nhds 0` is the neighbourhood filter $\mathcal{N}_0$.

To prove this result, we can first prove that $\lim_{x\to\infty} x = +\infty$, from which we (informally) have $\frac{1}{+\infty} = 0$ as limits. We can indicate this "intermediate goal" as follows:

```
example : Tendsto (fun x : ℝ ↦ 1 / x) atTop (nhds 0) := by
  have : Tendsto (fun x : ℝ ↦ x) atTop atTop := by
    ...
  ...
```

Here we used the `have` keyword, which acts similar to the `example` keyword, except that it works within the proof. As one would imagine, the proof for the intermediate goal is not difficult, though explaining the details would derail us from the intended goal. Instead, here I present my proof for the entire statement:

```
example : Tendsto (fun x : ℝ ↦ 1 / x) atTop (nhds 0) := by
  have : Tendsto (fun x : ℝ ↦ x) atTop atTop := by
    intro _ a
    exact a
  simpa [one_div] using this.inv_tendsto_atTop
```

Here, `simpa` (which stands for "**simp**lification + **a**ssumption") is a powerful tactic within Mathlib that allows attempting to closing a goal via a powerful automatic theorem proving algorithm. This tactic is not in Lean 4 itself, meaning that without Mathlib, such a powerful tactic would not be available for use in proofs. This is another reason why Mathlib might be useful for proofs, even if they don't involve mathematics.

3.3 Asymptotics

Intuitively, asymptotics are similar in concept to limits, so it is not surprising that asymptotic notations can also be unified under filters. First we define the notion of *eventuality*:

Definition 5. *Let X be a space, $\mathcal{F}$ be a filter on X and $p : X \to$* **Prop** *be a predicate. We say that p holds true* ***eventually*** *along the filter $\mathcal{F}$, denoted $\forall^f x \in \mathcal{F}, p(x)$, if $\{x \in X : p(x)\} \in \mathcal{F}$.*

Example 5. *Take $X = \mathbb{R}$ and $\mathcal{F} = \mathcal{N}_{+\infty}$. Then, p holds true eventually along $\mathcal{N}_{+\infty}$ if and only if $\{x \in X : p(x)\} \in \mathcal{N}_{+\infty}$, which is when $\{x \in X : p(x)\}$ contains an interval $[A, +\infty)$ for some $A \in \mathbb{R}$. This matches the informal definition of eventually.*

With this, we can define the big-O notation and the little-O notation. These are more general than the typical definition, where $f(x) \in O(g(x))$ requires f and g to have the same codomain.

Definition 6 (Big-O and little-o notations)**.** *Let X be any space, Y, Z be two normed spaces, and $\mathcal{F}$ be a filter on X. Further let $f : X \to Y$ and $g : X \to Z$ be two functions. We say $f = O_{\mathcal{F}}(g)$, or simply $f = O(g)$ (usually when $\mathcal{F} = \mathcal{N}_{+\infty}$), if there exists $c \in \mathbb{R}$ such that $\forall^f x \in \mathcal{F}, \|f(x)\| \leq c\|g(x)\|$. We say $f = o_{\mathcal{F}}(g)$, or simply $f = o(g)$, if for all positive real c, $\forall^f x \in \mathcal{F}, \|f(x)\| \leq c\|g(x)\|$.*

It is worth mentioning that often times, one does not strictly write equations in the form $f(x) = O(g(x))$. For example, it is understood that $x^5 + 2x + 1 = x^5 + O(x)$ along the filter $\mathcal{N}_{+\infty}$. What this is actually saying is that $\exists f \in O_{\mathcal{F}}(x)$ such that $\forall^f x \in \mathcal{F}, x^5 + 2x + 1 = x^5 + f(x)$.

4 Behrend's 3-AP-free Construction

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

The modern interpretation of Behrend's result divides the construction into five parts:

1. Construct an appropriate 3-AP-free additive semiring $X = X_{q,r}$ parametrised by certain parameters q, r;
2. Construct an injection $\varphi : X \to \mathbb{Z}$, which restricts to a bijective function $\widetilde{\varphi} : X \to Z \subseteq \mathbb{Z}$;
3. Prove that $\varphi(w) + \varphi(x) = \varphi(y) + \varphi(z) \implies w + x = y + z$[9];
4. Conclude that $\varphi(X)$ is also 3-AP-free;
5. Optimise parameters such that $X_{q,r}$ is as large as possible relative to the range of $\varphi(X)$.

In Behrend's construction, he takes the lattice points on the sphere $X_{q,r} := \{(x_0, \dots, x_{n-1}) \in [1,q]^n : x_0^2 + \dots + x_{n-1}^2 = r^2\}$, with addition being the usual vector addition. The fact that this is 3-AP-free follows from a rather trivial geometric fact: a line intersects a sphere at most two points.

Next, Behrend defines the function $\varphi : X_{q,r} \to \mathbb{Z}$, given by $\varphi(x_0, x_1, \dots, x_{n-1}) := \sum_{i=0}^{n-1} x_i (2q)^i$. In other words, φ converts a vector into the corresponding base-$2q$ integer. To prove the required properties of φ, we will need the following lemma.

Lemma 2. *Let $\mathbf{x} = (x_0, x_1, \dots, x_{n-1}) \in (-2q, 2q)^n$. Then, $\varphi(\mathbf{x}) = 0$ if and only if $\mathbf{x} = 0$.*

Proof. The $\Longleftarrow$ direction is trivial. We prove the $\implies$ direction by induction, where the idea is to look at modulo powers of $2q$. For the base case, $0 = \varphi(\mathbf{x}) = x_0 + \sum_{i=1}^{n-1} x_i (2q)^i \equiv x_0 \pmod{2q}$. Since $|x_0| < 2q$, the equivalence $x_0 \equiv 0 \pmod{2q}$ implies $x_0 = 0$. Now, suppose that $x_0 = x_1 = \dots = x_k = 0$ for some $k \in [0, n-1)$. Then, $\varphi(\mathbf{x}) = \sum_{i=k+1}^{n-1} x_i (2q)^i = x_{k+1}(2q)^{k+1} + (2q)^{k+2} \sum_{i=0}^{(n-2)-(k+2)} x_{i+k+2} (2q)^i$. Dividing by $(2q)^{k+1}$ and taking modulo $2q$, we see that $x_{k+1} \equiv 0 \pmod{2q}$ again, meaning $x_{k+1} = 0$. Hence by induction, $\mathbf{x} = 0$. $\square$

With this, we can prove the requirements for φ to be a Freiman isomorphism of order 2.

- φ is injective on $X_{q,r}$. This is because if $f(x) = f(y)$ for some $x, y \in X_{q,r}$, then $f(x) - f(y) = f(x - y) = 0$. Since $x, y \in [1,q]^n$, we have $x - y \in [1-q, q-1]^n \subseteq (-2q, 2q)^n$. By Lemma 2, we have $x - y = 0$, i.e. $x = y$.
- $\varphi(w) + \varphi(x) = \varphi(y) + \varphi(z) \implies w + x = y + z$. To see this, suppose that $\varphi(w) + \varphi(x) - \varphi(y) - \varphi(z) = \varphi(w + x - y - z) = 0$. Since $w - y, x - z \in [1 - q, q - 1]^n$, we have $w + x - y - z \in (-2q, 2q)^n$. By Lemma 2, we have $w + x - y - z = 0$, i.e. $w + x = y + z$.

[9]If the implication is bidirectional, which Behrend's *is*, then we say φ is a **Freiman isomorphism** (of order 2) [20].

- $w + x = y + z \implies \varphi(w) + \varphi(x) = \varphi(y) + \varphi(z)$. This follows directly from linearity of φ.

From this, it follows that $\varphi(X_{q,r})$ is indeed 3-AP-free:

Lemma 3. *Let X be a 3-AP-free space, Y be an arbitrary space, and $\varphi : X \to Y$ a map. If $\varphi(w) + \varphi(x) = \varphi(y) + \varphi(z) \implies w + x = y + z$ for all $w, x, y, z \in X$, then Y is also 3-AP-free.*

Proof. Suppose not. Then, there exists distinct $\varphi(x), \varphi(y), \varphi(z) \in \varphi(X)$ such that $\varphi(x) + \varphi(z) = 2\varphi(y) = \varphi(y) + \varphi(y)$. Since φ is a Freiman isomorphism, we have $x + z = y + y$, i.e. $(x, y, z) \subseteq X^3$ forms a 3-AP, contradiction. □

By Lemma 3, its image $\varphi(X) \subseteq \mathbb{Z}$ is also a 3-AP-free set.

The final step of the construction is mostly independent from the steps above, and involves optimising the parameters q, r such that $\varphi(X_{q,r})$ is the densest. Define $A(q, r) := \varphi(X_{q,r}) \subseteq [1, (2q)^n] \subseteq [1, N]$, by taking $q := \lfloor N^{1/n}/2 \rfloor$. Since r is the radius of the ball, only values $r \in [n, nq^2]$ are meaningful, and points in $[1, q]^n$ can be sorted into buckets labelled by $r \in [n, nq^2]$ by their norm. By the Pigeonhole Principle, there exists $r \in \mathbb{Z}$ such that $|X_{q,r}| \geq \frac{q^n}{nq^2}$. Hence, we can compute the (asymptotic) density as

$$\frac{|A(q,r)|}{N} = \frac{|X_{q,r}|}{N} \geq \frac{q^{n-2}}{nN} \approx \frac{N^{(n-2)/n}}{nN \cdot 2^{n-2}} \geq N^{-2/n} \cdot \frac{2^{2-n}}{n}$$

The logarithm is approximately $-\frac{2}{n} \log N + (2 - n) \log 2$, which is maximised when $n = \sqrt{2 \log N / \log 2}$. Then, $|A(q, r)|/N \geq e^{-c\sqrt{\log N}}$ with $c = 2\sqrt{2 \log 2}$, attaining Behrend's bound (modulo handwaving constants).

5 Green's Corner-free Construction

As noted in Section 1.2, there is often a close connection between large 3-AP-free sets and large corner-free sets. Indeed, it is possible to modify Behrend's 3-AP-free set construction into a corner-free set construction. However, we will not show that construction here, as the result has been superseded by that of [15] and soon after [9].

Instead, as the title of the chapter suggests, we will show the construction by Green from [9], simplified by the author in this work by unifying with the framework from Section 4. Below, $\mathbb{Z}_q = \{0, 1, \dots, q-1\} \subseteq \mathbb{Z}$, not cosets $\mathbb{Z}/q\mathbb{Z}$.

1. Constructing an appropriate corner-free "two-dimensional" additive semiring $X = X_{r,q,d} \subseteq \mathbb{Z}_q^d \times \mathbb{Z}_q^d$ with special properties, parametrised by certain parameters r, q, d;
2. Use the naive embedding $\zeta : \mathbb{Z}_q^d \to \mathbb{Z}$ by parsing vectors as base-q digits of integers;
3. Prove that for $(\zeta(x), \zeta(y)), (\zeta(x'), \zeta(y)), (\zeta(x), \zeta(y')) \in \tilde{\zeta}(X), \zeta(x') + \zeta(y) = \zeta(x) + \zeta(y') \implies x' + y = x + y'$ (using the special properties of construction);
4. Conclude that $\tilde{\zeta}(X)$ is also cornerfree;
5. Optimise parameters.

The similarity in structure between this and Behrend's 3-AP-free construction is clear. To motivate the construction of X, it is most convenient to start with step 3, which is the main technical part of the proof. As before, we try to prove (3.) by proving the equality holds for the i-th digit: $x'_i + y_i = x_i + y'_i$. A technical difficulty we face is that the idea of taking $\zeta(x') + \zeta(y) = \zeta(x) + \zeta(y')$ modulo q only gives us $x'_0 + y_0 \equiv x_0 + y'_0 \pmod{q}$, and that is not sufficient to prove that they are equal over the integers. An easy fix is to note that $x_0 - y_0 \equiv x'_0 - y'_0 \pmod{q}$, so if we constraint elements $(x, y) \in X$ to satisfy $x - y \in [k, k+q)$ for some fixed constant k, then equality over integer would hold.

With this constraint, step 3 goes through via induction, and step 4 follows by taking contrapose. The only obstacle that remains is adding additional constraints to X to ensure that it is corner-free. Green's *magic ingredient* is the parallelogram law, which we recall below.

Theorem 5.1: Parallelogram Law

Let $(X, \|\cdot\|)$ be a normed space. Then, for all $x, y \in X$, we have

$$2\|x\|^2 + 2\|y\|^2 = \|x+y\|^2 + \|x-y\|^2$$

As a special case, if $\|x\| = \|x+y\| = \|x-y\|$, then we can conclude $\|y\| = 0$. With this in mind, we can describe the full construction by Green.

Step 1: Let $X := \{(x, y) \in \mathbb{Z}_q^d \times \mathbb{Z}_q^d : \|x - y\|_2^2 = r \wedge k \leq x_i + y_i < k + q\}$, where k, q, d, r are constants to be determined.

Lemma 4. *The set X is corner-free.*

Proof. Suppose that $(x, y), (x + d, y), (x, y + d) \in X$ be a corner in X. By construction, we have $\|x - y\|_2^2 = \|x - y + d\|_2^2 = \|x - y - d\|^2 = r$. By the discussion above, we see that $\|d\| = 0$. $\square$

Step 2: Define $\zeta : \mathbb{Z}_q^d \to \mathbb{Z}$, sending $\zeta(\mathbf{x}) = \sum_{i=0}^{d-1} x_i q^i$, which is injective. Also define $\mathcal{A} := \tilde{\zeta}(X)$, where $\tilde{\zeta} : \mathbb{Z}_q^d \times \mathbb{Z}_q^d \to \mathbb{Z} \times \mathbb{Z}$ is just coordinate-wise application of ζ.

Step 3: Suppose that $\zeta(x') + \zeta(y) = \zeta(x) + \zeta(y')$ for elements $(x, y), (x', y), (x, y') \in X$. By expanding the definition of ζ and taking modulo q, we have $x_0' + y_0 \equiv x_0 + y_0' \pmod{q}$, i.e. $x_0 - x_0' \equiv y_0 - y_0' \pmod{q}$. By construction, $x_0 - x_0'$ and $y_0 - y_0'$ both only take value in $[k, k + q)$, so they must equal as integers. Now suppose that $x_i - x_i' = y_i - y_i'$ for $i = 0, 1, \ldots, k - 1$. Then, $\zeta(x) = \sum_{i=k}^{n-1} x_i q^i$, and similar for x', y, y'. By dividing out q^k and taking modulo q again, by a similar argument we have $x_k - x_k' = y_k - y_k'$. By induction, we have $x - x' = y - y'$.

Step 4: If not, then the corner in $\mathcal{A}$ would imply $x' + y = x + y'$, where $(x, y), (x', y), (x, y') \in X$. This implies $x' - x = y' - y$, meaning X has a corner, contradiction.

Step 5: The computation below is pretty routine on paper, but is tricky to formalise in Lean. Skip ahead to Section 7 for more information. As a result, all computations are asymptotic and we will be slightly handwavy regarding lower order terms.

To begin, consider the discretised torus $\mathbb{T} = \{(x, y) \in \mathbb{Z}_q^d \times \mathbb{Z}_q^d : k \le x_i + y_i < k + q\}$. Since each coordinate is independent, we have $|\mathbb{T}| = |\{(x, y) \in \mathbb{Z}_q^2 : k \le x_i + y_i < k + q\}|^d$. We also know $x_i + y_i \in [0, 2q - 2]$, so the only reasonable values of k are $k \in [0, q - 1)$. Now, note that $\{(x, y) \in \mathbb{Z}_q^2 : x_i + y_i \in [k, k + q)\} = \mathbb{Z}_q^2 \ \{(x, y) \in \mathbb{Z}_q^2 : x_i + y_i \in [0, k) \vee x_i + y_i \in [k + q, 2q - 2]\}$, which are the lattices points of $\mathbb{Z}^2$ inside two triangle regions. By basic geometry, one can find that $|\mathbb{T}| = \left(q^2 - \frac{k^2}{2} - \frac{(q-k)^2}{2} + O(k)\right)^d = \left(\frac{q^2 + 2qk - 2k^2}{2}\right)^d$. In particular, this is maximised when $k = q/2$, where $|\mathbb{T}| = \left(\frac{3}{4}q^2 + O(q)\right)^d$.

Next, note that the sets $X_r = \{(x, y) \in \mathbb{T} : \|x - y\|_2^2 = r$ for $r = 0, \ldots, d(q - 1)^2$ partition the $\mathbb{T}$. By the Pigeonhole Principle, there exists r such that $|X_r| \ge \frac{1}{d(q-1)^2+1}|\mathbb{T}| \ge \frac{1}{dq^2}\left(\frac{3}{4}q^2 + O(q)\right)^d$.

Now, consider a fixed value d and let $N := q^d$ be the size of the bounding square of $\mathcal{A}$. Then, $\mathcal{A} \subseteq [N] \times [N]$ is a corner-free set, and $|\mathcal{A}|/N^2 \ge \frac{1}{dq^2}\left(\frac{3}{4} + O\left(\frac{1}{q}\right)\right)^d$. Choose $q = (c + o(1))^d$ where c is a constant to be determined. Then, $d = \log N / \log q = \log N / (d \log(c + o(1)))$. Rearranging, we get $d = \sqrt{\frac{\log N}{\log(c+o(1))}} = \sqrt{\frac{\log N}{\log c + o(1)}} = \sqrt{\frac{\log N}{\log c} + o(1)} = \left(\sqrt{\frac{1}{\log_2 c}} + o(1)\right)\sqrt{\log_2 N}$.

Substituting this back into q, we have

$$q = \left((c + o(1))^{\sqrt{\frac{1}{\log_2 c}} + o(1)} \right)^{\sqrt{\log_2 N}} = \left(c^{\sqrt{\frac{1}{\log_2 c}} + o(1)} \right)^{\sqrt{\log_2 N}}$$

Substituting these into the main term, we have

$$\begin{aligned}(dq^2)^{-1} &= \left[\left(\sqrt{\frac{1}{\log_2 c}} + o(1) \right) \left(c^{2\sqrt{\frac{1}{\log_2 c}} + o(1)} \right)^{\log_2 N} \sqrt{\log_2 N} \right]^{-1} \\ &= \frac{1}{\sqrt{\log_2 N}} \left(c^{\left(-2\sqrt{\frac{1}{\log_2 c}} + o(1)\right) \log_2 N} \right)\end{aligned}$$

And

$$\left(\frac{3}{4} + O\left(\frac{1}{q}\right) \right)^d = \left(\frac{3}{4} + o(1) \right)^d = \left(\frac{3}{4} \right)^{d + o(1)} = \left(\frac{3}{4} \right)^{\left(\sqrt{\frac{1}{\log_2 c}} + o(1) \right) \sqrt{\log_2 N}}$$

Hence,

$$\begin{aligned}|\mathcal{A}|/N^2 &\geq (dq^2)^{-1} \left(\frac{3}{4} + O\left(\frac{1}{q} \right) \right)^d \\ &= \frac{1}{\sqrt{\log_2 N}} e^{\left(\left(-2\log c + \log\left(\frac{3}{4}\right)\right) \sqrt{\frac{1}{\log_2 c}} + o(1) \right) \log_2 N} \\ &= e^{\left(\left(-2\log c + \log\left(\frac{3}{4}\right)\right) \sqrt{\frac{1}{\log_2 c}} + o(1) \right) \log_2 N}\end{aligned}$$

We want to choose c such that $\left(-2\log c + \log\left(\frac{3}{4}\right)\right) \sqrt{\frac{1}{\log_2 c}}$ is maximised. With some elementary calculus, one can show that the optimal choice is $c = \exp\left(-\frac{\log\left(\frac{3}{4}\right)}{2} \right) = \frac{2}{\sqrt{3}}$, achieving $|\mathcal{A}|/N^2 \geq e^{(-\kappa + o(1)) \log_2 N}$ for $\kappa = 2\log 2\sqrt{2\log_2\left(\frac{4}{3}\right)} \approx 0.5485\ldots$, finishing the construction analysis.

We will defer the discussion of how the proof above is formalised in Lean to Section 7. In short, the key observation is that most asymptotics terms are $o(1)$, which are functions f that tends to 0 as the parameters go to infinity. This turns the asymptotics analysis from an asymptotics probllem into a limit problem.

6 Implementation: Construction

In the following three sections, we will describe the implementation of the ABCs of the proof: Implementation: Asymptotics, Implementation: Bridging the Parts and Implementation: Construction (though perhaps CABs is more appropriate). We show the Lean statements of the major results proven in the project, and explain some choices made along the way.

Just to clarify the terminology, *construction* refers to **Step 1 - 4** of Section 5, *bridge* refers to computing $|\mathbb{T}|$ for $k = q/2$, and *asymptotics* refer to computing an asymptotic lower bound for $|X_r|$.

6.1 Corner-free sets

Firstly, a `CornerFree` predicate of type $\mathbf{Set}(\alpha \times \alpha) \to \mathbf{Prop}$ is defined for any commutative group α:

```
@[to_additive "..."]
def MulCornerFree {α : Type*} [DecidableEq α] [CommGroup α] (s : Set (α × α)) : Prop :=
  ∀ x y d, (x, y) ∈ s → (x * d, y) ∈ s → (x, y * d) ∈ s → d = 1
```

The `@[to_additive "..."]` is an *attribute* defined by Mathlib. As the name suggests, this attribute automatically generalises a multiplicative definition into an additive one. More specifically, it automatically generates another definition `AddCornerFree`, given by:

```
def AddCornerFree {α : Type*} [DecidableEq α] [AddCommGroup α] (s : Set (α × α)) : Prop :=
  ∀ x y d, (x, y) ∈ s → (x + d, y) ∈ s → (x, y + d) ∈ s → d = 0
```

This definition is a straightforward generalisation of the notation of corner-free sets in $\mathbb{Z}^2$ to other additive / multiplicative groups. It is possible to generalise the definition to monoids (groups without inverses), but that made certain proofs harder, so we avoided that direction.

We also have an alternative definition `MulCornerFree'`, given by:

```
@[to_additive "..."]
def MulCornerFree' (s : Set (α × α)) : Prop :=
  ∀ x y x' y' d, (x, y) ∈ s → (x', y) ∈ s → (x, y') ∈ s → x' = x * d → y' = y * d → d = 1
```

For finite sets s, this version is decidable; see Section 9 for a detailed discussion.

6.2 Constructing $X_{r,q,d}$

Next, we construct the sets $X = X_{r,q,d} \subseteq \mathbb{Z}_q^d \times \mathbb{Z}_q^d$ as described in Section 5. We approach this by `filter`ing the elements we "want" out of the finite set of all elements in $\mathbb{Z}_q^d \times \mathbb{Z}_q^d$. Lean has a type for $\mathbb{Z}/q\mathbb{Z}$, namely the type `Fin q`. A naïve approach would be to define X to be

```
  def X : Finset ((Fin d → Fin q) × (Fin d → Fin q)) := univ.filter ...
```

However, this is an *incorrect* definition! The reason is that both Lean's `Fin q` type and the mathematical $\mathbb{Z}/q\mathbb{Z}$ type are groups, and as a consequence, $(q-1)+1=0$ within the group. However, for our construction, we do *not* want this behaviour. Instead, we would like to treat $\mathbb{Z}_q$ as a subset of $\mathbb{Z}$, with the addition operator induced from $\mathbb{Z}$. The solution we decided to go with is to use the *subtype* $\{a : \mathbb{Z} // a \in [0,q)\}$. To represent T^d for a type T, one abandoned idea was to use the `Module` and related API from Mathlib, which is used in linear algebra to represent vector spaces. However, we found it hard to use especially without an explicit coordinate, so we instead went with `Fin d → T`. The definitions below are slightly different but convey the same idea:

```
abbrev Vec' (d q : ℕ) := {f : Fin d → ℤ // 0 ≤ f ∧ f + 1 ≤ q}
```

With this, we can finally define X by `filter`ing the elements in $X_{r,q,d}$ out from the set of all elements in $\mathbb{Z}_q^d \times \mathbb{Z}_q^d$, determined via the `IsInCons` proposition below:

```
def norm' (v : Vec d) : ℤ := ∑ i, (v i) ^ 2

def IsInCons (r : ℕ) (x y : Vec' d q) : Prop :=
    norm' (x.val - y.val) = r ∧ (q ≤ 2 * (x.val + y.val) ∧ 2 * (x.val + y.val) + 1 ≤ 3 * q)

def X (r : ℕ) : Finset (Vec' d q × Vec' d q) := univ.filter (IsInCons r).uncurry
```

There are several design choices we made. Firstly, we defined our own norm function `norm'` instead of using Lean's $\|\cdot\|$ function, since that takes value on $\mathbb{R}$ and is inconvenient for our purpose, where $\mathbf{v} \in \mathbb{Z}^d$. Secondly, we put `q ≤ 2 * (x.val + y.val)` instead of `q / 2 ≤ x.val + y.val` directly. The reason is that in Lean, integer division is defined to be the floor division (see Section 2.5), which would yield the incorrect inequality.

The final part is to cast the vectors into integers via the map ζ, defined as `VecToInt` below, and assert that $\tilde{\zeta}(X)$ is corner-free. To be clear, `sorry`, a Lean keyword, is a *placeholder* term that stands for a completed proof.

```
def VecToInt : Vec' d q ↪ ℤ where
  toFun := fun v ↦ ∑ i, v.val i * q ^ i.val
  inj' := sorry

def VecPairToInt : Vec' d q × Vec' d q ↪ ℤ × ℤ where
  toFun := fun ⟨v₁, v₂⟩ ↦ ⟨VecToInt v₁, VecToInt v₂⟩
  inj' := sorry
```

```
theorem part1 : AddCornerFree ((@X d q r).map VecPairToInt : Set (ℤ × ℤ)) := by
  sorry
```

7 Implementation: Asymptotics

As shown back in Section 5, simplifying the asymptotics expressions is tedious and involves rearranging massive terms and moving asymptotics terms in and out, and there are many lemmas that went into proving the asymptotics simplification in the paper. Here, we demonstrate the issues we faced and tricks we discovered during the formalisation process through two examples.

7.1 Approximating Powers

Theorem 7.1

Fix $c \geq 1$. Prove that $c^d + 1 = (c + o(1))^d$ as $d : \mathbb{N} \to \infty$.

To make the problem more interesting, I tried to prove it without resorting to the closed form for the $o(1)$ term. Firstly, let's state this within Lean. Following the discussion under Section 3.3, we formalise the theorem statement as follows:

```
lemma aux' (c : ℝ) (hc : 1 ≤ c) :
    ∃ f : ℕ → ℝ, f =o[atTop] (1 : ℕ → ℝ) ∧ (∀ᶠ d in atTop, c ^ d + 1 = (c + f d) ^ d) := by
```

The proof idea is to observe that the "correct" solution satisfies $0 \leq f(d) \leq \frac{1}{cd}$ by Bernoulli's inequality. However, there are also false solutions to $c^d + 1 = (c + f(d))^d$ that we do not want, such as when taking $\overline{f}(d) = -2c - f(d)$ for even d. In the end, I split the proof of the theorem into two independent steps:

1. There exists a function $f : \mathbb{N} \to \mathbb{R}$ where $f \geq 0$ and $\forall^f d \in \mathcal{N}_{+\infty}, c^d + 1 = (c + f(d))^d$.

2. For all functions $f : \mathbb{N} \to \mathbb{R}$ where $f \geq 0$ and $\forall^f d \in \mathcal{N}_{+\infty}, c^d + 1 = (c + f(d))^d$, we have $f = o(1)$.

For (1), the approach taken is to use continuity of $x \mapsto x^d$, along with the fact that $c^d \leq c^d + 1 \leq c^d + cd \leq (c+1)^d$. The statement as well as the proof is shown below. In particular, continuity of $x \mapsto x^d$ is proven by `continuousOn_pow` and is used in `intermediate_value_Icc`.

Note that instead of $\forall^f d \in \mathcal{N}_{+\infty}$, I explicitly put $\forall d \geq 1$. Clearly, if $\forall d \geq 1, p(d)$, then $[1, +\infty) \subseteq \{d : p(d)\}$, and so $\{d : p(d)\} \in \mathcal{N}_{+\infty}$. Also, I am pretty sure the `Classical.choose` call is not needed, but hey, if it works, it works.

```
lemma aux_part1 (c : ℝ) (hc : 1 ≤ c) :
    ∃ f : ℕ → ℝ, 0 ≤ f ∧ (∀ d, 1 ≤ d → c ^ d + 1 = (c + f d) ^ d) := by
  suffices ∀ d : ℕ, 1 ≤ d → ∃ f : ℝ, f ≥ 0 ∧ c ^ d + 1 = (c + f) ^ d by
    use fun d ↦ if hd : 1 ≤ d then Classical.choose (this d hd) else 0
    constructor
    · intro d
```

```lean
      simp only [Pi.zero_apply]
      by_cases hd : 1 ≤ d <;> simp only [hd, dite_true, dite_false]
      · exact (Classical.choose_spec (this d hd)).left
      · rfl
    · intro d hd
      simp only [hd, dite_true]
      exact (Classical.choose_spec (this d hd)).right
  intro d hd
  have h₁ : c ^ d ≤ c ^ d + 1 := (lt_add_one _).le
  have h₂ : c ^ d + 1 ≤ (c + 1) ^ d := by
    refine (add_le_add_left (Nat.one_le_cast.mpr hd) (c ^ d)).trans ?_
    convert pow_add_mul_self_le_pow_one_add d hc zero_le_one
    rw [mul_one]
  have := intermediate_value_Icc (lt_add_one c).le (continuousOn_pow d)
  obtain ⟨f, ⟨hf₁, hf₂⟩⟩ := this (mem_Icc.mpr ⟨h₁, h₂⟩)
  use f - c, ?_, ?_
  · rw [ge_iff_le, sub_nonneg]
    exact (mem_Icc.mp hf₁).left
  · beta_reduce at hf₂
    rw [← add_sub_assoc, add_sub_cancel_left, hf₂]
```

For (2), we note that $(c + f(d))^d = c^d + 1 \leq \left(c + \frac{1}{d}\right)^d$, so $f(d) \leq \frac{1}{d}$. This combined with $f \geq 0$ shows that $f = o(1)$. The proof is given below. Notice that the two (outer) `have` clauses are precisely what we stated in the previous sentence. This is a common theme in formalising proofs, where `have` statements can be used to "signpost" what the big picture of the proof looks like.

```lean
lemma aux_part2 (c : ℝ) (hc : 1 ≤ c) :
    ∀ f : ℕ → ℝ, (0 ≤ f ∧ ∀ d, 1 ≤ d → c ^ d + 1 = (c + f d) ^ d) → (f =o[atTop] (1 : ℕ → ℝ)) := by
  intro f ⟨hf₁, hf₂⟩
  have {d} (hd : 1 ≤ d) : (c + f d) ^ d ≤ (c + 1 / (d : ℝ)) ^ d := by
    rw [← hf₂ d hd]
    have : c + 1 / (d : ℝ) = c * (1 + 1 / (c * d : ℝ)) := by
      rw [mul_add, mul_one, div_mul_eq_div_div, mul_div, mul_one_div_cancel (by linarith)]
    convert pow_add_mul_self_le_pow_one_add d hc ?_
    · rw [mul_one_div_cancel (by positivity)]
    · norm_num

  have : ∀ᶠ d in atTop, ‖f d‖ ≤ ‖1 / (d : ℝ)‖ := by
    simp_rw [norm_eq_abs, eventually_atTop]
    refine ⟨1, fun d hd ↦ ?_⟩
    specialize this hd
    rw [abs_eq_self.mpr (hf₁ _), abs_eq_self.mpr (by norm_num)]
    contrapose! this
    gcongr

  refine IsBigO.trans_isLittleO (IsBigO.of_bound' this) ?_
  simp_rw [Pi.one_def, isLittleO_iff, eventually_atTop, norm_div, norm_one, mul_one,
    RCLike.norm_natCast]
  intro c hc
  use max 1 ⌈1 / c⌉₊
  intro b hb
  refine (one_div_le ?_ hc).mpr ?_
  · exact Nat.cast_pos.mpr (le_of_max_le_left hb)
  · exact Nat.ceil_le.mp (le_of_max_le_right hb)
```

7.2 Simplifying d

The second example we will look at is the following simplification steps from Section 5.

$$\sqrt{\frac{\log N}{\log(c+o(1))}} = \sqrt{\frac{\log N}{\log c + o(1)}} = \sqrt{\frac{\log N}{\log c} + o(1)} = \left(\sqrt{\frac{1}{\log_2 c}} + o(1)\right)\sqrt{\log_2 N}$$

Each equality step is almost trivial to mathematicians, but formalising them takes a surprisingly large amount of effort. For example, consider the first equality, which boils down to proving $\log(c+o(1)) = \log c + o(1)$ (where $1 < c$). Using the filters language, for every $f(x) \in o(1)$ we want to find $g(x) \in o(1)$ such that $\log(c + f(x)) =^f \log c + \log g(x)$, where $=^f$ is a short hand for eventually always equal. This is easy, as we simply take $g(x) = 1 + \frac{f(x)}{c}$. However, what follows is the difficult part: proving $\log(c+f(x)) =^f \log(c) + \log\left(1 + \frac{f(x)}{c}\right)$. In particular, the sum of log formula will fail when $f(x) \le -c$, so it requires some manuevering to prove the following lemma:

```
example {c : ℝ} (hc : 1 < c) (f : ℕ → ℝ) (hf : f =o[atTop] (fun _ ↦ 1 : ℕ → ℝ)) :
    (fun N : ℕ ↦ log (c + f N)) =ᶠ[atTop] (fun N : ℕ ↦ log c + log (1 + f N / c)) := by
  rw [isLittleO_one_iff, atTop_basis.tendsto_iff (nhds_basis_Ioo_pos _)] at hf
  obtain ⟨N, ⟨_, hN⟩⟩ := hf c (zero_lt_one.trans hc)
  rw [EventuallyEq, eventually_atTop]
  use N
  intro b hb
  rw [← log_mul, mul_add, mul_div_cancel₀, mul_one]
  · linarith
  · linarith
  · /- 1 + f b / c ≠ 0 -/
    have : -c < f b := (zero_sub c) ▸ (Set.mem_Ioo.mp (hN b hb)).left
    have : -c / c < f b / c := by gcongr
    have hc : c ≠ 0 := by linarith
    rw [neg_div, (div_eq_one_iff_eq hc).mpr rfl] at this
    linarith
```

The key to this proof (and the other two equalities) is to recall that $f = o(1) \iff \lim f = 0$. This simple observation turns the equalities from complicated *asymptotic* big-O notations to *limit* problems, which are significantly easier. More specifically, the first line of `rw` rewrites $f = o(1)$ into $\lim f = 0$, then converts it into the $\varepsilon - \delta$ definition. The final "bulletpoint" is trying to prove $1+f(b)/c \neq 0$ when b is large enough, and everything in between is extracting the correct hypotheses.

A slightly more interesting example is the second equality, which reduces to proving $\frac{1}{1+o(1)} = 1+o(1)$. Instead of converting every $o(1)$ into explicit functions, we can instead rearrange first to get $\frac{1}{1+o(1)} - 1 = o(1)$ as our goal state. This is equivalent to proving that for $f(x) \to 0$, $\frac{1}{1+f(x)} - 1 \to 0$, which is again trivial. Although Mathlib does not yet have a tactic for directly substituting limits, one can still prove the result pretty easily. The proof given below has excessive number of `have` statements just for clarity.

```
example {f : ℕ → ℝ} (hf : f =o[atTop] (fun _ ↦ 1 : ℕ → ℝ)) :
    ∃ g : ℕ → ℝ, g =o[atTop] (fun _ ↦ 1 : ℕ → ℝ)
      ∧ (fun N ↦ 1 / (1 + f N)) =ᶠ[atTop] (fun N ↦ 1 + g N) := by
  use fun N ↦ 1 / (1 + f N) - 1
  constructor
  · rw [isLittleO_one_iff] at hf ⊢
    have h₁ : Tendsto (fun N ↦ 1 + f N)         atTop (nhds (1 + 0))         := hf.const_add 1
    have h₂ : Tendsto (fun N ↦ (1 + f N)⁻¹)     atTop (nhds (1 + 0)⁻¹)       := h₁.inv₀ (by simp)
    have h₃ : Tendsto (fun N ↦ (1 + f N)⁻¹ - 1) atTop (nhds ((1 + 0)⁻¹ - 1)) := h₂.sub_const 1
    simpa using h₃
  · simp
```

8 Implementation: Bridging the Parts

Going into the project, the bridge was projected to be by far the easiest, but that estimation was incorrect. To demonstrate the issue, consider a triangle with $A = 1 + 2 + \cdots + \lfloor \frac{q}{2} \rfloor = \lfloor \frac{\lfloor \frac{q}{2} \rfloor (\lfloor \frac{q}{2} \rfloor + 1)}{2} \rfloor$ lattice points. We would like to obtain its highest order terms i.e. $A = \frac{q^2}{8} + O(q)$. As it turns out, this is quite hard. One method considered during the project was to use the rough bound $\frac{q-k}{k} \leq \lfloor \frac{q}{k} \rfloor$, arriving at the following:

```
example {b : ℤ} (hb : 2 ≤ b) :
    (((b - 2 : ℚ) / 2) * ((b - 2 : ℚ) / 2 - 1) - 2) / 2 ≤ ((b / 2) * (b / 2 - 1) / 2 : ℤ) := by
  sorry /- left unproven -/
```

However, this leads to an interesting question: how do we know whether the statement written down is correct?

9 Correctness via `eval`

After a formal proof has been formalised in a theorem prover, there is still the question of "does this proven statement *really* correspond to the claimed statement on paper?" This section aims to give several ways to boost such confidence.

9.1 Decidability typeclasses

In usual mathematics, most statements are either `True` or `False`. However, it is not always possible, even with a computer, to determine which one it is. To distinguish the statements that *can* be decided, Lean has a typeclass instance called `Decidable`. The definition of `Decidable` makes it clear what the typeclass represents:

```
class inductive Decidable (p : Prop) where
  /-- Prove that `p` is decidable by supplying a proof of `¬p` -/
  | isFalse (h : Not p) : Decidable p
  /-- Prove that `p` is decidable by supplying a proof of `p` -/
  | isTrue (h : p) : Decidable p
```

For example, consider the following proposition that checks whether n is a square.

```
import Mathlib.Data.Nat.Parity

def MyIsSquare (n : ℕ) : Prop := ∃ k : ℕ, n = k ^ 2
```

Without any extra lemmas, `MyIsSquare` is currently non-`Decidable`. To see this, imagine a computer trying to decide[10] whether `MyIsSquare 37` holds. The only way it can do so is by testing natural numbers k incrementally, and halting if $n = k^2$ is met. This never happens for $n = 37$, so the computer will run infinitely. Indeed, Lean does not accept this as a `Decidable` predicate:

```
instance : DecidablePred MyIsSquare := by infer_instance /- fails -/
```

However, we can make the lemma `Decidable` by adding a lemma, telling the computer that the candidates k can only go up to a finite bound:

```
lemma MyIsSquare_iff_bounded {n : ℕ} : MyIsSquare n ↔ ∃ k : ℕ, k ≤ n ∧ n = k ^ 2 := by sorry

instance : DecidablePred MyIsSquare := by
  intro n
  simp_rw [MyIsSquare_iff_bounded]
  /- ⊢ Decidable (∃ k ≤ n, n = k ^ 2) -/
  infer_instance
-/
```

[10]In other words, "evaluate" the **Prop** to either `True` or `False`.

The advantage of proving `Decidable` instances is that it allows the Lean kernel to act as the "computer" and verify computations for you. For example, we can now type

```
#eval MyIsSquare 36 /- true -/
#eval MyIsSquare 37 /- false -/
```

In this project, most objects (such as `MulCornerFree`) are carefully defined to be `Decidable`. Hence, one can verify the construction X is correct by typing

```
#eval (A (d := 2) (q := 4) 5)
/-
{(![0, 1], ![2, 2]),
 (![0, 2], ![2, 1]),
 (![1, 0], ![2, 2]),
 (![1, 2], ![2, 0]),
 (![2, 0], ![1, 2]),
 (![2, 1], ![0, 2]),
 (![2, 2], ![0, 1]),
 (![2, 2], ![1, 0])}
-/

#eval (A (d := 2) (q := 4) 5).map VecPairToInt
/-
{(3, 8), (6, 5), (1, 8), (7, 2), (2, 7), (5, 6), (8, 3), (8, 1)}
-/

#eval AddCornerFree ((A (d := 2) (q := 5) 5).map VecPairToInt : Set (ℤ × ℤ))
/-
true
-/
```

9.2 Random Testing via `slim_check`

Another option for verifying whether the formalised objects are correct would be the `slim_check` tactic from Mathlib. It is simple to use:

```
example {a b : ℤ} : (a / b - 1 : ℚ) ≤ (a / b : ℤ) := by
  slim_check /- Successful -/
```

10 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 near-weekly meetings. I would also like to thank Julian Berman, who is the creator of the `lean.nvim` plugin. Without the full power of my NeoVim setup, this project would have been drastically slowed down. Finally, I would like to thank the Mathlib community as a whole, and especially to Bhavik, David, Eric, Jireh, Karl, Kendall, Kevin, Mario, Thomas, Yaël and others (in alphabetical order), who helped with simplifying the proofs for various results and answered many of my questions.

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. 3).

[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] Jordan S. Ellenberg and Dion Gijswijt. *On large subsets of F_q^n with no three-term arithmetic progression*. 2016. arXiv: 1605.09223 [math.CO] (cit. on p. 4).

[6] P. Erdös. *Some remarks on the theory of graphs*. en. 1947. DOI: 10.1090/s0002-9904-1947-08785-1. URL: http://dx.doi.org/10.1090/S0002-9904-1947-08785-1 (cit. on p. 3).

[7] 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. 6).

[8] 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. 6).

[9] Ben Green. “Lower bounds for corner-free sets”. In: *New Zealand Journal of Mathematics* 51 (July 2021), pp. 1–2. ISSN: 1179-4984. DOI: 10.53733/86. URL: http://dx.doi.org/10.53733/86 (cit. on pp. 1, 4, 18).

[10] 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. 4).

[11] 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. 3).

[12] 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).

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

[14] James Lepowsky et al. "The Mathematical Work of the 1998 Fields Medalists". In: *NOTICES OF THE AMS* 46.1 (Jan. 1999) (cit. on p. 3).

[15] Nati Linial and Adi Shraibman. *Larger Corner-Free Sets from Better NOF Exactly-N Protocols*. 2021. DOI: 10.48550/ARXIV.2102.00421. URL: https://arxiv.org/abs/2102.00421 (cit. on pp. 4, 18).

[16] Gareth Ma. *Yet Another Talk about Prime Numbers*. 2024. URL: https://github.com/grhkm21/wimp-2024/blob/master/main.pdf (cit. on p. 3).

[17] Per Martin-Löf and Giovanni Sambin. *Intuitionistic type theory*. Vol. 9. Bibliopolis Naples, 1984 (cit. on p. 5).

[18] Patrick Massot. *Topology and filters*. 2020. URL: https://www.youtube.com/watch?v=hhOPRaR3tx0 (cit. on p. 11).

[19] Sam Mattheus and Jacques Verstraete. *The asymptotics of $r(4, t)$*. 2024. arXiv: 2306.04007 [math.CO] (cit. on p. 3).

[20] Melvyn B Nathanson and Imre z Ruzsa. "Additive number theory: inverse problems and the geometry of sumsets". In: *Bulletin of the London Mathematical Society* 31.148 (1999), p. 108 (cit. on p. 16).

[21] Egbert Rijke. "Introduction to homotopy type theory". In: (2022). arXiv: 2212.11082 [math.LO] (cit. on pp. 5, 7).

[22] 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. 3).

[23] 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. 3).

[24] 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. 3).

[25] 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. 3).