Formalization and Linear Algebra

Katherine Buesing

2025-08-06

Outline

1 Introduction
- Table of Contents
- Formalization and Lean

2 Basis of a Direct Sum
- Definitions
- Linear Independence of Transverse Subspaces
- Basis of a Direct Sum Construction
- Additional Basis of Direct Sum Lemmas

3 Reflexivity of a Bilinear Form iff Alternating or Symmetric
- Definitions
- Grove's Lemma
- Reflexivity implies Alternating or Symmetric
- Reflexivity iff Alternating or Symmetric

4 The Orthogonal Complement of a Nondegenerate Subspace
- Definitions
- p and eq Constructions
- Theorem Statement

5 Conclusion
- Closing Remarks
- References

What is Formalization?

- Formalization is a way of creating mathematical proofs or definitions using a programming language, such as Lean. However, other proof assistants exist such as Isabelle or Coq.

What is Formalization?

- Formalization is a way of creating mathematical proofs or definitions using a programming language, such as Lean. However, other proof assistants exist such as Isabelle or Coq.
- Formalization allows us to have 100-percent certainty that a statement is correct.

What is Formalization?

- Formalization is a way of creating mathematical proofs or definitions using a programming language, such as Lean. However, other proof assistants exist such as Isabelle or Coq.
- Formalization allows us to have 100-percent certainty that a statement is correct.
- Although the current body of work of formalized mathematics is not very large, someday it may be vast enough that new mathematics can be formalized at the same time as its discovery.

What is Lean?

- Lean is a computer language based on dependent type theory, which means that an object's type can depend on its value.

What is Lean?

- Lean is a computer language based on dependent type theory, which means that an object's type can depend on its value.
- For example, Java is a strongly typed language, which sounds like it might be similar, but isn't necessarily.

What is Lean?

- Lean is a computer language based on dependent type theory, which means that an object's type can depend on its value.
- For example, Java is a strongly typed language, which sounds like it might be similar, but isn't necessarily.
- In Java, a number has the same type as another number, regardless of its value. In Lean, 0 has a different type from 1, even though they are both natural numbers. This is because its type depends on its value.

What is Lean?

- Lean is a computer language based on dependent type theory, which means that an object's type can depend on its value.
- For example, Java is a strongly typed language, which sounds like it might be similar, but isn't necessarily.
- In Java, a number has the same type as another number, regardless of its value. In Lean, 0 has a different type from 1, even though they are both natural numbers. This is because its type depends on its value.
- This dependent type theory allows us to be able to precisely formalize mathematic expressions.

0 vs. 1

```
#check 0
#check 1
```

```
0 : ℕ
1 : ℕ
```

Theorem Statement

Theorem

Given a vector space V and two subspaces W_1 and W_2 such that the intersection of W_1 and W_2 is the zero element. Pick bases b_1 and b_2 for subspaces W_1 and W_2, respectively. Then we can conclude that the disjoint union of b_1 and b_2 is also linearly independent.

```
theorem lin_indep_by_transverse_subspaces
   (k V : Type) [Field k] [AddCommGroup V]
   [Module k V] (I₁ I₂ : Type) [Fintype I₁]
   [Fintype I₂] (b₁ : I₁ → V) (b₂ : I₂ → V)
   (b1_indep : LinearIndependent k b₁)
   (b2_indep : LinearIndependent k b₂)
   (W₁ W₂ : Submodule k V) (h_int : W₁ ⊓ W₂ = ⊥)
   (hbw1 : ∀ i, b₁ i ∈ W₁) (hbw2 : ∀ i, b₂ i ∈ W₂)
   [DecidableEq I₁] [DecidableEq I₂]
   : LinearIndependent k (Sum.elim b₁ b₂) :=
```

Definition Statement

Definition

Given a vector space V, and two subspaces W_1 and W_2 such that the intersection of W_1 and W_2 is the zero element, and their direct sum is equal to V. Pick bases b_1 and b_2 of W_1 and W_2, respectively. Then, $b_1 \cup b_2$ is a basis for V.

```
def basis_of_direct_sum (W₁ W₂ : Submodule k V)
        {ι₁ ι₂ : Type} [Fintype ι₁] [Fintype ι₂]
        (B₁ : Basis ι₁ k W₁) (B₂ : Basis ι₂ k W₂)
        (hspan : W₁ ⊔ W₂ = (⊤: Submodule k V))
        (hindep : W₁ ⊓ W₂ = (⊥:Submodule k V))
        [DecidableEq ι₁] [DecidableEq ι₂]
        [FiniteDimensional k V]:
       Basis (ι₁ ⊕ ι₂) k V
```

Additional Lemmas

```
lemma left_mem_basis_direct_sum {ι₁ ι₂ :Type}
    (W₁ W₂ : Submodule k V) (B₁ : Basis ι₁ k W₁)(B₂ : Basis ι₂ k W₂)
    [FiniteDimensional k V] [Fintype ι₁] [DecidableEq ι₁] [Fintype ι₂]
    [DecidableEq ι₂] (hspan : W₁ ⊔ W₂ = (⊤: Submodule k V))
    (hindep : W₁ ⊓ W₂ = (⊥:Submodule k V)) (i:ι₁) :
        (basis_of_direct_sum W₁ W₂ B₁ B₂ hspan hindep) (Sum.inl i) ∈ W₁ := by
```

```
lemma right_mem_basis_direct_sum {ι₁ ι₂ :Type}
    (W₁ W₂ : Submodule k V) (B₁ : Basis ι₁ k W₁) (B₂ : Basis ι₂ k W₂)
    [FiniteDimensional k V] [Fintype ι₁] [DecidableEq ι₁]
    [Fintype ι₂]  [DecidableEq ι₂] (hspan : W₁ ⊔ W₂ = (⊤: Submodule k V))
    (hindep : W₁ ⊓ W₂ = (⊥:Submodule k V)) (i:ι₂) :
        (basis_of_direct_sum W₁ W₂ B₁ B₂ hspan hindep) (Sum.inr i) ∈ W₂ := by
```

What is a Bilinear Form?

Definition

A bilinear form β on a vector space V over a field k is a function $\beta : V \times V \to k$ that is linear in both variables.

Types of Bilinear Forms

Reflexive Bilinear Forms

A bilinear form β on a vector space V is reflexive if
$\forall u, v \in V, \beta(u, v) = 0 \implies \beta(v, u) = 0$

Symmetric Bilinear Forms

A bilinear form β on a vector space V is symmetric if
$\forall u, v \in V, \beta(u, v) = \beta(v, u)$

Alternating Bilinear Forms

A bilinear form β on a vector space V is alternating if
$\forall v \in V, \beta(v, v) = 0$

Lemma Statement

Lemma

Suppose that β is a bilinear form on a vector space V that satisfies the following condition:

- $\beta(u, v)\beta(w, u) = \beta(v, u)\beta(u, w)$

$\forall u, v, w \in V$. Then, we can conclude that β is a symmetric or alternating bilinear form.

```
lemma proptwopointsix {β: BilinForm k V}
(h : ∀ (u v w : V), (((β u) v) * ((β w) u)) =
(((β v) u) * ((β u) w))):
β.IsAlt ∨ β.IsSymm :=
```

Theorem Statement

Theorem

Let β be a reflexive bilinear form on a vector space V. Then, β is also symmetric or alternating.

```
theorem refl_is_alt_or_symm {β: BilinForm k V} (h: β.IsRefl)
[FiniteDimensional k V] :
    β.IsAlt  ∨ β.IsSymm :=
```

Theorem Statement

Theorem

β is a reflexive bilinear form on a vector space V if and only if β is alternating or symmetric.

```
theorem refl_iff_alt_or_symm {β : BilinForm F V}
: β.IsRefl ↔ (β.IsAlt ∨ β.IsSymm) := by
  constructor
  · intro h
    exact refl_is_alt_or_symm h
  · intro h
    cases h with
    | inl h₁ => exact IsAlt.isRefl h₁
    | inr h₂ => exact IsSymm.isRefl h₂
```

Orthogonal Complement

Definition

Given a subspace W of a vector space V, the orthogonal complement of W is the set of all vectors that are orthogonal to every vector in W. Additionally, the orthogonal complement of W is also a subspace of V.

Nondegenerate Definitions

Theorem

Let β be a bilinear form on V, $M = [\beta(v_i, v_j)]$, and $v_1, ..., v_n$ a basis of V. The following are equivalent:

- $det(M) \neq 0$
- $\forall w \in V \;\; \beta(v, w) = 0 \implies v = 0$
- $\forall v \in V \;\; \beta(v, w) = 0 \implies w = 0$
- *β is a nondegenerate bilinear form*

Definition

A nondegenerate subspace W is a subspace with a nondegenerate bilinear form β such that the determinant of the matrix representation of β restricted to the subspace W is nonzero.

Block Diagonal Matrix Definition

Definition

A two-block diagonal matrix M is a m by m matrix such that $m = \iota_1 + \iota_2$, and the following are true:

- The upper left hand ι_1 by ι_1 block of the matrix can have any values
- The lower right hand ι_2 by ι_2 block of the matrix can have any values
- The rest of the values of the matrix are zero

A two-block diagonal matrix can be represented by $\begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix}$ where A is an ι_1 by ι_1 matrix with any values and B is an ι_2 by ι_2 matrix with any values.

p Construction

Definition

p is a predicate that takes a matrix M, which is an m by m matrix where $m = \iota_1 + \iota_2$, and maps it to True for every element in the upper left ι_1 by ι_1 block of the matrix, and False for every other element.

```
def p (ι₁ ι₂ : Type) [Fintype ι₁] [Fintype ι₂] [DecidableEq ι₁]
[DecidableEq ι₂]: ι₁ ⊕ ι₂ → Prop := by
  intro i
  exact (∃ (y : ι₁), i = Sum.inl y)
```

This predicate is the construction we use to extract the blocks from a block diagonal matrix. Predicates such as these are one example of a construction you can make in Lean that is not a proof.

eq Construction

```
def eq (ι₁ ι₂ : Type) [Fintype ι₁] [DecidableEq ι₁]
       [Fintype ι₂]  [DecidableEq ι₂]
       : { i : ι₁ ⊕ ι₂ // ¬ p ι₁ ι₂ i } ≃ ι₂ where
```

This equivalence is a construction we need to use so that Lean is able to directly infer that ι_2 and $\neg p \iota_1 \iota_2$ are equivalent. This is another example of something we construct in Lean that is not a proof, as this is a function.

Theorem Statement

Theorem

Let V be a vector space with a subspace W. Assume you have a nondegenerate, reflexive bilinear form β, and let W be a nondegenerate subspace. The orthogonal complement of W is also nondegenerate.

```
theorem ortho_complement_nondeg (β:BilinForm k V)
[FiniteDimensional k V] (bnd : BilinForm.Nondegenerate β)
(W :Submodule k V) (wnd : Nondeg_subspace β W) (href : β.IsRefl)
  [DecidableEq ↑(Basis.ofVectorSpaceIndex k ↥W)]
  [DecidableEq (BilinForm.orthogonal β W)]
  [DecidablePred (p ↑(Basis.ofVectorSpaceIndex k ↥W)
  ↑(Basis.ofVectorSpaceIndex k ↥(BilinForm.orthogonal β W)))]
  {brefl : LinearMap.BilinForm.IsRefl β }:
  Nondeg_subspace β (BilinForm.orthogonal β W) := by
```

Proof Excerpts

```
have k₂ : ∀ i, ¬(p ι₁ ι₂) i → ∀ j , (p ι₁ ι₂) j → M i j = 0 := by
      intro x j₀ y j₁
      unfold p at j₀
      unfold p at j₁
      unfold M
      have g₀ : B y ∈ W := by
        unfold B
        rcases j₁ with < y₁, hy₁ >
        rw[hy₁]
        apply left_mem_basis_direct_sum W (BilinForm.orthogonal β W) b₁ b₂ k₁ k₀
      have g₁ : B x ∈ (BilinForm.orthogonal β W) := by
        unfold B
        have g₁₀ : ∃ z, x = Sum.inr z := by
          exact not_left_in_right x j₀
        rcases g₁₀ with < x₁, hx₁ >
        rw[hx₁]
        apply right_mem_basis_direct_sum W (BilinForm.orthogonal β W) b₁ b₂ k₁ k₀
```

```
have k₄ : ∀ i, ∀ j, (M₂ i j =
(BilinForm.toMatrix b₂ (β.restrict (BilinForm.orthogonal β W))) (eq  ι₁ ι₂ i) (eq ι₁ ι₂ j))
      intro x₀ y₀
      unfold M₂ Matrix.toSquareBlockProp M BilinForm.toMatrix
      simp
      refine DFunLike.congr ?_ ?_
      · ext v
        unfold B
        rw[eq_eq_not_p]
      · unfold B
        rw[eq_eq_not_p]
```

Dependent Type Theory

- Dependent type theory in Lean can make it difficult to infer which mathematical objects are equivalent; sometimes Lean is able to infer equalities between types, but other times we have to construct these equalities ourselves.

Dependent Type Theory

- Dependent type theory in Lean can make it difficult to infer which mathematical objects are equivalent; sometimes Lean is able to infer equalities between types, but other times we have to construct these equalities ourselves.
- Certain theorems work with certain types but not others. This forces us to think about how we choose to represent mathematical objects and the different ways they can exist.

Dependent Type Theory

- Dependent type theory in Lean can make it difficult to infer which mathematical objects are equivalent; sometimes Lean is able to infer equalities between types, but other times we have to construct these equalities ourselves.
- Certain theorems work with certain types but not others. This forces us to think about how we choose to represent mathematical objects and the different ways they can exist.
- For example, a basis could be both simply a basis and also a set of linearly independent vectors that span a space in human mathematics.

Dependent Type Theory

- Dependent type theory in Lean can make it difficult to infer which mathematical objects are equivalent; sometimes Lean is able to infer equalities between types, but other times we have to construct these equalities ourselves.
- Certain theorems work with certain types but not others. This forces us to think about how we choose to represent mathematical objects and the different ways they can exist.
- For example, a basis could be both simply a basis and also a set of linearly independent vectors that span a space in human mathematics.
- In Lean it can only be one of these things, even if they have the same value. While some type coercions can be performed, we still have to make a conscious decision about the best way to represent each object.

References

1. Avigad, J., de Moura, L., Kong, S., & Ullrich, S. Theorem Proving in Lean 4. https://leanprover.github.io/theoremprovinginlean4/
2. Avigad, J., & Massot, P. (2020). Mathematics in Lean. https://leanprover-community.github.io/mathematicsinlean/index.html
3. Browning, T., & Lutz, P. (2022). Formalizing Galois Theory. Experimental Mathematics, 31(2), 413–424.
4. Grove, L. (2002). *Classical Groups and Geometric Algebra*.
5. Massot, P. (2021). Why formalize mathematics?. https://www.imo.universite-paris-saclay.fr/ patrick.massot/files/exposition/whyformalize.pdf

Thank you!

Special thanks to Dr. George McNinch, the REU, and NSF.