Composition Direction of Seymour's Theorem for Regular Matroids — Formally Verified

Anonymous author
Anonymous affiliation

Anonymous author
Anonymous affiliation

Anonymous author
Anonymous affiliation

Anonymous author
Anonymous affiliation

Anonymous author
Anonymous affiliation

Anonymous author
Anonymous affiliation

Anonymous author
Anonymous affiliation

Anonymous author
Anonymous affiliation

Anonymous author
Anonymous affiliation

Anonymous author
Anonymous affiliation

Anonymous author
Anonymous affiliation

Anonymous author
Anonymous affiliation

Abstract

Seymour's decomposition theorem is a hallmark result in matroid theory presenting a structural characterization of the class of regular matroids. Formalization of matroid theory faces many challenges, most importantly that only a limited number of notions and results have been implemented so far. In this work, we formalize the proof of the forward (composition) direction of Seymour's theorem for regular matroids. To this end, we develop a library in Lean 4 that implements definitions and results about totally unimodular matrices, vector matroids, their standard representations, regular matroids, and 1-, 2-, and 3-sums of matrices and binary matroids given by their standard representations. Using this framework, we formally state Seymour's decomposition theorem and implement a formally verified proof of the composition direction in the setting where the matroids have finite rank and may have infinite ground sets.

2012 ACM Subject Classification Mathematics of computing → Matroids and greedoids; General and reference → General conference proceedings; General and reference → Verification

Keywords and phrases totally unimodular matrices, regular matroids, Seymour's decomposition theorem, calculus of inductive constructions

Digital Object Identifier 10.4230/LIPIcs.ITP.2026.75

1 Introduction

Seymour's regular matroid decomposition theorem is a hallmark structural result in matroid theory [9, 12, 4, 7]. It states that, on the one hand, any 1-, 2-, and 3-sum of two regular matroids is regular, and on the other hand, any regular matroid can be decomposed into matroids that are graphic, cographic, or isomorphic to R_{10} by repeated 1-, 2-, and 3-sum decompositions.

The interest in matroids comes from the fact that they capture and generalize many mathematical structures and properties, such as linear independence (captured by vector matroids), graphs (graphic matroids), and extensions of fields (algebraic matroids). Another advantage of matroids is that they admit a relatively short definition, making them amenable to formalization. As for Seymour's theorem, it not only presents a structural characterization of the class of regular matroids, but also leads to several important applications, such as polynomial algorithms for testing if a matroid is binary and for testing if a matrix is totally

17th International Conference on Interactive Theorem Proving (ITP 2026).
Editor: 68; Article No. 75; pp. 75:1–75:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

unimodular. Additionally, Seymour's theorem can offer a structural approach for solving certain combinatorial optimization problems, for example, it leads to the characterization and efficient algorithms for the cycle polytope.

Formalization of results about matroids faces several challenges. One of them is that the support for them is limited. In Mathlib, only selected basic definitions for matroids are implemented, such as maps, duals, and minors. However, many other fundamental notions are not yet implemented, including representability and regularity, the splitter theorem and the separation algorithm. Part of the difficulty stems from the fact that classically, matroids are defined only in the finite case (i.e., when the ground set and the rank are finite), while Mathlib implements matroids more generally, allowing them to be infinite and to have infinite rank. Additionally, the proofs presented in the existing literature require substantial additional work to make them easily amenable to formalization.

The goal of our work was to develop a general and reusable library proving a result that is at least as strong as the forward (composition) direction of classical Seymour's theorem (i.e., stated for finite matroids). Moreover, our aim was to make our library modular and extensible by ensuring compatibility with matroids in Mathlib [8].

To achieve our goals, we made the following compromises. First, we focused on the implementation of the proof of the composition direction, while only stating the decomposition direction. Second, we assumed finiteness where it would simplify proofs, while making sure that the final results held for finite matroids (in fact, they hold for matroids with potentially infinite ground set and finite rank). Finally, we tailored our implementation specifically to Seymour's theorem, avoiding introducing additional matroid notions if possible. Our project makes the following contributions:

- Formalized definition and selected properties of totally unimodular matrices, some of which were added to Mathlib.
- Implemented definitions and formally proved selected results about vector matroids, their standard representations, regular matroids, and 1-, 2-, and 3-sums of matrices and vector matroids given by their standard representations.
- Implemented a formally verified proof of the composition direction of Seymour's theorem, i.e., that any 1-, 2-, and 3-sum of two regular matroids is regular, in the case where the matroids may have infinite ground sets and have finite rank.
- Implemented a formally verified proof that graphic and cographic matroids are regular.
- Stated the decomposition direction of Seymour's theorem, i.e., that any regular matroid of finite rank can be decomposed into graphic matroids, cographic matroids, and matroids isomorphic to R_{10} by repeated 1-, 2-, and 3-sum decompositions.

Our formalization[1] is conceptually split into two parts: "implementation" and "presentation". Implementation is contained in the `Seymour` folder and encompasses all definitions and lemmas used to obtain our results. Presentation is contained in the `Seymour.lean` file, which repeats selected definitions and theorems comprising the key final results of our contribution. Every definition in the "presentation" file is checked to be definitionally equal to its counterpart from the "implementation" using the `recall` or the `example` command. Similarly, we `recall` every theorem presented here and then use the `#guard_msgs in #print axioms` command to check that the implementation of its proof (including the entire dependency tree) depends only on the three axioms `[propext, Classical.choice, Quot.sound]`, which are standard for Lean projects that use classical logic.

[1] link removed for blinded version

We refer to the statements of the final results and the definitions they (transitively) depend on as *trusted code*. The `Seymour.lean` file repeats all nontrivial trusted code, so that the reader can believe [10] our results without having to examine the entire implementation, assuming that the reader also uses the Lean compiler to check that all proofs are correct. Note that basic definitions from Lean and Mathlib are part of the trusted code but are not repeated in `Seymour.lean`, and we let the reader decide whether to blindly trust them or read them as well.

While working on our project, we leveraged the LeanBlueprint[2] tool to help guide our formalization efforts. In particular, we used it to create theoretical blueprints and dependency graphs, which allowed us to get a clearer overview of the results we were formalizing, as well as their dependencies. In our workflow, we first created a write-up encompassing the classical results from [12]. Based on this write-up, we developed a self-contained theoretical blueprint for our formalization by filling in gaps, fleshing out technical details, and sometimes re-working certain proofs. We followed this blueprint during the development of our library, keeping it up to date and turning it into documentation of our code.

We use Lean version 4.18.0 and we import Mathlib library revision aa936c3 (dated 2025-04-01).

We made the code snippets in this paper as faithful to the content of the repository as possible, though we made some omissions. In particular, proofs inside definitions were replaced by the `sorry` keyword in the paper, while the repository contains full implementation.

2 Theory Underpinning the Formalization

There are two classical sources presenting the proof of Seymour's decomposition theorem: [9] and [12], each with their own advantages and disadvantages.

Oxley2011 [9] develops a general theory of matroids and has a broader focus. It introduces many abstract notions and proves many statements about them, and Seymour's theorem and its dependencies are also stated and proved in terms of these abstract notions alongside many other results. The advantages of following [9] would be the higher reusability, generality, and extensibility of the formalization. Indeed, since [9] introduces a lot of foundational notions and results, the resulting implementation could serve as the basis for formalization of many other results from classical matroid theory. Moreover, [9] is more general than [12] in certain aspects, for example, [12] defines 1- and 2-sums only for binary matroids, while their definitions in [9] do not have this restriction. Finally, it seems that the approach to theory of infinite matroids [3] is more closely aligned with the approach of [9] than [12], which might make it easier to generalize formalizations based on the former than the latter to the infinite matroid setting. However, proof formalization following [9] would face many challenges. First, the support for matroids in Mathlib [8] at the time we carried out our project was quite limited. Thus a lot of time would be dedicated to developing low-level definitions and results about them, especially in the infinite matroid setting to ensure compatibility with Mathlib. Second, certain intermediate results could turn out difficult to formally prove. From our experiments, proving the equivalence of multiple characterizations of regular matroids turned out hard to formalize. Finally, [9] leaves many technical steps as exercises for the reader, most crucially leaving out the proof of regularity of 3-sum, and contains many proofs that crucially rely on graph theory which was not supported in Mathlib. This would make it challenging to convert the proofs to their formalized versions.

[2] `https://github.com/PatrickMassot/leanblueprint`

In contrast, Truemper2016 [12] focuses on decomposition and composition of matroids, with Seymour's theorem being one of the most prominent theorems that it builds towards. Truemper2016 [12] more frequently than Oxley2011 [9] utilizes explicit matrix representations in definitions, theorems, and proofs, especially when it comes to 1-, 2-, and 3-sums of regular matroids. Thus, following [12] would require implementing fewer intermediate definitions and results to begin working with Seymour's theorem itself. Moreover, Mathlib's support for matrices and linear independence was more extensive than for matroids, so this would allow us to build upon more things that were already available. However, following the approach of [12] had several important limitations. As mentioned earlier, it would be less general and potentially less amenable to generalization to the infinite matroid setting than [9]. Moreover, faithfully following [12] would mean implementing similar definitions and theorems on several levels of abstraction. More specifically, 1-, 2-, and 3-sums would need to be implemented separately for matrices, binary matroids defined by standard representation matrices, and binary matroids in general, and the results about the sums of these objects would need to be proved and propagated accordingly. Last but not least, similar to [9], one would need to fill in the omitted technical details and re-work proofs that could be extremely challenging to formalize directly, especially those involving graph-theoretic arguments.

Ultimately, we decided to follow the approach of [12] over [9] for formalizing Seymour's theorem, as it aligned more closely with our goals and values. We aimed to formalize the statement of Seymour's theorem and the proof of the composition direction, so having to implement fewer intermediate definitions and lemmas and being able to use more tools from Mathlib was a big plus. Though we did not mind limiting the generality of our contributions to classical results, our final results go beyond that and hold for matroids of finite rank with potentially infinite ground sets. The completeness of the presentation in [12] allowed us to develop a theoretical blueprint, where we fleshed out the technical details, circumvented problematic intermediate results, and streamlined the proofs, especially in the case of 3-sums.

3 Proof Outline and Design Choices

Before delving into technical details, we outline the structure of our formal proof and explain key design decisions. Our development mirrors the theoretical decomposition: we implement each matroid sum (1-, 2-, and 3-sum) at three levels (matrix, standard representation, and abstract matroid) to manage complexity. For each sum, we first prove that the matrix-level construction preserves total unimodularity, then lift this result to the matroid level via the StandardRepr abstraction. The 1-sum and 2-sum proofs closely follow and streamline the arguments in Truemper's work, while the 3-sum case required a new approach. We re-designed the 3-sum proof to avoid formalizing a difficult graph-theoretic re-signing argument: instead, we re-sign each summand only once and introduce an intermediate structure `MatrixLikeSum3` to capture the combined matrix blocks. This strategic design lets us systematically derive total unimodularity for 3-sums (reusing parts of the 2-sum argument) and circumvents the need for a complex graph argument in Lean. We also split our code into an "implementation" (detailed definitions and lemmas) and a "presentation" (key results with Lean's `#guard_msgs` checks), ensuring the proof is both modular and trustworthy.

4 Preliminaries

This section reviews Mathlib declarations our code relies on.

Throughout this paper, we write $\mathbb{Z}_n$ to denote `ZMod n` for any positive integer n, most often in the case $\mathbb{Z}_2$ denoting `ZMod 2`, which is also written as `Z2` in the code.

4.1 Matroids

Matroids have many equivalent definitions [9, 12, 3]. In Mathlib, the structure `Matroid` captures the definition via the *base axioms* from [3]: a *matroid* is a pair $M = (E, \mathcal{B})$ where E is a (potentially infinite) ground set and $\mathcal{B} \subseteq 2^E$ is a collection of sets such that:

(i) $\mathcal{B} \neq \emptyset$.

(ii) For all $B_1, B_2 \in \mathcal{B}$ and all $b_1 \in B_1 \setminus B_2$, there exists $b_2 \in B_2 \setminus B_1$ such that $(B_1 \setminus \{b_1\}) \cup \{b_2\} \in \mathcal{B}$.

(iii) For all $X \subseteq E$ and $I \subseteq X$ such that $I \subseteq B_1$ for some $B_1 \in \mathcal{B}$, there exists a maximal J such that $I \subseteq J \subseteq X$ and $J \subseteq B_2$ for some $B_2 \in \mathcal{B}$.

A set $B \in \mathcal{B}$ is called a *base*, and 2 is known as the *base exchange property*. Additionally, if a set $I \subseteq E$ is a subset of any base, then I is called *independent*. The definition above generalizes the classical notion of matroids [9, 12], which can only have finite ground sets. In our work, we construct matroids using the equivalent *independence axioms*, available in Mathlib as `IndepMatroid`. We use the assumption that the matroid has a finite rank (`RankFinite` in Mathlib). Note that the ground set is allowed to be infinite.

4.2 Totally Unimodular Matrices

In our work, regular matroids are defined in terms of totally unimodular matrices [9, 12]. Before introducing their definition, let us review how matrices and submatrices are implemented in Mathlib. A matrix with rows indexed by `m`, columns indexed by `n`, and entries of type α is represented by `Matrix m n` α, implemented as a (curried [11]) binary function `m → n →` α. Thus, the elements of matrix `A` can be accessed with `A i j`. Similarly, `Matrix.submatrix` is defined so that `(A.submatrix f g) i j = A (f i) (g j)` holds. Note that `Matrix.submatrix` may repeat and reorder rows and columns. For example, if

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad \texttt{f = ![0]}, \quad \texttt{g = ![2, 2, 0, 0]},$$

then `A.submatrix f g` $= \begin{bmatrix} 3 & 3 & 1 & 1 \end{bmatrix}$, typed as a matrix, not a vector.

Now, a matrix A over a commutative ring R is called *totally unimodular* if every finite square submatrix of A (not necessarily contiguous, with no row or column taken twice) has determinant in $\{-1, 0, 1\}$. We implement this definition as follows:

```
def Matrix.IsTotallyUnimodular {m n R : Type*} [CommRing R] (A : Matrix m n R) : Prop :=
  ∀ k : ℕ, ∀ f : Fin k → m, ∀ g : Fin k → n, f.Injective → g.Injective →
    (A.submatrix f g).det ∈ Set.range SignType.cast
```

Here, `SignType` is an inductive type with three values: `zero`, `neg`, and `pos`; and `SignType.cast` maps them to `(0:R)`, `(-1:R)`, and `(1:R)`, respectively.

Note that the indexing functions `f` and `g` are required to be injective in the definition, but this condition can be lifted. Indeed, lemma `Matrix.isTotallyUnimodular_iff` shows that one can equivalently check the determinants of all finite square submatrices, not just ones without repeated rows and columns.

Note that the definition of total unimodularity and lemma `Matrix.isTotallyUnimodular_iff` have been incorporated in Mathlib.

Keep in mind that the determinant is computed over R, so for certain commutative rings, all matrices are trivially totally unimodular, for example, for $R = \mathbb{Z}_3$.

4.3 Types and Subsets

In our project, we often have the following terms in the context:

```
(α : Type) (E : Set α) (I : Set α) (hIE : I ⊆ E)
```

Depending on the situation, there are three ways we may treat the set `I`. First, it may be viewed as a set of elements of type α, its original type, so we simply write `I`. Second, we may need to re-type `I` as a set of elements of the type `E.Elem`. Then we write `E ↓∩ I` using notation from Mathlib. Finally, `I` may be used as a set of elements of the type `I.Elem`. In this case, we write `Set.univ` of the correct type, which is usually inferred from the context.

4.4 Block Matrices

In this project, we often construct matrices by composing them from blocks using the following Mathlib definitions:

- `Matrix.fromRows A₁ A₂` constructs $\begin{array}{|c|}\hline A_1 \\ \hline A_2 \\ \hline \end{array}$
- `Matrix.fromCols A₁ A₂` constructs $\begin{array}{|c|c|}\hline A_1 & A_2 \\ \hline \end{array}$
- `Matrix.fromBlocks A₁₁ A₁₂ A₂₁ A₂₂` constructs $\begin{array}{|c|c|}\hline A_{11} & A_{12} \\ \hline A_{21} & A_{22} \\ \hline \end{array}$

5 Re-typing Matrix Dimensions

When constructing matroids, we often need to convert a block matrix whose blocks are indexed by disjoint sets into a matrix indexed by unions of those index sets. Although the contents of the matrix stay the same, both its dimensions change their type from a `Sum` of sets to a `Set` union of those sets. To this end, we implemented

```
def Subtype.toSum {α : Type*} {X Y : Set α}
    [∀ a, Decidable (a ∈ X)] [∀ a, Decidable (a ∈ Y)]
    (i : (X ∪ Y).Elem) : X.Elem ⊕ Y.Elem :=
  if hiX : i.val ∈ X then Sum.inl ⟨i, hiX⟩ else
  if hiY : i.val ∈ Y then Sum.inr ⟨i, hiY⟩ else
  (i.property.elim hiX hiY).elim
```

This allows us to re-type matrix dimensions and thus define the matrix transformation `Matrix.toMatrixUnionUnion` so that `A.toMatrixUnionUnion i j = A i.toSum j.toSum`.

We also define a function `Matrix.toMatrixElemElem` for convenience, but it is not a part of the trusted code.

6 Vector Matroids

Vector matroids [9, 12] is the most fundamental matroid class formalized in our work, serving as the basis for binary and regular matroids in later sections. A *vector matroid* is constructed from a matrix A by taking the column index set as the ground set and declaring a set I to be independent if the set of columns of A indexed by I is linearly independent. To this end, we implemented the definition

```
def Matrix.toMatroid {α R : Type*} {X Y : Set α} [DivisionRing R] (A : Matrix X Y R) :
    Matroid α := sorry
```

Note that although linear independence is defined over semirings R, in the definition above we need to assume that R is a `DivisionRing`, otherwise the resulting structure would not satisfy the augmentation property of a matroid.

7 Standard Representations

The *standard representation* [9, 12] of a vector matroid is the following structure:

```
structure StandardRepr (α R : Type*)
    [DecidableEq α] where
  X : Set α
  Y : Set α
  hXY : Disjoint X Y
  B : Matrix X Y R
  decmemX : ∀ a, Decidable (a ∈ X)
  decmemY : ∀ a, Decidable (a ∈ Y)
```

In essence, this is a wrapper for the standard representation matrix B indexed by disjoint sets X and Y, bundled together with the membership decidability for X and Y. The standard representation matrix B corresponds to the full representation matrix $\boxed{\mathbb{1} \mid B}$ with the conversion implemented as

```
def StandardRepr.toFull {α R : Type*} [DecidableEq α] [Zero R] [One R]
    (S : StandardRepr α R) : Matrix S.X (S.X ∪ S.Y).Elem R :=
  ((Matrix.fromCols 1 S.B) · ∘ Subtype.toSum)
```

Thus, the vector matroid given by its standard representation is constructed as follows:

```
def StandardRepr.toMatroid {α R : Type*} [DecidableEq α] [DivisionRing R]
    (S : StandardRepr α R) : Matroid α :=
  S.toFull.toMatroid
```

In this matroid, the ground set is $X \cup Y$, and a set $I \subseteq X \cup Y$ is independent if the columns of $\boxed{\mathbb{1} \mid B}$ indexed by I are linearly independent over R.

Below are several results we prove about standard representations, which are either used in the proof of regularity of 1-, 2-, and 3-sums, or could be useful for downstream projects.

First, we show that if the row index set X of a standard representation is finite, then X is a base in the resulting matroid:

```
lemma StandardRepr.toMatroid_isBase_X
    {α R : Type*} [DecidableEq α] [Field R]
    (S : StandardRepr α R) [Fintype S.X] : S.toMatroid.IsBase S.X
```

This lemma characterizes what sets can serve as row index sets of standard representations and motivates the corresponding hypotheses in the code snippets below.

Next, we prove that a full representation of a vector matroid can be transformed into a standard representation of the same matroid, with a given base as the row index set:

```
lemma Matrix.exists_standardRepr_isBase
    {α R : Type*} [DecidableEq α] [DivisionRing R]
    {X Y G : Set α} (A : Matrix X Y R) (hAG : A.toMatroid.IsBase G) :
    ∃ S : StandardRepr α R, S.X = G ∧ S.toMatroid = A.toMatroid
```

In classical literature on matroid theory [9, 12], this follows by simply performing a sequence of elementary row operations akin to Gaussian elimination. Our formal proof used a different approach, utilizing Mathlib's results about bases and linear independence. First, we showed that the columns indexed by `G` form a basis of the module generated by all columns of `A`. Then we proved that performing a basis exchange yields the correct standard representation matrix.

We also prove an analog of the above lemma that additionally preserves total unimodularity of the representation matrix:

```
lemma Matrix.exists_standardRepr_isBase_isTotallyUnimodular
    {α R : Type*} [DecidableEq α] [Field R]
    {X Y G : Set α} [Fintype G] (A : Matrix X Y R)
    (hAG : A.toMatroid.IsBase G) (hA : A.IsTotallyUnimodular) :
    ∃ S : StandardRepr α R, S.X = G ∧ S.toMatroid = A.toMatroid ∧ S.B.IsTotallyUnimodular
```

Classical literature [9, 12] observes that elementary row operations preserve total unimodularity and then simply refers to the proof of the previous lemma. Unfortunately, we could not take advantage of such a reduction, as it would be hard to verify that total unimodularity is preserved in our prior approach. Thus, we implemented an inductive proof essentially following the ideas of [9, 12]. Note that this lemma takes stronger assumptions than the previous one, namely `G` has to be finite and multiplication in `R` has to commute.

Another result we prove is that two standard representations of the same vector matroid over $\mathbb{Z}_2$ with the same finite row index set must be identical:

```
lemma ext_standardRepr_of_same_matroid_same_X
    {α : Type*} [DecidableEq α]
    {S₁ S₂ : StandardRepr α Z2} [Fintype S₁.X]
    (hSS : S₁.toMatroid = S₂.toMatroid) (hXX : S₁.X = S₂.X) : S₁ = S₂
```

Although this particular lemma is not employed later in our project, it captures an important result that a binary matroid has an essentially unique standard representation [9, 12]. Nevertheless, we make use of a very similar result:

```
lemma support_eq_support_of_same_matroid_same_X
    {F₁ : Type u₁} {F₂ : Type u₂}
    {α : Type max u₁ u₂ v} [DecidableEq α]
    [DecidableEq F₁] [DecidableEq F₂]
    [Field F₁] [Field F₂]
    {S₁ : StandardRepr α F₁}
    {S₂ : StandardRepr α F₂}
    [Fintype S₂.X]
    (hSS : S₁.toMatroid = S₂.toMatroid) (hXX : S₁.X = S₂.X) :
    let hYY : S₁.Y = S₂.Y := sorry
    hXX ▸ hYY ▸ S₁.B.support = S₂.B.support
```

This states that two standard representations of a vector matroid with identical (finite) row index sets have the same support, i.e., the zeros in them appear on identical positions. Crucially, this holds for any two standard representations over any two fields (where equality is decidable), and we later use it for $\mathbb{Q}$ and $\mathbb{Z}_2$.

8 Regular Matroids

Regular matroids [9, 12] are the core subject of Seymour's theorem. A matroid is *regular* if it can be constructed (as a vector matroid) from a rational totally unimodular matrix:

```
def Matroid.IsRegular {α : Type*} (M : Matroid α) : Prop :=
  ∃ X Y : Set α, ∃ A : Matrix X Y ℚ, A.IsTotallyUnimodular ∧ A.toMatroid = M
```

One key result we prove is that every regular matroid is in fact *binary*, i.e., can be constructed from a binary matrix:

```
lemma Matroid.IsRegular.isBinary
    {α : Type*} [DecidableEq α]
    {M : Matroid α} (hM : M.IsRegular) :
    ∃ X : Set α, ∃ Y : Set α, ∃ A : Matrix X Y Z2, A.toMatroid = M
```

Another important lemma we prove about regular matroids is their equivalent characterization in terms of totally unimodular signings. First, let us introduce the necessary definitions. We say that a matrix A is a *signing* of matrix U if their values are identical up to signs:

```
def Matrix.IsSigningOf {X Y R : Type*} [LinearOrderedRing R] {n : ℕ}
    (A : Matrix X Y R) (U : Matrix X Y (ZMod n)) : Prop :=
  ∀ i : X, ∀ j : Y, |A i j| = (U i j).val
```

We then say that a binary matrix U *has a totally unimodular signing* if it has a signing matrix A that is rational and totally unimodular:

```
def Matrix.IsTuSigningOf {X Y : Type*} (A : Matrix X Y ℚ) (U : Matrix X Y Z2) : Prop :=
  A.IsTotallyUnimodular ∧ A.IsSigningOf U
```

```
def Matrix.HasTuSigning {X Y : Type*} (U : Matrix X Y Z2) : Prop :=
  ∃ A : Matrix X Y ℚ, A.IsTuSigningOf U
```

Now, we can state the characterization: given a standard representation over $\mathbb{Z}_2$, its matrix has a totally unimodular signing if and only if the matroid obtained from the representation is regular.

```
lemma StandardRepr.toMatroid_isRegular_iff_hasTuSigning {α : Type*} [DecidableEq α]
    (S : StandardRepr α Z2) [Finite S.X] : S.toMatroid.IsRegular ↔ S.B.HasTuSigning
```

Out of all definitions in this section, only `Matroid.IsRegular` is a part of the trusted code.

9 The 1-Sum

Let $B_\ell \in \mathbb{Z}_2^{X_\ell \times Y_\ell}$ and $B_r \in \mathbb{Z}_2^{X_r \times Y_r}$ be standard representation matrices where X_ℓ, Y_ℓ, X_r, Y_r are pairwise disjoint sets. The *1-sum* $B = B_\ell \oplus_1 B_r$ of B_ℓ and B_r is

$$B = \begin{bmatrix} B_\ell & 0 \\ 0 & B_r \end{bmatrix} \in \mathbb{Z}_2^{(X_\ell \cup X_r) \times (Y_\ell \cup Y_r)}.$$

A matroid M is a *1-sum* of matroids M_ℓ and M_r if there exist standard $\mathbb{Z}_2$ representation matrices B_ℓ, B_r, and B (for M_ℓ, M_r, and M, respectively) of the form above.

All matroid sums are implemented on three levels: the `Matrix` level, the `StandardRepr` level, and the `Matroid` level. The `Matrix` level defines the standard representation matrix of the output matroid:

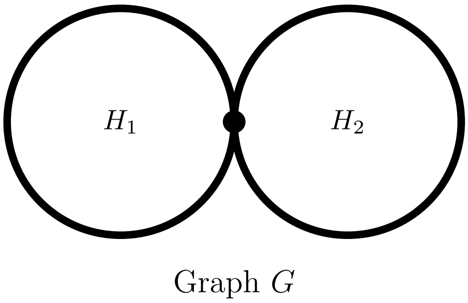

Figure 1 The 1-sum of graphic matroids. The graph G is composed of two subgraphs H_1 and H_2 sharing a single vertex. The cycle matroid of G is the 1-sum of the cycle matroids of H_1 and H_2, since the two subgraphs have disjoint edge sets.

```
def matrixSum1 {R : Type*} [Zero R] {Xℓ Yℓ Xr Yr : Type*}
    (Aℓ : Matrix Xℓ Yℓ R) (Ar : Matrix Xr Yr R) : Matrix (Xℓ ⊕ Xr) (Yℓ ⊕ Yr) R :=
  Matrix.fromBlocks Aℓ 0 0 Ar
```

The StandardRepr level (standardReprSum1) converts the output matrix indices from Sum types to set unions, provides a proof that the resulting row and column index sets are disjoint, and checks whether the operation is valid—returning none if preconditions are not met. The Matroid level defines a predicate—when M is a 1-sum of M_ℓ and M_r:

```
def Matroid.IsSum1of {α : Type*} [DecidableEq α]
    (M : Matroid α) (Mℓ Mr : Matroid α) : Prop :=
  ∃ S Sℓ Sr : StandardRepr α Z2,
  ∃ hXY : Disjoint Sℓ.X Sr.Y,
  ∃ hYX : Disjoint Sℓ.Y Sr.X,
  standardReprSum1 hXY hYX = some S
  ∧ S.toMatroid = M
  ∧ Sℓ.toMatroid = Mℓ
  ∧ Sr.toMatroid = Mr
```

In addition to basic API about the 1-sum, we also provide a theorem Matroid.IsSum1of.eq_disjointSum that establishes the equality between the disjoint sum (defined in Mathlib) and the 1-sum (defined in our project) of binary matroids.

10 The 2-Sum

Let $B_\ell \in \mathbb{Z}_2^{X_\ell \times Y_\ell}$ and $B_r \in \mathbb{Z}_2^{X_r \times Y_r}$ be standard representation matrices where $X_\ell \cap X_r = \{x\}$, $Y_\ell \cap Y_r = \{y\}$, and X_ℓ is disjoint with Y_r and X_r is disjoint with Y_ℓ. Let $A_\ell = B_\ell(X_\ell \setminus \{x\}, Y_\ell)$, $A_r = B_r(X_r, Y_r \setminus \{y\})$, $r = B_\ell(x, Y_\ell) \neq 0$, and $c = B_r(X_r, y) \neq 0$. The *2-sum* $B = B_\ell \oplus_2 B_r$ is defined as

$$B = \begin{bmatrix} A_\ell & 0 \\ c \otimes r & A_r \end{bmatrix}.$$

A matroid M is a *2-sum* of matroids M_ℓ and M_r if there exist standard $\mathbb{Z}_2$ representation matrices B_ℓ, B_r, and B (for M_ℓ, M_r, and M, respectively) of the form above.

The implementation of the 2-sum follows the same three-level structure as the 1-sum. The Matrix level places the two given matrices along the main diagonal of the resulting block matrix, with the bottom-left block containing the outer product of the two given vectors:

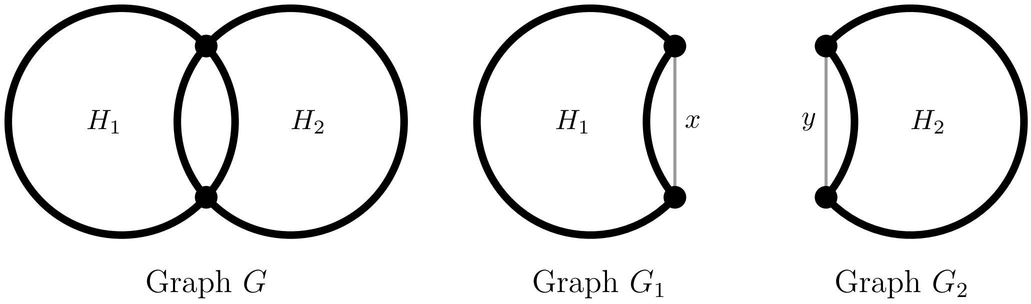

Figure 2 The 2-sum of graphic matroids. The graph G consists of two subgraphs H_1 and H_2 sharing an edge (two vertices). The graphs G_1 and G_2 are obtained by separating G along the shared edge: G_1 contains H_1 plus an additional edge x parallel to the shared edge, and G_2 contains H_2 plus an additional parallel edge y. The cycle matroid of G is the 2-sum of the cycle matroids of G_1 and G_2.

```
def matrixSum2 {R : Type*} [Semiring R] {X_ℓ Y_ℓ X_r Y_r : Type*}
    (A_ℓ : Matrix X_ℓ Y_ℓ R) (r : Y_ℓ → R) (A_r : Matrix X_r Y_r R) (c : X_r → R) :
    Matrix (X_ℓ ⊕ X_r) (Y_ℓ ⊕ Y_r) R :=
  Matrix.fromBlocks
    A_ℓ 0 (fun i j => c i * r j) A_r
```

The `StandardRepr` level (`standardReprSum2`) first slices the last row of $\texttt{S}_\ell\texttt{.B}$ and the first column of $\texttt{S}_r\texttt{.B}$ as the two separate vectors (`r` and `c`), naming the two remaining matrices $\texttt{A}_\ell$ and $\texttt{A}_r$. To identify the special row and column, we need a specific element `x` in $\texttt{S}_\ell\texttt{.X} \cap \texttt{S}_r\texttt{.X}$ and a specific element `y` in $\texttt{S}_\ell\texttt{.Y} \cap \texttt{S}_r\texttt{.Y}$ with no other element in any pairwise intersection among the four indexing sets. The following picture shows how $\texttt{S}_\ell\texttt{.B}$ and $\texttt{S}_r\texttt{.B}$ are taken apart:

$$\texttt{S}_\ell\texttt{.B} = \begin{array}{|c|}\hline A_\ell \\ \hline r \\ \hline\end{array}\,, \quad \texttt{S}_r\texttt{.B} = \begin{array}{|c|c|}\hline c & A_r \\ \hline\end{array}$$

The `Matroid` level is again a predicate—when M is a 2-sum of M_ℓ and M_r:

```
def Matroid.IsSum2of {α : Type*} [DecidableEq α]
    (M : Matroid α) (M_ℓ M_r : Matroid α) : Prop :=
  ∃ S S_ℓ S_r : StandardRepr α Z2,
  ∃ x y : α,
  ∃ hXX : S_ℓ.X ∩ S_r.X = {x},
  ∃ hYY : S_ℓ.Y ∩ S_r.Y = {y},
  ∃ hXY : Disjoint S_ℓ.X S_r.Y,
  ∃ hYX : Disjoint S_ℓ.Y S_r.X,
  standardReprSum2 hXX hYY hXY hYX = some S
  ∧ S.toMatroid = M
  ∧ S_ℓ.toMatroid = M_ℓ
  ∧ S_r.toMatroid = M_r
```

11 The 3-Sum

The 3-sum of binary matroids is defined as follows. Let X_ℓ, Y_ℓ, X_r, and Y_r be sets with the following properties:

- $X_\ell \cap X_r = \{x_2, x_1, x_0\}$ for some distinct x_0, x_1, and x_2
- $Y_\ell \cap Y_r = \{y_0, y_1, y_2\}$ for some distinct y_0, y_1, and y_2
- $X_\ell \cap Y_\ell = X_\ell \cap Y_r = X_r \cap Y_\ell = X_r \cap Y_r = \emptyset$

Let $B_\ell \in \mathbb{Z}_2^{X_\ell \times Y_\ell}$ and $B_r \in \mathbb{Z}_2^{X_r \times Y_r}$ be matrices of the form

$$B_\ell = \left[\begin{array}{c|cc|c} \multicolumn{3}{c|}{A_\ell} & 0 \\ & 1 & 1 & 0 \\ \hline D_\ell & \multicolumn{2}{c|}{D_0} & 1 \\ & & & 1 \end{array}\right], \quad B_r = \left[\begin{array}{cc|c|c} 1 & 1 & 0 & 0 \\ \hline \multicolumn{2}{c|}{D_0} & 1 & \\ & & 1 & A_r \\ \hline \multicolumn{2}{c|}{D_r} & & \end{array}\right]$$

where D_0 is invertible. Then the *3-sum* $B = B_\ell \oplus_3 B_r$ is

$$B = \left[\begin{array}{c|cc|c|c} \multicolumn{3}{c|}{A_\ell} & \multicolumn{2}{c}{0} \\ & 1 & 1 & 0 & \\ \hline D_\ell & \multicolumn{2}{c|}{D_0} & 1 & \\ & & & 1 & A_r \\ \hline D_{\ell r} & \multicolumn{2}{c|}{D_r} & & \end{array}\right] \quad \text{where } D_{\ell r} = D_r \cdot D_0^{-1} \cdot D_\ell$$

Here $D_0 \in \mathbb{Z}_2^{\{x_0,x_1\}\times\{y_0,y_1\}}$, $\left[\begin{array}{cc|c} 1 & 1 & 0 \\ \hline \multicolumn{2}{c|}{D_0} & 1 \\ & & 1 \end{array}\right] \in \mathbb{Z}_2^{\{x_2,x_0,x_1\}\times\{y_0,y_1,y_2\}}$, and the indexing is kept consistent between B_ℓ, B_r, and B. A matroid M is a *3-sum* of matroids M_ℓ and M_r if they admit standard representations over $\mathbb{Z}_2$ with matrices B, B_ℓ, and B_r of the form above. The `Matroid`-level predicate `Matroid.IsSum3of` is defined similarly to those for 1- and 2-sums.

12 Sums Preserve Regularity

In our library, the final theorems that regularity is preserved under 1-, 2-, and 3-sums are stated as follows.

```
theorem Matroid.IsSum1of.isRegular {α : Type*} [DecidableEq α] {M Mℓ Mᵣ : Matroid α} :
  M.IsSum1of Mℓ Mᵣ → M.RankFinite → Mℓ.IsRegular → Mᵣ.IsRegular → M.IsRegular

theorem Matroid.IsSum2of.isRegular {α : Type*} [DecidableEq α] {M Mℓ Mᵣ : Matroid α} :
  M.IsSum2of Mℓ Mᵣ → M.RankFinite → Mℓ.IsRegular → Mᵣ.IsRegular → M.IsRegular

theorem Matroid.IsSum3of.isRegular {α : Type*} [DecidableEq α] {M Mℓ Mᵣ : Matroid α} :
  M.IsSum3of Mℓ Mᵣ → M.RankFinite → Mℓ.IsRegular → Mᵣ.IsRegular → M.IsRegular
```

Note that these three theorems are stated for matroids and have the same interface. Moreover, when applying one of these results, a user is able to provide different representations for

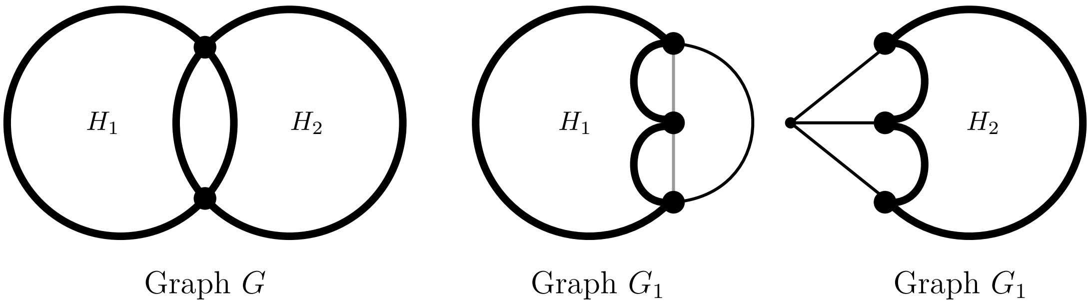

Figure 3 The 3-sum of graphic matroids. The graph G consists of two subgraphs H_1 and H_2 sharing a triangle (three vertices and three edges). The graphs G_1 and G_2 are obtained by separating G along this triangle: each summand retains the three boundary vertices with additional edges forming the shared triangle structure. The cycle matroid of G is the 3-sum of the cycle matroids of G_1 and G_2.

witnessing that M is a 1-, 2-, or 3-sum of M_ℓ and M_r, for witnessing that M has finite rank, and for witnessing that M_ℓ and M_r are regular.

We split the proof of each of these theorems into three stages corresponding to the three abstraction layers used for the definitions: `Matroid`, `StandardRepr`, and `Matrix`.

The final `Matroid`-level theorems are reduced to the respective lemmas for standard representations by applying `StandardRepr.toMatroid_isRegular_iff_hasTuSigning` and `StandardRepr.finite_X_of_toMatroid_rankFinite` in all three proofs (for the 1-, 2-, and 3-sums). The reductions from the `StandardRepr` level to the `Matrix` level for 1- and 2-sums is straightforward—plug the standard representation matrices and their (rational) signings into `matrixSum1` and `matrixSum2`, respectively. For 3-sums, this reduction is more involved, as we additionally apply the following lemma to simplify the assumption on D_0:

```
lemma Matrix.isUnit_2x2 (A : Matrix (Fin 2) (Fin 2) Z2) (hA : IsUnit A) :
  ∃ f : Fin 2 ≃ Fin 2, ∃ g : Fin 2 ≃ Fin 2,
    A.submatrix f g = 1 ∨ A.submatrix f g = !![1, 1; 0, 1]
```

Therefore, up to reindexing, D_0 is either $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ or $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$. Performing the reduction at this stage allows us to invoke `Matrix.isUnit_2x2` only once and then simply consider the two special forms of D_0.

On the `Matrix` level, our formal proof that 1-sums preserve total unimodularity of matrices is nearly identical to [12]. For 2-sums, we streamlined the proof by reformulating it as a forward argument by induction. For 3-sums, the entire argument was significantly reworked to simplify and streamline the approach of [12]. On a high level, we make two major changes, which we discuss in detail below.

The first key difference is that we re-sign the summands only once, rather than multiple times. Like in [12], we start with totally unimodular signings exhibiting regularity of the two summands. Then we multiply their rows and columns by ± 1 factors (which preserves total unimodularity) so that the submatrix $\begin{array}{|c|c|c|} \hline 1 & 1 & 0 \\ \hline \multicolumn{2}{|c|}{\multirow{2}{*}{D_0}} & 1 \\ \cline{3-3} \multicolumn{2}{|c|}{} & 1 \\ \hline \end{array}$ is signed in both summands simultaneously as either $\begin{bmatrix}1 & 1 & 0\\ 1 & 0 & 1\\ 0 & -1 & 1\end{bmatrix}$ or $\begin{bmatrix}1 & 1 & 0\\ 1 & 1 & 1\\ 0 & 1 & 1\end{bmatrix}$, depending on whether D_0 is $\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$ or $\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}$. Thus, we get totally unimodular signings of the summands that coincide on the intersection, which allows us to define the *canonical* signing of the entire 3-sum: use the same signs as in the re-signed summands everywhere except for the bottom-left block, which is signed via $D'_{\ell r} = D'_r \cdot (D'_0)^{-1} \cdot D'_\ell$, and the 0 block, which remains as is.

The main advantage of our approach is that we avoid chained constructions and proofs of properties of such constructions, and we do not need to define Δ-sums. Moreover, unlike [12], our proof does not rely on the general lemma about re-signing totally unimodular matrices. This detail is crucial, as the proof of this lemma in [12] involves a graph-theoretic argument, which would be very challenging to formalize in Lean with the current tools available in Mathlib.

The second major difference from the approach of [12] is that our main argument does not deal with signings of 3-sums directly. Instead, we work with a matrix family called `MatrixLikeSum3` in our code. This allows us to split the proof of regularity of 3-sums into three clear steps. First, we show that pivoting on a non-zero entry in the top-left block of any matrix from this family produces a matrix that also belongs to this family. Next, we utilize the result from the first step to prove that every matrix in this family is totally unimodular. We do this via a similar argument to the proof that 2-sums of totally unimodular matrices are totally unimodular. Finally, we show that every canonical signing of the 3-sum matrix defined above is included in this matrix family and is thus totally unimodular. Overall, this proof takes a more systematic approach to deriving properties of signings of 3-sums and using them to prove their total unimodularity. Additionally, it conveniently reuses a large portion of the argument for 2-sums.

In some proofs, we worked with large case splits with up to 896 cases. To handle such situations, we used `all_goals try` followed by one or more tactics, discharging multiple goals at once without selecting them by hand or repeating the proof. We repeatedly applied this method to discharge the remaining goals in waves until the proof was complete.

13 Related Work

In Lean 4, the largest library formalizing matroid theory is due to Peter Nelson[3]. It implements infinite matroids following [3] together with many key notions and results about them. The definition that is fully formalized and is the most related to our work is `Matroid.disjointSum`. For binary matroids, this definition is equivalent to the 1-sum implemented in this paper. Moreover, it can be used for any matroids with disjoint ground sets, while our implementation is restricted to vector matroids constructed from $\mathbb{Z}_2$ matrices. Peter Nelson's repository also makes progress towards formalizing other related notions, such as representable matroids, though this work is still ongoing. It is also worth noting that the results in Mathlib[4] have been copied over from this repository and comprise a strict subset of it.

Building upon Peter Nelson's work, Gusakov2024's thesis [5] formalizes the proof of Tutte's excluded minor theorem and to this end implements definitions and results about

[3] https://github.com/apnelson1/lean-matroids
[4] https://github.com/leanprover-community/mathlib4/tree/master/Mathlib/Combinatorics/Matroid

representable matroids. The thesis formalizes representations and standard representations of matroids, which we also do in our work, but it takes a different approach. In particular, instead of working with matrix representations, the thesis implements a representation of `Matroid` α as a mapping from the entire type α to a vector space, which maps non-elements of the matroid to the zero vector and independent sets to linearly independent vectors. The advantage of this approach is that certain proofs become easier to formalize, but this comes at a cost of making it harder to match the implementation with the theory and believe the correctness of the code.

There are also two Lean 3 repositories due to Artem Vasilyev[5] and Bryan Gin-ge Chen[6] dedicated to formalization of matroid theory. Both of them work with finite matroids following [9] and implement basic definitions and properties of matroids concerning circuits, bases, and rank functions. These results are completely subsumed by the current implementation of matroids in Mathlib.

Jonas Keinholz [6] formalizes the classical definition of (finite) matroids [9, 12] in Isabelle/HOL along with other basic ideas such as minors, bases, circuits, rank, and closure. More recently, Wan2025 use Keinholz's formalization to design a verification framework using a Locale that checks if a given collection of subsets of a given set is a matroid. The authors then showcase the verification algorithm by checking that the 0-1 knapsack problem does not conform to the matroid structure, while the fractional knapsack problem does. In comparison, Lean 4's Mathlib implements a more general definition of matroids and formalizes more results about them than either Matroids-AFP or Wan2025, but Lean lacks a procedure for formally verifying if a collection of sets has matroid structure.

In the HOL Light GitHub repository[7], John Harrison formalizes finitary matroids. The formalization closely follows the field theory notes of Pete L. Clark[8]. In particular, finitary matroids are defined in terms of a closure operator with similar properties as those proposed in [3]. This repository also includes a formal proof that this notion of (finitary) matroids is equivalent to the definition of a matroid using independent sets. Unlike Lean 4's Mathlib formalization (which includes formalizations of the closure operator and the notions of spanning sets), however, this notion of infinite matroids does not respect the notion of duality that is defined for matroids in [9, 12] as noted by Bruhn2013.

Grzegorz Bancerek and Yasunari Shidama [1] formalize matroids in Mizar. Their formalization includes basic notions like rank, basis, and cycle as well as examples like the matroid of linearly independent subsets for a given vector space. Overall, the scope of the Mizar formalization is comparable to the Isabelle/HOL formalization, except that the Mizar formalization allows for infinite matroids. In this sense, it is comparable to the Lean definition in Mathlib, which also allows for infinite matroids. However, whereas Mizar uses independence axioms to define matroids, Lean uses base axioms for the main definition and provides an API for constructing matroids via independence axioms.

14 Conclusion

In this work, we formally stated Seymour's decomposition theorem for regular matroids and implemented a formally verified proof of the forward (composition) direction of this theorem

[5] `https://github.com/VArtem/lean-matroids`
[6] `https://github.com/bryangingechen/lean-matroids`
[7] `https://github.com/jrh13/hol-light/blob/master/Library/matroids.ml`
[8] `https://plclark.github.io/PeteLClark/Expositions/FieldTheory.pdf`

in the setting where the matroids have finite rank and may have infinite ground sets. To this end, we developed a modular and extensible library in Lean 4 formalizing definitions and lemmas about totally unimodular matrices, vector matroids, regular matroids, and 1-, 2-, and 3-sums of matrices, standard representations of vector matroids, and matroids. Our implementation is 8728 lines long. Our work demonstrates that one can effectively use Lean and Mathlib to formally verify advanced results from matroid theory and extend classical results to a more general setting.

Formalizing Seymour's theorem presented several challenges. First, the limited matroid theory in Mathlib meant we had to develop many fundamentals from scratch (e.g. representability and regularity definitions). We addressed this by introducing a *StandardRepr* structure to bridge matrices and matroids, enabling us to work around the absence of a general matroid representation theory. Moreover, some proofs required managing enormous case splits (our 3-sum proof involved up to 896 subcases). We tackled this with structured automation - for instance, using `all_goals try` tactics to discharge many cases at once - thereby keeping the proof tractable in Lean. We also avoided certain combinatorial arguments (such as the graph-theoretic re-signing lemma from [12]) that would be cumbersome to formalize, opting for alternative approaches better suited to Lean.

The most natural continuation of our project is proving the decomposition direction of Seymour's theorem, stated as `Matroid.IsRegular.isGood_of_rankFinite` in our library. Our work can also serve as a starting point for formalizing Seymour's theorem for matroids of infinite rank [2].

References

1. Grzegorz Bancerek and Yasunari Shidama. Introduction to matroids. *Formalized Mathematics*, 16(4):325–332, 2008.
2. Nathan Bowler and Johannes Carmesin. The ubiquity of psi-matroids, 2013.
3. Henning Bruhn, Reinhard Diestel, Matthias Kriesell, Rudi Pendavingh, and Paul Wollan. Axioms for infinite matroids, 2013.
4. Jim Geelen and Bert Gerards. Regular matroid decomposition via signed graphs. *Journal of Graph Theory*, 48(1):74–84, 2005.
5. Alena Gusakov. Formalizing the excluded minor characterization of binary matroids in the lean theorem prover. Master's thesis, University of Waterloo, 2024.
6. Jonas Keinholz. Matroids. *Archive of Formal Proofs*, November 2018. `https://isa-afp.org/entries/Matroids.html`, Formal proof development.
7. Sandra R. Kingan. On seymour's decomposition theorem. *Annals of Combinatorics*, 19(1):171–185, Mar 2015.
8. The mathlib community. The Lean Mathematical Library. In Jasmin Blanchette and Cătălin Hritcu, editors, *CPP 2020*, pages 367–381. ACM, 2020.
9. James Oxley. *Matroid Theory*. Oxford University Press, Oxford, 02 2011.
10. Robert Pollack. How to believe a machine-checked proof. *BRICS Report Series*, 4(18), January 1997.
11. Moses Schönfinkel. Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92:305–316, 1924.
12. Klaus Truemper. *Matroid Decomposition*. Leibniz Company, 2016.