Formalization and Finite Algebra

Clea Bergsman, Katherine Buesing, Sahan Wijetunga

2025-07-24

Outline

Equivalent Bilinear Forms

Definition

Two bilinear forms β_1 and β_2 on the respective vector spaces V_1 and V_2 are **equivalent** if there is a vector space isomorphism $\Phi : V_1 \to V_2$ such that $\beta_2(\Phi v, \Phi w) = \beta_1(v, w)$ for all v, $w \in V_1$

Proving Equivalence of Bilinear Forms

Proof Statement

- Given two bilinear forms, β_1 and β_2, on the respective vector spaces V_1 and V_2 and a basis b_1 for β_1
- Show that β_1 is equivalent to $\beta_2 \iff \exists$ a basis b_2 of V_2 such that M_1 given by $[\beta_1\ (b_1\ i,\ b_1\ j)]$ is equal to M_2 given by $[\beta_2\ (b_2\ i,\ b_2\ j)]$

Proving Equivalence of Bilinear Forms $\rightarrow$

Proof Statement

Given that β_1 and β_2 are equivalent, show that M_1 is equivalent to M_2.

Steps:

1. Define $\Phi : V_1 \rightarrow V_2$ with two properties:
   - Equivalence: Φ is an invertible linear transformation
   - Compatibility : $\beta_1(v, w) = \beta_2(\Phi v, \Phi w)$
2. Construct b_2 as a basis from b_1 using Φ
   - If $b_1 = x_1, ..., x_n$ then $b_2 = \Phi x_1, ..., \Phi x_n$
3. Compatibility condition tells you $M_1 = M_2$

Proving Equivalence of Bilinear Forms $\leftarrow$

Proof Statement

Given a basis b_2 of V_2 such that $M_1 = M_2$, show that β_1 is equivalent to β_2.

Steps:

1. Define $\Phi : V_1 \to V_2$ where $\Phi(b_1\ i) = b_2\ i$
   - Note: Φ is invertible because it is a linear map between bases
2. Check compatibility condition holds
   - Compatibility is true on a basis, so must check that compatibility is true for all vectors
3. Show that all vectors can be written as a linear combination of basis vectors, therefore compatibility holds

Sums in Bilinear Forms

Lemma

$$\beta(\sum_i t_i \bullet b_i, \sum_j s_j \bullet b_j) = \sum_i \sum_j t_i * s_j \; \beta(b_i, b_j)$$

Where β is a bilinear form, t_i, $s_j \in k$, and b_i, b_j are basis vectors for V

Recall:

Definition

A **bilinear form** is a map $\beta : V \times W \to K$, where V and W are K-vector spaces and K is a field, when

1. $\beta(v_1 + v_2, w) = \beta(v_1, w) + \beta(v_2, w)$
2. $\beta(v, w_1 + w_2) = \beta(v, w_1) + \beta(v, w_2)$
3. $\beta(\lambda v, w) = \beta(v, \lambda w) = \lambda\beta(v, w)$

hold for all $v \in V$, $w \in W$, and $\lambda \in K$.

```
lemma equiv_of_series {ι:Type} [Fintype ι]
    (β:BilinForm k V) (b : Basis ι k V) (s t : ι → k):
(β (Fintype.linearCombination k ⇑b t))
(Fintype.linearCombination k ⇑b s) =
∑ i:ι, (∑ j:ι, (t i) * (s j) * (β (b i) (b j))) := by
  unfold Fintype.linearCombination
  dsimp
  rw [LinearMap.BilinForm.sum_left]
  apply Finset.sum_congr
  rfl
  intro i h
  rw [LinearMap.BilinForm.sum_right]
  apply Finset.sum_congr
  rfl
  intro j g
  rw [LinearMap.BilinForm.smul_left]
  rw [mul_comm]
  rw [LinearMap.BilinForm.smul_right]
  ring
```

Proving Equivalence of Bilinear Forms in Lean

- First we must define the equivalence relation, or isomorphism, between V_1 and V_2

```
structure equiv_of_spaces_with_form
 (β₁:BilinForm k V₁) (β₂:BilinForm k V₂)
where
equiv : V₁ ≃[k] V₂
compat : ∀ (v w : V₁), (β₁ v) w = (β₂ (equiv  v)) (equiv w)

def equiv_from_bases (b₁:Basis ι k V₁) (b₂:Basis ι k V₂)
  : V₁ ≃[k] V₂ :=
  LinearEquiv.trans b₁.repr (b₂.repr.symm)
```

Proof Statement and $\to$ Case

- Proof Statement

```
theorem equiv_via_matrices {ι:Type} [Fintype ι] [DecidableEq ι]
  (β₁:BilinForm k V₁)   (β₂:BilinForm k V₂) (b₁ : Basis ι k V₁)
  (i j : ι) (s t : ι → k) :
Nonempty (equiv_of_spaces_with_form β₁ β₂) ↔
∃ b₂:Basis ι k V₂, ∀ i j : ι,
(BilinForm.toMatrix b₁ β₁) i j = (BilinForm.toMatrix b₂ β₂) i j
```

- $\to$ Case: Given Φ, show $M_1 = M_2$

```
  constructor
  intro <N>
  let b₂ : Basis ι k V₂ := Basis.map b₁ N.equiv
  use b₂
  unfold b₂
  unfold BilinForm.toMatrix
  simp
  intro i j
  rw [N.compat (b₁ i) (b₁ j)]
```

← Case

- ← Case: Given b_2, show compatibility holds on Φ

```
intro h₁
  rcases h₁ with <b₂, h₁>
  refine Nonempty.intro ?_
  let eq : V₁ ≃[k] V₂ := by apply equiv_from_bases; exact b₁; exact b₂
  have identify_bases : ∀ i:ι, b₂ i = eq (b₁ i) := by
    intro i; unfold eq;  rw [← equiv_from_bases_apply b₁ b₂ i]
  apply equiv_of_spaces_with_form.mk
  intro v w
  swap
  exact eq
  have sum_v : v = (Fintype.linearCombination k ⇑b₁) (b₁.repr v):=
    by symm; apply fintype_linear_combination_repr
  have sum_w : w = (Fintype.linearCombination k ⇑b₁) (b₁.repr w):=
    by symm; apply fintype_linear_combination_repr
  nth_rw 1 [sum_v, sum_w]
  rw [equiv_of_series]
  nth_rw 2 [sum_v, sum_w]
  rw [ Fintype.linearCombination_apply, Fintype.linearCombination_apply]
  rw [ map_sum eq, map_sum eq]
  rw [equiv_of_bilin_series]
  apply Finset.sum_congr
  rfl
  intro i h
  apply Finset.sum_congr
  rfl
```

← Case Continued

```
intro j h
rw [map_smul eq, map_smul eq]
rw [LinearMap.BilinForm.smul_left]
rw [mul_comm]
rw [LinearMap.BilinForm.smul_right]
rw [mul_comm]
rw [← identify_bases, ← identify_bases]
rw [← BilinForm.toMatrix_apply b₁ β₁ i j,
  ← BilinForm.toMatrix_apply b₂ β₂]
rw [h₁ i j]
ring
```

Orthogonal Complement of a Nondegenerate Subspace

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

Theorem

The orthogonal complement of a nondegenerate subspace W is also nondegenerate.

Pen and Paper Proof

- Let V be a vector space over a field k with a nondegenerate bilinear form β, and let W be a nondegenerate subspace of V.
- Then, the direct sum of W and the orthogonal complement of W is equal to V, and their intersection is the zero element, which follows from nondegeneracy.
- Let us pick a basis b_1 for W and a basis b_2 for the orthogonal complement of W. Then, we know that the union of these bases is equal to a basis for V, which follows from the theorem we proved in our last presentation.
- We know that the matrix representation, M, of β on our basis $b_1 \cup b_2$ is a block diagonal matrix because b_1 and b_2 are orthogonal sets of vectors

Pen and Paper Proof Continued

- Then, we know $det(M) = det(M_1) * det(M_2)$, where M_1 and M_2 are the diagonal blocks.
- Since β is a nondegenerate bilinear form, we know $det(M)$ is nonzero, and therefore $det(M_2)$ is nonzero.
- M_2 is equivalent to the matrix representation for β restricted to the orthogonal complement of W, and since its determinant is nonzero, we can conclude that the orthogonal complement of W is a nondegenerate subspace

Lean Helper Lemmas

```
theorem finrank_sup_eq_neg_finrank_inf_add
{u v : Type} {K : Type u} {V : Type v}
[DivisionRing K] [AddCommGroup V] [Module K V]
  (s t : Submodule K V) [FiniteDimensional K ↥s]
  [FiniteDimensional K ↥t] :
  Module.finrank K ↥(s ⊔ t) = Module.finrank K ↥s +
  Module.finrank K ↥t - (Module.finrank K ↥(s ⊓ t)) := by
    rw[Nat.sub_eq_of_eq_add']
    rw[← Submodule.finrank_sup_add_finrank_inf_eq]
    rw[add_comm]
```

This theorem is very similar to one that was written in mathlib, but this is an example of how sometimes making a helper lemma for your specific problem can be helpful even if in general the statement is redundant

Lean Helper Lemmas Continued

```
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
        unfold basis_of_direct_sum
        unfold Sum.elim
        simp

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
        unfold basis_of_direct_sum
        unfold Sum.elim
        simp
```

Lean Definitions

```
def Nondeg_subspace (β : BilinForm k V) (W:Submodule k V) : Prop :=
  BilinForm.Nondegenerate (BilinForm.restrict β W)

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)
```

Lean Proof

```
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
    let ι₁ := (Basis.ofVectorSpaceIndex k ↥W)
    let ι₂ := (Basis.ofVectorSpaceIndex k ↥(BilinForm.orthogonal β W))
    have k₀ : W ⊓ (BilinForm.orthogonal β W) = ⊥ := by
      rw[IsCompl.inf_eq_bot]
      exact (BilinForm.restrict_nondegenerate_iff_isCompl_orthogonal brefl).mp wnd
    have k₁ : W ⊔ (BilinForm.orthogonal β W) = ⊤ := by
      ext x
      constructor
      · simp
      · simp
        let Wplus := W ⊔ β.orthogonal W
        have k₁₀ : Wplus = W ⊔ β.orthogonal W := by
            rfl
```

Lean Proof

```
        have k₁₁ : Module.finrank k (Wplus) = Module.finrank k V := by
          rw[k₁₀]
          rw[finrank_sup_eq_neg_finrank_inf_add]
          rw[k₀]
          simp
          rw[LinearMap.BilinForm.finrank_orthogonal]
          · rw[← add_comm]
            refine Nat.sub_add_cancel ?_
            apply Submodule.finrank_le
          · exact bnd
          · exact href
          · exact V
          · exact k
        apply Submodule.eq_top_of_finrank_eq at k₁₁
        rw[← k₁₀]
        rw[k₁₁]
        simp
    let b₁ : Basis ι₁ k W := Basis.ofVectorSpace k W
    let b₂ : Basis ι₂ k (BilinForm.orthogonal β W) :=
    Basis.ofVectorSpace k (BilinForm.orthogonal β W)
    let B : Basis (ι₁ ⊕ ι₂) k V := by
      apply basis_of_direct_sum
      · exact b₁
      · exact b₂
      · exact k₁
      · exact k₀
```

Lean Proof

```
let M : Matrix (ι₁ ⊕ ι₂) (ι₁ ⊕ ι₂) k := BilinForm.toMatrix B β
let M₁ := (M.toSquareBlockProp (p ↑ι₁ ↑ι₂))
let M₂ := (M.toSquareBlockProp fun i ↦ ¬p (↑ι₁) (↑ι₂) i)
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
  rw[LinearMap.BilinForm.mem_orthogonal_iff] at g₁
  rw[BilinForm.toMatrix_apply]
  exact href (B y) (B x) (href (B x) (B y) (href (B y) (B x) (g₁ (B y) g₀)))
have k₃ : M.det = M₁.det * M₂.det := by
  rw[Matrix.twoBlockTriangular_det M (p ι₁ ι₂) k₂]
```

Lean Proof

```
have k₄ : M₂ = (BilinForm.toMatrix b₂ (β.restrict (BilinForm.orthogonal β W))) := by
  sorry
have k₅ : M₂.det ≠ 0 := by
  intro h
  rw[h] at k₃
  rw[mul_zero] at k₃
  have k₅₀ : M.det ≠ 0 := by
    exact (BilinForm.nondegenerate_iff_det_ne_zero B).mp bnd
  exact k₅₀ k₃
unfold Nondeg_subspace
rw[k₄] at k₅
apply Matrix.nondegenerate_of_det_ne_zero at k₅
exact (BilinForm.nondegenerate_toMatrix_iff b₂).mp k₅
```

Tensor Products

Let R be a ring, M_R a right R-module and ${}_RN$ a left R-module. The tensor product $M \otimes_R N$ is an abelian group such that

$$\mathsf{Bil}(M, N; P) \simeq \mathsf{Hom}(M \otimes_R N, P)$$

for all abelian groups P. If N is also a right S-module then one can define a right S-module structure on $M \otimes_R N$ such that for all right S-modules P,

$$\mathsf{Hom}_S(M \otimes_R N, P) \simeq \mathsf{Hom}_R(M, \mathsf{Hom}_S(N, P)).$$

The elements of form $m \otimes n$ generates $M \otimes_R N$.

Extension by scalars

Let F be a field, V a vector space over F, and K/F a field extension. Then we define

$$V_K := K \otimes_F V$$

which we call V extended by scalars from K. If $\mathcal{B}$ is a basis for V, then $\mathcal{B}_K := \{1 \otimes v : v \in \mathcal{B}\}$ is a basis for V_K.
If $A \in F^{n \times n}$ defines an F-linear map $V \to V$ with basis $\mathcal{B}$, then A viewed in $K^{n \times n}$ defines a corresponding K-linear map $V_K \to V_K$ with basis $\mathcal{B}_K$. These maps commute with the natural map $V \to V_K$.

Quadratic Forms

A quadratic form ϕ is a map $V \to F$ that satisfies

- $\phi(a \cdot v) = a^2 \cdot \phi(v)$
- $(v, w) \mapsto \phi(v + v) - \phi(v) - \phi(w)$ is bilinear

Given any symmetric bilinear form $B : V \times V \to F$, $\phi(v) := B(v, v)$ is a quadratic form. This gives us maps

$$\mathsf{Quad}(V) \to \mathsf{Bil}(V)$$

$$\mathsf{Bil}(V) \to \mathsf{Quad}(V)$$

which are inverses up to scaling by 2.

Quadratic Forms (Extension by Scalars)

Given $B : V \times V \to F$ we can define $B_K : V_K \times V_K \to K$. Hence a quadratic form $\phi : V \to F$ can be extended $V_K \to K$.

When $K = F(t)$ we write $\phi_{F(t)}$ and $V(t) := V_{F(t)}$. Similarly for $\phi_{F[t]}$ and $V[t] := V_{F[t]}$.

Theorem

Cassels-Pfister Theorem

Let ϕ be a Quadratic form on V. Then $\text{im}(\phi_{F(t)}) \cap F[t] = \text{im}(\phi_{F[t]})$.

Corollary

Let $f \in F[t]$ be a sum of squares in $F(t)$. Then f is a sum of n squares in $F[t]$.

```
/-- The values taken by the extension of a quadratic map `ϕ: V → F` to `V(X) → F(X)`
    that are in `F[X]` are taken by the extension `V[X] → F[X]` as well.
-/
theorem CasselsPfisterTheorem (ϕ: QuadraticForm F V) [Invertible (2: F)]:
  (↑)⁻¹' (Set.range (ϕ.baseChange (F(X))))
    = Set.range (ϕ.baseChange F[X]) := ...
```

Proof

First proved by Cassels in 1964, then generalized by Pfister in 1965.

Tools used in proof

- extension of scalars
- extension of quadratic forms
- hyperbolic 2 space
- degree over polynomial module
- conversions between structures

Extension by scalars in Mathlib

Tensor product of modules over commutative semirings.

This file constructs the tensor product of modules over commutative semirings. Given a semiring `R` and modules over it `M` and `N`, the standard construction of the tensor product is `TensorProduct R M N`. It is also a module over `R`.

It comes with a canonical bilinear map `TensorProduct.mk R M N : M →ₗ[R] N →ₗ[R] TensorProduct R M N`.

Given any bilinear map `f : M →ₗ[R] N →ₗ[R] P`, there is a unique linear map `TensorProduct.lift f : TensorProduct R M N →ₗ[R] P` whose composition with the canonical bilinear map `TensorProduct.mk` is the given bilinear map `f`. Uniqueness is shown in the theorem `TensorProduct.lift.unique`.

Notation

- This file introduces the notation `M ⊗ N` and `M ⊗[R] N` for the tensor product space `TensorProduct R M N`.
- It introduces the notation `m ⊗ₜ n` and `m ⊗ₜ[R] n` for the tensor product of two elements, otherwise written as `TensorProduct.tmul R m n`.

Extension of quadratic forms by scalars in Mathlib

Mathlib.LinearAlgebra.QuadraticForm.TensorProduct

```
def QuadraticForm.baseChange                                                    source
      {R : Type uR} (A : Type uA) {M₂ : Type uM₂} [CommRing R] [CommRing A]
      [AddCommGroup M₂] [Algebra R A] [Module R M₂] [Invertible 2]
      (Q : QuadraticForm R M₂) :
  QuadraticForm A (TensorProduct R A M₂)
```

The base change of a quadratic form.

▶ Equations

Polynomial Module equivalence

We can view $V[t] = F[t] \otimes_F V$ similarly to (formal) polynomials as sums $v_0 + v_1 X^1 + v_2 X^2 + \dots$.

This interpretation is implemented in Mathlib as **PolynomialModule** $F\ V$, with data $\mathbb{N} \to_0 V$.

```
/-- There is an `R[X]` linear equivalence `(R[X] ⊗[R] M) ≃ₗ[R[X]]
    (PolynomialModule R M)` The `toFun` construction comes from
    `TensorMap`. The `left_inv` and `right_inv` conditions are proved
    using `TensorProduct.induction_on`, `Polynomial.induction_on` and
    `PolynomialModule.induction_on`
--/
def PolynomialModuleEquivTensorProduct  :
  (R[X] ⊗[R] M) ≃ₗ[R[X]] (PolynomialModule R M) := by
```

Degree

Degree is fairly easy to define over $\mathbb{N} \to_0 V$ as the 'maximum' or 'supremum' of the support. It would be much more difficult to define degree on $F[t] \otimes_F V$.

```
def Polynomial.degree                                              source
      {R : Type u} [Semiring R] (p : Polynomial R) :
    WithBot ℕ
```

degree p is the degree of the polynomial p, i.e. the largest X-exponent in p. degree p = some n when p ≠ 0 and n is the highest power of X that appears in p, otherwise degree 0 = ⊥.

▼ Equations

- `p.degree = p.support.max`

The downward chain

We have a map $(\mathbb{N} \to_0 V) \to F[t] \otimes_F V$ from the prior equivalence. We also have a natural map $F[t] \to F(t)$ (from Mathlib), which gives us a map $F[t] \otimes_F V \to F(t) \otimes_F V$.

$$
\begin{tikzcd}
\text{PolynomialModule F V} \arrow[d, shift right] \\
F[t] \otimes_F V \arrow[u, shift right] \arrow[d] \\
F(t) \otimes_F V
\end{tikzcd}
$$

Implicit Conversions

```
@[coe]
noncomputable def coeRatFunc: V[F] → V(F) := toRatFuncTensor

noncomputable scoped instance : Coe (V[F]) (V(F)) := ⟨coeRatFunc⟩
```

```
@[coe]
noncomputable abbrev coePolynomialModule: PolynomialModule F V → V[F] := PolynomialEquiv

noncomputable scoped instance : Coe (PolynomialModule F V) (V[F]) := ⟨coePolynomialModule⟩
```

Notation

```
scoped[RationalFunctionFields] notation:9000 F "(X)" => RatFunc F
scoped[RationalFunctionFields] notation:1000 V "[" F:1000 "]" => F[X] ⊗[F] V
scoped[RationalFunctionFields] notation:1000 V "(" F:1000 ")"  => F(X) ⊗[F] V
```

References

1. Conrad, K. (n.d.). Bilinear Forms. https://kconrad.math.uconn.edu/blurbs/linmultialg/bilinearform.pdf
2. Elman RS, Karpenko N, Merkurjev A. The Algebraic and Geometric Theory of Quadratic Forms. American Mathematical Society; 2008.

Thank you!

Special thanks to Dr. George McNinch and the REU