Formalizing the Solution to the Cap Set Problem

Sander R. Dahmen
Department of Mathematics, Vrije Universiteit Amsterdam, The Netherlands
s.r.dahmen@vu.nl

Johannes Hölzl
Department of Computer Science, Vrije Universiteit Amsterdam, The Netherlands
johannes.hoelzl@posteo.de

Robert Y. Lewis
Department of Computer Science, Vrije Universiteit Amsterdam, The Netherlands
r.y.lewis@vu.nl

Abstract

In 2016, Ellenberg and Gijswijt established a new upper bound on the size of subsets of $\mathbb{F}_q^n$ with no three-term arithmetic progression. This problem has received much mathematical attention, particularly in the case $q = 3$, where it is commonly known as the *cap set problem*. Ellenberg and Gijswijt's proof was published in the *Annals of Mathematics* and is noteworthy for its clever use of elementary methods. This paper describes a formalization of this proof in the Lean proof assistant, including both the general result in $\mathbb{F}_q^n$ and concrete values for the case $q = 3$. We faithfully follow the pen and paper argument to construct the bound. Our work shows that (some) modern mathematics is within the range of proof assistants.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Type theory; Mathematics of computing → Number-theoretic computations; Computing methodologies → Combinatorial algorithms; Software and its engineering → Formal methods

Keywords and phrases formal proof, combinatorics, cap set problem, Lean

Supplement Material Links to our formalization and supporting documents are hosted at the URL `https://lean-forward.github.io/e-g/`

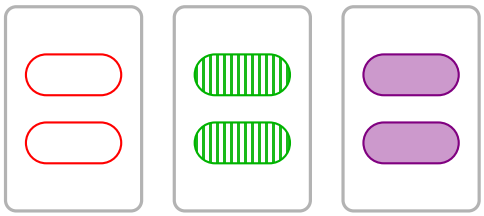
Funding *Sander R. Dahmen*: NWO Vidi grant No. 639.032.613, New Diophantine Directions.
Johannes Hölzl: ERC grant agreement No. 713999, Matryoshka.
Robert Y. Lewis: ERC grant agreement No. 713999, Matryoshka.

Acknowledgements We are grateful to the Lean `mathlib` maintainers and contributors on whose work this project is based. We thank Jeremy Avigad and Jasmin Blanchette for helpful comments on this paper, Dion Gijswijt for suggesting an improvement to our asymptotic bound argument, and Manuel Eberl for pointing us to related work.

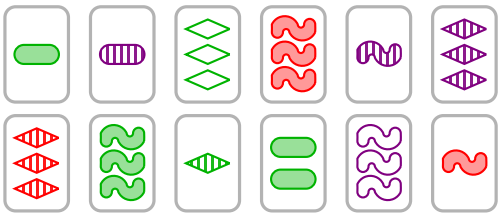
[This is a preprint of a paper that will appear in the proceedings of ITP 2019.]

1 Introduction

As proof assistants improve and their libraries grow, these tools are increasingly used to formalize results at the cutting edge of computer science. At some prestigious conferences such as *Principles of Programming Languages* (POPL), it is common for papers establishing new metatheoretical results about programming languages to be accompanied by formal proofs. In the field of mathematics, however, the picture looks very different. Even though early proof assistants were developed by and for mathematicians [10, 27], there are still very few mathematicians who use these tools in their work. With a small number of noteworthy exceptions (e.g. Gouëzel and Schur [21] and Hales, et al. [23]), no current work in pure mathematics work gets formalized; most of the results formalized in papers at *Interactive*



(a) A valid triple. Each card has the same shape and the same number of shapes. Each card has a different color and a different fill.



(b) A collection of twelve cards that contains no valid triple.

Figure 1 The cap set problem can be interpreted in the game Set, where it concerns an upper bound on the size of a collection of cards that contains no valid triple.

Theorem Proving (ITP) or *Certified Programs and Proofs* (CPP) have already made it into undergraduate or introductory graduate textbooks.

Researchers often point to the *depth* of mathematical theory to explain this difference. While programming language formalizations can be sprawling and difficult, they rarely depend on large background libraries, and often involve repetitive arguments that are amenable to automation. In comparison, mathematics builds upwards on centuries of earlier work, and one cannot formalize modern results without first formalizing the necessary foundation. The few existing formal developments of cutting-edge mathematics tend to focus on results that are difficult to verify by hand—justifying the effort needed to develop libraries—or fall in subfields of mathematics where the background theory is less intimidating.

The combinatorial proof described in this paper belongs in the latter category. Let G be an abelian group. A three-term *arithmetic progression* of elements of G is a sequence $a, a+g, a+g+g$ where $a, g \in G$ and g is nonzero. Let $r_3(G)$ denote the cardinality of a largest subset of G containing no three-term arithmetic progression. We will focus on the group $(\mathbb{Z}/3\mathbb{Z})^n = \{(a_1, \ldots, a_n) \mid a_i \in \{0, 1, 2\}\}$, where vector addition is pointwise and modulo 3; a subset of this group with no three-term arithmetic progression is known as a *cap set*. The *cap set problem* asks whether there is a constant $c < 3$ such that $r_3((\mathbb{Z}/3\mathbb{Z})^n)$ grows in n no faster than c^n.

Readers familiar with the card game Set (Figure 1) may understand the cap set problem in different terms. A card in Set has four features, where each feature has three possible values. (A card has one, two, or three copies of a shape; the shape is an oval, a diamond, or a squiggle; the shape is solid, striped, or empty; the shape is purple, red, or green.) A triple of cards is said to be *valid* if, for each feature, either all three cards have the same value or all three cards have different values. During game play, players search a collection of cards for valid triples. The number $r_3((\mathbb{Z}/3\mathbb{Z})^4)$ is the maximum size of a collection of distinct cards in which no valid triples can be found, and the cap set problem concerns the growth rate of this value as the number of features is increased.

The cap set problem is surprisingly difficult to analyze and has attracted attention over the past decades from leading combinatorialists. Croot, Lev, and Pach [9] solved a closely related problem in 2016. Building on their work, Ellenberg and Gijswijt soon showed that $r_3((\mathbb{Z}/3\mathbb{Z})^n)$ is $o(2.756^n)$, a major breakthrough. In fact, they proved a more general result about finite fields. Their 2017 paper in the *Annals of Mathematics* [18] is noteworthy in that the core of the proof does not use any complicated theoretical machinery. Rather, it relies on a clever shift of context, casting the problem in terms of polynomials of bounded degree. While their final proof of the asymptotics does make use of relatively high-powered methods,

Tao [30] and Zeilberger [33] indicate how these calculations can be made elementary. We also note that Tao [30] reformulates Ellenberg and Gijswijt's proof in a more symmetric way, using what is now called "slice rank." Although this is arguably a more natural way to express things, the underlying arguments are essentially the same.

This paper describes a formalization of Ellenberg and Gijswijt's argument, carried out in the Lean proof assistant. While unavoidably more verbose, our computation of an upper bound for $r_3((\mathbb{Z}/p\mathbb{Z})^n)$ faithfully follows Ellenberg and Gijswijt's proof. To verify the asymptotics, we work out a new elementary argument (inspired by Zeilberger's approach and a suggestion by Gijswijt). Ellenberg and Gijswijt use a technique known as the *polynomial method* to translate the problem to one about vector spaces of polynomials. We expect that our library contributions will be useful for proving other results that follow this approach.

A recent project begun at the Vrije Universiteit Amsterdam aims to bring together traditional mathematicians, formalizers, and tool developers to incorporate modern number theory into proof assistants.[1] The current paper shows that the goals of this project are within reach: we have formalized a paper published in the *Annals* less than two years ago.

The more general components of our formalization have been incorporated into the Lean mathematics library `mathlib`, which is available on GitHub.[2] The remainder of the formalization can be found with the supplementary material linked at the beginning of this paper. The code blocks presented below should be read as schematic, not literal. We sometimes change names, remove namespaces, omit universe levels, and swap implicit and explicit arguments for the sake of formatting and presentation.

2 Mathematical Background

Ellenberg and Gijswijt study a generalization of the cap set problem that holds for arbitrary finite fields (including $\mathbb{Z}/p\mathbb{Z}$ for any prime p). For the rest of this discussion, we fix a positive integer n and prime power q, and let $\mathbb{F}_q$ denote a finite field with cardinality q.

For $d \in \mathbb{R}$ with $0 \le d \le (q-1)n$, consider all n-variable monomials whose degree in each variable is at most $q-1$ and whose total degree is at most d, i.e.

$$M_n^d := \left\{ \prod_{i=1}^{n} x_i^{a_i} \in \mathbb{F}_q[x_1, \ldots, x_n] \;\middle|\; 0 \le a_i \le q-1 \text{ and } \sum_{i=1}^{n} a_i \le d \right\}.$$

Let $m_d := |M_n^d|$. Ellenberg and Gijswijt [18, Theorem 4] establish an upper bound for the size of generalized cap sets in terms of $m_{(q-1)n/3}$.

▶ **Theorem 1** (Ellenberg–Gijswijt). *Let $\alpha, \beta, \gamma \in \mathbb{F}_q$ such that $\alpha + \beta + \gamma = 0$ and $\gamma \neq 0$. Let A be a subset of $\mathbb{F}_q^n$ such that the equation $\alpha a_1 + \beta a_2 + \gamma a_3 = 0$ has no solutions with $a_1, a_2, a_3 \in A$ apart from those with $a_1 = a_2 = a_3$. Then $|A| \le 3m_{(q-1)n/3}$.*

If $(\alpha, \beta, \gamma) = (1, -2, 1)$, then the equation $\alpha a_1 + \beta a_2 + \gamma a_3 = 0$ is equivalent to $a_2 - a_1 = a_3 - a_2$; any solution to this, other than $a_1 = a_2 = a_3$, corresponds to a three term arithmetic progression.

To answer the cap set problem, it remains to determine good asymptotics for $m_{(q-1)n/3}$ as n tends to ∞.

[1] `https://lean-forward.github.io/`
[2] `https://github.com/leanprover-community/mathlib/`

▶ **Theorem 2.** *For every q there exists $c \in \mathbb{R}$ with $0 < c < q$ such that $m_{(q-1)n/3} = \mathcal{O}(c^n)$ as $n \to \infty$.*

Thus, with notation from Theorem 1, $|A| = \mathcal{O}(c^n)$ for some $0 < c < q$. For particular values of q we can write down explicit values of c. In the case of the original cap set problem, where $q = 3$ (and $\alpha = \beta = \gamma = 1$, also noting that $-2 = 1$ in $\mathbb{Z}/3\mathbb{Z}$), the proof method yields the following theorem; the exact value c already appears in Zeilberger [33].

▶ **Theorem 3.** *Let $c := \frac{3}{8}\sqrt[3]{207 + 33\sqrt{33}} < 2.755105$. Then $r_3\left((\mathbb{Z}/3\mathbb{Z})^n\right) \leq 3c^n$, and thus $r_3\left((\mathbb{Z}/3\mathbb{Z})^n\right) = o(2.755105^n)$ (as $n \to \infty$).*

The proof of Theorem 1 follows the *polynomial method*. (For a general introduction to the polynomial method, see e.g. Guth [22] or Tao [29].) Broadly speaking, this approach aims to analyze finite combinatorial objects by describing them through a system or space of polynomials. Techniques from algebraic geometry, or sometimes algebraic topology or simply linear algebra, can then be employed to study these polynomials; the results should translate back to properties of the original combinatorial objects of interest.

The polynomial method has been employed over the last decade to solve a large variety of open problems in arithmetic combinatorics and number theory. However, the scope and limitations of the method are still not well understood. In particular, its applicability to the cap set problem was unexpected, at least until the breakthrough of Croot, Lev, and Pach [9]. The main approach to the cap set problem for the previous half century was through Fourier theory methods.

We sketch here an overview of the proof of Theorem 1; more details can be found in Section 4. Let α, β, γ, and A be as stated in the theorem. We introduce the $\mathbb{F}_q$-vector space spanned by M_n^d, i.e.

$$S_n^d := \left\{ \sum_{m \in M_n^d} c_m m \;\middle|\; c_m \in \mathbb{F}_q \right\}.$$

Consider the $\mathbb{F}_q$-vector subspace V of S_n^d consisting of all polynomials $p \in S_n^d$ that vanish on the complement of $-\gamma A = \{-\gamma a \mid a \in A\}$ inside $\mathbb{F}_q^n$, i.e.

$$V := \{p \in S_n^d \mid \forall a \in \mathbb{F}_q^n \setminus (-\gamma A),\ p(a) = 0\}.$$

This is the setup of the polynomial method, the idea being that this space of polynomials V contains valuable information on $|-\gamma A| = |A|$ via $\dim(V)$. The strategy is to get good lower and upper bounds on $\dim(V)$. Namely, it holds that

$$\dim(V) \geq m_d - q^n + |A| \qquad \text{and} \qquad \dim(V) \leq 2m_{d/2}. \tag{1}$$

The lower bound is reasonably straightforward: it follows from rank-nullity and the remark that $|\mathbb{F}_q^n \setminus (-\gamma A)| = q^n - |A|$. The upper bound is more involved; the key to it is the following.

▶ **Proposition 4** (Proposition 2 from [18])**.** *Let $A \subseteq \mathbb{F}_q^n$ and $\alpha, \beta, \gamma \in \mathbb{F}_q$ with $\alpha + \beta + \gamma = 0$. Let $P \in S_n^d$ such that for all $a, b \in A$ with $a \neq b$ we have $P(\alpha a + \beta b) = 0$. Then*

$$|\{a \in A \mid P(-\gamma a) \neq 0\}| \leq 2m_{d/2}.$$

In addition, an elementary combinatorial argument gives us

$$q^n - m_d \leq m_{(q-1)n-d}. \tag{2}$$

Combining (1) and (2) and taking $d = 2(q-1)n/3$ gives us Theorem 1, i.e.

$$|A| \leq 3m_{(q-1)n/3}.$$

To establish the asymptotic behavior of this bound, Ellenberg and Gijswijt apply Cramér's theorem on large deviations. Tao [30] describes a more elementary approach via Stirling's approximation for the factorial function. Zeilberger [33] gives another even more elementary approach using recurrence sequences. Inspired by Zeilberger's paper, we worked out yet another approach, which lends itself very well to formalization in Lean. This was the initial approach we followed through; it is briefly described in Appendix A. Finally, thanks to a remark from Dion Gijswijt on our preprint, we arrive at a further significant simplification of the asymptotics proof, which we present below.

Our starting point is the combinatorial observation

$$m_d = \sum_{j=0}^{\lfloor d \rfloor} c_j^{(n)} \tag{3}$$

where $c_j^{(n)}$ is the coefficient of x^j in the polynomial $\left(1 + x + \ldots x^{q-1}\right)^n$. Let $r \in \mathbb{R}$ with $0 < r < 1$ and write $e := \lfloor (q-1)n/3 \rfloor$. Note that the $c_j^{(n)}$ are nonnegative and that $r^e \leq r^j$ for integers $0 \leq j \leq e$. Now

$$m_{(q-1)n/3} \cdot r^e = \sum_{j=0}^{e} c_j^{(n)} r^e \leq \sum_{j=0}^{e} c_j^{(n)} r^j \leq \sum_{j=0}^{(q-1)n} c_j^{(n)} r^j = \left(1 + r + \ldots + r^{q-1}\right)^n.$$

Dividing by $r^e \geq (r^{(q-1)/3})^n$ and defining

$$C_{r,q} := \frac{1 + r + \ldots + r^{q-1}}{r^{(q-1)/3}} = \frac{1 - r^q}{(1-r)r^{(q-1)/3}} \tag{4}$$

we arrive at our main asymptotics estimate

$$m_{(q-1)n/3} \leq C_{r,q}^n.$$

Elementary analysis gives us that for every $q > 1$ there exists some $0 < r < 1$ such that $C_{r,q} < q$, yielding Theorem 2. Specializing at $q = 3$ and $r = (\sqrt{33} - 1)/8$ gives the precise version of the cap set problem in Theorem 3. Similarly, minimizing $C_{r,q}$ for other values of q immediately leads to other growth rates, including those given by Zeilberger [33].

3 Lean and its Mathematics Library

The Lean proof assistant, developed principally by Leonardo de Moura, was first released in 2014 [11]. Lean implements a version of the calculus of inductive constructions (CIC) [8] with support for quotient types and classical reasoning. Since the release of Lean 3 in 2017 [17], there has been a concerted effort to develop `mathlib`, a comprehensive library for use in mathematics and computer science [4]. This library is built on the latest release of Lean, version 3.4.2. Some of the text in this section is adapted from a paper by the third author [26], which describes another formalization based on `mathlib`.

The datatypes available in `mathlib` include the concrete types commonly found in mathematics, among them $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{C}$; finite sets and multisets over a base type; univariate and multivariate polynomials; and embeddings and isomorphisms between types.

```
class semigroup (α : Type) extends has_mul α :=
(mul_assoc : ∀ a b c : α, a * b * c = a * (b * c))

class monoid (α : Type) extends semigroup α, has_one α :=
(one_mul : ∀ a : α, 1 * a = a) (mul_one : ∀ a : α, a * 1 = a)

class group (α : Type) extends monoid α, has_inv α :=
(mul_left_inv : ∀ a : α, a⁻¹ * a = 1)

lemma one_inv (α : Type) [group α] : 1⁻¹ = (1 : α) :=
inv_eq_of_mul_eq_one (one_mul 1)
```

Figure 2 A sample of the bottom of the algebraic hierarchy. The lemma `one_inv` can be applied to any α for which Lean can infer an instance of `group` α.

The algebraic hierarchy of mathlib is designed using *type classes*, which endow a base type with extra structure in the forms of operations, properties, and notation [28, 32]. Lean's type class resolution mechanism automatically manages inheritance between type classes (Figure 2). If a type class `T'` extends (directly or by transitivity) a type class `T`, any theorem proved over `T` will apply to any type that instantiates `T'`. The algebraic hierarchy begins with semigroups and monoids and extends to rich structures including fields, Noetherian rings, and principal ideal domains. Van Doorn, von Raumer, and Buchholz [31] also explain how type classes are used to define an algebraic hierarchy in Lean.

The project described in this paper makes heavy use of the linear algebra and multivariate polynomial developments in `mathlib`. As with the algebraic hierarchy, these developments are built around type classes. The linear algebra theory in particular is modeled after the one found in Isabelle/HOL, reworked to use bundled submodules and bundled linear functions.

The fundamental type class in linear algebra is `module` α β, which assumes a ring structure on α and an abelian group structure on β, and endows β with a well-behaved scalar multiplication operation from α. When α is a field, this extends to the type class `vector_space` α β. Many of the typical theorems and constructions from linear algebra are defined over this type class, including the existence of bases, the rank-nullity theorem for linear maps, and the matrix representation of maps between finite-dimensional spaces. General instances establish that a family of vector spaces over an index type forms a vector space itself, and that a field α instantiates `vector_space` α α; combined, these allow us to consider the type of n-tuples of field elements, `fin n` $\to$ α, as a vector space over α.

Polynomials are another important instance of a vector space. Given a type σ used to index variables, we identify a monomial with a finitely supported function from σ to $\mathbb{N}$. A multivariate polynomial is a finitely supported function mapping monomials into a coefficient ring α. We use the infix notation $\to_0$ for functions of finite support.

```
def mv_polynomial (σ α : Type) [comm_semiring α] := (σ →₀ ℕ) →₀ α
```

When α is a field, this type forms a vector space over α. Important operations on polynomials include `eval`, which evaluates the polynomial in α given an assignment $\sigma \to \alpha$, and `total_degree`, which computes the maximum degree over all monomials in a polynomial.

Many contributions were made to `mathlib` in the course of this project. In addition to extending the linear algebra, polynomial, and finitely supported function theories, we added various results about big operators and series, finite sets and multisets, and orders of elements in finite groups (to show, for example, that $a^q = a$ for $a \in \mathbb{F}_q$).

Another type class that plays an important role in our formalization is `fintype` α, which

provides functions for listing and counting the elements of α. The standard finite types instantiate this class, including the type `fin n` of natural numbers less than `n`. When α and β instantiate `fintype`, so does the function type α → β.

The `mathlib` library is designed with a focus on classical logic. Type-valued declarations are defined computably when possible, but classical logic is used freely in propositions. Our formalization is similarly classical.

Readers unused to Lean syntax should note that explicit arguments to declarations are enclosed in parentheses `()`, implicit arguments are enclosed in curly brackets `{}`, and type class arguments are enclosed in square brackets `[]`. Only explicit arguments are given by the user when applying a declaration. Implicit arguments are inferred from later arguments and the expected type, and type class arguments are inferred by type class resolution.

Another important feature of Lean syntax is its projection notation. As an example, let terms `F : polynomial α` and `a : α` be given. The operator

```
polynomial.eval : α → polynomial α → α
```

evaluates a polynomial at an argument. Because the head symbol of the type of `F` is `polynomial`, matching the namespace of `eval`, we can abbreviate `polynomial.eval a F` with the more concise `F.eval a`. This notation can be nested:

```
polynomial.eval a (polynomial.derivative F)
```

shortens to `F.derivative.eval a`.

4 The Cap Set Bound

As described in Section 2, Ellenberg and Gijswijt's solution to the cap set problem [18] proceeds in two parts. The first part establishes an upper bound on the size of a cap set in terms of the dimension of a vector space of polynomials; the second part shows the asymptotic behavior of this bound. Our formalization is similarly divided. This section describes the formal construction of the bound, and Section 5 explains the verification of the asymptotics. Our construction of the bound closely follows Ellenberg and Gijswijt's paper.

At the outset of our efforts, the first author produced a detailed paper proof[3] of the result, drawing from Ellenberg and Gijswijt and from Zeilberger [33] and adapting the asymptotics part significantly. The most recent approach to this part was added after initially submitting this paper, and was subsequently also formalized. The theorem names in the following sections match the corresponding statements in the paper proof.

The theorems here hold over an arbitrary finite field. We will take a fixed parameter `α : Type` instantiating the type classes `[fintype α]` and `[discrete_field α]`, and use `q` to abbreviate the cardinality `fintype.card α`. In this section, we also fix a parameter `n : ℕ`, representing the length of the tuples in the set whose cardinality we will bound.

The goal of this section, then, is to define a function `m` and prove the following theorem, which corresponds to the informal statement of Theorem 1 above:

```
theorem theorem_12_1 {α : Type} [discrete_field α] [fintype α]
  (n : ℕ) {a b c : α} (hc : c ≠ 0) (habc : a + b + c = 0)
  (hn : n > 0) {A : finset (fin n → α)}
  (ha : ∀ x y z ∈ A, a • x + b • y + c • z = 0 → x = y ∧ x = z) :
  A.card ≤ 3 * m α n (1 / 3 * ((card α - 1) * n))
```

[3] This writeup is available at `https://lean-forward.github.io/e-g/`

Ellenberg and Gijswijt's key insight is to translate the question to one concerning vector spaces of multivariate polynomials. After setting up this translation, this bound will follow from a sequence of intermediate lemmas.

4.1 Setting Up the Polynomial Method

The type `mv_polynomial (fin n) α` forms a vector space, by results established in `mathlib` (Section 3). We will focus our attention on a particular subspace. We define `M` to be the set of monomials in `n` variables where the exponent of each variable is strictly less than `q`. This set is linearly independent with respect to `α`.

```
def M : finset (mv_polynomial (fin n) α) :=
(finset.univ.image
  (λ f : fin n →₀ fin q, f.map_range fin.val rfl)).image
    (λ d : fin n →₀ ℕ, monomial d (1:α))
```

For `d : ℚ`, we make the following definitions:

- `M'` is the subset of `M` whose elements have total degree at most `d`.
- `S'` is the span of `M'`; this is a subspace of `mv_polynomial (fin n) α`.
- `m` is the dimension of `S'`.

Since `M'` is linearly independent, it follows that the cardinality of `M'` is equal to `m`.

```
def M' (d : ℚ) : finset (mv_polynomial (fin n) α) :=
M.filter (λ m, d ≥ mv_polynomial.total_degree m)

def S' (d : ℚ) : submodule α (mv_polynomial (fin n) α) :=
submodule.span α ((M' d) : set (mv_polynomial (fin n) α))

def m (d : ℚ) : ℕ := (vector_space.dim α (S' d)).to_nat

lemma M'_card (d : ℚ) : (M' d).card = m d
```

Much of the following argument will be carried out in a subspace of `S'`. We first describe this subspace generically. Given a subspace of polynomials `T` and a set of vectors `A`, we define `zero_set T A` to be the set of polynomials in `T` that evaluate to 0 at all elements of `A`. By basic properties of polynomial evaluation, this set is a subspace of `T`.

```
parameters (T : subspace α (mv_polynomial (fin n) α))
           (A : finset (fin n → α))

def zero_set : set (mv_polynomial (fin n) α) :=
{p ∈ T.carrier | ∀ a ∈ A, mv_polynomial.eval a p = 0}

def zero_set_subspace : subspace α (mv_polynomial (fin n) α) :=
{ carrier := zero_set,
  zero := ⟨submodule.zero, by simp⟩,
  add := λ _ _ hx hy,
   ⟨submodule.add hx.1 hy.1, λ _ hp, by simp [hx.2 hp, hy.2 hp]⟩,
  smul := λ _ _ hp,
   ⟨submodule.smul hp.1, λ _ hx, by simp [hp.2 hx]⟩ }
```

Our target theorem takes as parameters `a b c : α` and `A : finset (fin n → α)` satisfying certain properties, in particular that `c ≠ 0`. Let these terms be given. We define `neg_cA` to be the image of `A` under multiplication by `-c`, and `V` to be the zero set of `S'` with respect to the complement of `neg_cA`.

```
def neg_cA : finset (fin n → α) := A.image (λ z, (-c) · z)

def V : subspace α (S' d) :=
zero_set_subspace (S' d) (finset.univ \ neg_cA)

def V_dim : ℕ := (vector_space.dim α V).to_nat
```

Our goal—an upper bound on the cardinality of `A`, in terms of `m`—will follow from a number of lemmas controlling the dimension of `V`.

4.2 Lemma 1: Bounding the Dimension from Below

The first lemma establishes a lower bound for the dimension of `V` in terms of `m`, `q`, and `A.card`. We prove this via a generic result that holds for every `zero_set_subspace` of a finite-dimensional space.

```
theorem lemma_9_2 (T : subspace α (mv_polynomial (fin n) α))
  (A : finset (fin n → α)) :
  (vector_space.dim α zero_set_subspace).to_nat + A.card ≥
    (vector_space.dim α T).to_nat
```

This lemma is an exercise in linear algebra. It follows quickly from the rank-nullity theorem. The formal proof takes little work with our additions to the linear algebra theory in `mathlib`.

We now set a parameter `d : ℚ` which will remain fixed until the end of this section. After specializing `lemma_9_2` and performing a cardinality computation, we obtain the following:

```
theorem lemma_12_2 : q^n + V_dim ≥ m d + A.card
```

The `mathlib` definition of `vector_space.dim` takes values in the type `cardinal`, since vector spaces are not restricted to finite dimensions. (Perhaps confusingly, `finset.card` and `fintype.card` take values in ℕ.) In our setting, the vector space `S'`, and hence its subspace `V`, is finite dimensional. The cast `cardinal.to_nat` is thus well behaved.

4.3 Lemmas 2 and 3: Bounding the Dimension from Above

Next we establish an upper bound for the dimension of `V`. It is conceptually clearest to achieve this via two lemmas, one which bounds the dimension above by an intermediate value, and one which bounds this value above by `m`.

To prove the first lemma, we define the support set of a polynomial to be the set of points on which it does not evaluate to 0:

```
def sup (p : mv_polynomial (fin n) α) : finset (fin n → α) :=
finset.univ.filter (λ x, p.eval x ≠ 0)
```

A general argument about finite sets shows that there is some polynomial in `V` with maximal support.

```
lemma exi_max_sup :
  ∃ P ∈ V, ∀ P' ∈ V, sup P ⊆ sup P' → sup P = sup P'
```

We define `P` to be this polynomial and `P_sup` to be `sup P`, allowing us to state the following:

```
theorem lemma_12_3 : P_sup.card ≥ V_dim
```

The proof of this lemma involves some algebraic manipulation of the evaluation function `mv_polynomial.eval`. It invokes yet another polynomial subspace, the zero set of `V` with respect to `P_sup`.

In order to relate `P_sup` to other more interesting constants, we must prove a second lemma:

```
theorem lemma_12_4 : P_sup.card ≤ 2 * m (d/2)
```

This lemma is a special case of Proposition 4 (Section 2), stated here in Lean:

```
theorem proposition_11_1 {p :  mv_polynomial (fin n) α}
  (A : finset (fin n → α)) : p ∈ S' n d →
  (∀ (x : fin n → α), x ∈ A → ∀ (y : fin n → α), y ∈ A →
    x ≠ y → p.eval (a · x + b · y) = 0) →
  (A.filter (λ x, p.eval (-c · x) ≠ 0)).card ≤ 2 * m (d / 2)
```

Proving this proposition requires the most intricate argument of our formalization. We note that this is in line with Ellenberg and Gijswijt's paper; their corresponding Proposition 2 makes up nearly a third of the non-expository content. Some of the intricacy comes from another shift of representation. Every student of linear algebra learns that linear transformations between finite-dimensional vector spaces can be represented by matrices, and it is standard in mathematics to conflate the two concepts. While our lemma (after unfolding the definition of `P_sup`) is stated in terms of the linear transformation `p.eval`, Ellenberg and Gijswijt's argument proceeds more naturally in the matrix setting. Formalizing their argument required significant library development to unify the treatment of linear transformations and matrices in Lean. We expect that this development will be reusable in future results that depend on linear algebra.

Briefly, the proof of `proposition_11_1` proceeds as follows. Given terms `a b : α`, `x y : fin n → α`, and `p : mv_polynomial (fin n) α` with `p ∈ S' d`, the term `p.eval (a · x + b · y)` can be written as a linear combination of evaluated monomials in `M' d`. We define an `A × A` matrix `B` such that `B x y = p.eval (a · x + b · y)`. In fact, we can factor the matrix `B` and express it in the following form:

```
lemma B_eq_sum_matrix : B =
  split_left.sum (λ _ _, matrix.vec_mul_vec _ _) +
  split_right.sum (λ  _ _, matrix.vec_mul_vec _ _)
```

(We direct interested readers to our formalization for the details of this computation.) Here, the cardinalities of the finite sets `split_left` and `split_right` are at most `m (d/2)`. Since the product of two vectors `matrix.vec_mul_vec` has rank 1, this implies that `B` has rank at most `2 * m (d / 2)`. But in fact, `B` is a diagonal matrix, from which we can infer that its rank is equal to the cardinality we wish to bound.

4.4 Lemma 4: A Combinatorial Calculation

Our next lemma, largely independent of the previous ones, relates different values of `m`.

```
theorem lemma_12_5 : q^n ≤ m ((q-1)*n - d) + m d
```

This lemma follows from a combinatorial argument on `fin n → fin q`, the type of `n`-tuples of natural numbers less than `q`. First, we define functions to map such a tuple to the monomial with corresponding exponents, and in reverse:

```
def monom : (fin n → fin q) → mv_polynomial (fin n) α
def monom_exps : mv_polynomial (fin n) α → (fin n → fin q)
```

Note that these functions are inverses when we restrict `fin n → fin q` to the subset `M`.

We then define five terms of type `finset (fin n → fin q)`, including the universal set:

```
I := finset.univ
B := {v ∈ I // (total_degree (monom v)) ≤ d}
C := {v ∈ I // (total_degree (monom v)) > d}
D := {v ∈ I // (total_degree (monom v)) < (q-1)*n - d}
E := {v ∈ I // (total_degree (monom v)) ≤ (q-1)*n - d}
```

There are a number of straightforward cardinality calculations that follow. Among them, we show that `B.card = m d`, since `B` is the image of `M' d` under `monom_exps`. It similarly holds that `E.card = m ((q-1)*n - d)`. The function sending the tuple $(a_1, \ldots, a_n)$ to $(q-1-a_1, \ldots, q-1-a_n)$ is a bijection and maps `C` to `D`; thus these sets have the same cardinality. Combining these calculations leads us to our goal.

Thanks to the large library of `finset` operations in `mathlib`, the proof of this lemma is basically frictionless. Indeed, the least pleasant part is checking that the bijection used is in fact a bijection, an argument that involves some trivial natural number arithmetic.

4.5 Lemma 5: Connecting These Lemmas

We have nearly achieved our goal for this section. Combining the previous four lemmas via linear arithmetic, we obtain the following:

```
theorem lemma_12_6 : A.card ≤ 2 * m (d/2) + m ((q-1)*n - d) :=
by linarith using [lemma_12_2, lemma_12_3, lemma_12_4, lemma_12_5]
```

Finally, abstracting the parameter `d` and instantiating it with `2/3*(q-1)*n` delivers our desired bound.

```
theorem theorem_12_1 : A.card ≤ 3*(m (1/3*((q-1)*n)))
```

5 Asymptotics

We have shown an upper bound for the cardinality of a cap set `A` in terms of `n`. To be precise, this bound is proportional to the number of monomials in `n` variables with total degree at most `(q-1)*n/3`, where `q` is the cardinality of the underlying finite field.

Our goal was to investigate the growth rate of this bound, in terms of `n`. In particular, we would like to show that it grows at a rate bounded above by `c^n`, for some `c < q`. Ellenberg and Gijswijt apply Cramér's theorem, a fairly deep result in probability theory (not to be confused with Cramer's rule), to derive this fact. But this detour is not necessary, and formalizing Cramér's theorem would be a significant undertaking on its own. We verify the growth rate of the size of `A` using more elementary methods. While the results of this section could be stated in terms of $\mathcal{O}$-notation [1], we favor a more explicit style, which allows us to state the $q = 3$ result in very concrete terms.

Our goal is the following general statement:

```
theorem general_cap_set {α : Type} [discrete_field α] [fintype α] :
  ∃ B C : ℝ, B > 0 ∧ C > 0 ∧ C < card α ∧
    ∀ {a b c : α} {n : ℕ} {A : finset (fin n → α)},
      c ≠ 0 → a + b + c = 0 →
      (∀ x y z ∈ A, a · x + b · y + c · z = 0 → x = y ∧ x = z) →
      A.card ≤ B * C ^ n
```

Our motivating example is concerned with the case where the underlying field is $\mathbb{Z}/3\mathbb{Z}$. In this case, we can be more explicit about the growth rate:

```
theorem cap_set {n : ℕ} {A : finset (fin n → ℤ/3ℤ)} :
  (∀ x y z ∈ A, x + y + z = 0 → x = y ∧ x = z) →
    A.card ≤ 3 * (((3/8) ^ 3 * (207 + 33 * sqrt 33)) ^ (1/3)) ^ n
```

Since we have that

$$\sqrt[3]{\left(\frac{3}{8}\right)^3 \left(207 + 33\sqrt{33}\right)} \approx 2.755,$$

this result answers the cap set problem in the affirmative.

To prove `general_cap_set`, we will show an alternate representation for `m` and develop an argument that bounds this value from above in terms of `n` and `d`. This argument involves some combinatorial calculations similar to those presented in Section 4.4.

In the previous section we worked with a fixed parameter `n`, the length of the vectors. It is now necessary to abstract over this parameter. (We will keep the base field α and its cardinality `q` fixed.) Note that `m` depends on both `n` and a rational input `d`.

5.1 Expressing `m` as a Sum of Coefficients

Our first lemma will show that we can write `m` as a sum of coefficients depending on n and d. On paper, we define

$$c_j^{(n)} := \left| \left\{ (a_1, \ldots, a_n) \;\middle|\; a_i \in \{0, 1, \ldots, q-1\} \text{ and } \sum_{i=1}^{n} a_i = j \right\} \right|.$$

We again face a choice of how to represent these values in Lean. In Section 4.4, we represented such tuples $(a_1, \ldots, a_n)$ with the type `fin n → fin q`. This type is very convenient when `n` is fixed, but a following lemma will proceed by induction on `n`, and the function representation is cumbersome in this kind of argument. We choose instead to represent these tuples with the type `vector (fin q) n`, defined to be the subtype of `list (fin q)` whose elements have fixed length `n`. To connect with earlier results stated using the function representation, we will show a bijection between the two types. Moving between representations like this is aided by library support for establishing bijections and showing that relevant properties are preserved, and with the right support, it is far easier to carry out arguments in the "natural" setting.

With this in mind, we define:

```
def sf (n j : ℕ) : finset (vector (fin q) n) :=
finset.univ.filter (λ f, (f.nat_sum = j))

def cf (n j : ℕ) : ℕ := (sf n j).card
```

Following the bijection between representations of tuples, and reusing some of the cardinality computations from Section 4.4, we show that `m n d` is equal to the sum of `cf q n j` for `0 ≤ j ≤ ⌊d⌋`:

```
theorem lemma_13_8 (n : ℕ) {d : ℚ} (hd : d ≥ 0) :
  m n d = (finset.range (⌊d⌋.nat_abs + 1)).sum (cf n)
```

To get a better handle on `m`, we would like a more algebraic representation of `cf`. As an intermediate step, we turn again to the setting of polynomials, this time univariate:

we will show that for each j and n, $c_j^{(n)}$ is equal to the jth coefficient of the polynomial $(1 + x + \ldots + x^{q-1})^n$.

It is in this argument that we benefit from using the list representation for tuples, as we need to prove:

```
lemma cf_mul (n j : ℕ) : cf (n+2) j =
  (finset.range (j + 1)).sum (λ i, (cf 1 (j - i)) * cf (n + 1) i)
```

This combinatorial puzzle requires lifting $(n+1)$-tuples to $(n+2)$-tuples. Any $(n+2)$-tuple of natural numbers less than q whose values sum to j can be constructed by appending its last value k to an $(n+1)$-tuple whose values sum to $i = j - k$. The number of such $(n+2)$-tuples, then, is the sum of the number of such $(n+1)$-tuples where i ranges from 0 to $\max(q-1, j)$. Since `cf 1 k` is 0 when `k > q` and 1 otherwise, this sum is equal to the expression in `cf_mul`.

Counting arguments like this can make for entertaining puzzles on paper, but the pain of formalizing them can be compounded by using the wrong representation. We found that the lifting of tuples required for this argument was much more natural under the list representation for tuples; casts in the function representation became unwieldy.

With this identity, and proceeding by induction on `n`, we can define the polynomial $1 + x + \ldots + x^{q-1}$ and show our desired result:

```
def one_coeff_poly (m : ℕ) : polynomial ℕ :=
(finset.range m).sum (λ k, (polynomial.X : polynomial ℕ) ^ k)

theorem lemma_13_9 (hq : q > 0) :
  ∀ n j : ℕ, ((one_coeff_poly q) ^ n).coeff j = cf n j
```

5.2 Concrete Bounds on `m`

We can now write `m` in terms of the coefficients `cf`. We will use this representation to establish a concrete upper bound on the values of `m`. This upper bound will be in terms of another auxiliary value:

```
def crq (r : ℝ) (q : ℕ) :=
((one_coeff_poly q).eval₂ coe r) / r ^ ((q-1)/3)
```

Note that for `p : polynomial ℕ` and `r : ℝ`, `p.eval₂ coe r` embeds the coefficients of `p` into the real numbers and evaluates the resulting polynomial at `r`.

For every `r` between 0 and 1, `crq` bounds `m`:

```
theorem theorem_14_1 {r : ℝ} (hr : 0 < r) (hr2 : r < 1) : m ((q -
    1)*n / 3) ≤ (crq r q) ^ n
```

This result is derived from `theorem_13_8` and `theorem_13_9`, with the additional fact that summing the monomials of a polynomial over its support is the same as evaluating the polynomial.

```
lemma finset_sum_range {r : ℝ} (hr : 0 < r) (hr2 : r < 1) :
  (finset.range ((q - 1) * n + 1)).sum (λ j, r ^ j * (cf q n j)) =
    ((one_coeff_poly q) ^ n).eval₂ coe r
```

Since `crq 1 q = q` and the derivative of `crq` with respect to `r` is positive at `r = 1`, we have from elementary calculus:

```
theorem lemma_13_15 : ∃ r : ℝ, 0 < r ∧ r < 1 ∧ crq r q < q
```

Instantiating `theorem_14_1` with this `r`, invoking `theorem_12_1`, and abstracting the type parameter α leads us to the theorem `general_cap_set` stated at the beginning of this section.

We finally return to the original cap set problem with `q = 3`. Pen and paper calculations show that `crq r 1` is minimized in `r` at `r := (real.sqrt 33 - 1) / 8`. Aided by the numeral and ring normalization tactics in `mathlib`, we establish that `0 < r < 1` and that `crq r 3 = ((3 / 8)^3 * (207 + 33*real.sqrt 33))^(1/3)`. We apply `theorem_14_1` to this `r` to conclude:

```
theorem cap_set {n : ℕ} {A : finset (fin n → ℤ/3ℤ)} :
  (∀ x y z ∈ A, x + y + z = 0 → x = y ∧ x = z) →
    A.card ≤ 3 * (((3/8) ^ 3 * (207 + 33 * sqrt 33)) ^ (1/3)) ^ n
```

6 Related Work

We are not aware of any existing formal developments that relate directly to the cap set problem or the polynomial method. Since the core library components of our proof are in combinatorics and number theory, linear algebra, and the theory of polynomials, we provide here a survey of formalizations in these areas. This incomplete list is meant to indicate the depth and flavor of such projects.

The combinatorial arguments we employ are fairly simple results about involutions and the cardinalities of finite sets; similar developments exist in the libraries of most modern proof assistants. Gonthier's proof of the four color theorem in Coq [19] includes some more sophisticated proofs. Dubois, Giorgetti, and Genestier [14] also provide a Coq library for enumerative combinatorics, again more sophisticated than what is needed in our proof.

While the result of Ellenberg and Gijswijt is most clearly characterized as combinatorics, it is also of interest in number theory. There has been recent attention toward formalizing results in this area, including Eberl's work on analytic number theory in Isabelle/HOL [16] and Lewis' work on the p-adic numbers in Lean [26]. Chyzak, Mahboubi, Sibut-Pinote, and Tassi's Coq proof that $\zeta(3)$ is irrational [7] is also relevant.

Finite fields play an important role in combinatorics and number theory and are needed to state our general result. Chan and Norrish's mechanization of the AKS algorithm [5] shows an approach to their study in HOL4, which makes for an interesting contrast with our approach in a dependently typed system. Their subsequent work [6] relates to ours in its study of polynomials over finite fields.

There are many formal proof developments of linear algebra. Our additions to `mathlib` were partially inspired by the impressive work of Gonthier in Coq [20], Lee [25] and Aransay and Divasón [2, 13] in Isabelle/HOL, and Harrison in HOL Light [24].

Our formalization focuses in particular on the vector space of polynomials, also seen in Divasón, Joosten, Thiemann, and Yamada [12]. As with linear algebra, polynomials are a fundamental object of study in mathematics, and they appear in most proof assistant libraries. Some recent results concerning polynomials include Bernard, Bertot, Rideau, and Strub [3] and Eberl [15].

7 Conclusion

We have formalized Ellenberg and Gijswijt's solution to the cap set problem, a recent and celebrated result in combinatorics. Our formalization is evidence that verifying certain cutting-edge mathematics is possible without enormous investments of time or resources. This effort was undertaken as part of the Lean Forward project, which aims to develop tools, tactics, and libraries to formalize modern results in number theory and related areas. Much of the background theory we have implemented will be of future use in this project.

At the outset of our efforts, the first author produced a detailed paper proof of the result, drawing from Ellenberg and Gijswijt and from Zeilberger [33] and adapting the asymptotics part significantly. We used this writeup as a blueprint for our formalization. It was heartening to see that the blueprint translated very directly to Lean. We were able to work at a similar level of abstraction as the original sources without any complications introduced by the proof assistant.

Our proof of the asymptotics is a significant simplification of the original arguments. While in principle this could have been found without any interactive theorem proving, it was ultimately due to the formalization process, including the necessity to explore alternative paths of this part of the proof and feedback from Gijswijt on an earlier version of this paper, that this simplification was established.

As usual, it is difficult to compare the length of formal proofs with their paper counterparts, since the background assumptions and level of detail differ significantly. Nevertheless, we can provide some approximate information. Ellenberg and Gijswijt's paper contains just over two pages of mathematical work. Our blueprint is seventeen pages long; the first six pages are preliminary material, and two pages correspond to an obsolete argument (Appendix A). The remaining nine pages correspond to around 2000 lines of our formalization. (This does not represent our entire effort: thousands more lines of general definitions and proofs were added to `mathlib` as part of this project.) The ratio of 2000 lines of formal proof to two pages of paper proof is perhaps misleading, since we take a more verbose approach to checking the asymptotic behavior of the upper bound. (Ellenberg and Gijswijt take only one paragraph to invoke Cramér's theorem.) A better comparison is the part of the proof described in Section 4: 900 formal lines subsume a page and a half of paper proof. The corresponding section of our detailed writeup is just under five pages.

This formalization, and `mathlib` more generally, rely heavily on hierarchies of type classes. In some sections of our proof—particularly those involving linear subspaces of the type of multivariate polynomials—we found that type class inference behaved erratically. The backtracking search performed by Lean's elaborator is sensitive to many features, and import order and additional instances can greatly affect the depth and speed of the search. We ended up revising the hierarchy in parts of `mathlib` to simplify this. A moral we have taken from this project is that "misleading" instances that lead the elaborator down a long and ultimately unsuccessful path can be nearly as dangerous as circular instances.

A An Earlier Proof of Asymptotics

After submission of our paper, Dion Gijswijt suggested a further simplification to the approach we used for controlling the asymptotic behavior of the bound. The argument we present above in Sections 2 and 5 follows this suggestion. For the sake of completeness, we present here our original approach, which may be of interest in its own right.

A.1 Informal Description

We will bound the coefficients of the polynomials from (3):

$$m_d = \sum_{i=0}^{\lfloor d \rfloor} \Big(\text{coefficient of } x^i \text{ in the polynomial } \left(1 + x + \ldots x^{q-1}\right)^n\Big). \tag{5}$$

We can work in an algebraic manner as follows, thus avoiding Cauchy's residue theorem from complex analysis. Let k be any field, $f \in k[x]$, $i \in \mathbb{N}$, $\zeta \in k^*$ of finite order l, and $r \in k^*$. If $l > \max(\deg(f), i)$, then

$$l \cdot \big(\text{coefficient of } x^i \text{ in the polynomial } f\big) = \sum_{j=0}^{l-1} \frac{f(r\zeta^j)}{r^i \zeta^{ij}}. \tag{6}$$

The key ingredient for proving this statement is the following special case of the geometric sum, where ζ and l are as above and $h \in \mathbb{Z}$.

$$\sum_{j=0}^{l-1} \zeta^{hj} = \begin{cases} 0 & \text{if } l \nmid h \\ l & \text{if } l \mid h \end{cases}$$

Repeatedly applying (6) to (5) with $k = \mathbb{C}$, $\zeta = \exp(2\pi\sqrt{-1}/l)$ for any $l > n(q-1)$, and $r \in \mathbb{R}$ satisfying $0 < r < 1$, as well as calculating and estimating quite a bit, we obtain that

$$m_{(q-1)n/3} \leq B_{r,q} C_{r,q}^n$$

for some constants $B_{r,q}, C_{r,q} \in \mathbb{R}_{>0}$ depending only on r and q. Specifically, we can take $C_{r,q}$ as in (4).

A.2 Formalization

We pick up at the beginning of Section 5.2, where we have not yet established an algebraic representation for `cf`. It is necessary to get a better handle on the coefficients of `one_coeff_poly ^ n`. A brief detour into estimates with complex numbers will result in the following bound:

```
theorem lemma_13_10 (n : ℕ) {r : ℝ} (hr : r > 0) :
  cf n j ≤ (((one_coeff_poly q)^n).eval₂ coe r) / r^j
```

Note that for `p : polynomial ℕ` and `r : ℝ`, `p.eval₂ coe r` embeds the coefficients of `p` into the real numbers and evaluates the resulting polynomial at `r`. This operation is generic, and we will soon embed this same polynomial into $\mathbb{C}$.

To obtain the bound in `lemma_13_10`, we will use a general result about complex polynomials. We derive this directly, but we note that it also follows from general considerations about Laurent polynomials:

```
def ζk (k : ℤ) : ℂ := exp (2*π*I/k)

lemma pick_out_coef {f : polynomial ℂ} {i k : ℕ} (h1 : k > i)
  (h2 : k > nat_degree f) {r : ℝ} (h3 : r > 0) :
  (coeff f i) * k =
    (range k).sum (λ j, (eval (r*(ζk k)^j) f)/(r^i * (ζk k)^(i*j)))
```

When we instantiate `f` with the embedding of `one_coeff_poly ^ n` into $\mathbb{C}$, we see that this complex sum is in fact a nonnegative real number for each `i`, since it is equal to `cf i n`. We can thus approximate its absolute value using the triangle inequality to derive `lemma_13_10` above.

We can now write `m` in terms of the coefficients `cf`, and for each positive real `r`, we can bound `cf` from above in terms of `r`. It remains to establish a concrete upper bound on `m`.

We will do so using the same auxiliary value used in Section 5.2:

```
def crq (r : ℝ) (q : ℕ) :=
((one_coeff_poly q).eval₂ coe r) / r ^ ((q-1)/3)
```

It is convenient to first establish a bound in the case where `n` is divisible by 3. The proof of this bound combines `lemma_13_8` and `lemma_13_10` with some elementary results about geometric sums.

```
theorem lemma_13_11 (N : ℕ) {r : ℝ} (hr : 0 < r) (hr2 : r < 1) :
  m (3*N) ((q-1)*N) ≤ (1/(1-r)) * ((crq r q))^(3*N)
```

Recall that `m n d` is the number of monomials in `n` variables with total degree at most `d`. This number is clearly monotonic increasing in `d`; it is also easy to recognize that it is monotonic increasing in `n`, although formalizing this takes slightly more work. From these considerations and the previous lemma, we deduce:

```
theorem theorem_13_13 (n : ℕ) {r : ℝ} (hr : 0 < r) (hr2 : r < 1) :
  (m n ((q - 1)*n / 3)) ≤ ((crq r q)^2 / (1 - r)) * (crq r q)^n
```

As earlier, we can now derive from elementary calculus:

```
theorem lemma_13_15 : ∃ r : ℝ, 0 < r ∧ r < 1 ∧ crq r q < q
```

Instantiating `theorem_13_13` with this `r`, invoking `theorem_12_1`, and abstracting the type parameter α leads us to the theorem `general_cap_set`.

We finally return to the original cap set problem with `q = 3`. Since we have used the same function `crq` as in Section 5.2, we can optimize it in `r` in the same way to find the value `r := (real.sqrt 33 - 1) / 8`. Aided by the numeral and ring normalization tactics in `mathlib`, we establish that `0 < r < 1` and that `crq r 3 = ((3 / 8)^3 * (207 + 33*real.sqrt 33))^(1/3)`. We compute the rough approximation `(crq r q)^2 / (1 - r) ≤ 198` to conclude:

```
theorem cap_set {n : ℕ} {A : finset (fin n → ℤ/3ℤ)} :
  (∀ x y z ∈ A, x + y + z = 0 → x = y ∧ x = z) →
    A.card ≤ 198 * (((3/8) ^ 3 * (207 + 33 * sqrt 33)) ^ (1/3)) ^ n
```

References

1. Reynald Affeldt, Cyril Cohen, and Damien Rouhling. Formalization Techniques for Asymptotic Reasoning in Classical Analysis. *Journal of Formalized Reasoning*, October 2018. URL: https://hal.inria.fr/hal-01719918.
2. Jesús Aransay and Jose Divasón. Formalization and execution of linear algebra: From theorems to algorithms. In Gopal Gupta and Ricardo Peña, editors, *Logic-Based Program Synthesis and Transformation*, pages 1–18, Cham, 2014. Springer International Publishing.
3. Sophie Bernard, Yves Bertot, Laurence Rideau, and Pierre-Yves Strub. Formal proofs of transcendence for e and pi as an application of multivariate and symmetric polynomials. In *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2016, pages 76–87, New York, NY, USA, 2016. ACM. doi:10.1145/2854065.2854072.

4 Mario Carneiro. The Lean 3 mathematical library (presentation), July 2018. URL: http://robertylewis.com/files/icms/Carneiro_mathlib.pdf.

5 Hing-Lun Chan and Michael Norrish. Mechanisation of AKS algorithm: Part 1 – the main theorem. In Christian Urban and Xingyuan Zhang, editors, *Interactive Theorem Proving*, pages 117–136, Cham, 2015. Springer International Publishing.

6 Hing-Lun Chan and Michael Norrish. Proof pearl: Bounding least common multiples with triangles. In Jasmin Christian Blanchette and Stephan Merz, editors, *Interactive Theorem Proving*, pages 140–150, Cham, 2016. Springer International Publishing.

7 Frédéric Chyzak, Assia Mahboubi, Thomas Sibut-Pinote, and Enrico Tassi. A computer-algebra-based formal proof of the irrationality of $\zeta(3)$. In Gerwin Klein and Ruben Gamboa, editors, *Interactive Theorem Proving*, pages 160–176, Cham, 2014. Springer International Publishing.

8 Thierry Coquand and Christine Paulin. Inductively defined types. In *COLOG-88 (Tallinn, 1988)*, volume 417 of *Lec. Notes in Comp. Sci.*, pages 50–66. Springer, Berlin, 1990. doi:10.1007/3-540-52335-9_47.

9 Ernie Croot, Vsevolod F. Lev, and Péter Pál Pach. Progression-free sets in $\mathbb{Z}_4^n$ are exponentially small. *Ann. of Math. (2)*, 185(1):331–337, 2017. doi:10.4007/annals.2017.185.1.7.

10 N. G. de Bruijn. Automath, a language for mathematics. In Jörg H. Siekmann and Graham Wrightson, editors, *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*, pages 159–200. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983. doi:10.1007/978-3-642-81955-1_11.

11 Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The Lean theorem prover, 2014. URL: http://leanprover.github.io/files/system.pdf.

12 Jose Divasón, Sebastiaan Joosten, René Thiemann, and Akihisa Yamada. A formalization of the Berlekamp-Zassenhaus factorization algorithm. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2017, pages 17–29, New York, NY, USA, 2017. ACM. doi:10.1145/3018610.3018617.

13 Jose Divasón and Jesús Aransay. Rank-nullity theorem in linear algebra. *Archive of Formal Proofs*, January 2013. http://isa-afp.org/entries/Rank_Nullity_Theorem.html, Formal proof development.

14 Catherine Dubois, Alain Giorgetti, and Richard Genestier. Tests and proofs for enumerative combinatorics. In Bernhard K. Aichernig and Carlo A. Furia, editors, *Tests and Proofs*, pages 57–75, Cham, 2016. Springer International Publishing.

15 Manuel Eberl. Symmetric polynomials. *Archive of Formal Proofs*, September 2018. http://isa-afp.org/entries/Symmetric_Polynomials.html, Formal proof development.

16 Manuel Eberl. Nine chapters of analytic number theory in Isabelle/HOL. In *Interactive Theorem Proving*, 2019.

17 Gabriel Ebner, Sebastian Ullrich, Jared Roesch, Jeremy Avigad, and Leonardo de Moura. A metaprogramming framework for formal verification. *Proceedings of the ACM on Programming Languages*, 1(ICFP):34, 2017.

18 Jordan S. Ellenberg and Dion Gijswijt. On large subsets of $\mathbb{F}_q^n$ with no three-term arithmetic progression. *Ann. of Math. (2)*, 185(1):339–343, 2017. doi:10.4007/annals.2017.185.1.8.

19 Georges Gonthier. The four colour theorem: Engineering of a formal proof. In Deepak Kapur, editor, *Computer Mathematics*, pages 333–333, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

20 Georges Gonthier. Point-free, set-free concrete linear algebra. In Marko van Eekelen, Herman Geuvers, Julien Schmaltz, and Freek Wiedijk, editors, *Interactive Theorem Proving*, pages 103–118, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

21 Sébastien Gouëzel and Vladimir Shchur. A corrected quantitative version of the Morse lemma. *arXiv preprint arXiv:1810.04579*, 2018.

22 Larry Guth. *Polynomial methods in combinatorics*, volume 64 of *University Lecture Series*. American Mathematical Society, Providence, RI, 2016.

23 Thomas Hales, Mark Adams, Gertrud Bauer, Tat Dat Dang, John Harrison, Hoang Le Truong, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Tat Thang Nguyen, et al. A formal proof of the Kepler conjecture. In *Forum of Mathematics, Pi*, volume 5. Cambridge University Press, 2017.

24 John Harrison. The HOL Light theory of euclidean space. *J. Autom. Reason.*, 50(2):173–190, February 2013. doi:10.1007/s10817-012-9250-9.

25 Holden Lee. Vector spaces. *Archive of Formal Proofs*, August 2014. http://isa-afp.org/entries/VectorSpace.html, Formal proof development.

26 Robert Y. Lewis. A formal proof of Hensel's lemma over the p-adic integers. In *Proceedings of the 8th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2019, pages 15–26, New York, NY, USA, 2019. ACM. doi:10.1145/3293880.3294089.

27 Roman Matuszewski and Piotr Rudnicki. Mizar: the first 30 years. *Mechanized Mathematics and Its Applications*, 4(1):3–24, March 2005.

28 Bas Spitters and Eelis van der Weegen. Type classes for mathematics in type theory. *Mathematical Structures in Computer Science*, 21(4):795–825, 2011.

29 Terence Tao. Algebraic combinatorial geometry: the polynomial method in arithmetic combinatorics, incidence combinatorics, and number theory. *EMS Surv. Math. Sci.*, 1(1):1–46, 2014. doi:10.4171/EMSS/1.

30 Terence Tao. A symmetric formulation of the Croot-Lev-Pach-Ellenberg-Gijswijt capset bound, May 2016. URL: http://terrytao.wordpress.com/2016/05/18.

31 Floris van Doorn, Jakob von Raumer, and Ulrik Buchholtz. Homotopy type theory in Lean. In Mauricio Ayala-Rincón and César A. Muñoz, editors, *Interactive Theorem Proving*, pages 479–495. Springer International Publishing, 2017.

32 P. Wadler and S. Blott. How to make ad-hoc polymorphism less ad hoc. In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '89, pages 60–76, New York, NY, USA, 1989. ACM. doi:10.1145/75277.75283.

33 Doron Zeilberger. A motivated rendition of the Ellenberg–Gijswijt gorgeous proof that the largest subset of F_3^n with no three-term arithmetic progression is $O(c^n)$, with $c = \sqrt[3]{(5589+891\sqrt{33})}/8 = 2.755104613023633000221 27\ldots$. *arXiv preprint arXiv:1607.01804*, 2016.