Formalizing the Bruck–Ryser–Chowla Theorem: Combinatorial Design Theory in Lean

Eric Jonathan Wang
Carnegie Mellon University, Pittsburgh, PA, USA

Elif Uskuplu
Indiana University, Bloomington, IN, USA

Abstract

We present a formalization of combinatorial design theory in Lean 4, with a focus on balanced incomplete block designs (BIBDs) and their algebraic properties. The flagship result is the Bruck–Ryser–Chowla theorem, which gives the best known necessary conditions for the existence of a symmetric BIBD, formalized here in a proof assistant for the first time. Reaching this result required us to develop substantial infrastructure beyond combinatorics: we formalize Witt's cancellation theorem for quadratic forms, prove new results on matrix congruence and block matrices, and extend Mathlib's linear algebra library in several directions. We also provide the first formalization of Fisher's inequality in Lean and the first formalization of the Kramer–Mesner theorem in any proof assistant, along with a reusable double-counting argument that supports standard combinatorial reasoning. The cross-domain nature of these contributions reflects a distinctive feature of design theory itself: it draws on and feeds back into many areas of mathematics, making it a particularly rewarding target for formalization within a large-scale library like Mathlib.

2012 ACM Subject Classification Theory of computation → Logic and verification; Software and its engineering → Formal software verification

Keywords and phrases Lean theorem prover, combinatorial design theory, BIBD, matrix theory, Bruck–Ryser–Chowla theorem, quadratic forms

Digital Object Identifier 10.4230/LIPIcs.ITP.2026.19

Supplementary Material *Software (Source Code)*: `https://github.com/ericwang2004/combinatorial-designs-formalization`
archived at `swh:1:dir:f89852141f28ef205935f5f7d5f22c1e5dbf0bac`

Funding Our research was conducted at the 2025 Indiana University REU program supported by NSF grant DMS-2447563. The first author acknowledges travel support from the Hoskinson Center for Formal Mathematics.

Acknowledgements We thank Lean's Zulip chat for answering our questions, as well as our anonymous referees for their thorough comments.

Declaration about GenAI/LLM use We employed AI assistance for the following tasks: checking the text for typos, drafting documentation comments for our Lean code, and identifying related work in the literature and in Mathlib. AI was not used in the formalization itself.

1 Introduction

Large-scale libraries such as Mathlib [16] now cover substantial portions of undergraduate and graduate mathematics in Lean [5], and recent efforts like the Polynomial Freiman–Ruzsa conjecture [1] and Fermat's Last Theorem [3] continue to extend this reach. Aiming to contribute to this growing effort, we present a formalization of combinatorial design theory (CDT) in Lean 4, targeting integration with Mathlib.

The notion of a combinatorial *design* originated in the statistician Ronald Fisher's 20th century work on the *design* of agricultural experiments [10], particularly in connection to the analysis of variance (ANOVA) method, where the arrangement of treatments across experimental units (the "blocks" of a design) was used to control for variability [4]. CDT has, since then, enjoyed connections to other areas of mathematics. From geometry, a projective plane of order n is a special case of a design known as a $(n^2 + n + 1, n + 1, 1)$-BIBD. From coding theory, the generator matrix of a linear code is the incidence matrix of a BIBD [11]. From cryptography, secret sharing schemes and authentication codes can be understood in design-theoretic terms [18, 20]. Design-theoretic proofs themselves draw heavily from linear algebra and number theory.

CDT is a particularly rich target for formalization as it sits at the crossroads of many disciplines. A well-developed CDT library would therefore serve as connecting tissue between many existing and future formalizations. Prior to our work, the only formalization of design theory in generality was carried out in Isabelle/HOL by Edmonds and Paulson [7, 6], who developed a hierarchy of design structures and proved results up to Fisher's inequality [8]. We discuss the relationship between their work and ours in Section 2.

The choice of Lean 4 was driven by several concrete advantages. Mathlib's extensive coverage of linear algebra and matrix theory gave us a strong foundation, and Lean's dependent type theory offers advantages over HOL-based systems for encoding the parameterized structures that arise naturally in design theory. Lean is also a full-featured programming language, which matters for CDT because constructions, parameter calculations, and existence checks all benefit from a capable programming layer underneath the logic. These considerations led us to build a new Mathlib-compatible CDT library from scratch rather than attempting to port the existing Isabelle development.

Our primary reference is Stinson's *Combinatorial Designs: Constructions and Analysis* [19]. We focus on BIBDs, designing the type hierarchy to be as general as practical. The formalization includes Fisher's inequality, the Kramer–Mesner theorem, and, as the flagship result, the Bruck–Ryser–Chowla (BRC) theorem, which gives the best known necessary conditions for the existence of a symmetric BIBD. Reaching BRC required matrix-theoretic infrastructure not available in Mathlib, including results on matrix congruence and direct sums, as well as a formalization of Witt's cancellation theorem for quadratic forms via a proof route different from Stinson's. Altogether, the project provides both domain-specific contributions to CDT and general-purpose additions to Mathlib's linear algebra library.

Section 2 introduces the mathematical background, presents the design hierarchy, and compares our approach with the Isabelle formalization. Section 3 covers incidence matrices and the first layer of important results, including Fisher's inequality and the Kramer–Mesner theorem. Section 4 describes the formalization of the Bruck–Ryser–Chowla theorem. Section 5 discusses directions for future work.

2 The Design Library

A *design* is a pair $(X, \mathcal{B})$ where X is a set of *points* and $\mathcal{B}$ is a collection (multiset) of subsets of X called *blocks*. In Lean, we represent a design as a structure parameterized by types `X` and ι, where `X` is the point set and ι is an index set for the blocks:

```
structure Design (ι X : Type*) where
  blocks : ι → Finset X
```

The choice to index blocks by a type ι rather than using `Multiset (Finset X)` is deliberate; the two representations are mathematically equivalent, and we discuss the tradeoffs in Section 2.2.

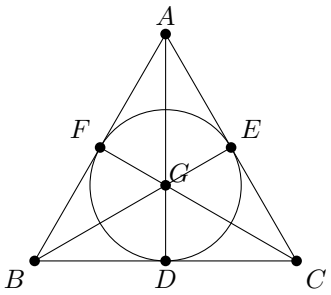
▶ **Example 1.** Take $X = \{0, 1, 2\}$ and $\mathcal{B} = [\{0, 1\}, \{1, 2\}, \{0\}, \{0\}]$. Then $(X, \mathcal{B})$ is a design. To encode this using the formal definition, we set `X` to `Fin 3`, ι to `Fin 4`, and define `blocks` by

$$0 \mapsto \{0, 1\},\ 1 \mapsto \{1, 2\},\ 2 \mapsto \{0\},\ 3 \mapsto \{0\}.$$

▶ **Example 2** (Fano plane)**.** Take

$$\begin{aligned} X &= \{A, B, C, D, E, F, G\}, \\ \mathcal{B} &= [\{A, F, B\}, \{B, D, C\}, \{C, E, A\}, \{A, G, D\}, \\ &\quad \{B, G, E\}, \{C, G, F\}, \{D, E, F\}]. \end{aligned}$$

The design $(X, \mathcal{B})$ is the *Fano plane*. It can be drawn as follows, where the lines and the circle represent the blocks.



We build a hierarchy of design structures by progressively adding constraints, leading to the following chain of definitions:

- A *block design* (`BlockDesign`) has uniform block size k.
- An *r-regular design* (`RegularDesign`) has each point occurring in exactly r blocks. The parameter r is the *replication number*.
- A *regular block design* (`RegularBlockDesign`) is both regular and a block design.
- A *t-wise balanced design* (`BalancedDesign`) has the property that any t distinct points appear together in exactly λ blocks.
- A *pairwise balanced design* (`PBD`) is a 2-wise balanced design.
- An *incomplete design* (`IncompleteDesign`) is a block design such that $k < v$ where $v = \#X$.
- A *regular pairwise balanced design* (`RPBD`) is both regular and pairwise balanced.
- A *nontrivial RPBD* (`nontrivialRPBD`) is an RPBD with at least one block B satisfying $0 < \#B < \#X$. This is the natural setting for Fisher's inequality.
- A *t-design* (`TDesign`) is a t-wise balanced incomplete block design with $t \leq k$.
- A *balanced incomplete block design* (`BIBD`) is a 2-design. A (v, k, λ)-BIBD has v points, block size k, and balance number λ, with $2 \leq k < v$.

Figure 1 shows the inheritance diagram of these structures in our Lean formalization.

The basic parameter relations of BIBDs can be summarized as follows.

▶ **Theorem 3.** *Let Φ be a (v, k, λ)-BIBD with b blocks. Then:*

1. *Every point occurs in exactly $r = \lambda(v - 1)/(k - 1)$ blocks. That is, Φ is r-regular.*
2. *The number of blocks satisfies $bk = vr$.*
3. *If $b = v$ (a* symmetric *BIBD), then $r = k$ and any two blocks intersect in exactly λ points.*

The first two parts follow from double-counting arguments (Section 3); the third relies on Fisher's inequality to establish that the incidence matrix has full rank. In our formalization, each result is proved at the most general level at which it holds.

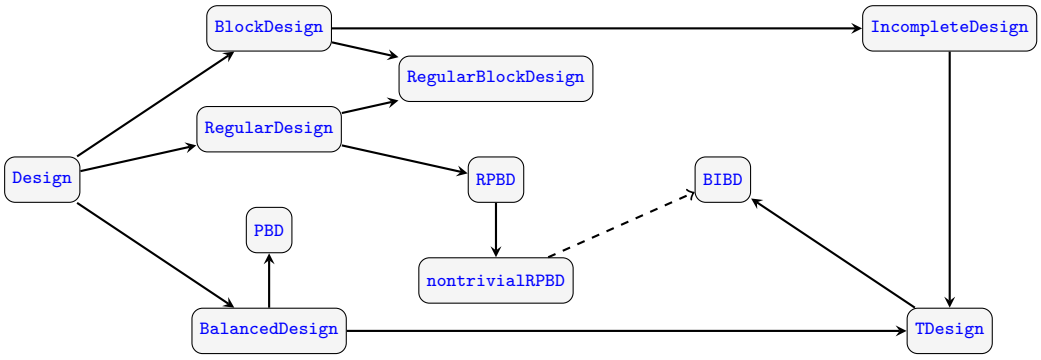


Figure 1 The design type hierarchy in our Lean formalization. Solid arrows point from general to specialized structures (`extends`). The dashed arrow is a consequence of Theorem 3.

2.1 Library Overview

Table 1 gives an overview of the Lean files comprising our formalization. The library comprises 14 files totaling approximately 3,400 lines of Lean code, containing 93 theorems and lemmas and 111 definitions, structures, and instances. There are no uses of `sorry` or `axiom`.

Several components of this library serve dual roles. The files `MatrixLemmas.lean` and `MatrixCongruence.lean` are directly required by the main results: the determinant and rank lemmas underpin Fisher's inequality and the incidence matrix characterizations, while matrix congruence is central to the BRC proof. At the same time, these results are general enough to be useful beyond our library. The file `WittCancellation.lean` formalizes Witt's cancellation theorem for quadratic forms, which is likewise essential for BRC. To our knowledge, this constitutes the first formalization of Witt's cancellation theorem for quadratic forms in any proof assistant.

The file `Diagonalizable.lean`, which proves that every symmetric matrix over a field of characteristic $\neq 2$ is congruent to a diagonal one, was developed for an earlier approach to Witt cancellation that we later replaced (see Section 4); we retain it as a standalone contribution.

The library also includes initial steps toward topics we plan to develop further: Hadamard matrices and their correspondence with certain symmetric BIBDs, and difference sets as an algebraic method for constructing designs. These are currently at an early stage but are included as part of the infrastructure for the broader CDT library we envision.

The auxiliary results we proved for Fisher's inequality, the Kramer–Mesner theorem, and the Bruck–Ryser–Chowla theorem were initially motivated by the needs of those specific proofs. However, as the library grew, we found that many of these lemmas had potential applications elsewhere. We chose to keep and sometimes generalize results that seemed reusable. In this way, the project also contributes to Lean's linear algebra library. All definitions follow Mathlib conventions and build on existing Mathlib infrastructure, so that integration requires minimal refactoring. Specifically, we intend to contribute the files `Diagonalizable.lean`, `MatrixCongruence.lean`, `MatrixLemmas.lean`, and `WittCancellation.lean`.

2.2 Comparison with the Isabelle Formalization

The only prior formalization of combinatorial design theory in full generality is the work of Edmonds and Paulson in Isabelle/HOL [6], later extended in Edmonds's thesis [9]. Their library covers a broad range of design types and constructions, and was subsequently used as the basis for a formalization of Fisher's inequality [8]. We compare the two formalizations along several axes.

Table 1 Overview of the Lean files in our CDT library.

File	Contents
`Defs.lean`	Core design hierarchy.
`Basic.lean`	Parameter relations for BIBDs: replication number, $bk = vr$, double-counting infrastructure, coercions between design types.
`IncidenceMatrix.lean`	Incidence matrix definition, algebraic characterization of RPBDs ($MM^\top = \lambda J + (r-\lambda)I$).
`Isomorphism.lean`	Design isomorphisms, automorphism group, correspondence between isomorphisms and incidence matrix permutations.
`Construction.lean`	Dual, sum, complement, derived, and residual design constructions with closure proofs. Construction of a symmetric BIBD from a difference set.
`Examples.lean`	Concrete examples including the Fano plane as a $(7, 3, 1)$-BIBD, with an explicit automorphism, no (22, 7, 2)-BIBD exists.
`FisherInequality.lean`	Fisher's inequality for nontrivial RPBDs via rank arguments.
`KramerMesner.lean`	Kramer–Mesner theorem: construction of t-designs from group actions and orbit matrices.
`SymmetricBIBD.lean`	Block intersection property for symmetric BIBDs, conversion from nontrivial RPBDs to BIBDs, and the Bruck–Ryser–Chowla theorem.
`MatrixLemmas.lean`	Auxiliary matrix results: determinant of $aJ+bI$, rank lemmas, unit matrices from full rank.
`MatrixCongruence.lean`	Matrix congruence relation, Lagrange's four squares applied to matrix identity, direct sums, and their interaction with congruence.
`Diagonalizable.lean`	Every symmetric matrix over a field with char $\neq 2$ is congruent to a diagonal matrix. A standalone result based on diagonalizable bilinear forms, retained for its general utility.
`WittCancellation.lean`	Witt's cancellation theorem for quadratic forms, formulated via isometry equivalences.
`HadamardMatrix.lean`	Hadamard matrix definition, basic properties.

Representation. The most fundamental difference lies in how designs are represented. In Isabelle, designs are formalized as locales whose basic data is a point set `V` (a finite set) and a block collection `B` (a multiset of sets). Our Lean formalization instead uses an indexed family `blocks : ι → Finset X`, where ι is an arbitrary type. The two representations are mathematically equivalent when both the point set and the index set are finite: one can form a multiset from an indexed family, and conversely extract an index type from a multiset. However, the choice has significant downstream consequences, and each approach reflects the strengths of its proof assistant.

The indexed representation makes incidence matrices immediate: the type `Matrix X ι α` gives a matrix whose rows are indexed by points and columns by block indices, with no additional setup. In Isabelle, where matrices are indexed by natural numbers, the multiset must first be converted to a list in order to assign column indices. This motivated the introduction of an `ordered_incidence_system` locale with explicit list representations for both points and blocks, together with lemmas for translating between the list and set/multiset views.

Double-counting arguments also benefit from the indexed approach. The standard technique of counting a set $S \subseteq X \times \mathcal{B}$ in two ways translates to switching the order of summation over the finite types X and ι, which is directly supported by Mathlib's `Finset.sum` infrastructure. With multisets, analogous arguments require multiset-specific counting lemmas rather than the general `Fintype` summation machinery.

Design isomorphisms are similarly cleaner in the indexed setting: an isomorphism consists of a bijection $X \simeq Y$ and a bijection $\iota \simeq \iota'$ compatible with the block maps, which is straightforward to state as a Lean structure. In the multiset setting, one instead checks that the image of the block multiset under the point bijection equals the target multiset, which requires reasoning about multiset equality (i.e., permutation equivalence).

On the other hand, the multiset representation has its own advantages. Repeated blocks are encoded directly through multiplicity, and operations like filtering by a predicate are natural. Lean's `Multiset` type, defined as the quotient `List α / Perm`, requires working with `Quotient.lift` to define functions, which can be less ergonomic than plain function types. Isabelle's mature multiset library makes this less of an issue there. The Kramer–Mesner theorem is an example where we found it useful to pass through the multiset view even within our indexed framework, since the proof constructs a block collection by taking multiplicities from a solution vector.

These differences are not accidental: they reflect the type systems of the respective proof assistants. Lean's structures take type parameters, making `Design ι X` natural. Isabelle's locales take value parameters (sets, multisets), making the multiset-based formulation a better fit for its architecture.

Design hierarchy. The Isabelle formalization starts from `incidence_system`, a pair of a set and a multiset of subsets satisfying $B \subseteq V$ for each block B. Adding finiteness yields `finite_incidence_system`, and further constraints produce the various design types. Since all design locales in Isabelle build on `finite_incidence_system`, the entire hierarchy is grounded in finite structures from the start.

Our Lean hierarchy takes a different approach. The base structure `Design` imposes no finiteness or decidable equality constraint on the point set; its only requirement is that blocks be finite subsets of X (`Finset X`). Finiteness of the point set enters at the level of `IncompleteDesign` (recall this means $k < \#X$): to speak of $\#X$ presupposes that X is finite. Decidable equality of X comes into play at `RegularDesign`, to ensure that the predicate `x ∈ blocks i` is decidable. These choices mean that our definition of `Design` does not directly correspond to any single locale in the Isabelle hierarchy. In practice, the finiteness and decidability assumptions are always present for the results we prove, but keeping the base type general leaves room for future extensions.

Breadth versus depth. The Isabelle library is notably broader: it includes group divisible designs (GDDs), resolvable designs, covering and packing designs, Wilson's construction, and connections to graph theory, among other topics. Their hierarchy encompasses over 30 design-related locales.

Our library is narrower in scope but reaches deeper into the algebraic direction. While the Isabelle formalization covers results up to Fisher's inequality (proved via linear algebraic methods in a separate development [8]), our formalization proceeds to the Kramer–Mesner theorem and the Bruck–Ryser–Chowla theorem, both of which are first formalizations in any proof assistant. The BRC theorem in particular required substantial infrastructure in matrix congruence and quadratic forms that has no counterpart in the Isabelle development. We note that an independent effort[1] to formalize BRC in Isabelle exists but remains incomplete.

[1] `https://github.com/cafeinst/Bruck_Ryser_Chowla`

Our library is designed to support the same kind of horizontal expansion that the Isabelle library has achieved. The type hierarchy is intentionally general, and the infrastructure for incidence matrices, counting arguments, and algebraic characterizations provides a foundation for adding new design families.

2.3 Fisher's Inequality

Since Fisher's inequality is the only major theorem formalized in both libraries, it merits a direct comparison. The Isabelle formalization [8] proves three variations: a *uniform* version for BIBDs using the rank argument, a *generalized* version for constant-intersect designs, and a *dual* version for PBDs. Our Lean formalization proves a single version for nontrivial RPBDs, corresponding most closely to the uniform version in proof technique. Table 2 summarizes the comparison; we highlight the most significant differences below.

Generality and result strength. The Isabelle uniform version assumes a (v, k, λ)-BIBD, while ours is stated for `nontrivialRPBD`, strictly generalizing it (every BIBD is a nontrivial RPBD via `BIBD_to_RPBD`). Moreover, the Isabelle theorem exposes only the cardinality bound $v \leq b$, whereas our statement preserves the intermediate rank: $v \leq \mathrm{rank}(M) \leq b$. In the symmetric case this yields $\mathrm{rank}(M) = \#X$, a prerequisite for the block intersection property (Theorem 10), the proof that symmetric nontrivial RPBDs are BIBDs (Theorem 11), and the Bruck–Ryser–Chowla theorem (Section 4). Proving $\lambda < r$ for nontrivial RPBDs, where no uniform block size is available, requires a separate 12-line combinatorial argument (Section 3.4).

Determinant computation. This is the step where the two formalizations diverge most sharply. In Isabelle, the determinant of $NN^\top$ is computed by elementary row and column operations across four lemmas totaling 129 non-blank lines of code. In Lean, a single algebraic identity (Theorem 16, 14 lines) replaces the entire computation by applying Mathlib's matrix determinant lemma; the rank calculation follows in 6 lines. The resulting lemmas are also reused in the even case of the Bruck–Ryser–Chowla theorem.

The remaining differences in Table 2 (type transfer, ordering infrastructure) are direct consequences of the representational choices discussed above: polymorphic incidence matrices eliminate type lifting, and type-indexed matrices eliminate ordering boilerplate. The Isabelle formalization additionally includes the generalized and dual variations, which use the linear algebra bound method and have no counterpart in our library.

3 From Counting to Linear Algebra

This section presents the combinatorial and algebraic core of the library. We begin with a counting principle that underpins many results in design theory, then move to incidence matrices and their algebraic characterization of designs. These tools together yield Fisher's inequality, the Kramer–Mesner theorem, and several structural results on symmetric BIBDs.

3.1 A Counting Principle

A fundamental result about BIBDs is that any BIBD is also an RPBD: every point occurs in the same number of blocks.

▶ **Theorem 4.** *In a (v, k, λ)-BIBD $(X, \mathcal{B})$, every $x \in X$ occurs in $r = \lambda(v-1)/(k-1)$ blocks.*

Table 2 Comparison of Fisher's inequality formalizations. Non-blank line counts are from the AFP source [8] for Isabelle and from `FisherInequality.lean` and `MatrixLemmas.lean` for Lean. Isabelle counts include only the uniform (BIBD) version and its direct prerequisites.

Aspect	Isabelle (BIBD)	Lean (nontrivialRPBD)
Assumption	(v, k, λ)-BIBD	nontrivialRPBD
Result	$v \leq b$	$v \leq \text{rank}(M) \leq b$
Field type	`real` (after lifting)	polymorphic α
$MM^\top$ structure	integer matrix, locale	polymorphic, `rpbdCondition`
Det/rank method	row/column ops (129 lines)	matrix det. lemma (20 lines)
Type transfer	`lift_01_mat`, hom. lemmas	unnecessary
$\lambda < r$	from BIBD parameters	separate proof (12 lines)
Ordering setup	`ordered_incidence_system`	unnecessary
Main theorem	17 lines	24 lines

Proof sketch. Fix $x \in X$ and count $S = \{(y, B) \mid x \neq y,\ x \in B,\ y \in B\}$ in two ways: by choosing y first, $\#S = \lambda(v-1)$; by choosing B first, $\#S = r(k-1)$. ◀

The value r is called the *replication number* of the BIBD. Similar double counting establishes:

▶ **Theorem 5.** *Any (v, k, λ)-BIBD with b blocks and replication number r satisfies $bk = vr$.*

Proof. Count $S = \{(x, B) \mid x \in B\} \subseteq X \times \mathcal{B}$ in two ways. There are v ways to choose x, then r blocks containing x, giving $\#S = vr$. There are b ways to choose B, then k elements in B, giving $\#S = bk$. ◀

The key ingredient in both proofs is the following counting principle, which formalizes the idea of "counting a set in two ways."

▶ **Theorem 6** (Counting principle). *Let α, β be finite types and P, Q predicates on α and $\alpha \times \beta$ respectively. Suppose there are n elements x satisfying $P(x)$, and for each such x, there are k elements y satisfying $Q(x, y)$. Then*

$$\#\{(x, y) \mid P(x) \wedge Q(x, y)\} = kn.$$

In Lean, the statement is:

```
theorem card_dependent {α β : Type*} [Fintype α] [Fintype β]
    (P : α → Prop) [DecidablePred P]
    (Q : α → β → Prop) [∀ x, DecidablePred (Q x)]
    {k : ℕ} (hk : ∀ x, P x → #{y | Q x y} = k) :
    #{(x, y) | P x ∧ Q x y} = k * #{x | P x} := by...
```

The formalized proof constructs an explicit bijection from the dependent product to a Cartesian product; the details are routine but require `DecidablePred P` instances for Lean's finite set machinery. To derive Theorem 4, we apply the counting principle twice with different predicate choices and equate the results via `card_of_swap`.

3.2 Incidence Matrices

The *incidence matrix* of a design $\Phi = (X, \mathcal{B})$ with $X = \{x_1, \ldots, x_v\}$ and $\mathcal{B} = [B_1, \ldots, B_b]$ is the $v \times b$ matrix $M(\Phi)$ where $M(\Phi)_{i,j} = 1$ if $x_i \in B_j$ and 0 otherwise. In Lean:

```
def toIncMat (α : Type*) [One α] [Zero α] (Φ : Design ι X) :
    Matrix X ι α := of (fun x i ↦ if x ∈ Φ.blocks i then 1 else 0)
```

Note that the definition of `Matrix` in Mathlib allows arbitrary size for matrices, therefore, even if ι and `X` are infinite, the incidence matrix definition still makes sense.

▶ **Example 7.** The incidence matrix of the design described in Example 1 is

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}.$$

The type `Matrix X ι α` is a synonym for `X → ι → α`: rows are indexed by points, columns by block indices, and entries have type α, which will typically be a commutative ring or a field.

Conversely, any zero-one matrix gives rise to a design:

```
def fromIncMat (α : Type*) [DecidableEq α] [One α] (M : Matrix X ι α) :
  Design ι X where blocks := fun i ↦ {x | M x i = 1}
```

3.3 Algebraic Characterization of RPBDs

Results about designs tend to rely on algebraic properties of their incidence matrices. The proofs we present below were chosen for the ease with which they can be formalized; in several cases they differ from the textbook proofs we followed [19], and we found the matrix-based arguments to be shorter and more natural in a proof assistant.

Crucially, we have the following algebraic characterization:

▶ **Theorem 8.** *A zero-one matrix M is the incidence matrix of an RPBD with balance number λ and replication number r if and only if $MM^\top = \lambda J + (r-\lambda)I$, where J is the all-ones matrix and I is the identity matrix.*

Proof. The entry $(MM^\top)_{i,j}$ counts blocks containing both x_i and x_j, which equals r when $i = j$ and λ when $i \neq j$, giving the forward direction. The converse follows by reading off regularity and balance from the diagonal and off-diagonal entries of the matrix equation. ◀

In Lean, we express this as a predicate:

```
def rpbdCondition (α : Type*) [Ring α][DecidableEq m] (l r : α)
    (M : Matrix m n α) : Prop :=
  isZeroOne M ∧ M * Mᵀ = l • (allOnes _ _ _) + (r - l) • 1
```

Theorem 8 enables a shift in proof strategy: once $MM^\top = \lambda J + (r-\lambda)I$ is established, we can reason about designs using matrix algebra rather than combinatorial counting. This shift is very productive in a proof assistant, since Mathlib's infrastructure for matrix multiplication and linear algebra makes such arguments concise.

3.4 Fisher's Inequality

Fisher's inequality states that in any nontrivial RPBD the number of blocks is at least the number of points. The proof proceeds by showing that the incidence matrix has full rank.

We first establish a key lemma: in any nontrivial RPBD, $\lambda < r$. This is proved by exhibiting a point x that belongs to a nontrivial block B (one with $0 < \#B < \#X$) and a point $y \notin B$. Then

$$\lambda = \#\{i \mid \{x, y\} \subseteq \Phi.\text{blocks } i\} < \#\{i \mid x \in \Phi.\text{blocks } i\} = r,$$

since every block containing both x and y also contains x, but B contains x without containing y. In Lean:

```
theorem l_lt_r_of_nontrivialRPBD (Φ : nontrivialRPBD ι X l r) : l < r := by...
```

This inequality ensures that the matrix $\lambda J + (r - \lambda)I$ is nonsingular, which is the essential step.

▶ **Theorem 9** (Fisher's Inequality). *Let $\Phi = (X, \mathcal{B})$ be a nontrivial RPBD. Then $\#X \leq \#\mathcal{B}$.*

Proof. Let M be the incidence matrix of Φ. By Theorem 8, $MM^\top = \lambda J + (r - \lambda)I$. Since $\lambda < r$, the matrix determinant lemma, which is already in Mathlib as

```
Matrix.det_one_add_replicateCol_mul_replicateRow
```

shows that $\det(\lambda J + (r - \lambda)I) \neq 0$, so $\operatorname{rank}(MM^\top) = \#X$.

Then $\#X = \operatorname{rank}(MM^\top) \leq \operatorname{rank}(M) \leq \#\mathcal{B}$. ◀

Our Lean statement captures both the rank bound and the cardinality bound, since the full-rank property is needed later for the Bruck–Ryser–Chowla theorem:

```
theorem b_ge_rank_ge_v_of_nontrivialRPBD (α : Type*) [Field α]
    [LinearOrder α] [IsStrictOrderedRing α]
    (Φ : nontrivialRPBD ι X l r) : let M := toIncMat α Φ.toDesign;
    (Fintype.card ι) ≥ rank M ∧ rank M ≥ (Fintype.card X) := by...
```

When $\#X = \#\mathcal{B}$ (the symmetric case), this immediately gives $\operatorname{rank}(M) = \#X$, which we record as the lemma `rank_eq_v_of_symmNontrivialRPBD`. This full-rank property is a key ingredient in the proofs that follow.

3.5 Symmetric BIBDs

A *symmetric* BIBD is a (v, k, λ)-BIBD with $\#X = \#\mathcal{B}$, i.e., with as many blocks as points. By Theorem 5, this is equivalent to $k = r$. The Fano plane (Example 2) is an example. The incidence matrix of a symmetric BIBD is square (but not necessarily symmetric as a matrix). Since it has full rank by Fisher's inequality, it is invertible.

▶ **Theorem 10** (Block intersection property). *Let Φ be a symmetric (v, k, λ)-BIBD. Then for any pair of distinct blocks B_i and B_j, $\#(B_i \cap B_j) = \lambda$.*

Proof. Let M be the incidence matrix. Note that $(M^\top M)_{i,j} = \#(B_i \cap B_j)$. Since M is the incidence matrix of a symmetric BIBD, we have $MJ = JM$ (both sides have all entries equal to k). By Theorem 8, $MM^\top M = (\lambda J + (r - \lambda)I)M = M(\lambda J + (r - \lambda)I)$. Since M is invertible, $M^\top M = \lambda J + (r - \lambda)I$, which gives the result. ◀

▶ **Theorem 11.** *Any nontrivial RPBD $(X, \mathcal{B})$ with $\#X = \#\mathcal{B}$ is a BIBD.*

Proof. Let M be the incidence matrix, λ the balance number, and r the replication number. It suffices to show that every block has the same size. Since $M^\top \mathbf{1} = (\#B_1, \ldots, \#B_b)^\top$ and $M\mathbf{1} = r\mathbf{1}$, we get $MM^\top \mathbf{1} = (\lambda v + r - \lambda)\mathbf{1}$. Invertibility of M then forces $M^\top \mathbf{1} = \frac{\lambda v + r - \lambda}{r}\mathbf{1}$, so every block has size $(\lambda v + r - \lambda)/r$. ◀

▶ Remark 12. The proofs of Theorem 10 and Theorem 11 differ from the textbook proofs in Stinson [19]. Stinson's proofs manipulate double summations involving row vectors of $M^\top$. We found it difficult to directly translate these arguments into Lean, while matrix manipulations are significantly more manageable. Our proofs use the invertibility of M in place of the linear independence arguments implicit in Stinson's approach, resulting in formalizations that are not only shorter but arguably cleaner on paper.

3.6 Constructions on Designs

Given some existing designs, how can we put them together to form a new one? There are many known constructions, including sum, dual, complement, residual, derived, and difference sets, to name the few that we formalized in `Construction.lean`. Section 3.7 describes a very general, group-theoretic construction of t-designs. Our purpose here is to highlight just a couple of these constructions and how our design hierarchy sheds light on their structure.

Given designs $\Phi_1 = (X, \mathcal{B}_1)$ and $\Phi_2 = (X, \mathcal{B}_2)$, their *sum* is their disjoint union $\Phi = (X, \mathcal{B}_1 \sqcup \mathcal{B}_2)$. In Lean, we have:

```
def Design.sum (Φ₁ : Design ι₁ X) (Φ₂ : Design ι₂ X) : Design (ι₁ ⊕ ι₂) X where
  blocks := Sum.elim Φ₁.blocks Φ₂.blocks
```

Moreover, it is straightforward to see that various levels of the design hierarchy are closed under the sum construction. For instance, the sum of two block designs with a common block size k is once again a block design with block size k. Similarly, the sum of a (v, k, λ_1)-BIBD and a (v, k, λ_2)-BIBD is a $(v, k, \lambda_1 + \lambda_2)$-BIBD. Our design hierarchy gives us a clean vocabulary in which to express these closure properties:

```
def BlockDesign.sum (Φ₁ : BlockDesign ι₁ X k) (Φ₂ : BlockDesign ι₂ X k) :
    BlockDesign (ι₁ ⊕ ι₂) X k where
  toDesign := Design.sum Φ₁.toDesign Φ₂.toDesign
  uniform := by...
def BIBD.sum (Φ₁ : BIBD ι₁ X k l₁) (Φ₂ : BIBD ι₂ X k l₂) :
    BIBD (ι₁ ⊕ ι₂) X k (l₁ + l₂) where...
-- analogous defs for other levels of the hierarchy
```

The *dual* of a design Φ is the design whose incidence matrix is $M(\Phi)^\top$:

```
def Design.dual (α : Type*) [DecidableEq α] [One α] [Zero α] (Φ : Design ι X) :
    Design X ι := fromIncMat α (toIncMat α Φ)ᵀ
```

Unlike the case for sums, levels of the design hierarchy are not in general closed under taking duals. For instance, let Φ be a (v, k, λ)-BIBD with replication number r. Let Φ' be the dual of Φ. Since every block of Φ has size k, every point in Φ' occurs in k blocks (the points and blocks "switch roles" going from Φ to Φ'). Likewise, since every point in Φ occurs in r blocks, every block of Φ' has r points. Therefore Φ' is a regular block design. However, it can be shown that Φ' need not in general be balanced. That is, the regular block design is the highest level of the design hierarchy in which we can place the dual of a BIBD:

```
def BIBD.dual (Φ : BIBD ι X k l) [Inhabited X] :
    RegularBlockDesign X ι (rep Φ) k where...
```

3.7 Kramer–Mesner Theorem

The Kramer–Mesner theorem [14] provides a method for constructing t-designs using group actions. Let G be a finite group acting on a finite set X of v points. The action of G extends naturally to finite subsets of X: for $g \in G$ and $S \subseteq X$, we have $g \cdot S = \{g \cdot x : x \in S\}$. This partitions the k-subsets of X into orbits $\mathcal{K}_1, \ldots, \mathcal{K}_n$ and the t-subsets into orbits $\mathcal{T}_1, \ldots, \mathcal{T}_m$ under G.

Define the *Kramer–Mesner matrix* $A_{k,t} = (a_{ij})$ where

$$a_{ij} = \#\{K \in \mathcal{K}_i : T \subseteq K\}$$

for any fixed representative $T \in \mathcal{T}_j$. This count is independent of the choice of representative, so $A_{k,t}$ is well defined. A t-(v, k, λ) design $\mathcal{D} = (X, \mathcal{B})$ is called *G-invariant* if the block collection $\mathcal{B}$ is closed under the action of G, that is, for every block $B \in \mathcal{B}$ and every $g \in G$, the set $g \cdot B$ is also a block in $\mathcal{B}$. Equivalently, $\mathcal{B}$ is a union of complete orbits of k-subsets under G.

▶ **Theorem 13** (Kramer–Mesner). *A G-invariant t-(v, k, λ) design exists if and only if there is a nonnegative integer vector z such that*

$$z\, A_{k,t} = \lambda\, \mathbf{u}, \tag{1}$$

where $\mathbf{u}$ denotes the all-ones row vector of length m.

The Kramer–Mesner theorem reduces the existence question for t-designs to finding a nonnegative integer solution to a linear system. By working with orbits under G rather than all subsets, the matrix $A_{k,t}$ can be dramatically smaller, making otherwise infeasible searches practical; this approach has led to the discovery of the first simple (no repeated blocks) 7-designs [2] and, recently, new Steiner systems $S(2, 6, 91)$ [13]

Our formalization certifies both directions of the equivalence for general t-designs, which means not only that a solution to the linear system yields a valid design (forward direction), but also that every G-invariant design arises from some solution (converse), so no designs are missed by the method. Since Lean is simultaneously a programming language and a proof assistant, this opens the door to building a verified design construction tool: one could implement the orbit enumeration, assemble the Kramer–Mesner matrix, solve the integer linear system, and extract a certified t-design, all within a single environment where each step carries a machine-checked correctness guarantee.

The formal statement in Lean is:

```
theorem kramerMesner_iff (t k l : ℕ) (hvk' : k < Fintype.card X ∧ t ≤ k) :
    (∃ (ι : Type u) (_ : Fintype ι) (_ : DecidableEq ι)
    (D : TDesign ι X t k l), IsGInvariant ι X G D.toDesign) ↔
    (∃ z : Matrix (Fin 1) {O : jOrbits X G // orbitCard O = k} ℤ,
      (∀ O, 0 ≤ z 0 O) ∧ KramerMesnerCondition X G t k l z) := by...
```

where `KramerMesnerCondition` is Equation (1). The BIBD case is recovered as a direct corollary by setting $t = 2$.

Proof sketch. The forward direction extracts a solution from a G-invariant design: the solution vector z records the multiplicity of each orbit in the block collection. The key step is showing that G-invariance ensures a well-defined orbit decomposition.

The converse constructs a design from a solution vector z. For each orbit $\mathcal{K}_j$ of k-subsets, take z_j copies and form their multiset union. The resulting block collection has uniform block size k by construction. The balance property follows from the Kramer–Mesner equation: for any t-subset T in orbit $\mathcal{T}_i$, the number of blocks containing T is $\sum_j z_j \cdot a_{ij} = \lambda$. The constructed design is automatically G-invariant since the block collection is a union of complete orbits. ◀

▶ Remark 14. The Kramer–Mesner proof is where we found it most natural to pass through the multiset view of designs. The construction builds a block collection as a multiset from the solution vector, and a key lemma `card_filter_multiset_eq_card_filter_indexed` bridges the two representations by equating filtered multiset cardinalities with index counts. This illustrates why the indexed and multiset representations are complementary rather than competing: each is better suited to certain parts of the proof.

4 The Bruck–Ryser–Chowla Theorem

While Kramer–Mesner supplies a powerful method for constructing designs, the Bruck–Ryser–Chowla theorem (BRC) places strong limitations on the parameter sets for which symmetric designs can exist. For example, is there a symmetric $(22, 7, 2)$-BIBD? A symmetric $(43, 7, 1)$-BIBD? While our structural results so far are not strong enough to decide these questions, we will see that BRC rules them out on purely number theoretic grounds.

▶ **Theorem 15** (Bruck–Ryser–Chowla). *Suppose there exists a symmetric (v, k, λ)-BIBD.*

- *If v is even, then $k - \lambda$ is a perfect square.*
- *If v is odd, then there exists a nontrivial integer solution (x, y, z) to the diophantine equation $x^2 = (k - \lambda)y^2 + (-1)^{(v-1)/2}\lambda z^2$.*

The formal statement is:

```
theorem bruck_ryser_chowla_even [Inhabited X]
    (hv : Even (Fintype.card X)) (Φ : BIBD X X k l) : IsSquare (k - l) := by...

theorem bruck_ryser_chowla_odd [Inhabited X] {u : ℕ}
    (hv : Fintype.card X = 2 * u + 1) (Φ : BIBD X X k l) :
    ∃ x y z : ℤ,
    (x ≠ 0 ∨ y ≠ 0 ∨ z ≠ 0) ∧ x^2 = (k - l) * y^2 + (-1)^u * l * z^2 := by...
```

The proof of the even case relies on a determinant calculation, whose statement in natural language and Lean are as follows.

▶ **Theorem 16.** *Let $\mathbb{F}$ be a field and $a, b \in \mathbb{F}$ with $b \neq 0$. Then*

$$\det(aJ + bI) = b^n \left(1 + \frac{an}{b}\right),$$

where $J, I \in \mathbb{F}^{n\times n}$ are the all-ones and identity matrices as usual.

```
theorem det_ones_add_diagonal {α : Type*} (n : Type*)
    [Field α] [Fintype n] [DecidableEq n] (a b : α) (hb : b ≠ 0) :
    det (a • allOnes n n α + b • 1)
      = b ^ Fintype.card n * (1 + a / b * Fintype.card n) := by...
```

This lemma leads to a straightforward proof of the even v case of BRC.

Proof (Theorem 15, v even). Let Φ be a symmetric (v, k, λ)-BIBD. Let r be the replication number of Φ, and M its incidence matrix. Compute

$$\begin{aligned}\det(M^2) = \det(MM^\top) &= \det(\lambda J + (r - \lambda)I) \\ &= (k + (v - 1)\lambda)(k - \lambda)^{v-1} = k^2(k - \lambda)^{v-1},\end{aligned}$$

where we used Theorem 16 to evaluate the determinant and Theorem 4 for the last equality. Since M is an integer matrix, $\det(M)$ is an integer, so the result follows. ◀

▶ **Example 17.** The theorem `no_22_7_2_BIBD` shows that no $(22, 7, 2)$-BIBD exists. The proof computes $r = 7$ from the replication number formula, deduces $b = v = 22$ from $bk = vr$, and applies `bruck_ryser_chowla_even` to obtain $n^2 = 5$ for some natural number n. Since $4^2 > 5$, we have $n \leq 3$, and the `interval_cases` tactic automatically discharges each case $n = 0, 1, 2, 3$.

The odd v case of BRC has a considerably more interesting proof and formalization. Before coming to that, we take a detour through the theory of quadratic forms and the main ingredients of the proof.

Let $\mathbb{F}$ be a field and $M \in \mathbb{F}^{n\times n}$ a matrix. Then M determines a quadratic form $p_M : \mathbb{F}^n \to \mathbb{F}$ via $p_M(x) = \langle x, Mx\rangle$. The characterization of equivalence of quadratic forms in terms of their matrix representations is given by the notion of matrix congruence. Given matrices $M, N \in \mathbb{F}^{n\times n}$, we say that M and N are *congruent*, written $M \sim N$, if there is an invertible matrix $Q \in \mathbb{F}^{n\times n}$ such that $M = QNQ^\top$. Note that $\sim$ is an equivalence relation on $\mathbb{F}^{n\times n}$. The following theorem is easy to prove.

▶ **Theorem 18.** *Let $\mathbb{F}$ be a field with $\mathrm{char}(\mathbb{F}) \neq 2$, and let $M, N \in \mathbb{F}^{n\times n}$ be symmetric matrices. Then $M \sim N$ if and only if their associated quadratic forms p_M, p_N are equivalent.*

▶ Remark 19. The forward direction ($M \sim N \Rightarrow p_M \simeq p_N$) holds for arbitrary matrices over any field: if $M = A^\top NA$ with A invertible, then $x^\top Mx = (Ax)^\top N(Ax)$ for all x. The two hypotheses are only needed for the converse. Symmetry ensures that a quadratic form uniquely determines its matrix (otherwise M and $\frac{1}{2}(M + M^\top)$ give the same form but need not be congruent), and $\mathrm{char}(\mathbb{F}) \neq 2$ is required to recover the symmetric bilinear form from the quadratic form via polarization.

In Lean, we formulated matrix congruence as follows.

```
structure MatCongr {n α : Type*} [Fintype n] [DecidableEq n] [CommRing α]
    (P N : Matrix n n α) where
  A : Matrix n n α
  inv : Invertible A
  cong : P = A * N * Aᵀ
infixl:55 " ∼ₘ " => MatCongr
```

The last line in the above definition allows us to express matrix congruence in infix notation, just as we would in natural language (the m subscript is there to prevent notation clashes). Then the two directions of Theorem 18 are stated as:

```
def MatCongr_toIsometryEquiv {M N : Matrix m m α}
    (h : M ∼ₘ N) : M.toQuadraticMap'.IsometryEquiv N.toQuadraticMap' := by...

def IsometryEquiv_toMatCongr [Invertible (2 : α)] {M N : Matrix m m α}
    (hM : M.IsSymm) (hN : N.IsSymm)
    (e : M.toQuadraticMap'.IsometryEquiv N.toQuadraticMap') : M ∼ₘ N := by...
```

The other crucial ingredient in the proof is the direct sum of matrices and the Witt cancellation theorem. The *direct sum* of $M \in \mathbb{F}^{m\times m}$ and $N \in \mathbb{F}^{n\times n}$, written $M \oplus N$, is the block matrix

$$M \oplus N = \begin{pmatrix} M & 0 \\ 0 & N \end{pmatrix} \in \mathbb{F}^{(m+n)\times(m+n)}.$$

This definition is conveniently expressed in Lean using `Matrix.fromBlocks` from Mathlib as:

```
abbrev matDirectSum (M : Matrix m m α) (N : Matrix n n α) :
    Matrix (m ⊕ n) (m ⊕ n) α := fromBlocks M 0 0 N
infixl:60 " ⊕ₘ " => matDirectSum
```

So far, it seems that our Lean definitions have been rather uninteresting, literal translations of their natural language counterparts. But here, an unexpected subtlety arises in connection with associativity of $\oplus$. Fix square matrices M, N, P. On paper, we would not hesitate to write down

$$(M \oplus N) \oplus P \sim M \oplus (N \oplus P).$$

Indeed, these are equal as matrices, thus a fortiori congruent. Notice, however, that we cannot even express this in Lean! We are only able to speak of matrix congruence between matrices of the same type. But, if `M : Matrix m m α`, `N : Matrix n n α`, and `P : Matrix p p α`, then the types of the left-hand and right-hand expressions are $(m \oplus n) \oplus p$ and $m \oplus (n \oplus p)$. These types are isomorphic, but not literally the same.

The inability of our definition of matrix congruence to accommodate "obvious" congruences between matrices of different types seems to be a major shortcoming. It is natural to attempt a *heterogeneous* definition of congruence, such as:

```
structure hMatCongr (N : Matrix n n α) (M : Matrix m m α) extends n ≃ m where
    A : Matrix m m α
    inv : Invertible A
    cong : reindex toEquiv toEquiv N = A * M * Aᵀ
```

This definition faithfully captures the notion of matrix congruence in the following sense: if M and N are heterogeneously congruent matrices over the *same* index type, then $M \sim N$ (this is formalized in `MatrixCongruence.lean`). Roughly, the reason is that reindexing a matrix M by a permutation σ of its indices gives $P_\sigma^\top M P_\sigma$, where P_σ is the permutation matrix associated with σ.

Although the definition `hMatCongr` is correct, we found it difficult to work with. For instance, one of the basic properties of matrix congruence is transitivity. The proof is one line for `MatCongr`. In contrast, in the case of `hMatCongr`, the shortest proof we found is already a mouthful, making use of the fact that an equivalence of types $m \simeq n$ induces an equivalence of `Matrix m m α` and `Matrix n n α` as vector spaces over α, available in Mathlib as `Matrix.reindexAlgEquiv`.

In general, there is a tradeoff between having to contend with reindexings everywhere and losing the ability to express the proof of BRC as naturally as possible. We found that the former difficulty outweighs the latter. Accordingly, we adopt the original definition `matCongr` and sacrifice the ability to directly express associativity of $\oplus$. Instead, we work with the following version of associativity:

```
def matCongrAssocOfMatCongr {M' : Matrix m m α} {O' : Matrix o o α}
    (h : M ⊕ₘ N ⊕ₘ O ∼ₘ M' ⊕ₘ N' ⊕ₘ O') :
    M ⊕ₘ (N ⊕ₘ O) ∼ₘ M' ⊕ₘ (N' ⊕ₘ O') := by...
```

This lemma allows us to make all the associativity arguments we need, just not as one single line of equational reasoning as we might have wanted. We now state the last two ingredients of the proof of BRC.

▶ **Theorem 20.** *Let n be a multiple of 4 and $\mathbb{F}$ a field of characteristic 0. Then for any positive integer m, we have $mI \sim I$, where $I \in \mathbb{F}^{n \times n}$ is the identity.*

▶ **Theorem 21** (Witt Cancellation)**.** *Let M, N, P be symmetric matrices over a field $\mathbb{F}$ in which $2 \neq 0$. Suppose $M \oplus N \sim M \oplus P$. Then $N \sim P$.*

In Lean, the statements of these results are as follows:

```
noncomputable def matCongrOneOfFourDiv [CharZero α] (hn : 4 ∣ Fintype.card n)
    {m : ℤ} (mpos : 0 < m) :
    (m : α) • (1 : Matrix n n α) ≈ₘ (1 : Matrix n n α) := by...

noncomputable def oplusLeftCancel [Invertible (2 : α)]
    (hN : IsSymm N) (hP : IsSymm P)
    (h : M ⊕ₘ N ≈ₘ M ⊕ₘ P) : N ≈ₘ P := by...
```

The proof of Theorem 20 relies on Lagrange's four square theorem, available in Mathlib as `Nat.sum_four_squares`.

The proof of Witt cancellation [15] requires more explanation, as we chose to work at the level of quadratic forms rather than matrices. The classical statement is naturally about quadratic forms: if $Q \perp Q_1$ and $Q \perp Q_2$ are isometric over a field $\mathbb{F}$ with $\operatorname{char} \mathbb{F} \neq 2$, then Q_1 and Q_2 are isometric. Our proof proceeds by induction on the dimension of Q. The key inductive step shows that if two nondegenerate quadratic forms take the same nonzero value at some vector, there exists an isometry sending one vector to the other; this is established by composing at most two reflections, built using Mathlib's `Module.reflection`, and relying on the identity $Q(x+y)+Q(x-y) = 2(Q(x)+Q(y))$. A crucial ingredient is the Mathlib result `QuadraticForm.equivalent_weightedSumSquares`, which states that every quadratic form over a field (with $2 \neq 0$) is isometric to a weighted sum of squares $\sum_i w_i x_i^2$. We decompose the common summand Q into this form and cancel one rank-one component at a time.

The decision to work with quadratic forms was informed by practical experience. Our original plan, outlined in Section 2.1, was to prove cancellation directly at the matrix level, which motivated the development of `Diagonalizable.lean`. However, that approach became unwieldy: threading invertibility hypotheses through some congruence step and managing block matrix bookkeeping added substantial overhead. The quadratic form route turned out to be cleaner, largely because the key diagonalization result was already in Mathlib and because quadratic form isometries (`QuadraticMap.IsometryEquiv`) compose more naturally than matrix congruences. Once cancellation is established for quadratic forms, we transfer it to matrices via the correspondence between symmetric matrices and quadratic forms (`MatCongr_toIsometryEquiv` and `IsometryEquiv_toMatCongr`), yielding `oplusLeftCancel`. We note that our Lean statement of `oplusLeftCancel` does not require the cancelled matrix M to be symmetric, even though the textbook formulation of Theorem 21 assumes all three matrices are. The reason is that any matrix M defines the same quadratic form $x \mapsto x^\top M x$ as its symmetric part $(M + M^\top)/2$, so symmetry of M is immaterial for the quadratic form argument. Symmetry of N and P is still needed because converting the resulting isometry back into a matrix congruence relies on the fact that a symmetric matrix is uniquely determined by its quadratic form.

The tools are now in place to prove the harder case of the Bruck–Ryser–Chowla theorem. We present only a sketch of the proof given in [17] to give a sense of the kind of calculation we formalized.

Proof sketch of Theorem 15, v odd. Suppose $v \equiv 1 \pmod 4$. All matrices in the following calculation have entries in $\mathbb{Q}$, and $I_n \in \mathbb{Q}^{n\times n}$ denotes the $n \times n$ identity:

$$
\begin{aligned}
I_{v-1} \oplus I_1 \oplus -\lambda I_1 &\sim I_v \oplus -\lambda I_1 && (I_{v-1} \oplus I_1 = I_v)\\
&\sim (k-l) I_v \oplus -\frac{k-\lambda}{\lambda} I_1 && (\text{Theorem 8 plus some work})\\
&\sim (k-\lambda) I_{v-1} \oplus (k-\lambda) I_1 \oplus -\frac{k-\lambda}{\lambda} I_1 && (I_{v-1} \oplus I_1 = I_v)\\
&\sim I_{v-1} \oplus (k-\lambda) I_1 \oplus -\frac{k-\lambda}{\lambda} I_1. && (\text{Theorem 20})
\end{aligned}
$$

Then Theorem 21 allows us to cancel the I_{v-1} from both sides, leaving

$$\begin{pmatrix} 1 & 0 \\ 0 & -\lambda \end{pmatrix} \sim \begin{pmatrix} k-\lambda & 0 \\ 0 & -\frac{k-\lambda}{\lambda} \end{pmatrix}.$$

Thus the quadratic forms $p(x,y) = x^2 - \lambda y^2$ and $q(x,y) = (k-\lambda)x^2 - \frac{k-\lambda}{\lambda}y^2$ are equivalent over $\mathbb{Q}$, which implies the conclusion. The case that $v \equiv 3 \pmod 4$ proceeds along similar lines. ◀

▶ Remark 22. The proof we formalize follows Ryser [17] rather than the textbook proof in Stinson [19]. Stinson works with indeterminates: he introduces a 4×4 matrix C satisfying $CC^\top = (k-\lambda)I_4$ via Lagrange's four-square decomposition, performs an explicit change of variables in groups of four, and then eliminates the linear forms $L_1, \ldots, L_{v-1}$ one at a time. Formalizing this would require managing v-dependent sequential substitutions and tracking rational dependencies at each stage. Ryser's proof replaces the entire elimination with a single application of Theorem 20 and then Theorem 21, which is better suited to our matrix congruence infrastructure. The core of the odd case formalization is the lemma `brcKey`, which shows that the diagonal matrices $\mathrm{diag}(1, \ldots, 1, -\lambda)$ and $\mathrm{diag}(k-\lambda, \ldots, k-\lambda, -(k-\lambda)/\lambda)$ of order $v+1$ are congruent over $\mathbb{Q}$, using the bordered matrix A' (obtained by appending an all-ones column and a row of k/λ to the incidence matrix) as the witnessing matrix. Invertibility of A' is established by a determinant argument. The two sub-cases ($v \equiv 1$ and $v \equiv 3 \pmod 4$) share this lemma and diverge only in how the direct sum is regrouped to expose a block whose dimension is divisible by 4, so that Theorem 20 applies.

A second point concerns the passage from the 2×2 matrix congruence to the integer solution. Both Ryser and Stinson treat this as immediate, but in the formalization it requires explicit work: `matCongr_two_by_two_condition` extracts rational solutions from the congruence, and we then clear denominators by writing $x = p/q$, $z = r/s$ and multiplying through by $(qs)^2$ to obtain integer solutions. This accounts for roughly 15 lines in each parity case.

5 Conclusion and Future Work

We have presented a formalization of combinatorial design theory in Lean 4, including Fisher's inequality, the Kramer–Mesner theorem for general t-designs, and the Bruck–Ryser–Chowla theorem for symmetric BIBDs. Reaching these results required infrastructure for matrix congruence and Witt's cancellation theorem that is of independent interest for Mathlib's linear algebra library.

The library is BIBD-centric, but supports natural extensions. The most immediate is to broaden the hierarchy with resolvable designs, Latin squares, Steiner triple systems, and group divisible designs. A second target is projective planes: an $(n^2+n+1, n+1, 1)$-BIBD with $n \geq 2$ is a projective plane of order n. Mathlib already contains an axiomatic treatment of projective planes as configurations[2], but no existence or nonexistence results. Connecting our BIBD hierarchy to these definitions would let Fisher's inequality, the block intersection property, and BRC flow automatically to projective planes, immediately ruling out orders $n = 6, 14, 21, 22, 30$.

[2] `https://leanprover-community.github.io/mathlib4_docs/Mathlib/Combinatorics/Configuration.html`

More broadly, combinatorial designs connect to coding theory [12, 11], cryptography [20], and statistical experiment design [10, 4]; in each setting, our formalized parameter constraints can serve as verified feasibility checks. On the computational side, the Kramer–Mesner theorem provides the theoretical backbone for a verified design construction tool within Lean's combined programming and proving environment.

A recurring lesson from this project is that the right algebraic perspective can make combinatorial arguments substantially easier to formalize. In particular, the proofs of the block intersection property (Remark 12) and BRC (Section 4) replaced counting arguments with matrix-based reasoning that was not only easier to formalize but arguably cleaner than the textbook counterparts. We found Mathlib's linear algebra infrastructure to be a major asset, while the main friction points were in finite set and multiset reasoning, where automation lags behind.

References

1 Aaron Anderson, Mantas Bakšys, Jonas Bayer, Mauricio Collares, Rémy Degenne, Yaël Dillies, Ben Eltschig, Sébastien Gouëzel, Kalle Kytölä, Rob Lewis, Paul Lez, Luccioli Lorenzo, Heather Macbeth, Patrick Massot, Arend Mellendijk, Kyle Miller, Pietro Monticone, Kim Morrison, Oliver Nash, Utensil Song, Terence Tao, Floris van Doorn, Sky Wilshaw, and Lawrence Wu. Formalization of the Polynomial Freiman-Ruzsa Conjecture of Marton, November 2023. URL: https://github.com/teorth/pfr.

2 Anton Betten, Adalbert Kerber, Axel Kohnert, Reinhard Laue, and Alfred Wassermann. The discovery of simple 7-designs with automorphism group pγl(2, 32). In Gérard Cohen, Marc Giusti, and Teo Mora, editors, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pages 131–145, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg. doi:10.1007/3-540-60114-7_10.

3 Kevin Buzzard and Richard Taylor. The fermat's last theorem project, 2024. URL: https://leanprover-community.github.io/blog/posts/FLT-announcement/.

4 William G. Cochran and Gertrude M. Cox. *Experimental Designs*. John Wiley & Sons, New York, 2nd edition, 1957.

5 Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language. In *28th International Conference on Automated Deduction*, pages 625–635. Springer International Publishing, 2021. doi:10.1007/978-3-030-79876-5_37.

6 Chelsea Edmonds and Lawrence C. Paulson. Combinatorial design theory. *Archive of Formal Proofs*, 2021. , Formal proof development. URL: https://isa-afp.org/entries/Design_Theory.html.

7 Chelsea Edmonds and Lawrence C. Paulson. A modular first formalisation of combinatorial design theory. In Fairouz Kamareddine and Claudio Sacerdoti Coen, editors, *Intelligent Computer Mathematics*, pages 3–18, Cham, 2021. Springer International Publishing. doi:10.1007/978-3-030-81097-9_1.

8 Chelsea Edmonds and Lawrence C. Paulson. Fisher's inequality: Linear algebraic proof techniques for combinatorics. *Arch. Formal Proofs*, 2022, 2022. URL: https://www.isa-afp.org/entries/Fishers_Inequality.html.

9 Chelsea Louise Edmonds. *Formalising Combinatorial Structures and Proof Techniques in Isabelle/HOL*. PhD thesis, University of Cambridge, September 2023.

10 Ronald A. Fisher. *The Design of Experiments*. Oliver and Boyd, Edinburgh, 1935.

11 Sarah J. Johnson and Steven R. Weller. A family of irregular LDPC codes with low encoding complexity. *IEEE Communications Letters*, 7(2):79–81, 2003. doi:10.1109/LCOMM.2002.808375.

12 E. F. Assmus Jr. and J. D. Key. *Designs and Their Codes*, volume 103 of *Cambridge Tracts in Mathematics*. Cambridge University Press, 1992.

13 Michael Kiermaier, Vedran Krčadinac, Vladimir D. Tonchev, Renata Vlahović Kruc, and Alfred Wassermann. Some new Steiner designs $S(2, 6, 91)$. *Journal of Algebraic Combinatorics*, 62:article no. 38, 2025. doi:10.1007/s10801-025-01468-6.

14 Earl S. Kramer and Dale M. Mesner. t-designs on hypergraphs. *Discrete Mathematics*, 15(3):263–296, 1976. doi:10.1016/0012-365X(76)90030-3.

15 Tsit-Yuen Lam. *Introduction to Quadratic Forms over Fields*, volume 67 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 2005.

16 The mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*. Association for Computing Machinery, January 2020. doi:10.1145/3372885.3373824.

17 Herbert John Ryser. The existence of symmetric block designs. *Journal of Combinatorial Theory, Series A*, 32:103–105, 1982. doi:10.1016/0097-3165(82)90068-1.

18 D. R. Stinson. Universal hashing and authentication codes. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 74–85, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

19 Douglas R. Stinson. *Combinatorial Designs: Constructions and Analysis*. Springer, New York, 2004.

20 Douglas R. Stinson. Combinatorial designs and cryptography, revisited. In *50 Years of Combinatorics, Graph Theory, and Computing*, pages 335–358. CRC Press, 2020.