Proofs of Properties of finite-dimensional Vector Spaces using Isabelle/HOL

Jose Divasón Mallagaray
Supervised by Jesús María Aransay Azofra

Universidad de La Rioja
Departamento de Matemáticas y Computación

Seminario de Informática "Mirian Andrés"
Logroño, 25 de Octubre de 2011

INTRODUCTION

Project

1. The objective of this project is to formalize concepts and theorems of linear algebra, concretly of vector spaces, using Isabelle/HOL.
2. We have followed a Halmos' book: *Finite-dimensional vector spaces*.
3. The project has been written in English.

We will try to formalize the first 16 sections in Halmos:

Sections

1. Fields
2. Vector Spaces
3. Examples
4. Comments
5. Linear Dependence
6. Linear Combinations
7. Bases
8. Dimension

Sections

9. Isomorphism
10. Subspaces
11. Calculus of Subspaces
12. Dimension of a Subspace
13. Dual Spaces
14. Brackets
15. Dual Bases
16. Reflexivity

Main Theorems

Theorem 1

Every linearly independent set can be extended to a basis.

Theorem 2

Any two finite bases of a finite dimensional vector space have the same cardinality.

Theorem 3

An n-dimensional vector space V over a field $\mathbb{K}$ is isomorphic to $\mathbb{K}^n$.

Theorem 4

There exists an isomorphism between a vector space V and the dual space of its dual.

Isabelle

- **Isabelle:** The theorem proving assistant in which we have made the development.
- **Isar:** Intelligible semi-automated reasoning.
- **HOL:** Higher-order logic.
- **HOL-Algebra:** A library of linear algebra implemented in Isabelle using HOL.
- **Locales:** A kind of module in which we can fix variables and declare assumptions.

Example of Isabelle code

```
locale vector_space = K: field K + V: abelian_group V
  for K (structure) and V (structure) +
  fixes scalar_product:: "'a => 'b => 'b" (infixr "·" 70)
  assumes mult_closed: "⟦x ∈ carrier V;a ∈ carrier K⟧
  ⟹ a · x ∈ carrier V"
  and mult_assoc: "⟦x ∈ carrier V; a ∈ carrier K; b ∈ carrier K⟧
  ⟹ (a ⊗K b) · x = a · (b · x)"
  and mult_1: "⟦x ∈ carrier V⟧ ⟹ 1K · x = x"
  and add_mult_distrib1:
  "⟦x∈ carrier V; y ∈ carrier V; a ∈ carrier K⟧
  ⟹ a · (x ⊕V y)= a·x ⊕V a·y"
  and add_mult_distrib2:
  "⟦x∈ carrier V; a ∈ carrier K; b ∈ carrier K⟧
  ⟹ (a ⊕K b) · x = a·x ⊕V b·x"
```

INDEXED SETS

Indexed Sets

1. In mathematics, we usually represent a set of n elements this way:

$$A = \{a_1, \ldots, a_n\}$$

2. Really a set doesn't have an order by default (but we can give one for it).
3. This is not important...unless the order has influence on the proof.

- We have implemented the type *indexed set* as a pair of a set and a function that goes from naturals to the set:

type_synonym *('a) iset = "'a set × (nat => 'a)"*

- An indexing of a set will be any bijection between the set of the natural numbers less than its cardinality (because we start counting from 0) and the set:

inj_on f A = (∀x∈A. ∀y∈A. f x = f y ⟶ x = y)

bij_betw f A B = (inj_on f A ∧ f ` A = B)

definition *indexing :: "('a iset) => bool"*
  where *"indexing (A,f) = bij_betw f {..<card (A)} A"*

We have defined operations to insert and remove one element of an indexed set:

definition *indexing_ext :: "('a iset) => 'a => (nat => nat => 'a)"*
  where *"indexing_ext (A,f) a = (λn. λk. if k < n then f k*
  else if k = n then a else f (k - 1))"

definition *insert_iset :: "'a iset => 'a => nat => 'a iset"*
  where *"insert_iset (A,f) a n*
  = (insert a A, indexing_ext (A,f) a n)"

definition *remove_iset :: "'a iset => nat => 'a iset"*
  where *"remove_iset (A,f) n = (A - {f (n)},*
  (λk. if k < n then f (k) else f (k + 1)))"

We present an induction rule created to prove theorems and properties of indexed sets:

```
lemma
  indexed_set_induct2 [case_names indexing finite empty insert]:
  assumes "indexing (A, f)"
  and "finite A"
  and "∀f. indexing ({}, f) ==> P {} f"
  and step: "∀a A f n. [|a ∉ A;
            [| indexing (A, f) |] ==> P A f;
            finite (insert a A);
            indexing ((insert a A), (indexing_ext (A, f) a n));
            0 ≤ n; n ≤ card A |] ==>
            P (insert a A) (indexing_ext (A, f) a n)"
  shows "P A f"
  using `finite A` and `indexing (A, f)`
proof (induct arbitrary: f)
...
qed
```

THEOREM 1

Previous Result

If the set of non-zero vectors $x_1, \ldots, x_n$ is linearly dependent, then there exists at least one x_k, $2 \leq k \leq n$, which is a linear combination of the preceding ones.

Note that the given order is very important, so the use of indexed sets is indispensable.

Theorem 1

Every linearly independent set of a finite vector space V can be extended to a basis.

Theorem 1

Every linearly independent set of a finite vector space V can be extended to a basis.

- Let $A = \{a_1, \dots, a_n\}$ an independent set and $B = \{b_1, \dots, b_m\}$ a basis of V. We apply the previous result to the set: $C = \{ \underbrace{a_1, \dots, a_n}_{Elements\ of\ A}, \underbrace{b_1, \dots, b_m}_{Elements\ of\ B} \}$
- Since the first n elements are in an independent set (they are contained in A), hence the element which is a linear combination of the preceding ones is in B.
- Let b_i that element, then we remove it and we obtain: $C' = \{a_1, \dots, a_n, b_1, \dots, b_{i-1}, b_{i+1}, \dots b_m\}$
- If C' is independent we have already finished (the basis is C'), if not we iterate the process.

PROBLEMS

- $C = \{ \overbrace{a_1, \ldots, a_n}^{Elements\ of\ A}, \overbrace{b_1, \ldots, b_m}^{Elements\ of\ B} \}$ could be a multiset.
  SOLUTION: $C = A \cup (B - A)$.
- There could be some elements of B which are linear combination of the preceding ones (there is no unicity). **SOLUTION:** Take the least.
- The iterative reasonings are hard to be implemented in Isabelle. The functions in HOL are total. **SOLUTION:** Partial functions (tail recursive).

We define two functions: *remove_ld* and *iterate_remove_ld*.

- The first one removes the least element of a dependent set which is a linear combination of the preceding ones.

```
definition remove_ld :: "'c iset => 'c iset"
  where "remove_ld (A,f) =
  (let n = (LEAST k::nat. ∃y ∈ A. ∃g.
  g ∈ coefficients_function (carrier V)
  ∧ (1::nat) ≤ k ∧ k < (card (A))
  ∧ f k = y
  ∧ y = linear_combination g (f ` {i::nat. i<k}))
  in remove_iset A n)"
```

- The second one iterates the previous function until achieving an independent set.

```
partial_function (tailrec) iterate_remove_ld :: "'c set => (nat
=> 'c) => 'c set"
  where "iterate_remove_ld A f
  = (if linear_independent A then A
  else iterate_remove_ld (fst (remove_ld (A, f)))
  (snd (remove_ld (A, f))))"
```

There are three important results which *iterate_remove_ld* must satisfy to demonstrate the theorem:

1. The result is a linearly independent set (about 100 lines).
2. The result is a spanning set (about 130 lines).
3. The independent set A is contained in the result of the function (about 350 lines).

The total number of lines necessary to prove this theorem were 984.

```
lemma extend_not_empty_independent_set_to_a_basis:
  assumes "linear_independent A"
  and "A≠{}"  shows "∃S. basis S ∧ A ⊆ S"
proof -
  def X ≡"B-A"
  have "linear_independent(iterate_remove_ld (A∪X) h)"
  proof (rule linear_independent_iterate_remove_ld)
    ...
  qed
  have "span(iterate_remove_ld (A∪X) h)=carrier V"
  proof (rule iterate_remove_ld_preserves_span)
    ...
  qed
  have "A ⊆ (iterate_remove_ld (A∪X) h)"
  proof (rule A_in_iterate_remove_ld)
    ...
  qed
  ...
qed
```

THEOREM 2

Swap theorem

If A is a linearly independent set of V and B is any spanning set of V, then $card(A) \leq card(B)$.

Corollary: theorem 2

Any two finite bases of a finite dimensional vector space have the same cardinality.

$$swap\_function\ (\{a_1, \ldots, a_n\} \times \{b_1, \ldots, b_m\}) \\ = (\{a_2, \ldots, a_n\} \times \{a_1, b_1, \ldots, b_{i-1}, b_{i+1}, \ldots, b_m\})$$

Where b_i is the first element which is a linear combination of the preceding ones $(a_1, \ldots, b_{i-1})$.

This function satisfies, amongst others, the following properties:

- It preserves the linear independence in the first component.
- It preserves the span in the second component.

swap_function $((A,f) \times (B,g))$:

- **First component:**
  - We remove the first element of A, in other words: the function returns the set $A - \{a_1\}$ (and the corresponding indexation).
- **Second component:**
  - If $a_1 \in B$ then simply we change the indexation moving that element to the first position of B.
  - If $a_1 \notin B$, then we add it in the first position of B and after that we will remove the first element which is a linear combination of the preceding ones using the function *remove_ld*.

The implementation in Isabelle:

```
definition swap_function :: "('c iset × 'c iset)
  => ('c iset × 'c iset)"
  where "swap_function ((A,f),(B,g)) = (remove_iset_0 A,
  if f 0 ∈ B then
  insert_iset (remove_iset (B,g) (obtain_position (f 0) B))
  (f 0) 0
  else remove_ld (insert_iset (B,g) (f 0) 0))"
```

Swap theorem

If A is a linearly independent set of V and B is any spanning set of V, then $card(A) \leq card(B)$.

- Suppose that $card(A) > card(B)$ and then we apply *swap_function* $card(B)$ times.
- We will obtain that in the second component of the result there will be only elements of A (but not all). This is because we will have removed *card* B elements of B in the second component (one in each iteration, so we will have removed all elements of B).

$$swap\_function^{card(B)} \left(\{a_1, \ldots, a_n\} \times \{b_1, \ldots, b_m\}\right) \\ = \left(\{a_{card(B)+1}, \ldots, a_n\} \times \{\underbrace{a_1, \ldots, a_{card(B)}}_{C}\}\right)$$

- Let be C that set, we will have:
  - $C \subset A$ (strict).
  - $span(C) = V$ (because the second component was a spanning set and the function preserves the span). So C is a spanning set.
- Let be $x \in A$ but $x \notin C$ (this element exists because $C \subset A$ strictly). As C is a spanning set, we can express x as a linear combination of elements of C.
- However, this is a contradiction with A being linearly independent (because $C \cup \{x\}$ would be linearly dependent and as $C \cup \{x\} \subseteq A$ then A would be dependent).

PROBLEMS

- We can't follow a similar reasoning than in theorem 1 to prove the result: now we need to have control in the number of iterations.
- Need to separate in cases the function to avoid a multiset again and to be able to apply *remove_ld*.
- We have to make use of the power of a function...however, this is not implemented in Isabelle. We have to make it:

```
instantiation "fun" :: (type, type) power
begin
definition one_fun :: "'a => 'a"
  where one_fun_def: "one_fun = id"
definition times_fun :: "('a => 'a) => ('a => 'a) => 'a => 'a"
  where "times_fun f g = (∀x. f (g x))"
instance
proof
qed
end
```

- Once we have defined the power of a function, we have to prove the properties that *swap_function* satisfies in case that we apply the function once and after that generalize them using induction. The following lemma is indispensable:

```
corollary fun_power_suc_eq:
  shows "(f^(n+1)) x = f ((f^n) x)"
  using fun_power_suc by (metis id_o o_eq_id_dest)
```

- This is a long and tedious process: the proofs of all necessary properties and lemmas to make the demonstration take up 1800 lines.

THEOREM 3

What is $\mathbb{K}^n$?

Definition of $\mathbb{K}^n$

$$\mathbb{K}^n = \underbrace{\mathbb{K} \times \mathbb{K} \times \cdots \times \mathbb{K}}_{n} = \{(x_1, \ldots, x_n) | x_i \in \mathbb{K}\ \forall i, 1 \leq i \leq n\}$$

And in Isabelle?

- First we define the type vector, a pair of a function and a natural:
  types *'a vector = "(nat => 'a) * nat"*
  - The function maps naturals to elements of a set.
  - The natural is the length of the vector minus one.
  - **Example:** To represent (a_1, a_2, a_3, a_4) we have a vector $(f, 3)$ where $f(0) = a_1$, $f(1) = a_2$, $f(2) = a_3$ and $f(3) = a_4$.
  - **Problem:** we don't have unicity of representation.

- **definition** *K_n_carrier :: "'a set => nat => ('a vector) set"*
  where *"K_n_carrier A n = {v. ((∀i<n. ith v i ∈ A))*
  ∧ (∀i≥n. ith v i = **0***) ∧ (vlen v = (n - 1))}"*
- **definition**
  K_n_add :: " nat => 'a vector => 'a vector => 'a vector"
  (infixr *"⊕ı" 65)*
  where *"K_n_add n = (λv w. ((λi. ith v i* $\oplus_R$ *ith w i), n - 1))"*
- **definition** *K_n_zero :: "nat => 'a vector"*
  where *"K_n_zero n = ((λi.* $\mathbf{0}_R$*), n - 1)"*
- **definition** *K_n_mult :: "nat => 'a vector => 'a vector => 'a vector"*
  where *"K_n_mult n = (λv w. ((λi. ith v i* $\otimes_R$ *ith w i), n - 1))"*
- **definition** *K_n_one :: "nat => 'a vector"*
  where *"K_n_one n = ((λi.* $\mathbf{1}_R$*), n - 1)"*

Definition of $\mathbb{K}^n$ in Isabelle

Finally using the definition of carrier, add, zero, mult and one we can define the concept of $\mathbb{K}^n$:

```
definition K_n :: "nat => 'a vector ring"
  where
  "K_n n = (| carrier = K_n_carrier (carrier R) n,
              mult = (λv w. K_n_mult n v w),
              one = K_n_one n,
              zero = K_n_zero n,
              add = (λv w. K_n_add n v w)|)"
```

We need to check that $\mathbb{K}^n$ is a vector space, so we need to define its scalar product: $a \odot (b_1, \ldots, b_n) = (a \cdot b_1, \ldots, a \cdot b_n)$

```
definition K_n_scalar_product :: "'a => 'a vector => 'a vector"
(infixr "⊙" 65) where "a ⊙ b = (λn::nat. a ⊗R ith b n, vlen b)"

lemma vector_space_K_n:
shows "vector_space R (K_n n) (op ⊙)"
unfolding K_n_def
proof (intro vector_spaceI)
```

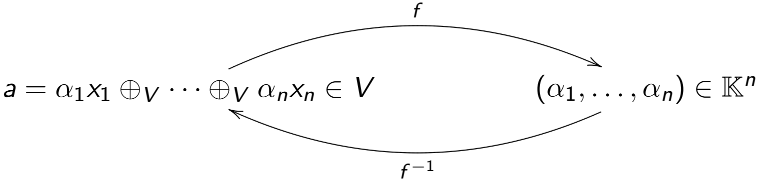
Definition of isomorphism between vector spaces

Two vector spaces V and W over the same field $\mathbb{K}$ are isomorphic if there exists a linear map $f\colon V \to W$ such that is a bijection.

Theorem 3

An n-dimensional vector space V over a field $\mathbb{K}$ is isomorphic to $\mathbb{K}^n$.

Let $X = \{x_1, \ldots, x_n\}$ be a basis of V. The isomorphism between V and $\mathbb{K}^n$ is easy to understand:



FROM $\mathbb{K}^n$ TO V

- Let $(\alpha_1, \ldots, \alpha_n)$ be a vector of $\mathbb{K}^n$. Hence, the corresponding $a \in V$ will be $a = \alpha_1 x_1 \oplus_V \cdots \oplus_V \alpha_n x_n$.
- How could we make it in Isabelle? We use that $\{x_i\}_{i \in \{1 \ldots n\}}$ are a basis and thus every $a \in V$ can be uniquely determined as the finite sum $\sum_{i=1}^{n} \alpha_i x_i = \alpha_1 x_1 \oplus_V \cdots \oplus_V \alpha_n x_n = a$. We only have to multiply each component of $(\alpha_1, \ldots, \alpha_n)$ with the corresponding element of the basis $X = \{x_1, \ldots, x_n\}$ and finally sum all again to obtain the linear combination which will be equal to a:
- In order to do that we will define a function named *iso_K_n_V*. To terminate the proof we have to demonstrate that this function is also a *linear map*.

```
definition iso_K_n_V :: "'a vector => 'c"
  where "iso_K_n_V x
  = finsum V (λi. fst x i · indexing_X i) {..<dimension}"
```

FROM V TO $\mathbb{K}^n$

- Let $a \in V$. We know that we can express it as a linear combination of the elements of the basis and this linear combination is unique: $a = \alpha_1 x_1 \oplus_V \cdots \oplus_V \alpha_n x_n$. The corresponding element in $\mathbb{K}^n$ is $(\alpha_1, \ldots, \alpha_n)$.
- Hence we need to manage to represent $(\alpha_1, \ldots, \alpha_n)$ using that $a = \alpha_1 x_1 \oplus_V \cdots \oplus_V \alpha_n x_n$. We will do it in the next way, we can write $(\alpha_1, \ldots, \alpha_n)$ as a finite sum of elements of the canonical basis of $\mathbb{K}^n$: $(\alpha_1, \ldots, \alpha_n) = \alpha_1 \cdot (1, 0, \ldots, 0) \oplus_{\mathbb{K}^n} \cdots \oplus_{\mathbb{K}^n} \alpha_n \cdot (0, \ldots, 0, 1)$
- So we have to take the scalars of the linear combination of the elements of the basis of V ($\{x_1, \ldots, x_n\}$) for a and multiply them (with the scalar product of $\mathbb{K}^n$) with the corresponding vector of the canonical basis. Finally we will sum all to obtain $(\alpha_1, \ldots, \alpha_n)$.

```
definition iso_V_K_n :: "'c => 'a vector"
  where "iso_V_K_n x =
  finsum (K_n dimension) (λi. (K_n_scalar_product (lin_comb (x)
(indexing_X i)) (x_i i dimension))) {..<dimension}"
```

MANAGEMENT

Task	Estimated time
Previous learning	12
Fields	17
Vector spaces	2.4
Examples	3.6
Comments	36
Linear dependence	24
Linear combinations	45
Basis	23.5
Dimension	25.25
Isomorphism	29.5
Subspaces	10.5
Calculus of subspaces	17
Dimension of a subspace	9.5
Dual spaces	12
Brackets	1
Dual bases	21.5
Reflexivity	10
Documentation	140
TOTAL HOURS	439.75

File	Lines
Previous	55
Field2	326
Vector_Space	42
Examples	57
Comments	329
Linear_dependence	532
Linear_combinations	1921
Indexed_set	1226
Basis	1962
Dimension	2235
Isomorphism	3465
Subspaces	234
TOTAL	12387

CONCLUSIONS AND FURTHER WORK

CONCLUSIONS

- Formalization requires a steep learning curve.
- Proofs in a book are not fully formal.
- Comparison between the length in the book and the formalized proof.
- Iterative proofs vs rewritting proofs.

FURTHER WORK

- To continue with the development of the following sections in Halmos.
- ForMath project.

THANKS FOR YOUR ATTENTION.

The complete Isabelle code and the memoir are available in

`www.unirioja.es/cu/jodivaso`