Vector-Spaces

Jose Divasón Mallagaray

October 26, 2011

Contents

theory *Previous*
imports *Main*
begin

1 Previous general results

We present here some result and theorems which will be used in our develoepement. There are general properties, not centered in any section of our implementation.

lemma *less-than-Suc-union*:
shows $\{i.\ i < Suc\ (n::nat)\} = \{i.\ i < n\} \cup \{n\}$
⟨*proof*⟩

Next two lemmas is a non-elegant trick which makes possible work with premisses that contains multiples *op* $\wedge$

lemma *conjI3*: $[\![A;\ B;\ C]\!] \Longrightarrow A \wedge B \wedge C$
⟨*proof*⟩

lemma *conjI4*: $[\![A;\ B;\ C;\ D]\!] \Longrightarrow A \wedge B \wedge C \wedge D$
⟨*proof*⟩

lemma *conjI5*: $[\![A;\ B;\ C;\ D;\ E]\!] \Longrightarrow A \wedge B \wedge C \wedge D \wedge E$
⟨*proof*⟩

lemma *conjI6*:
shows $[\![A;\ B;\ C;\ D;\ E;\ F]\!] \Longrightarrow A \wedge B \wedge C \wedge D \wedge E \wedge F$
⟨*proof*⟩

Next lemmas prove some properties of the bijections between subsets of a given set.

lemma *bij-betw-subset*:
assumes *b*: *bij-betw f A B* **and** *sb*: $C \subseteq A$
shows *bij-betw f C* (*f* ‘ *C*)
⟨*proof*⟩

lemma
bij-betw-image-minus:
assumes *b*: *bij-betw f A B* **and** *a*: $a \in A$
shows *f* ‘ $(A - \{a\}) = B - \{f\ a\}$
⟨*proof*⟩

end
theory *Field2*
imports *Previous*
~~/*src*/*HOL*/*Algebra*/*Ring*
begin

2 Previous relations between algebraic structures.

We can create a lemma to check if one algebraic structure is a domain.

lemma *domainI*:
fixes *R* (**structure**)
assumes *cring*: *cring R*
and *one-not-zero*: $\mathbf{1} \sim= \mathbf{0}$
and *integral*: $\bigwedge a\ b$. [| $a \otimes b = \mathbf{0}$; $a \in$ *carrier R*; $b \in$ *carrier R* |] ==> $a = \mathbf{0} \mid b = \mathbf{0}$
shows *domain R*
⟨*proof*⟩

Similarly with a field:

lemma *fieldI*:
fixes *R* (**structure**)
assumes *dom*: *domain R*
and *field-Units*: *Units R* = *carrier R* − {**0**}
shows *field R*
⟨*proof*⟩

A field is an additive monoid

lemma (**in** *field*) *field-impl-monoid*:
monoid (| *carrier* = *carrier R*, *mult* = *add R*, *one* = *zero R* |)
⟨*proof*⟩

A field is a multiplicative monoid:

lemma *field-is-monoid*: **fixes** *K* (**structure**)
assumes *field-K*: *field K* **shows** *monoid K*

⟨*proof*⟩

Every field is a ring

lemma *field-is-ring*: **fixes** K (**structure**)
  assumes *field-K*: *field* K **shows** *ring* K
⟨*proof*⟩

2.1 Previous properties

First of all we are going to introduce some properties of fields. Most of them are also satisfied in rings and in previous algebraic structures, so they will be trivial for us.

This property is trivial and proved in the library:

lemma (**in** *field*) *r-zero*:
  $x \in$ *carrier* R ==> $x \oplus \mathbf{0} = x$
⟨*proof*⟩

However, we can make a long proof of the preceding fact.

lemma (**in** *field*) *r-zero2*: $x \in$ *carrier* R ==> $x \oplus \mathbf{0} = x$
⟨*proof*⟩
    print-facts
    ⟨*proof*⟩
  find-theorems *?x* $\oplus$ *?y* = *?y* $\oplus$ *?x*
  ⟨*proof*⟩

This is also in the library (for commutative groups):

lemma (**in** *field*) *a-comm*:
  !! x y. $[\![x \in$ *carrier* R; $y \in$ *carrier* $R]\!] \Longrightarrow x \oplus y = y \oplus x$
  ⟨*proof*⟩

But we can prove it: we have that the property is satisfied in a commutative group. We will prove that a field is a commutative group and then we will use the property.

lemma (**in** *field*) *a-comm2*:
  !! x y. $[\![x \in$ *carrier* R; $y \in$ *carrier* $R]\!] \Longrightarrow x \oplus y = y \oplus x$
⟨*proof*⟩

lemma (**in** *field*) *a-assoc*:
!! x y z. $[\![x \in$ *carrier* R; $y \in$ *carrier* R; $z \in$ *carrier* $R]\!] \Longrightarrow (x \oplus y) \oplus z = x \oplus (y \oplus z)$
⟨*proof*⟩

lemma (**in** *field*) *r-neg*:
$x \in$ *carrier* $R \Longrightarrow x \oplus (\ominus x) = \mathbf{0}$
⟨*proof*⟩

lemma (**in** *field*) *m-comm*:
!! $x\ y$. $[\![\ x \in carrier\ R;\ y \in carrier\ R]\!] \Longrightarrow x \otimes y = y \otimes x$
⟨*proof*⟩

lemma (**in** *field*) *m-assoc*:
!! $x\ y\ z$. $[\![x \in carrier\ R;\ y \in carrier\ R;\ z \in carrier\ R]\!] \Longrightarrow (x \otimes y) \otimes z = x \otimes (y \otimes z)$
⟨*proof*⟩

lemma (**in** *field*) *r-one*:
$x \in carrier\ R \Longrightarrow x \otimes \mathbf{1} = x$
⟨*proof*⟩

lemma (**in** *field*) *r-inv*:
$x \in Units\ R \Longrightarrow x \otimes inv\ x = \mathbf{1}$
⟨*proof*⟩

lemma (**in** *field*) *r-distr*:
$[\![x \in carrier\ R;\ y \in carrier\ R; z \in carrier\ R]\!] \Longrightarrow x \otimes (y \oplus z) = x \otimes y \oplus x \otimes z$
⟨*proof*⟩

lemma (**in** *field*) *l-one*:
$x \in carrier\ R \Longrightarrow \mathbf{1} \otimes x = x$
⟨*proof*⟩

2.2 Exercises in Halmos

Definition of field and some properties are already included in the library, so we don't make it.

Here we present some exercises proposed by Halmos. There are someone already solved in the library, so they will be trivial for us.

Exercise 1A

lemma (**in** *field*) *l-zero*:
$x \in carrier\ R \Longrightarrow \mathbf{0} \oplus x = x$
⟨*proof*⟩

Exercise 1B

lemma (**in** *field*) *a-l-cancel*:
$[\![x \in carrier\ R;\ y \in carrier\ R; z \in carrier\ R]\!] \Longrightarrow (x \oplus y = x \oplus z) = (y = z)$
⟨*proof*⟩

Exercise 1C

lemma (**in** *field*) *plus-minus-cancel*:
$[\![x \in carrier\ R;\ y \in carrier\ R]\!] \Longrightarrow x \oplus (y \ominus x) = y$
⟨*proof*⟩

Corollary of 1C. It is in the library.

corollary (**in** *field*) *minus-eq*:
$[\![y \in \mathit{carrier}\ R;\ x \in \mathit{carrier}\ R]\!] \Longrightarrow y \ominus x = y \oplus (\ominus x)$
⟨*proof*⟩

Exercise 1D

lemma (**in** *field*) *r-null*:
$x \in \mathit{carrier}\ R \Longrightarrow x \otimes \mathbf{0} = \mathbf{0}$
⟨*proof*⟩

lemma (**in** *field*) *l-null*:
$x \in \mathit{carrier}\ R \Longrightarrow \mathbf{0} \otimes x = \mathbf{0}$
⟨*proof*⟩

Exercise 1E

lemma (**in** *field*) *l-minus-one*:
$x \in \mathit{carrier}\ R \Longrightarrow (\ominus \mathbf{1}) \otimes x = \ominus x$
⟨*proof*⟩

Exercise 1F

lemma (**in** *field*) *prod-minus*:
assumes *x-in-R*: $x \in \mathit{carrier}\ R$
and *y-in-R*: $y \in \mathit{carrier}\ R$
shows $(\ominus x) \otimes (\ominus y) = x \otimes y$
⟨*proof*⟩

Exercise 1G

This exercise can be solved directly using integral property. However we will make it using $\mathit{Units} = \mathit{carrier}\ R - \{\mathbf{0}_R\}$. This is because field would not need to be derived from domain, the properties for domain follow from the assumptions of field (if we consider a field like a commutative ring in which $\mathit{Units} = \mathit{carrier}\ R - \{\mathbf{0}_R\}$

lemma (**in** *field*) *integral*:
assumes *x-y-eq-0*: $x \otimes y = \mathbf{0}$
and *x-in-R*: $x \in \mathit{carrier}\ R$
and *y-in-R*: $y \in \mathit{carrier}\ R$
shows $x = \mathbf{0} \mid y = \mathbf{0}$
⟨*proof*⟩

end

theory *Vector-Space*
imports *Field2*
begin

3 Definition of Vector Space

Here the definition of a vector space using locales and inherit. We need to fix a field, an abelian group and the scalar product relating both structures (an abelian group together a field would be a vector space with one specific scalar product but not with another). A vector space is an algebraic structure composed of a field, an abelian monoid and a scalar product which satisfies some properties.

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*: $[\![x \in carrier\ V; a \in carrier\ K]\!]$
$\Longrightarrow a \cdot x \in carrier\ V$
and *mult-assoc*: $[\![x \in carrier\ V;\ a \in carrier\ K;\ b \in carrier\ K\ ]\!]$
$\Longrightarrow (a \otimes_K b) \cdot x = a \cdot (b \cdot x)$
and *mult-1*: $[\![x \in carrier\ V]\!] \Longrightarrow \mathbf{1}_K \cdot x = x$
and *add-mult-distrib1*:
$[\![x \in carrier\ V;\ y \in carrier\ V;\ a \in carrier\ K]\!]$
$\Longrightarrow a \cdot (x \oplus_V y) = a \cdot x \oplus_V a \cdot y$
and *add-mult-distrib2*:
$[\![x \in carrier\ V;\ a \in carrier\ K;\ b \in carrier\ K]\!]$
$\Longrightarrow (a \oplus_K b) \cdot x = a \cdot x \oplus_V b \cdot x$

Using this lemma we can check if an algebraic structure is a vector space

lemma *vector-spaceI*:
fixes K (**structure**) **and** V (**structure**)
and *scalar-product* ::$'a$ => $'b$ => $'b$ (**infixr** · *70*)
assumes *field-K*: *field* K
and *abelian-group-V*: *abelian-group* V
and *mult-closed*:
$\bigwedge x\ a.\ [\![x \in carrier\ V; a \in carrier\ K]\!] \Longrightarrow a \cdot x \in carrier\ V$
and *mult-assoc*:
$\bigwedge x\ a\ b.\ [\![\ x \in carrier\ V;\ a \in carrier\ K;\ b \in carrier\ K\ ]\!]$
$\Longrightarrow (a \otimes_K b) \cdot x = a \cdot (b \cdot x)$
and *mult-1*: $\bigwedge x.\ [\![x \in carrier\ V]\!] \Longrightarrow \mathbf{1}_K \cdot x = x$
and *add-mult-distrib1*:
$\bigwedge x\ y\ a.\ [\![x \in carrier\ V;\ y \in carrier\ V;\ a \in carrier\ K]\!]$
$\Longrightarrow a \cdot (x \oplus_V y) = a \cdot x \oplus_V a \cdot y$
and *add-mult-distrib2*:
$\bigwedge x\ a\ b.\ [\![x \in carrier\ V;\ a \in carrier\ K;\ b \in carrier\ K]\!]$
$\Longrightarrow (a \oplus_K b) \cdot x = a \cdot x \oplus_V b \cdot x$
shows *vector-space K V scalar-product*
⟨*proof*⟩

end

theory *Examples*
imports *Vector-Space RealDef*
begin

4 Examples

context *vector-space*
begin

Here we show that every field is a vector space over itself (we interpret the scalar product as the ordinary multiplication of the field. We use make use of *vector-spaceI*.

lemma *field-is-vector-space*:
  assumes *field-K*: *field K*
  shows *vector-space K K op* $\otimes_K$
⟨*proof*⟩

end
end
theory *Comments*
imports *Examples*
begin

5 Comments

context *vector-space*
begin

Now some properties of vector spaces.

Halmos proposes some exercises, but most of them are properties already proved in abelian groups, rings... so they are in the library and using the inheritance of properties provided by locales we obtain them for vector spaces. Lemmas in which the scalar product appears need to be proved and we make it here.

We have two zeros: $\mathbf{0}_V$ and $\mathbf{0}$. We need to define separately the closure property in order to avoid confusions. Alternatively, we could specify the structure writing *V.zero-closed* and *K.zero-closed*.

lemma *zeroV-closed*: $\mathbf{0}_V \in$ *carrier V*
⟨*proof*⟩

lemma *zeroK-closed*: $\mathbf{0}_K \in$ *carrier K*
  ⟨*proof*⟩

A variation of *r-neg* ($x \in$ *carrier V* $\Longrightarrow x \oplus_V \ominus_V x = \mathbf{0}_V$):

lemma *r-neg'*:

assumes *x-in-V*: $x \in$ *carrier V*
shows $x \ominus_V x{=}\mathbf{0}_V$
⟨*proof*⟩

We want to prove that $a \cdot \mathbf{0}_V = \mathbf{0}_V$. First of all, we prove some auxiliary lemmas:

lemma *mult-zero-descomposition* [*simp*]:
assumes *a-in-K*: $a \in$ *carrier K*
shows $a \cdot \mathbf{0}_V \oplus_V a \cdot \mathbf{0}_V = a \cdot \mathbf{0}_V$
⟨*proof*⟩

lemma *plus-minus-assoc*:
assumes *x-in-V*: $x \in$ *carrier V*
and *y-in-V*: $y \in$ *carrier V* **and** *z-in-V*: $z \in$ *carrier V*
shows $x \oplus_V y \ominus_V z = x \oplus_V (y \ominus_V z)$
⟨*proof*⟩

Now we can complete theorem that we want to prove. It corresponds with exercise 1C in section 4 in Halmos.

lemma *scalar-mult-zeroV-is-zeroV*:
assumes *a-in-K*:$a \in$ *carrier K*
shows $a \cdot \mathbf{0}_V = \mathbf{0}_V$
⟨*proof*⟩

We apply a similar reasoning to prove that $\mathbf{0} \cdot x = \mathbf{0}_V$ (this corresponds with exercise 1D in section 4 in Halmos):

lemma *mult-zero-descomposition2*:
assumes *x-in-V*: $x \in$ *carrier V*
shows $\mathbf{0}_K \cdot x \oplus_V \mathbf{0}_K \cdot x = \mathbf{0}_K \cdot x$
⟨*proof*⟩

The exercise 1D in section 4 in Halmos is proved as follows:

lemma *zeroK-mult-V-is-zeroV*:
assumes *x-in-V*: $x \in$ *carrier V*
shows $\mathbf{0}_K \cdot x = \mathbf{0}_V$
⟨*proof*⟩

Another relevant property permit us to relate the additive inverse of the multiplicative unit with the additive inverse. It corresponds with exercise (1F) in section 4 in Halmos.

lemma *negate-eq*:
assumes *x-in-V*: $x \in$ *carrier V*
shows $(\ominus_K \ \mathbf{1}_K) \cdot x = \ominus_V x$
⟨*proof*⟩

The previous property can be proved not only for the multiplicative unit of $\mathbb{K}$ but for every element in its carrier. We redo the demonstration (the previous lemma could be proved as a corollary of this):

lemma *negate-eq2*:
assumes *x-in-V*: $x \in$ *carrier V*
and *a-in-K*: $a \in$ *carrier K*
shows $(\ominus_K\ a) \cdot x = \ominus_V\ (a \cdot x)$
⟨*proof*⟩

The next two lemmas prove exercise 1E, which says that the scalar product also satisfies an integral property (if $a \cdot b = 0_V$, either $a = 0_{\mathbb{K}}$ or $b = 0_V$):

lemma *mult-zero-uniq*:
assumes *x-in-V*: $x \in$ *carrier V* **and** *x-not-zero*: $x \neq \mathbf{0}_V$
and *a-in-K*: $a \in$ *carrier K* **and** *m-ax-0*: $a \cdot x = \mathbf{0}_V$
shows $a = \mathbf{0}_K$
⟨*proof*⟩

lemma *integral*:
assumes *x-in-V*: $x \in$ *carrier V*
and *a-in-K*: $a \in$ *carrier K*
and *m-ax-0*: $a \cdot x = \mathbf{0}_V$
shows $a = \mathbf{0}_K \mid x = \mathbf{0}_V$
⟨*proof*⟩

We present here some other properties which don't appear in Halmos but that will be useful in our development. For instance, distributivity of substraction with respect to the scalar product:

lemma *diff-mult-distrib1*:
assumes *x-in-V*: $x \in$ *carrier V*
and *y-in-V*: $y \in$ *carrier V*
and *a-in-K*: $a \in$ *carrier K*
shows $a \cdot (x \ominus_V y) = a \cdot x \ominus_V a \cdot y$
⟨*proof*⟩

The following result proves distributivity of substraction (of $\mathbb{K}$) with respect to the scalar product:

lemma *diff-mult-distrib2*:
assumes *x-in-V*: $x \in$ *carrier V*
and *a-in-K*: $a \in$ *carrier K*
and *b-in-K*: $b \in$ *carrier K*
shows $(a \ominus_K b) \cdot x = a \cdot x \ominus_V b \cdot x$
⟨*proof*⟩

The following result proves that the unary substraction of $\mathbb{K}$ and V is a self-cancelling operation by means of the scalar product:

lemma *minus-mult-cancel*:
assumes *x-in-V*: $x \in$ *carrier V* **and** *a-in-K*: $a \in$ *carrier K*
shows $(\ominus_K\ a) \cdot (\ominus_V\ x) = a \cdot x$
⟨*proof*⟩

A result proving that the scalar product is commutative over the elements of $\mathbb{K}$:

lemma *mult-left-commute*:
  assumes *x-in-V*: $x \in$ *carrier V*
  and *a-in-K*: $a \in$ *carrier K*
  and *b-in-K*:$b \in$ *carrier K*
  shows $a \cdot b \cdot x = b \cdot a \cdot x$
⟨*proof*⟩

A result proving that the scalar product is left-cancelling for the elements of $\mathbb{K}$ different from 0:

lemma *mult-left-cancel*:
  assumes *x-in-V*: $x \in$ *carrier V*
  and *y-in-V*: $y \in$ *carrier V*
  and *a-in-K*: $a \in$ *carrier K*
  and *a-not-zero*: $a \neq \mathbf{0}_K$
  shows $(a \cdot x = a \cdot y) = (x = y)$
⟨*proof*⟩

A similar result to the previous one but proving that the element of V can be also cancelled:

lemma *mult-right-cancel*:
  assumes *x-in-V*: $x \in$ *carrier V*
  and *a-in-K*: $a \in$ *carrier K*
  and *b-in-K*: $b \in$ *carrier K*
  and *x-not-zero*: $x \neq \mathbf{0}_V$
  shows $(a \cdot x = b \cdot x) = (a = b)$
⟨*proof*⟩

end
end
theory *Linear-dependence*
imports *Comments*
begin

6 Linear dependence

context *vector-space*
begin

In this section we will present the definition of linearly dependent set and linearly independent set. First of all we will introduce the definition of *linear-combination*.

A linear combination is a finite sum of vectors of V multiplicated by scalars. However, how can we specify the scalars? In a linear combination each vector will be multiplicated by one specific scalar, so this scalar depends on the vector. For that reason, we introduce the notion of *coefficients-function*.

definition *coefficients-function* :: *'b set* => (*'b* => *'a*) *set*
where *coefficients-function X*
= {*f*. *f* ∈ *X* → *carrier K* ∧ (∀ *x*. *x*∉*X* ⟶ *f x* = **0**$_K$)}

The explanation of the definition of coefficients function is as follows: given any set of vectors X, its coefficients functions will be every function which maps each of the vectors in X to scalars in $\mathbb{K}$. We impose an additional condition, in such a way that every element out of the set of vectors X is mapped to a distinguished element (in this case $\mathbf{0}$) of $\mathbb{K}$.

The first condition in the definition ($f \in X \to$ *carrier* K) is clear. A coefficients function is a function which maps, as we have said before, the elements of a given set X to their corresponding scalars in $\mathbb{K}$. The second condition ($\forall x.\ x \notin X \longrightarrow f\,x = \mathbf{0}$) requires further explanation: the reason to map every element out of the set X to a distinguished point is that this allows us to compare coefficients functions through the extensional equality of functions ($(f = g) = (\forall x.\ f\ x = g\ x)$). Thus, two coefficients function will be equal whenever they map every vector of X to the same scalar of $\mathbb{K}$ (this statement would not hold in the absence of the second condition).

Giving f a coefficients function and a certain x in *carrier* V then $f\,x$ (the scalar of the vector) will be in *carrier* K.

lemma *fx-in-K*:
assumes *x-in-V*: *x* ∈ *carrier V*
and *cf-f*: *f* ∈ *coefficients-function* (*carrier V*)
shows *f*(*x*) ∈ *carrier K*
⟨*proof*⟩

For every $x \in$ *carrier* V, multiplication between the scalar and the vector ($f\ x \cdot x$) is in *carrier* V.

lemma *fx-x-in-V*:
assumes *x-in-V*: *x* ∈ *carrier V*
and *cf-f*: *f* ∈ *coefficients-function* (*carrier V*)
shows *f*(*x*)·*x* ∈ *carrier V*
⟨*proof*⟩

Now we are going to define a linear combination. In Halmos, next section is about linear combinations, however we have to introduce now the definition because we will use it to define the linear dependence of a set. We will use the definition of sums over a finite set (*finsum*) which already exists in the Isabelle library. Note that we are defining a *linear-combination* with two parameters: second is the set of elements of V and first is the coefficients function which assigns each vector to its scalar.

Due to the definition of *finsum-def* we are only considering the case of a finite linear combination. The case of infinite linear combinations is undefined. This is not a problem for us, because we will work with finite vector spaces

and in our development we will only need linear combinations over finite sets. In addition, the sums in an infinite vector space are all finite because without additional structure the axioms of a vector space do not permit us to meaningfully speak about an infinite sum of vectors.

definition *linear-combination* :: $('b \Rightarrow 'a) \Rightarrow 'b\ set \Rightarrow 'b$
where *linear-combination f X* = *finsum V* $(\lambda y.\ f(y) \cdot y)$ *X*

In order to define the notion of linear dependence of a set we need to demand that this set be finite and a subset of the carrier. To abbreviate notation we will define these two premises as *good-set*.

definition *good-set* :: *'b set => bool*
where *good-set X* = (*finite X* $\wedge$ *X* $\subseteq$ *carrier V*)

Next two lemmas show both properties:

lemma *good-set-finite*:
assumes *good-set-X*: *good-set X*
shows *finite X*
⟨*proof*⟩

lemma *good-set-in-carrier*:
assumes *good-set-X*: *good-set X*
shows *X* $\subseteq$ *carrier V*
⟨*proof*⟩

Empty set is a *good-set*.

lemma [*simp*]: *good-set* {}
⟨*proof*⟩

Now, we can present the definition of linearly dependent set. A set will be dependent if there exists a linear combination equal to zero in which not all scalars are zero.

definition *linear-dependent* :: *'b set* $\Rightarrow$ *bool*
where *linear-dependent X* = (*good-set X*
$\wedge$ ($\exists f.\ f \in$ *coefficients-function* (*carrier V*) $\wedge$ *linear-combination f X* = $\mathbf{0}_V$
$\wedge \neg(\forall x \in X.\ f\ x = \mathbf{0}_K)$))

This definition is equivalent to the previous one:

definition *linear-dependent-2* :: *'b set* $\Rightarrow$ *bool*
where *linear-dependent-2 X* =
($\exists f.\ f \in$ *coefficients-function* (*carrier V*) $\wedge$ *good-set X*
$\wedge$ *linear-combination f X* = $\mathbf{0}_V \wedge \neg\ (\forall x \in X.\ f\ x = \mathbf{0}_K)$)

Next lemma, which is in the library, proves that are equivalent

lemma $(\exists f.\ X \wedge Y\ f) = (X \wedge (\exists f.\ Y\ f))$
⟨*proof*⟩

lemma *linear-dependent-eq-def*:
shows *linear-dependent X* = *linear-dependent-2 X*
⟨*proof*⟩

We introduce now the notion of a linearly independent set. We will prove later that linear dependence and independence are complementary notions (every set will be either dependent or independent).

definition *linear-independent* :: *'b set* ⇒ *bool*
where *linear-independent X* =
(*good-set X*
∧ (∀ *f*. (*f*∈ *coefficients-function* (*carrier V*) ∧ *linear-combination f X* = $\mathbf{0}_V$)
⟶ (∀ *x* ∈ *X*. *f*(*x*)=$\mathbf{0}_K$)))

Next lemmas prove that if we have a linear (in)dependent set hence we have a *good-set* (finite and in the carrier).

lemma *l-ind-good-set*: *linear-independent X* ⟹ *good-set X*
⟨*proof*⟩

lemma *l-dep-good-set*: *linear-dependent X* ⟹ *good-set X*
⟨*proof*⟩

The empty set is linearly independent.

lemma *empty-set-is-linearly-independent* [*simp*]:
shows *linear-independent* {}
⟨*proof*⟩

We can prove that linear independence is the opposite of linear dependence. For that, we first prove that every set which is not linearly independent must be linearly dependent:

lemma *not-independent-implies-dependent*:
assumes *good-set*: *good-set X*
shows ¬ *linear-independent X* ⟹ *linear-dependent X*
⟨*proof*⟩

Now we prove that every set which is linearly dependent is not linearly independent:

lemma *dependent-implies-not-independent*:
shows *linear-dependent X* ⟹ ¬ *linear-independent X*
⟨*proof*⟩

Hence the result:

lemma *dependent-if-only-if-not-independent*:
assumes *good-set*: *good-set X*
shows *linear-dependent X* ⟷ ¬ *linear-independent X*
⟨*proof*⟩

Analogously, we can prove that a set is not linearly dependent if and only if it is linearly independent. We use $[\![\neg P; \neg R \Longrightarrow P]\!] \Longrightarrow R$ and the previous lemma:

lemma *not-dependent-implies-independent*:
assumes *good-set*: *good-set X*
shows ¬ *linear-dependent X* ⟹ *linear-independent X*
⟨*proof*⟩

lemma *independent-implies-not-dependent*:
shows *linear-independent X* ⟹ ¬ *linear-dependent X*
⟨*proof*⟩

Finally, we obtain the equivalence of definitions:

lemma *independent-if-only-if-not-dependent*:
assumes *good-set*: *good-set X*
shows *linear-independent X* ⟷ ¬ *linear-dependent X*
⟨*proof*⟩

Every good set will be either dependent or independent (but not both at the same time). Note: the operator OR of this proof is not an exclusive OR, so really here we are proving that every set is either dependent or independent or both.

lemma *li-or-ld*:
assumes *good-set*:*good-set X*
shows *linear-dependent X* | *linear-independent X*
⟨*proof*⟩

In order to avoid that problem, we need to implement the operator exclusive OR:

definition *xor* :: *bool* ⇒ *bool* ⇒ *bool*
where *xor A B* ≡ (*A* ∧ ¬ *B*) ∨ (¬*A* ∧ *B*)

Now we can prove that every good set will be either dependent or independent (but not both at the same time):

lemma *li-xor-ld*:
assumes *good-set*:*good-set X*
shows *xor* (*linear-dependent X*) (*linear-independent X*)
⟨*proof*⟩

A corollary of these theorems using that the empty set is linearly independent: if we have a linearly dependent set, then it isn't the empty set:

lemma *dependent-not-empty*:
assumes *ld-A*: *linear-dependent A*
shows *A*≠{}
⟨*proof*⟩

Now we prove that every set X containing a linearly dependent subset Y is itself linearly dependent. This property is stated in Halmos but not proved, he says that the fact is clear.

The proof is easy but long. We want to achieve a linear combination of the elements of X equal to zero and where not all scalars are zero. We know that a subset Y of X is dependent, so there exists a linear combination of the elements of Y equal to zero where not all scalars are zero (we will denote its coefficients funcion as f). If we define a coefficients function for the set X where the scalars of the elements $y \in Y$ are $f(y)$ and 0_K for the rest of elements in X, then we will obtain a linear combination of elements of X equal to zero where not all scalars are zero (because not for all $x \in Y$ $f(x)$ is 0_K).

lemma *linear-dependent-subset-implies-linear-dependent-set*:
  assumes *Y-subset-X*: $Y \subseteq X$ **and** *good-set*: *good-set X*
  and *linear-dependent-Y*: *linear-dependent Y*
  shows *linear-dependent X*
⟨*proof*⟩

More properties and facts:

lemma *exists-subset-ld*:
  assumes *ld-X*: *linear-dependent X*
  shows $\exists\, Y.\ Y \subseteq X \wedge$ *linear-dependent Y*
  ⟨*proof*⟩

lemma *exists-subset-li*:
  assumes *ld-X*: *linear-dependent X*
  shows $\exists\, Y.\ Y \subseteq X \wedge$ *linear-independent Y*
⟨*proof*⟩

A set containing $\mathbf{0}_V$ is not an independent set:

lemma *zero-not-in-linear-independent-set*:
  assumes *li-A*: *linear-independent A*
  shows $\mathbf{0}_V \notin A$
⟨*proof*⟩

Every subset of an independent set is also independent. This property has been proved using *sledgehammer*.

lemma *independent-set-implies-independent-subset*:
  assumes *A-in-B*: $A \subseteq B$
  and *li-B*: *linear-independent B*
  shows *linear-independent A*
  ⟨*proof*⟩

We can even extend the notions of linearly dependent and independent sets to infinite sets in the following way. We shall say that a set is linearly independent if every finite subset of it is such.

definition *linear-independent-ext*:: *'b set* ⇒ *bool*
  where *linear-independent-ext X*
  = (∀ *A*. *finite A* ∧ *A* ⊆ *X* ⟶ *linear-independent A*)

Otherwise, it is linearly dependent.

definition *linear-dependent-ext*:: *'b set* ⇒ *bool*
  where *linear-dependent-ext X*
  = (∃ *A*. *A* ⊆ *X* ∧ *linear-dependent A*)

As expected, if we have a linearly independent set it will be also *linear-independent-ext* set.

lemma *independent-imp-independent-ext*:
  assumes *li-X*: *linear-independent X*
  shows *linear-independent-ext X*
⟨*proof*⟩

The same property holds for dependent sets:

lemma *dependent-imp-dependent-ext*:
  assumes *ld-X*: *linear-dependent X*
  shows *linear-dependent-ext X*
  ⟨*proof*⟩

Every finite set which is *linear-independent-ext* will also be *linear-independent*:

lemma *fin-ind-ext-impl-ind*:
  assumes *li-ext-X*: *linear-independent-ext X*
  and *finite-X*: *finite X*
  shows *linear-independent X*
  ⟨*proof*⟩

Similarly with the notion of linear dependence:

lemma *fin-dep-ext-impl-dep*:
  assumes *ld-ext-X*: *linear-dependent-ext X*
  and *gs-X*: *good-set X*
  shows *linear-dependent X*
  ⟨*proof*⟩

We can prove that also in the infinite case, the definitions of *linear-independent-ext* and *linear-dependent-ext* are complementary (every set will be of one type or the other). Let's see it:

lemma *not-independent-ext-implies-dependent-ext*:
  assumes *X-in-V*: *X* ⊆ *carrier V*
  shows ¬ *linear-independent-ext X* ⟹ *linear-dependent-ext X*
  ⟨*proof*⟩

lemma *not-dependent-ext-implies-independent-ext*:
  assumes *X-in-V*: *X* ⊆ *carrier V*
  shows ¬ *linear-dependent-ext X* ⟹ *linear-independent-ext X*

⟨*proof*⟩

lemma *independent-ext-implies-not-dependent-ext*:
shows *linear-independent-ext X* ⟹ ¬ *linear-dependent-ext X*
⟨*proof*⟩

lemma *dependent-ext-implies-not-independent-ext*:
shows *linear-dependent-ext X* ⟹ ¬ *linear-independent-ext X*
⟨*proof*⟩

corollary *dependent-ext-if-only-if-not-indepentent-ext*:
assumes *X-in-V*: *X* ⊆ *carrier V*
shows *linear-dependent-ext X* ⟷ ¬ *linear-independent-ext X*
⟨*proof*⟩

corollary *independent-ext-if-only-if-not-depentent-ext*:
assumes *X-in-V*: *X* ⊆ *carrier V*
shows *linear-independent-ext X* ⟷ ¬ *linear-dependent-ext X*
⟨*proof*⟩

end
end
theory *Indexed-Set*
imports *Main FuncSet Previous*
begin

7 Indexed sets

The next type definition, *iset*, represents the notion of an indexed set, which is a pair: a set and a function that goes from naturals to the set.

type-synonym ('*a*) *iset* = '*a set* × (*nat* => '*a*)

Now we define functions which make possible to separate an indexed set into the set and the function and we add them to the simplifier, since they are only meant to be abbreviations of the "fst" and "snd" operations:

definition *iset-to-set* :: '*a iset* => '*a set*
where *iset-to-set A* = *fst A*

definition *iset-to-index* :: '*a iset* => (*nat* => '*a*)
where *iset-to-index A* = *snd A*

lemmas [*simp*] = *iset-to-set-def iset-to-index-def*

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. Note: we will always work with finite sets. By default, the definition of *card* assigns to an infinite set cardinality equal to *0*.

definition *indexing* :: ('*a iset*) => *bool*
where *indexing A* = *bij-betw* (*iset-to-index A*)
{..<*card* (*iset-to-set A*)} (*iset-to-set A*)

Once we have the definition of *indexing*, we are going to prove some properties of it:

We introduce some lemmas presenting properties and alternative definitions of "indexing". For instance, whenever we have an indexing $A = (iset\_to\_set\ A, iset\_to\_index\ A)$ the index function will map naturals in the range $\{.. < card(A)\}$ to elements of $iset\_to\_set\ A$ and, moreover, the image set of the indexing function in such range will be whole set $iset\_to\_set\ A$.

lemma *indexing-equiv-img*:
assumes *ob*: *indexing A*
shows (*iset-to-index A*)
∈ {..<(*card* (*iset-to-set A*))} → (*iset-to-set A*)
∧ (*iset-to-index A*) ‘ {..<(*card* (*iset-to-set A*))}
= (*iset-to-set A*)
⟨*proof*⟩

The implication is also satisfied in the opposite direction:

lemma *img-equiv-indexing*:
assumes *f*: (*iset-to-index A*)
∈ {..<(*card* (*iset-to-set A*))} → (*iset-to-set A*)
∧ (*iset-to-index A*) ‘ {..<(*card* (*iset-to-set A*))}
= (*iset-to-set A*)
shows *indexing A*
⟨*proof*⟩

Now we present another alternative definition of indexing linking it with the notions of injectivity and surjectivity:

lemma *indexing-inj-surj*:
assumes *ob*: *indexing A*
shows *inj-on* (*iset-to-index A*) {..<(*card* (*iset-to-set A*))}
∧ (*iset-to-index A*) ‘ {..<(*card* (*iset-to-set A*))}
= (*iset-to-set A*)
⟨*proof*⟩

lemma *indexing-inj-surj-inv*:
assumes *inj-on* (*iset-to-index A*) {..<(*card* (*iset-to-set A*))}
∧ (*iset-to-index A*) ‘ {..<(*card* (*iset-to-set A*))} = (*iset-to-set A*)
shows *indexing A*
⟨*proof*⟩

One basic property is that the empty set with any function of appropriate type is an *indexing*:

lemma *indexing-empty*:
indexing ({}, *f*)

⟨*proof*⟩

We can obtain an equivalent notion of previous lemma writing the property in the unfolded definition of *indexing*.

lemma *indexing-empty-inv*:
shows *inj-on* (*iset-to-index* ({}, *f*)) {..<*card* (*iset-to-set* ({}, *f*))}
∧ *iset-to-index* ({}, *f*) ‘ {..<*card* (*iset-to-set* ({}, *f*))} = *iset-to-set* ({}, *f*) ⟨*proof*⟩

Now we are proving a basic but useful lemma: if we have an *indexing* of a set, then the image of a natural less than the cardinality of the set is an element of the set.

lemma *indexing-in-set*:
assumes *indexing* (*A*,*f*)
and $n < card\ A$
shows $f\ n \in A$
⟨*proof*⟩

We present two auxiliary lemmas about indexings and their behaviours as injective functions. The first one claims that if we have an *indexing* and two naturals (less than the cardinality of the set) with the same image, then the naturals are equal (which is a consequence of injectivity):.

lemma
indexing-impl-eq-preimage:
assumes *i*: *indexing* (*A*, *f*)
and *x*: $x \in \{..<card\ A\}$ **and** *y*: $y \in \{..<card\ A\}$
and *f*: $f\ x = f\ y$
shows $x = y$
⟨*proof*⟩

On the contrary, if we have the same assumptions than before but we consider that the image of both naturals are different, then the numbers are distinct.

lemma
indexing-impl-ndiff-image:
assumes *i*: *indexing* (*A*, *f*)
and *x*: $x \in \{..<card\ A\}$ **and** *y*: $y \in \{..<card\ A\}$
and *f*: $x \neq y$
shows $f\ x \neq f\ y$
⟨*proof*⟩

The following lemma proves that for any finite set A, there exist a natural number n and a function f such that f is an index function of A with $\{.. < n\}$ the collection of indexes. The prof is no constructive, is based on a lemma in the Isabelle library proving that every finite set is a mapping of a range of the naturals.

lemma *finite-imp-nat-seg-image-inj-on-Pi*:

assumes f: *finite A*
shows ($\exists\, n$::*nat*. $\exists f \in \{i.\ i < n\} \to A$.
((f ‘ $\{i.\ i < n\} = A$) $\wedge$ *inj-on* f $\{i.\ i < n\}$))
⟨*proof*⟩

The bijection is between the naturals up to card A and the set. Thanks to that we are giving to the set an indexation, we are representing a set more or less like a vector in C++: a structure with $card(A)$ components (from position 0 to $(card(A) - 1)$). Each component $f(i)$ tallies with one element of the set.

The following lemma extends the previous one, since we prove that n in the previous lemma is actually $card(A)$. The proof is carried out by induction on the finite set A, and the indexing function is explicitly given ($?f$ in the proof below):

lemma *finite-imp-nat-seg-image-inj-on-Pi-card*:
assumes f: *finite A*
shows ($\exists f \in \{i.\ i < (card\ A)\} \to A$. (($f$ ‘ $\{i.\ i < (card\ A)\} = A$)
$\wedge$ *inj-on* f $\{i.\ i < (card\ A)\}$))
⟨*proof*⟩

As a corollary, we prove that for each finite set there exists an indexing of it. This is the main theorem of this section and it will be very useful in the future to assign an order to a finite set (we will need it in future proofs).

corollary *obtain-indexing*:
assumes *finite-A*: *finite A*
shows $\exists f$. *indexing* (*A*,*f*)
⟨*proof*⟩

In addition, if we have an indexing we will know that the set is finite. This lemma will allow us to remove the premise $finite\ A$ whenever we have indexings. This is because Isabelle assigns 0 as the cardinality of an infinite set. Suppose that A is infinite. If we have an $indexing(A, f)$, hence f is a bijection between the set of naturals less than the cardinality of A (0 due to the implementation) and A. Then, $A = f`\{.. < card(A)\} = f`\{.. < 0\} = f`\{\} = \{\}$. However, we have supposed that A was infinite and $\{\}$ is not, so we have a contradiction and A is always finite.

lemma *indexing-finite*[*simp*]:
assumes *indexing-A*: *indexing* (*A*,*f*)
shows *finite A*
⟨*proof*⟩

After introducing the notion of indexed set, we need to introduce two basic operations over indexed sets: insert and remove. They will be generic with respect to the position where an element can be inserted or removed. For instance, given an indexed set $\{(a, 0), (b, 1), (c, 2)\}$ if we are to insert an element d, we will admit indexing $\{(d, 0), (a, 1), (b, 2), (c, 3)\}, \{(a, 0), (d, 1), (b, 2), (c, 3)\}$

and so on. In other words, inserting an element in a sorted set preserves the order of the elements, but maybe not their positions.

First we define the function which, for a given indexing A and an element a gives all possible indexings for the set *insert a* (*iset_to_set A*) preserving (*iset_to_index A*):

n is the position where 'a' will be inserted. It shoul be a natural number between 0 (first position) and card A (last position).

definition *indexing-ext* :: ('*a iset*) => '*a* => (*nat* => *nat* => '*a*)
where
indexing-ext A a =
(%*n*. %*k*. *if k* < *n then* (*iset-to-index A*) *k*
else if k = *n then a*
else (*iset-to-index A*) (*k* − *1*))

Now we present a basic property (it will be useful to be applied in induction proofs): If one *indexing-ext* generated from an indexation F and from one element $a \notin$ *index-to-set F* is good (is an indexing), then the indexation of F is also good (an indexing).

It is a long lemma (about 300 lines). The proof of injectivity must be separated in several different cases, depending on the position where we insert the element (after, before or exactly in the *nth* position):

lemma *indexing-indexing-ext*:
assumes *ob*:
indexing ((*insert x* (*iset-to-set F*)), (*indexing-ext F x n*))
and *n1*: *0* ≤ *n*
and *n2*: *n* ≤ *card* (*iset-to-set F*)
and *x-notin-F*: *x* ∉ (*iset-to-set F*)
shows *indexing F*
⟨*proof*⟩

From the above definitions we can define the operation insert for indexed sets. We don't assume that the new element (which is going to be inserted in the set) is not in the set, this will appear as a premise in the corresponding results.

Given any indexed set A, an element a and a position n, the operation *insert_iset* will introduce a in *iset_to_set A* in the position n (modifying accordingly the original indexation *iset_to_index A*).

definition *insert-iset* :: '*a iset* => '*a* => *nat* => '*a iset*
where
insert-iset A a n
= (*insert a* (*iset-to-set A*), *indexing-ext A a n*)

Next lemma claims that if we insert an element in an *indexing*, we are increasing the cardinality of the set in a unit. Logically, we need to assume

that the element which is going to be inserted is not in the set.

lemma *insert-iset-increase-card*:
assumes *indexing-A*: *indexing* (*A*,*f*)
and *a-notin-A*: $a \notin A$
shows *card* (*iset-to-set* (*insert-iset* (*A*,*f*) *a n*)) = *card A* + *1*
⟨*proof*⟩

Given an indexing (A, f), an element $a \notin A$ and a position $n \leq card(A)$, the result of inserting a in A in position n will be an indexing:

lemma *insert-iset-indexing*:
assumes *indexing-A*: *indexing* (*A*,*f*)
and *a-notin-A*: $a \notin A$
and *n2*: $n \leq$ (*card A*)
shows *indexing* (*insert-iset* (*A*,*f*) *a n*)
⟨*proof*⟩

We introduce the definition of a generic function *remove-iset* which removes the *nth* element of an indexed set. Logically, the position of the element which is going to be removed must be less than the cardinality of the set. The indexing must be also modified in such a way that every element above n will decrease its position in one unit. For instance, if we have the indexed set $\{(a, 0), (b, 1), (c, 2)\}$ and we remove the position 0, we will obtain $\{(b, 0), (c, 1)\}$.

definition *remove-iset* :: *'a iset => nat => 'a iset*
where *remove-iset A n* = (*fst A* − {(*snd A*) *n*},
(λ*k*. *if k* < *n then* (*snd A*) *k else* (*snd A*) (*Suc k*)))

Here an equivalent definition to *remove-iset ?A ?n* = (*fst ?A* − {*snd ?A ?n*}, λ*k*. *if k* < *?n then snd ?A k else snd ?A* (*Suc k*)):

lemma *remove-iset-def'*:
remove-iset (*A*, *f*) *n* = (*A* − {*f n*}, (λ*k*. *if k* < *n then f k else f* (*Suc k*)))
⟨*proof*⟩

The following lemma proves that, for any indexing, the result of removing an element in a valid position will be again an indexing. This is a long lemma (about 150 lines).

lemma
indexing-remove-iset:
assumes *i*: *indexing* (*B*, *h*)
and *n*: *n* < *card B*
shows *indexing* (*remove-iset* (*B*, *h*) *n*)
⟨*proof*⟩

The result of inserting an element in an indexed set in position n and then removing the element in position n is the original indexed set.

lemma

remove-iset-insert-iset-id:
assumes *x-notin-A*: $x \notin A$
and *n-l-c*: $n < card\ A$
shows *remove-iset* (*insert-iset* $(A, f)\ x\ n)\ n = (A, f)$
⟨*proof*⟩

Next lemma is a good example of proof by acumulation of facts, and it is ideal to structure it using *moreover* and finish it with *ultimately*. However, we can use $[\![A;\ B;\ C;\ D]\!] \Longrightarrow A \wedge B \wedge C \wedge D$ to abridge it:

The lemma claims that given an indexing (X, f), there exists an indexing $(insert\ x\ X, h)$ which places x in the last position (and keeps the elements of X in their original places).

lemma *indexation-x-union-X*:
assumes *finite*: *finite X* **and** *x-not-in-X*: $x \notin X$
and *f-buena*:$f \in \{i.\ i < (card\ X)\} \to X$ **and** *ordenFX*: $f\ `\ \{i.\ i < (card\ X)\} = X$
shows $\exists h.\ (h \in \{i.\ i < (card\ (insert\ x\ X))\} \to (insert\ x\ X)$
$\wedge\ h`\{i.\ i < (card\ (insert\ x\ X))\} = (insert\ x\ X)$
$\wedge\ h\ (card\ X) = x \wedge (\forall i.\ i<card(X) \longrightarrow h\ i = f\ i))$
⟨*proof*⟩

This is an indispensable lemma to prove the theorem that claims that an independent set can be completed to a basis. Given any pair of (disjoint) sets A and B, there exists an indexing function h which places the elements of A in the first $card(A)$ positions and then the elements of B. In the proof, the indexing function is explicitly provided:

lemma *indexing-union*:
assumes *disjuntos*: $A \cap B = \{\}$
and *finite-A*: *finite A*
and *A-not-empty*: $A \neq \{\}$ — If not the result is trivial.
and *finite-B*: *finite B*
shows $\exists h.$ *indexing* $(A \cup B, h) \wedge h`\ \{..<card(A)\} = A$
$\wedge\ h`\ (\{..<(card(A)+card(B))\} - \{..<card(A)\}) = B$
⟨*proof*⟩

Now we are going to define a new function which returns the position where an element a is in a set A. When we use this function it is very important to assume that $a \in A$, since functions are total in HOL, and without the premise $a \in A$ we would obtain an undefined value of the righ type. An alternative definition could be made writing LEAST instead of THE and then we could remove $n < card\ A$. Note that both THE and LEAST are based on the Hilbert's ϵ operator, which, in general, places us out of a constructive setting.

This function will be very important for the proof that each basis of a vector space has the same cardinality.

definition *obtain-position* :: *'c⇒'c iset ⇒ nat*
where *obtain-position a A* = (*THE n*. (*snd A*) *n* = *a*
∧ *n* < *card* (*fst A*))

Under the right premises, this natural number exists and is smaller than $card(A)$ which ensures that *obtain-position* is well-defined.

lemma *exists-n-obtain-position*:
assumes *a-in-A*: *a* ∈ *A*
and *indexing-A*: *indexing* (*A*,*f*)
shows ∃ *n*::*nat*. *f n* = *a*
⟨*proof*⟩

We proof that exists someone that also verifies $n < card\ A$

lemma *exists-n-and-less-card-obtain-position*:
assumes *a-in-A*: *a* ∈ *A*
and *indexing-A*: *indexing* (*A*,*f*)
shows ∃ *n*::*nat*. *f n* = *a* ∧ *n* < (*card A*)
⟨*proof*⟩

Thanks to the previous lemma and the injectivity of indexing functions, we can prove the existence and the unicity of *obtain-position*:

lemma *exists-n-and-is-unique-obtain-position*:
assumes *a-in-A*: *a*∈ *A*
and *indexing-A*: *indexing* (*A*,*f*)
shows ∃!*n*::*nat*. *f n* = *a* ∧ *n* < (*card A*)
⟨*proof*⟩

Now that we have proved that *obtain-position* is well defined, we prove that its result satisfies the required properties. The number which is returned by *obtain-position* is less than the cardinal of the set:

lemma *obtain-position-less-card*:
assumes *a-in-A*: *a* ∈ *A*
and *indexing-A*: *indexing* (*A*,*f*)
shows (*obtain-position a* (*A*,*f*)) < *card A*
⟨*proof*⟩

The function really returns the position of the element.

lemma *obtain-position-element*:
assumes *a-in-A*: *a* ∈ *A*
and *indexing-A*: *indexing* (*A*,*f*)
shows *f* (*obtain-position a* (*A*,*f*)) = *a*
⟨*proof*⟩

An element will not be in the set returned by the function *remove-iset* called with the position of that element.

lemma *a-notin-remove-iset*:
assumes *a-in-A*: *a* ∈ *A*

and *indexing-A*: *indexing* (*A*,*f*)
shows *a* ∉ *fst* (*remove-iset* (*A*,*f*) (*obtain-position a* (*A*,*f*)))
⟨*proof*⟩

Finally some important theorems to prove future properties of indexed sets. Isabelle has an induction rule to prove properties of finite sets. Unfortunately, this rule is of little help for proving properties of indexed sets, since the set and the indexing function must behave accordingly in the induction rule, and their inherent properties. Consequently, we have to introduce a special induction rule for indexed sets.

First an auxiliary lemma:

lemma *exists-indexing-ext*:
assumes *i*: *indexing* (*insert x A*, *f*)
shows ∃ *h*. ∃ *n* ∈ {..*card A*}. (*f* = (*indexing-ext* (*A*, *h*) *x*) *n*)
⟨*proof*⟩

The first one induction rule:

theorem
indexed-set-induct:
assumes *indexing* (*A*, *f*)
and *finite A*
and !!*f*. *indexing* ({}, *f*) ==> *P* {} *f*
and *step*: !!*a A f n*. [|*a* ∉ *A*; *finite A*; *indexing* (*A*, *f*);
0 ≤ *n*; *n* ≤ *card A*|] ==> *P* (*insert a A*) ((*indexing-ext* (*A*, *f*) *a*) *n*)
shows *P A f*
⟨*proof*⟩

This induction rule is similar to the proper of finite sets, ⟦*finite F*; *P* {}; ⋀*x F*. ⟦*finite F*; *x* ∉ *F*; *P F*⟧ ⟹ *P* (*insert x F*)⟧ ⟹ *P F*, but taking into account the indexing. Thus, if a property P holds for the empty set and one of its indexing functions, and when it holds for a given set A and an indexing function f, we now how to prove it for the pair *insert a A* (with $a \notin A$) and any of the extensions of f, then P holds for every indexing (A, f). The proof of the property is completed by induction over the set A, but keeping f free for later instantiation with the right indexing functions.

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 $P\ A\ f$
⟨*proof*⟩

end

theory *Linear-combinations*
imports *Linear-dependence Indexed-Set*
begin

8 Linear combinations

context *vector-space*
begin

To define the notion of linear dependence and independence we already introduced the definition of linear combination. Nevertheless, here we present some properties of linear combinations. We could have used them to simplify the proofs of some theorems in the previous section, but we have decided to keep the order of the sections in Halmos.

A *linear-combination* is closed, when considering a set $X \subseteq carrier\ V$ and a proper coefficients function f:

lemma *linear-combination-closed*:
assumes *good-set*: *good-set X*
and *f*: $f \in$ *coefficients-function* (*carrier V*)
shows *linear-combination f X* $\in$ *carrier V*
⟨*proof*⟩

A *linear-combination* over the empty set is equal to $\mathbf{0}_V$

lemma *linear-combination-of-zero*:
shows *linear-combination f* $\{\} = x \longleftrightarrow x = \mathbf{0}_V$
⟨*proof*⟩

From previous lemma we can obtain a corollary which will be useful as a simplification rule.

corollary *linear-combination-empty-set* [*simp*]:
shows *linear-combination f* $\{\} = \mathbf{0}_V$
⟨*proof*⟩

The computation of the linear combination of a unipuntual set is direct:

lemma *linear-combination-singleton*:
assumes *cf-f*: $f \in$ *coefficients-function* (*carrier V*)
and *x-in-V*: $x \in$ *carrier V*
shows *linear-combination f* $\{x\} = f\ x \cdot x$
⟨*proof*⟩

A *linear-combination* of *insert x X* is equal to $f\ x \cdot x \oplus_V$ *linear-combination f X*

lemma *linear-combination-insert*:
assumes *good-set-X*: *good-set X*
and *x-in-V*: $x \in$ *carrier V*
and *x-not-in-X*: $x \notin X$
and *cf-f*: $f \in$ *coefficients-function* (*carrier V*)
shows *linear-combination f* (*insert x X*)
$=$ *f x* $\cdot$ *x* $\oplus_V$ *linear-combination f X*
⟨*proof*⟩

If each term of the linear combination is zero, then the sum is zero.

lemma *linear-combination-zero*:
assumes *good-set-X*: *good-set X*
and *cf-f*: $f \in$ *coefficients-function* (*carrier V*)
and *all-zero*: $\bigwedge x.\ x \in X \Longrightarrow f\ (x) \cdot x = \mathbf{0}_V$
shows *linear-combination f X* $= \mathbf{0}_V$
⟨*proof*⟩

This is an auxiliary lemma which we will use later to prove that $a \cdot$ *linear-combination f X* $=$ *linear-combination* $(\lambda i.\ a \otimes f\ i)\ X$. We prove it doing induction over the finite set X. Firstly, we have to prove the property in case that the set is empty. After that, we suppose that the result is true for a set X and then we have to prove it for a set *insert x X* where $x \notin X$.

lemma *finsum-aux*:
⟦*finite X*; *X* $\subseteq$ *carrier V*; *a*$\in$ *carrier K*; *f* $\in X \to$ *carrier K*⟧
$\Longrightarrow$ *a* $\cdot$ $(\bigoplus_V y \in X.\ f\ y \cdot y) = (\bigoplus_V y \in X.\ a \cdot (f\ y \cdot y))$
⟨*proof*⟩

To multiply a linear combination by a scalar a is the same that multiplying each term of the linear combination by a.

lemma *linear-combination-rdistrib*:
⟦*good-set X*; *f* $\in$ *coefficients-function* (*carrier V*);
a $\in$ *carrier K*⟧ $\Longrightarrow$ *a* $\cdot$ (*linear-combination f X*)
$=$ *linear-combination* $(\%i.\ a \otimes f(i))\ X$
⟨*proof*⟩

Now some useful lemmas which will be helpful to prove other ones.

lemma *coefficients-function-g-f-null*:
assumes *cf-f*: $f \in$ *coefficients-function* (*carrier V*)
shows $(\lambda x.$ *if* $x \in Y$ *then* $f(x)$ *else* $\mathbf{0}_K)$
$\in$ *coefficients-function* (*carrier V*) ⟨*proof*⟩

This lemma is a generalization of the idea through we have proved *linear-dependent-subset-implies-linea* ⟦$Y \subseteq X$; *good-set X*; *linear-dependent Y*⟧ $\Longrightarrow$ *linear-dependent X*. Using it we could reduce its proof, but in Halmos the section of linear dependence goes before the one about linear combinations. The proof is based on dividing the linear combination into two sums, from which one of them is equal to 0_V. This lemma takes up about 130 code lines.

lemma *eq-lc-when-out-of-set-is-zero*:
assumes *good-set-A*: *good-set A* **and** *good-set-Y*: *good-set Y*
and *cf-f*: $f \in$ *coefficients-function* (*carrier V*)
shows *linear-combination* (λx. *if* $x \in Y$ *then* $f(x)$ *else* $\mathbf{0}_K$)
($Y \cup A$) = *linear-combination f Y*
⟨*proof*⟩

Another auxiliary lemma. It will be very useful to prove properties in future sections. If we have an equality of the form $\mathbf{0}_V = g\ x \cdot x \oplus_V$ *linear-combination g X*, then we can work out the value of x (there exists a coefficients function f such that $x =$ *linear_combination f X*. This coefficients function is explicitly defined by dividing each of the values $g(y)$ by $g(x)$).

lemma *work-out-the-value-of-x*:
assumes *good-set*: *good-set X*
and *coefficients-function-g*:
$g \in$ *coefficients-function* (*carrier V*)
and *x-in-V*: $x \in$ *carrier V*
and *gx-not-zero*: $g\ x \neq \mathbf{0}_K$
and *lc-descomposicion*: $\mathbf{0}_V = g(x) \cdot x \oplus_V$ *linear-combination g X*
shows $\exists f.\ f \in$ *coefficients-function* (*carrier V*)
$\wedge$ *linear-combination f X* = *x*
⟨*proof*⟩

Now we are going to prove a property presented in Halmos, section 6: if $\{x_i\}_{i \in \mathbb{N}}$ is linearly independent, then a necessary an sufficient condition that x be a linear combination of $\{x_i\}_{i \in \mathbb{N}}$ is that the enlarged set, obtained by adjoining x to $\{x_i\}_{i \in \mathbb{N}}$, be linearly dependent.

Here the first implication. The proof is based on definig a linear combination of the set *insert x X* equal to 0_V. As long as we know that *linear_combination f X* = *x* we define a coefficients function for *insert x X* where the coefficients of $y \in X$ are $f(y)$ and the coefficient of x is -1. A detail that is omitted in Halmos is that not every coefficient is zero since the coefficient of x is -1. The complete proof requires 102 lines of Isabelle code.

lemma *lc1*:
assumes *linear-independent-X*: *linear-independent X*
and *x-in-V*:*x* $\in$ *carrier V* **and** *x-not-in-X*:*x* $\notin X$
shows ($\exists f.\ f \in$ *coefficients-function* (*carrier V*) $\wedge$ *linear-combination f X* = *x*)
$\Longrightarrow$ *linear-dependent* (*insert x X*)
⟨*proof*⟩

And now we present the second implication. The proof is based on obtaining a linear combination of *insert x X* in which not all scalars are zero (we can do it since X is linearly dependent). Hence we prove that the scalar of x is not zero (if it is, hence X would be dependent and independent so

a contradiction). Then, we can express x as a linear combination of the elements of X.

lemma *lc2*:
assumes *linear-independent-X*: *linear-independent X*
and *x-in-V*: $x \in$ *carrier V*
and *x-not-in-X*: $x \notin X$
shows *linear-dependent* (*insert x X*)
$\Longrightarrow$ ($\exists$ *f*. *f*$\in$ *coefficients-function* (*carrier V*)
$\wedge$ *linear-combination f X* $= x$)
⟨*proof*⟩

Finally, the theorem proving the equivalence of both definitions.

lemma *lc1-eq-lc2*:
assumes *linear-independent-X*: *linear-independent X*
and *x-in-V*:$x \in$ *carrier V* **and** *x-not-in-X*:$x \notin X$
shows *linear-dependent* (*insert x X*) $\longleftrightarrow$
($\exists$ *f*. *f*$\in$ *coefficients-function* (*carrier V*)
$\wedge$ *linear-combination f X* $= x$)
⟨*proof*⟩

This lemma doesn't appears in Halmos (but it seems to be a similar result to the theorem **??**). The proof is based on obtaining a linear combination of the elements of $X \cup Y$ equal to 0_V where not all scalars are equal to $0_{\mathbb{K}}$. Hence we can express an element $y \in (X \cup Y)$ such that its scalar is not zero as a linear combination of the rest elements of $X \cup Y$. This is a long proof of 180 lines.

lemma *exists-x-linear-combination*:
assumes *li-X*: *linear-independent X*
and *ld-XY*: *linear-dependent* ($X \cup Y$)
shows $\exists\, y \in Y.\ \exists\, g.\ g \in$ *coefficients-function* (*carrier V*)
$\wedge\ y =$ *linear-combination g* ($X \cup (Y - \{y\})$)
⟨*proof*⟩

A corollary of the previous lemma claims that if we have a linearly dependent set, then there exists one element which can be expressed as a linear combination of the other elements of the set.

corollary *exists-x-linear-combination2*:
assumes *ld-Y*: *linear-dependent Y*
shows $\exists\, y \in Y.\ \exists\, g.\ g \in$ *coefficients-function* (*carrier V*)
$\wedge\ y =$ *linear-combination g* ($Y - \{y\}$)
⟨*proof*⟩

Every singleton set is linearly independent. This lemma could be in previous section, however we have to make use of some properties of linear combinations. We can repeat the proof without these properties, but it would be longer. We will use that $a \cdot x = \mathbf{0} \Longrightarrow a = \mathbf{0}$ because $x \neq \mathbf{0}$.

lemma *unipuntual-is-li*:

assumes *x-in-V*: $x \in$ *carrier* V **and** *x-not-zero*: $x \neq \mathbf{0}_V$
shows *linear-independent* $\{x\}$
⟨*proof*⟩ **thm** *finsum-insert*
⟨*proof*⟩

Now we are ready to prove the theorem 1 in section 6 in Halmos. It will be useful (really indispensable) in future proofs and it is basic in our developement. The theorem claims that in a linear dependent set exists an element which is a linear combination of the preceding ones.

NOTE: As we are assuming that $\mathbf{0}_V$ is not in the set, the element which is a linear combination of the preceding ones will be between the second and the last position of the set (1 and $card(A) - 1$ with the notation used in our implementation of indexed sets). The element in the first position (position 0) can't be a linear combination of the preceding ones because it would be a linear combination of the empty set, hence this element would be $\mathbf{0}_V$ and it is not in the set.

We make the proof using induction (we don't follow the proof of the book). At first, it seemed easier this way.

lemma
linear-dependent-set-contains-linear-combination:
assumes *ld-X*: *linear-dependent X*
and *not-zero*: $\mathbf{0}_V \notin X$
shows $\exists\, y \in X.\ \exists\, g.\ \exists\, k{::}nat.\ \exists f \in \{i.\ i{<}(card\ X)\} \to X.\ f`\{i.\ i{<}(card\ X)\} = X \wedge g \in coefficients\text{-}function\ (carrier\ V)$
$\wedge\ (1{::}nat) \leq k \wedge k < (card\ X) \wedge f\, k = y \wedge y = linear\text{-}combination\ g\ (f`\{i{::}nat.\ i{<}k\})$
⟨*proof*⟩

lemma
card-less-induct-good-set:
assumes *c*: *good-set A*
and *step*: $\bigwedge A.\ [\![\ (\bigwedge B.\ [\![\ card\ B < card\ A;\ good\text{-}set\ B\ ]\!] \Longrightarrow P\ B);\ good\text{-}set\ A\ ]\!] \Longrightarrow P\ A$
shows $P\ A$
⟨*proof*⟩

Really, the result that we need to prove corresponds closer to the next theorem than we have proved in the previous theorem *linear-dependent-set-contains-linear-combination*. We have to assume that the indexation is known beforehand. This will be necessary in the future, because we will remove dependent elements in regard a gived indexation of one set (so the removed element will be unique). We will apply this theorem iteratively to a set in future proofs, so if we

didn't fix the order beforehand we won't have unicity of the result (because the indexing could change in each step).

We will use the induction rule for indexed sets that we introduced before (*indexed-set-induct2*). This is a laborious and large theorem, of about 400 code lines.

theorem
linear-dependent-set-sorted-contains-linear-combination:
assumes *ld-A*: *linear-dependent A*
and *not-zero*: $\mathbf{0}_V \notin A$
and *i*: *indexing* (*A*, *f*)
shows $\exists\, y{\in}A.\ \exists\, g.\ \exists\, k{::}nat.$
$g \in$ *coefficients-function* (*carrier V*)
$\wedge$ (*1*::*nat*) $\leq k \wedge k <$ (*card A*)
$\wedge\ f\ k = y \wedge y =$ *linear-combination g* (*f*‘{*i*::*nat*. *i*<*k*})
⟨*proof*⟩

The proof can be also done without induction and then the proof of the theorem is shorter: "only" 200 code lines. The proof is a generalization of one of the cases in the induction above.

theorem
linear-dependent-set-sorted-contains-linear-combination2:
assumes *ld-A*: *linear-dependent A*
and *not-zero*: $\mathbf{0}_V \notin A$
and *i*: *indexing* (*A*, *f*)
shows $\exists\, y{\in}A.\ \exists\, g.\ \exists\, k{::}nat.$
$g \in$ *coefficients-function* (*carrier V*)
$\wedge$ (*1*::*nat*) $\leq k \wedge k <$ (*card A*)
$\wedge\ f\ k = y \wedge y =$ *linear-combination g* (*f*‘{*i*::*nat*. *i*<*k*})
⟨*proof*⟩

end
end
theory *Basis*
imports *Linear-combinations*
begin

9 Basis

context *vector-space*
begin

A finite spanning set is a finite set of vectors that can generate every vector in the space through such linear combinations.

definition *spanning-set* :: *'b set* ⇒ *bool*
where *spanning-set X* = (*good-set X*
$\wedge$ ($\forall\, x.\ x \in$ *carrier V* ⟶ ($\exists\, f.\ f \in$ *coefficients-function* (*carrier V*) $\wedge$ *linear-combination f X* = *x*)))

Even, we can talk about an infinite spanning set. We say that a set (finite or infinite) $X \subseteq$ *carrier* V is a spanning set (we will rename this definition as *spanning-set-ext*) if for every $x \in$ *carrier* V it is possible to choose a finite subset of X such that exists a linear combination of its elements equal to x.

As we have said before, the sums are all finite: we can not talk about an infinite sum of vectors without adding some concepts and more structure (the axioms of Vector Space do not allow it).

definition *spanning-set-ext* :: *'b set* ⇒ *bool*
where *spanning-set-ext* $X = (\forall x.\ x \in$ *carrier* $V \longrightarrow$
$(\exists A.\ \exists f.$ *good-set* $A \wedge A \subseteq X \wedge f \in$ *coefficients-function* (*carrier* V) $\wedge$ *linear-combination* $f\ A = x))$

Let's see the compatibility between the definitions:

Now we prove that every *spanning-set* is a *spanning-set-ext*:

lemma *spanning-imp-spanning-ext*:
assumes *sp-X*: *spanning-set* X
shows *spanning-set-ext* X
⟨*proof*⟩

Whenever we have a *spanning-set-ext* which is finite and $X \subseteq$ *carrier* V then it is a *spanning-set*.

lemma *gs-spanning-ext-imp-spanning*:
assumes *sp-X*: *spanning-set-ext* X
and *gs-X*: *good-set* X
shows *spanning-set* X
⟨*proof*⟩

A basis is an independent spanning set. We define it in general (X could be finite or infinite).

definition *basis* :: *'b set* ⇒ *bool*
where *basis* $X = (X \subseteq$ *carrier* $V \wedge$ *linear-independent-ext* $X \wedge$ *spanning-set-ext* $X)$

If we have a finite basis, then it is a good set.

lemma *finite-basis-implies-good-set*:
assumes *basis-B*: *basis* B
and *finite-B*: *finite* B
shows *good-set* B
⟨*proof*⟩

We introduce the definition of *span* of a determinated set A like the set of all elements which can be expressed as a linear combination of the elements of A.

definition *span* :: *'b set* => *'b set*

where *span A* = {*x*. ∃ *g* ∈ *coefficients-function* (*carrier V*). *x* = *linear-combination g A*}

First of all, we prove the behavior of *span* with respect to {}.

lemma
span-empty [*simp*]:
shows *span* {} = {$\mathbf{0}_V$}
⟨*proof*⟩

One auxiliar result says that $\mathbf{0}_V$ is in the span of every set.

lemma
span-contains-zero [*simp*]:
assumes *fin-A*: *finite A*
and *A-in-V*: *A* ⊆ *carrier V*
shows $\mathbf{0}_V$ ∈ *span A*
⟨*proof*⟩

Now we are going to prove that if we remove an element of a set which is a linear combination of the rest of elements then the span of the set is the same than the span of the set minus the element. This will be a fundamental property to be applied in the future. First of all, we do two auxiliar proofs.

This auxiliary lemma claims that given a coefficients funcion g of $A - \{a\}$ hence there exists another one (denoted by ga) such that *linear-combination g* (*A* − {*a*}) = *linear-combination ga A*. The coefficients function *ga* will be defined as follows: λ*x*. *if x* = *a then* **0** *else g x*.

lemma *exists-function-Aa-A*:
assumes *cf-g*: *g* ∈ *coefficients-function* (*carrier V*)
and *good-set-A*: *good-set A*
and *a-in-A*: *a* ∈ *A*
shows ∃ *ga* ∈ *coefficients-function* (*carrier V*).
($\bigoplus_V y \in A - \{a\}$. *g y* · *y*) = ($\bigoplus_V y \in A$. *ga y* · *y*)
⟨*proof*⟩

This auxiliary lemma is similar to the previous one. It claims that given a coefficients function h and another one g such that a = *linear-combination g* (*A* − {*a*}), there exists a coefficients function *ga* such that *linear-combination h A* = *linear-combination ga* (*A* − {*a*}). This coefficients funcion *ga* is defined as follows: λ*x*. *h a* ⊗ *g x* ⊕ *h x*. In other words, with these premises every linear combination of elements of A can be expressed as a linear combination of elements of $A - \{a\}$.

lemma *exists-function-A-Aa*:
assumes *cf-h*:*h* ∈ *coefficients-function* (*carrier V*)
and *cf-g*: *g* ∈ *coefficients-function* (*carrier V*)
and *a-lc-g-Aa*: *a* = *linear-combination g* (*A*−{*a*})
and *good-set-A*: *good-set A* **and** *a-in-A*: *a*∈*A*
shows ∃ *ga* ∈ *coefficients-function* (*carrier V*).

$(\bigoplus_V y{\in}A.\ h\ y \cdot y) = (\bigoplus_V y{\in}A - \{a\}.\ ga\ y \cdot y)$
⟨*proof*⟩

Now we present the theorem. The proof is done by double content of both span sets and we make use of the two previous lemmas.

theorem
span-minus:
assumes *good-set-A*: *good-set A*
and *a-in-A*: $a \in A$
and *exists-g*: $\exists\, g.\ g{\in}$ *coefficients-function* (*carrier V*)
$\wedge\ a =$ *linear-combination g* $(A - \{a\})$
shows *span A* = *span* $(A - \{a\})$
⟨*proof*⟩

A corollary of this theorem claims that for every linearly dependent set A, then $\exists\, a{\in}A.$ *span A* = *span* $(A - \{a\})$.

We also need to use *linear-dependent Y* $\Longrightarrow \exists\, y{\in}Y.\ \exists\, g.\ g \in$ *coefficients-function* (*carrier V*) $\wedge$ *y* = *linear-combination g* $(Y - \{y\})$

corollary
span-minus2:
assumes *ld-A*: *linear-dependent A*
shows $\exists\, a{\in}A.$ *span A* = *span* $(A - \{a\})$
⟨*proof*⟩

If an element y is not in the span of a set A, hence that element is not in that set. The proof is completed by *reductio ad absurdum*. If $a \in A$, then there is a linear combination of the elements of A, and thus $a \in span(A)$, which is a contradiction with one of the premises.

lemma *not-in-span-impl-not-in-set*:
assumes *y-notin-span*: $y \notin$ *span A*
and *cb-A*: *good-set A*
and *y-in-V*: $y \in$ *carrier V*
shows $y \notin A$
⟨*proof*⟩

If we have an element which is not in the span of an independent set, then the result of inserting this element into that set is a linearly independent set. The proof is done dividing the goal into cases. The case where $A \neq \{\}$ again is divided in cases with respect to the boolean *linear-independent* (*insert y A*). In the case where *linear-independent* (*insert y A*) is false, again we proceed by *reductio ad absurdum*. It is a long lemma of 129 lines.

lemma *insert-y-notin-span-li*:
assumes *y-notin-span*: $y \notin$ *span A*
and *y-in-V*: $y \in$ *carrier V*
and *li-A*: *linear-independent A*
shows *linear-independent* (*insert y A*)

⟨*proof*⟩

We can unify the concepts of *spanning-set*, *span* and *basis* and illustrate the relationships that exist among them.

The *span* of a *spanning-set* is *carrier V*.

lemma *span-basis-implies-spanning-set*:
assumes *span-A-V*: *span A* = *carrier V*
and *good-set-A*: *good-set A*
shows *spanning-set A*
⟨*proof*⟩

The opposite implication:

lemma *spanning-set-implies-span-basis*:
assumes *sg-A*: *spanning-set A*
shows *span A* = *carrier V*
⟨*proof*⟩

Now we present the relationship between *spanning-set* and *span*: if *span A* = *carrier V* then *A* is a *spanning set*.

lemma *span-V-eq-spanning-set*:
assumes *cb-A*: *good-set A*
shows *span A* = *carrier V* ⟷ *spanning-set A*
⟨*proof*⟩

Now we can introduce in Isabelle a new definition of basis (in the case of finite dimensional vector spaces). A finite basis will be a set *A* which is *linear-independent* and satisfies *span A* = *carrier V*. We use the previous lemma to check that it is equivalent to *basis X* = (*X* ⊆ *carrier V* ∧ *linear-independent-ext X* ∧ *spanning-set-ext X*).

lemma *basis-def'*:
assumes *cb-A*: *good-set A*
shows *basis A* ⟷ (*linear-independent A* ∧ *span A* = *carrier V*)
⟨*proof*⟩

If we have a finite basis, we can forget extended versions of linear independence and spanning set:

lemma *finite-basis*:
assumes *fin-A*: *finite A*
shows *basis A* ⟷ (*linear-independent A* ∧ *spanning-set A*)
⟨*proof*⟩

end

9.1 Finite Dimensional Vector Space

For working in a finite vector space we need to fix a finite basis.

The definition of finite dimensional vector spaces in Isabelle/HOL is direct. It consists of a vector space in which we assume that there exists a finite basis. Note that we have not proved yet that every vector space contains a basis.

locale *finite-dimensional-vector-space* = *vector-space* +
  fixes *X* :: *'c set*
  assumes *finite-X*: *finite X*
  and *basis-X*: *basis X*

context *finite-dimensional-vector-space*
begin

From this point the fixed basis is denoted by X.

We add to simplifier both premisses.

lemmas [*simp*] = *finite-X basis-X*

It is easy to show that the basis is a good set, is linearly independent and a spanning set.

lemma *good-set-X*:
  shows *good-set X*
  ⟨*proof*⟩

lemma *linear-independent-X*:
  shows *linear-independent X*
  ⟨*proof*⟩

lemma *spanning-set-X*:
  shows *spanning-set X*
  ⟨*proof*⟩

We add to simplifier these three lemmas.

lemmas [*simp*] = *good-set-X linear-independent-X spanning-set-X*

For all $x \in carrier\ V$ exists a linear combination of elements of the basis (we can write $x \in carrier\ V$ in combination of the elements of a basis).

lemma *exists-combination*:
  assumes *x-in-V*: $x \in carrier\ V$
  shows $\exists f.\ (f \in coefficients\text{-}function\ (carrier\ V) \wedge x = linear\text{-}combination\ f\ X)$
  ⟨*proof*⟩

Next lemma shows us that coordinates of a vector are unique for each basis

lemma *unique-coordenates*:
  assumes *x-in-V*: $x \in carrier\ V$
  and *cf-f*: $f \in coefficients\text{-}function\ (carrier\ V)$
  and *lc-f*: $x = linear\text{-}combination\ f\ X$
  and *cf-g*: $g \in coefficients\text{-}function\ (carrier\ V)$

and *lc-g*: $x = \textit{linear-combination}\ g\ X$
shows $\forall x \in X.\ g\ x = f\ x$
⟨*proof*⟩

We have fixed a finite basis and now we can prove some theorems about the *span*. Note that the concept of finitude of the basis is very important in the proofs.

The span of a basis is the total, so it's easy to prove that $\textit{carrier}\ V \subseteq \textit{span}\ X$. The other implication is also easy: we have only to unfold the definition and use $[\![\textit{good-set}\ ?X;\ ?f \in \textit{coefficients-function}\ (\textit{carrier}\ V)]\!] \Longrightarrow \textit{linear-combination}\ ?f\ ?X \in \textit{carrier}\ V$.

lemma *span-basis-is-V*: $\textit{span}\ X = \textit{carrier}\ V$
⟨*proof*⟩

The span of every set joined with a basis is the total. Before proving this theorem, we make two auxiliar lemmas.

First one:

lemma *exists-linear-combination-union-basis*:
assumes *fin-A*: *finite A*
and *A-in-V*: $A \subseteq \textit{carrier}\ V$
and *x-in-V*: $x \in \textit{carrier}\ V$
shows $\exists g.\ g \in \textit{coefficients-function}\ (\textit{carrier}\ V) \wedge x = \textit{linear-combination}\ g\ (A \cup X)$
⟨*proof*⟩

Second one

lemma *span-union-basis-eq*:
assumes *fin-A*: *finite A*
and *A-in-V*: $A \subseteq \textit{carrier}\ V$
shows $\textit{span}\ (A \cup X) = \textit{span}\ X$
⟨*proof*⟩

Finally the theorem: the span of every set joined with a basis is the total

corollary *span-union-basis-is-V*:
assumes *fin-A*: *finite A*
and *A-in-V*: $A \subseteq \textit{carrier}\ V$
shows $\textit{span}\ (A \cup X) = \textit{carrier}\ V$
⟨*proof*⟩

9.2 Theorem 1.

From this, we are going to center into the proof that every linearly independent set can be extended to a basis.

The function *remove-ld* takes an element of type *'a iset* and returns other element of that type in which in the set has been removed the first element

that is a combination of the preceding ones, and the indexation has removed the corresponding index.

In the next definition, making use of previous theorem:

⟦*linear-dependent Xa*; $\mathbf{0}_V \notin$ *Xa*⟧ ⟹ ∃ *y*∈*Xa*. ∃ *g k*. ∃*f*∈{*i*. *i* < *card Xa*} → *Xa*. *f* ‘ {*i*. *i* < *card Xa*} = *Xa* ∧ *g* ∈ *coefficients-function* (*carrier V*) ∧ *1* ≤ *k* ∧ *k* < *card Xa* ∧ *f k* = *y* ∧ *y* = *linear-combination g* (*f* ‘ {*i*. *i* < *k*}), we remove the *least* element that verifies the property that it can be expressed as a linear combination of the preceding ones. The existence of this element is guaranteed by the fact that the set is linearly dependent. If we iterate the function *remove-ld* we can be sure that it will terminate because sooner or later we will achieve a linearly independent set.

It is important to note that we have to provide a fixed indexation *f* for the elements to be removed are uniquely determined.

The function *remove-ld* must be only applied to an indexation of a linearly dependent set that does not contain $\mathbf{0}_V$, since these are the uniques conditions where we have ensured the existence of the element to be removed using:

linear-dependent-set-contains-linear-combination: ⟦*linear-dependent Xa*; $\mathbf{0}_V$ ∉ *Xa*⟧ ⟹ ∃ *y*∈*Xa*. ∃ *g k*. ∃*f*∈{*i*. *i* < *card Xa*} → *Xa*. *f* ‘ {*i*. *i* < *card Xa*} = *Xa* ∧ *g* ∈ *coefficients-function* (*carrier V*) ∧ *1* ≤ *k* ∧ *k* < *card Xa* ∧ *f k* = *y* ∧ *y* = *linear-combination g* (*f* ‘ {*i*. *i* < *k*}).

definition *remove-ld* :: *'c iset* => *'c iset*
where *remove-ld A* =
(*let n* = (*LEAST k*::*nat*. ∃ *y*∈(*fst A*). ∃ *g*.
g ∈ *coefficients-function* (*carrier V*)
∧ (*1*::*nat*) ≤ *k* ∧ *k* < (*card* (*fst A*))
∧ (*snd A*) *k* = *y*
∧ *y* = *linear-combination g* ((*snd A*) ‘ {*i*::*nat*. *i*<*k*}))
in remove-iset A n)

Next lemma expresses another notation for *remove-ld ?A* = *Let* (*LEAST k*. ∃ *y*∈*fst ?A*. ∃ *g*. *g* ∈ *coefficients-function* (*carrier V*) ∧ *1* ≤ *k* ∧ *k* < *card* (*fst ?A*) ∧ *snd ?A k* = *y* ∧ *y* = *linear-combination g* (*snd ?A* ‘ {*i*. *i* < *k*})) (*remove-iset ?A*).

lemma *remove-ld-def'*:
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 (*A* − {*f n*}, (λ*k*. *if k* < *n then f k else f* (*Suc k*))))
⟨*proof*⟩

Now we can prove some properties of the function *remove-ld*: it preserves the carrier, is monotone and decrease the cardinality.

lemma *remove-ld-preserves-carrier*:
assumes *b*: $B \subseteq$ *carrier* V
shows *fst* (*remove-ld* (B, h)) $\subseteq$ *carrier* V
⟨*proof*⟩

lemma *remove-ld-monotone*:
assumes *b*: $B \subseteq$ *carrier* V
shows *fst* (*remove-ld* (B, h)) $\subseteq B$
⟨*proof*⟩

lemma *remove-ld-decr-card*:
assumes *ld-A*: *linear-dependent* A
and *not-zero*: $\mathbf{0}_V \notin A$
and *indexing-A-f*: *indexing* (A, f)
shows *card* (*fst* (*remove-ld* (A, f))) = *card* $A - 1$
⟨*proof*⟩

corollary *remove-ld-decr-card2*:
assumes *ld-A*: *linear-dependent* A
and *not-zero*: $\mathbf{0}_V \notin A$
and *indexing-A-f*: *indexing* (A, f)
shows *card* (*fst* (*remove-ld* (A, f))) < *card* A
⟨*proof*⟩

This is an indispensable result: our function *remove-ld* preserves the propiety of *span*. For this proof is very important the theorem *span-minus*: $[\![$*good-set ?A*; *?a* $\in$ *?A*; $\exists\, g.\ g \in$ *coefficients-function* (*carrier* V) $\wedge$ *?a* = *linear-combination g* (*?A* $-$ {*?a*})$]\!] \Longrightarrow$ *span ?A* = *span* (*?A* $-$ {*?a*}).

lemma *remove-ld-preserves-span*:
assumes *ld-A*: *linear-dependent* A
and *not-zero*: $\mathbf{0}_V \notin A$
and *indexing-A-f*: *indexing* (A, f)
shows *span* (*fst* (*remove-ld* (A, f))) = *span* A
⟨*proof*⟩

The next function *iterate-remove-ld* has done that we have to install *Isabelle2011*. In previous versions we have to make use of *function* (*tailrec*), but this element had some bugs. In particular, we could not use *function* (*tailrec*) in the next definition.

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))))

declare *iterate-remove-ld.simps* [*simp del*]

Its behaviour is the next: from a set and a indexation of it, we apply recursively the operation *remove-ld* up to we achieve a linearly independent set.

The reiterated elimination of the linearly dependent elements would have to keep the span.

If we call to the function *iterate-remove-ld* with a linearly independent set, it will return us that set.

lemma *iterate-remove-ld-empty* [*simp*]: *iterate-remove-ld* {} *f* = {}
⟨*proof*⟩

lemma
iterate-remove-ld-li [*simp*]:
assumes *li-A*: *linear-independent A*
shows *iterate-remove-ld A f* = *A*
⟨*proof*⟩

Now we are going to prove some lemmas about indexings and *remove-iset*. Note that we can not put this lemmas in the file *Indexed-Set.thy* because the axioms *good-set* and *linear-dependent* are sometimes in the premises.

The next lemma express that the result of aplying *remove-iset* preserves the good set property. In our context we need to prove it for *remove-ld*...but it does not cease to be a particular case of *remove-iset*.

lemma
remove-iset-good-set:
assumes *c*: *good-set A*
and *i*: *indexing* (*A*, *h*)
shows *good-set* (*fst* (*remove-iset* (*A*, *h*) *n*))
⟨*proof*⟩

lemma
remove-ld-good-set:
assumes *c*: *good-set A*
and *i*: *indexing* (*A*, *h*)
shows *good-set* (*fst* (*remove-ld* (*A*, *h*)))
⟨*proof*⟩

Next theorem applies ⟦*indexing* (*?B*, *?h*); *?n* < *card ?B*⟧ ⟹ *indexing* (*remove-iset* (*?B*, *?h*) *?n*) to the function *remove-ld*. We can omit the good set condition: it is implicit in the fact that the set is linearly dependent.

theorem *indexing-remove-ld*:
assumes *l*: *linear-dependent A*
and *i*: *indexing* (*A*, *f*)
and *n*: $\mathbf{0}_V \notin A$
shows *indexing* (*remove-ld* (*A*, *f*))
⟨*proof*⟩

Next lemma shows us that first element of a indexed set is in the carrier. Note that we can not put this lemma in the file *Indexed Set* due to the axiom $A \subseteq$ *carrier V* (we have not a structure of carrier in that file).

lemma *f0-in-V*:
assumes *indexing-A*: *indexing* (*A*,*f*)
and *A-in-V*: $A \subseteq$ *carrier V*
and *A-not-empty*: $A \neq \{\}$ — Essential to cardinality
shows *f 0* $\in$ *carrier V*
⟨*proof*⟩

If A is independent, then its firts element is not zero.

lemma *f0-not-zero*:
assumes *indexing-A*: *indexing* (*A*,*f*)
and *li-A*: *linear-independent A*
and *A-not-empty*: $A \neq \{\}$
shows *f 0* $\neq \mathbf{0}_V$
⟨*proof*⟩

We can also prove that apply the function *insert-iset* return us a good set.

lemma *insert-iset-good-set*:
assumes *a-notin-A*: $a \notin A$
and *indexing*: *indexing* (*A*,*f*)
and *a-in-V*: $a \in$ *carrier V*
and *cb-A*: *good-set A*
shows *good-set* (*fst*(*insert-iset* (*A*,*f*) *a n*))
⟨*proof*⟩

Remove an element and after that insert it is a good set

lemma *good-set-insert-remove*:
assumes *B-in-V*: $B \subseteq$ *carrier V*
and *A-in-V*: $A \subseteq$ *carrier V*
and *A-not-empty*: $A \neq \{\}$
and *indexing-A*: *indexing* (*A*,*f*)
and *indexing-B*: *indexing* (*B*,*g*)
and *a-in-B*: $a \in B$
shows *good-set* (*fst* (*insert-iset* (*remove-iset* (*B*, *g*) (*obtain-position a* (*B*, *g*))) *a n*))
⟨*proof*⟩

The result of applying the function *iterate-remove-ld* to any finite set in *carrier V* will be always independent (the function finishes).

We are going to make the proof firstly by dividing in cases (with respect to the condiction *linear-independent A*) and after that by total induction over the cardinal of the set: $(\bigwedge x.\ (\bigwedge y.\ f\ y < f\ x \Longrightarrow P\ y) \Longrightarrow P\ x) \Longrightarrow P\ a$.

With respect to the induction, it is important to note that we can not make induction over the *structure* of the set, with the following induction rule for indexed set that we have introduced in section **??**:

indexed-set-induct2: $[\![$*indexing* (*A*, *f*); *finite A*; $\bigwedge f.$ *indexing* ({}, *f*) $\Longrightarrow P$ {} *f*; $\bigwedge a\ A\ f\ n.\ [\![a \notin A$; *indexing* (*A*, *f*) $\Longrightarrow P\ A\ f$; *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*)⟧ ⟹ *P A f*

If we make induction over the structure, we will have to prove the case *insert a A* and the induction hypothesis will say that the result is true for *A*. However, independently of in what position of the indexation we place the element *a*, we can not know if *remove-ld* (*insert a A*, *indexing-ext* (*A*, *f*) *a n*) will return the same set *A* or it will return another set. In other words: the result of inserting the element *a* in any position of the *A* set and after that removing the least element which is a linear combination of the preceding ones (*remove-ld* does it) is not equal to the original set. We can not ensure it even when we insert the element *a* in the least position that it can be expressed as a linear combination of the preceding ones: we can not be sure that *remove-ld* will remove that element. For example, in the set {(*1*, *0*), (*2*, *0*), (*0*, *1*)}, if we insert the element (*0*, *2*) in the least position where it is a linear combination of the preceding ones we achieve the set {(*1*, *0*), (*2*, *0*), (*0*, *1*), (*0*, *2*)}. However, if we apply *remove-ld* to this set, it will return {(*1*, *0*), (*0*, *1*), (*0*, *2*)} and this is not equal to the original set.

lemma
linear-independent-iterate-remove-ld:
assumes *A-in-V*: *A* ⊆ *carrier V*
and *not-zero*: **0**$_V$ ∉ *A*
and *indexing-A-f*: *indexing* (*A*, *h*)
shows *linear-independent* (*iterate-remove-ld A h*)
⟨*proof*⟩

Similarly to the previous theorem, we can prove that the function *iterate-remove-ld* preserves the *span*.

lemma *iterate-remove-ld-preserves-span*:
assumes *A-in-V*: *A* ⊆ *carrier V*
and *indexing-A-f*: *indexing* (*A*,*h*)
and *not-zero*: **0**$_V$ ∉ *A*
shows *span* (*iterate-remove-ld A h*) = *span A*
⟨*proof*⟩

If we have an *indexing* (*A* ∪ *B*, *h*) where elements of an independent set A are in its first positions and after those the elements of a set B, then A will be in *remove-ld* (*A* ∪ *B*, *h*) (we will have removed an element of B). The premisse of *A* ∩ *B* = {} is indispensable in order to avoid the notion of multisets. In the book, Halmos doesn't worry about this: he simply create a set with all elements of A in the first positions and after that all elements of B...but what does it happen if a element of B are in A? we will have a multiset because we have the same element in two positions. However, this is not a limitation for our theorem if we make a trick like these: *A*∪*B* = *A*∪(*B*−*A*). Using that we avoid the problem.

lemma *A-in-remove-ld*:
assumes *indexing*: *indexing* (*A*∪*B*,*h*)
and *ld-AB*: *linear-dependent* (*A*∪*B*)
and *surj-h-A*:*h*‘ {..<*card*(*A*)}= *A*

and *li-A*: *linear-independent A*
and *zero-not-in*: $\mathbf{0}_V \notin (A \cup B)$
and *disjuntos*: *A*∩*B*={}
shows *A* ⊆ *fst* (*remove-ld* ((*A*∪*B*),*h*))
⟨*proof*⟩

This lemma is an extended version of previous one. It shows that we are removing one element of the second set and preserving the indexation.

lemma *descomposicion-remove-ld*:
assumes *indexing*: *indexing* (*A*∪*B*,*h*)
and *B-not-empty*: *B*≠{} — Due to cardinality, it is indispensable.
and *surj-h-A*:*h*‘ {..<*card*(*A*)}= *A*
and *surj-h-B*:*h*‘ ({..<(*card*(*A*)+*card*(*B*))}−{..<*card*(*A*)})=*B*
and *li-A*: *linear-independent A*
and *zero-not-in*: $\mathbf{0}_V \notin (A \cup B)$
and *ld-AB*: *linear-dependent* (*A*∪*B*)
and *disjuntos*: *A*∩*B*={}
shows ∃ *y*. *fst* (*remove-ld* ((*A*∪*B*),*h*))=*A*∪(*B*−{*y*}) ∧ *y*∈*B*
∧ (*snd* (*remove-ld* (*A*∪*B*, *h*))) ‘ ({..<*card A* + *card* (*B*−{*y*})} − {..<*card A*})
= (*B*−{*y*})
∧ *snd* (*remove-ld*((*A*∪*B*), *h*)) ‘ {..<*card A*}=*A* ∧ *indexing* (*A* ∪ (*B*−{*y*}), *snd*
(*remove-ld* (*A*∪*B*, *h*)))
⟨*proof*⟩

Finally an important lemma proved using (⋀*x*. (⋀*y*. *?f y* < *?f x* ⟹ *?P y*)
⟹ *?P x*) ⟹ *?P ?a* such as we do in *linear-independent-iterate-remove-ld* and in *iterate-remove-ld-preserves-span*. We need above lemmas to prove it. It shows us that *iterate-remove-ld* does not remove any element of A if elements of A are in first positions and A is linearly independent.

lemma *A-in-iterate-remove-ld*:
assumes *indexing*: *indexing* (*A*∪*B*,*h*)
and *B-in-V*: *B* ⊆ *carrier V*
and *surj-h-A*:*h*‘ {..<*card*(*A*)}= *A*
and *surj-h-B*:*h*‘ ({..<(*card*(*A*)+*card*(*B*))}−{..<*card*(*A*)})=*B*
and *li-A*: *linear-independent A*
and *zero-not-in*: $\mathbf{0}_V \notin (A \cup B)$
and *disjuntos*: *A*∩*B*={}
shows *A* ⊆ (*iterate-remove-ld* (*A*∪*B*) *h*)
⟨*proof*⟩

Now we are in position to prove that every independent set can be extended to a basis. First we prove it for any non-empty set.

lemma *extend-not-empty-independent-set-to-a-basis*:

assumes *li-A*: *linear-independent A*
and *A-not-empty*: $A \neq \{\}$
shows $\exists B.$ *basis* $B \wedge A \subseteq B$
⟨*proof*⟩

And finally the theorem (case empty is trivial since we add all elements of our fixed basis X to it.

theorem *extend-independent-set-to-a-basis*:
assumes *li-A*: *linear-independent A*
shows $\exists B.$ *basis* $B \wedge A \subseteq B$
⟨*proof*⟩

We have proved that any independent set can be extended to a basis, but in anywhere we have proved that there exists a basis (we have supposed it as a premisse in the case of finite dimensional vector spaces). The proof that every vector space has a basis is not made in Halmos: some additional results as Zorn's lemma, chains or well-ordering are required. See http://planetmath.org/encyclopedia/EveryVectorSpaceHasABasis.html for a detailed proof.

However, we can prove the existence of a basis in a particular case: when *carrier V* is finite.

To prove this result, we are going to apply the function *iterate-remove-ld* to *carrier* $V - \{\mathbf{0}_V\}$. This function requires that $\mathbf{0}_V$ doesn't belong to the set where we apply it, so we will not apply it to *carrier V*, but to *carrier* $V - \{\mathbf{0}_V\}$. This function will return us a linearly independent set which span is the same as the span of *carrier* $V - \{\mathbf{0}_V\}$. Proving that *span* (*carrier* $V - \{\mathbf{0}_V\}$) = *carrier V* we will obtain the result (because *carrier* $V - \{\mathbf{0}_V\}$ is a spanning set).

Let's see the proof. Firstly, we can see that the set V is a *spanning-set*. It is trivial.

lemma *spanning-set-V*:
assumes *finite-V*: *finite* (*carrier V*)
shows *spanning-set* (*carrier V*)
⟨*proof*⟩

Thanks to that, the span of V is itself (trivially).

lemma *span-V-is-V*:
assumes *finite-V*: *finite* (*carrier V*)
shows *span* (*carrier V*) = *carrier V*
⟨*proof*⟩

Now we need to prove that *spanning-set* (*carrier* $V - \{\mathbf{0}_V\}$).

lemma *spanning-set-V-minus-zero*:
assumes *finite-V*: *finite* (*carrier* $V - \{\mathbf{0}_V\}$)

shows *spanning-set* (*carrier* $V - \{\mathbf{0}_V\}$)
⟨*proof*⟩

As a corollary we have that *span* (*carrier* $V - \{\mathbf{0}_V\}$) = *carrier* V

corollary *span-V-minus-zero-is-V*:
assumes *finite-V*: *finite* (*carrier* $V - \{\mathbf{0}_V\}$)
shows *span* (*carrier* $V - \{\mathbf{0}_V\}$)=*carrier* V
⟨*proof*⟩

Finally, the theorem:

theorem *finite-V-implies-ex-basis*:
assumes *finite-V*: *finite* (*carrier* V)
shows $\exists B.$ *basis* B
⟨*proof*⟩

A similar result than *spanning-set-V-minus-zero* is the next. We will use this theorem in the future.

lemma *spanning-set-minus-zero*:
assumes *finite-B*: *finite* B
and *B-in-V*: $B \subseteq$ *carrier* V
and *sg-B*: *spanning-set* B
shows *spanning-set* ($B - \{\mathbf{0}_V\}$)
⟨*proof*⟩

Every finite or infinite vector space contains a *spanning-set-ext* (in particular, *carrier* V fullfills this condition):

lemma *spanning-set-ext-carrier-V*:
shows *spanning-set-ext* (*carrier* V)
⟨*proof*⟩

lemma *vector-space-contains-spanning-set-ext*:
shows $\exists A.$ *spanning-set-ext* $A \wedge A \subseteq$ *carrier* V
⟨*proof*⟩

end
end
theory *Dimension*
imports *Basis*
begin

10 Dimension

context *finite-dimensional-vector-space*
begin

Now we are going to prove that every basis of a finite vector space has the same cardinality than any other basis.

First of all, we are going to define a function that remove the first element of an iset. We will use the function *remove-iset*. Note that this redefinition is essential: we can not iterate *remove-iset* because is *remove-iset*: $: iset \times \mathbb{N} \rightarrow iset$

definition *remove-iset-0* :: *'e iset* => *'e iset*
where *remove-iset-0 A* = *remove-iset A 0*

A property about this function and the empty set:

lemma *remove-iset-empty*:
shows *fst* (*remove-iset-0* ({},*f*))={}
⟨*proof*⟩

Now the definition of the function by means of we are going to prove the theorem.

definition *swap-function* :: (*'c iset* × *'c iset*)
=> (*'c iset* × *'c iset*)
where *swap-function A* = (*remove-iset-0* (*fst A*),
if (((*snd*(*fst A*) *0*)) ∈ *fst*(*snd A*)) *then*
insert-iset (*remove-iset* (*snd A*)
(*obtain-position* ((*snd*(*fst A*) *0*)) (*snd A*))) (*snd*(*fst A*) *0*) *0*
else
remove-ld (*insert-iset* (*snd A*) ((*snd*(*fst A*) *0*)) *0*))

From this, we will prove some basic properties that *swap-function* satisfies.

The set of the first component of the result is finite:

lemma *finite-fst-swap-function*:
assumes *indexing-A*: *indexing* (*A*,*f*)
shows *finite* (*iset-to-set*(*fst*(*swap-function* ((*A*,*f*),(*B*,*g*)))))
⟨*proof*⟩

The set of the first component of the result is in the carrier:

lemma *swap-function-fst-in-carrier*:
assumes *A-in-V*: *A* ⊆ *carrier V*
shows *iset-to-set*(*fst*(*swap-function* ((*A*,*f*),(*B*,*g*)))) ⊆ *carrier V*
⟨*proof*⟩

If the first set is not empty, then the set of the first component of the result is contained (strictly) in it.

lemma *fst-swap-function-subset-fst*:
assumes *indexing-A*: *indexing* (*A*,*f*)
and *A-not-empty*: *A*≠{} — INDISPENSABLE: IF NOT THE EMPTY CASE WILL NOT BE STRICT
shows *iset-to-set*(*fst*(*swap-function* ((*A*,*f*),(*B*,*g*)))) ⊂ *A*
⟨*proof*⟩

If we not demand that content be strict, then the result is trivial.

lemma *fst-swap-function-subseteq-fst*:
shows *iset-to-set*(*fst*(*swap-function* ((*A*,*f*),(*B*,*g*)))) ⊆ *A*
⟨*proof*⟩

We are goint to prove that the set of the second component of the result is a *good-set*. To prove it we will make use of ⟦*B* ⊆ *carrier V*; *A* ⊆ *carrier V*; *A* ≠ {}; *indexing* (*A*, *f*); *indexing* (*B*, *g*); *a* ∈ *B*⟧ ⟹ *good-set* (*fst* (*insert-iset* (*remove-iset* (*B*, *g*) (*obtain-position a* (*B*, *g*))) *a n*)).

lemma *swap-function-snd-good-set*:
assumes *B-in-V*: *B* ⊆ *carrier V*
and *A-in-V*: *A* ⊆ *carrier V*
and *A-not-empty*: *A*≠{}
and *indexing-A*: *indexing* (*A*,*f*)
and *indexing-B*: *indexing* (*B*,*g*)
shows *good-set* (*iset-to-set*(*snd*(*swap-function* ((*A*,*f*),(*B*,*g*)))))
⟨*proof*⟩

corollary *swap-function-snd-in-carrier*:
assumes *B-in-V*: *B* ⊆ *carrier V*
and *A-in-V*: *A* ⊆ *carrier V*
and *A-not-empty*: *A*≠{}
and *indexing-A*: *indexing* (*A*,*f*)
and *indexing-B*: *indexing* (*B*,*g*)
shows (*iset-to-set*(*snd*(*swap-function* ((*A*,*f*),(*B*,*g*))))) ⊆ *carrier V*
⟨*proof*⟩

If the first set is independent, our function will preserve it.

lemma *fst-swap-function-preserves-li*:
assumes *li-A*: *linear-independent A*
shows *linear-independent* (*iset-to-set*(*fst*(*swap-function* ((*A*,*f*),(*B*,*g*)))))
⟨*proof*⟩

If the first element of the iset (A,f) is in B, the function will preserve the second set (but it will have changed the indexation, putting that element in first position of B).

lemma *swap-function-preserves-B-if-fst-element-of-A-in-B*:
assumes *f0-in-B*: *f* 0 ∈*B*
and *indexing-A*: *indexing* (*A*,*f*)
and *indexing-B*: *indexing* (*B*,*g*)
shows *iset-to-set*(*snd*(*swap-function* ((*A*,*f*),(*B*,*g*))))=*B*
⟨*proof*⟩

This is an auxiliar lemma which says that if we insert an element into a spanning set, the result will be a linearly dependent set. We will need this result to assure the existence of the element to remove of the second set using the function *swap-function* through the theorem ⟦*linear-dependent A*; $\mathbf{0}_V \notin$ *A*; *indexing* (*A*, *f*)⟧ ⟹ ∃ *y*∈*A*. ∃ *g k*. *g* ∈ *coefficients-function* (*carrier V*)

$\wedge\ 1 \leq k \wedge k < card\ A \wedge f\ k = y \wedge y = linear\text{-}combination\ g\ (f\ `\ \{i.\ i < k\})$

lemma *linear-dependent-insert-spanning-set*:
assumes *f0-notin-B*: $f\ 0 \notin B$
and *indexing-A*: *indexing* (A,f)
and *indexing-B*: *indexing* (B,g)
and *A-in-V*: $A \subseteq carrier\ V$
and *B-in-V*: $B \subseteq carrier\ V$
and *A-not-empty*: $A \neq \{\}$ — Essential to cardinality
and *sg-B*: *spanning-set B*
shows *linear-dependent* (*iset-to-set* (*insert-iset* (B,g) $(f\ 0)$ *0*))
⟨*proof*⟩

This result is similar to *linear-dependent-insert-spanning-set* but using sets directly, not isets.

lemma *spanning-set-insert*:
assumes *sg-B*: *spanning-set B*
and *finite-B*: *finite B*
and *B-in-V*: $B \subseteq carrier\ V$
and *a-in-V*: $a \in carrier\ V$
shows *spanning-set* (*insert a B*)
⟨*proof*⟩

Our function will preserve that the second term is a *spanning-set*.

lemma *swap-function-preserves-sg*:
assumes *indexing-A*: *indexing* (A,f)
and *indexing-B*: *indexing* (B,g)
and *B-in-V*: $B \subseteq carrier\ V$
and *A-not-empty*: $A \neq \{\}$ — Essential to cardinality
and *li-A*: *linear-independent A*
and *sg-B*: *spanning-set B*
and *zero-notin-B*: $\mathbf{0}_V \notin B$
shows *spanning-set* (*iset-to-set*(*snd*(*swap-function* $((A,f),(B,g))$))))
⟨*proof*⟩

swap-function preserves the cardinality of the second iset.

lemma *snd-swap-function-preserves-card*:
assumes *indexing-A*: *indexing* (A,f)
and *indexing-B*: *indexing* (B,g)
and *B-in-V*: $B \subseteq carrier\ V$
and *A-not-empty*: $A \neq \{\}$
and *li-A*: *linear-independent A*
and *sg-B*: *spanning-set B*
and *zero-notin-B*: $\mathbf{0}_V \notin B$
shows *card* (*iset-to-set* (*snd* (*swap-function* $((A,f),(B,g))$))) = *card B*
⟨*proof*⟩

Next lemmas shows us how our function decreases the cardinality of the first term.

lemma *fst-swap-function-decr-card*:
assumes *indexing-A*: *indexing* (*A*,*f*)
shows *card* (*iset-to-set*(*fst*(*swap-function* ((*A*,*f*),(*B*,*g*))))) = *card A* − *1*
⟨*proof*⟩

Now we are going to prove that exists an element of the second iset such that if we apply the *swap-function*, the second term will be able to be writen as the second set removing that element and adding the first element of the first set.

We will prove it by cases, first the case that B is not empty

lemma *swap-function-exists-y-in-B-not-empty*:
assumes *indexing-A*: *indexing* (*A*,*f*)
and *indexing-B*: *indexing* (*B*,*g*)
and *B-in-V*: *B* ⊆ *carrier V*
and *A-not-empty*: *A* ≠ {}
and *B-not-empty*: *B* ≠ {}
and *li-A*: *linear-independent A*
and *sg-B*: *spanning-set B*
and *zero-notin-B*: $\mathbf{0}_V \notin B$
shows ∃ *y*∈*B*. *iset-to-set* (*snd*(*swap-function* ((*A*,*f*),(*B*,*g*)))) = (*insert* (*f 0*) (*B*−{*y*}))
⟨*proof*⟩ **thm** *descomposicion-remove-ld*
⟨*proof*⟩

And now the case that B is empty. It is an inconsistent case: if B is empty and a spanning set, then the vector space is $\{\mathbf{0}_V\}$. A is not empty, so $A=\{\mathbf{0}_V\}$. However, we will have a contradiction: A will be dependent ($\{\mathbf{0}_V\}$ is dependent) and also independent (by hypothesis).

lemma *swap-function-exists-y-in-B-empty*:
assumes *indexing-A*: *indexing* (*A*,*f*)
and *A-not-empty*: *A*≠{}
and *B-empty*: *B*={}
and *li-A*: *linear-independent A*
and *sg-B*: *spanning-set B*
shows ∃ *y*∈*B*. *iset-to-set* (*snd*(*swap-function* ((*A*,*f*),(*B*,*g*))))=(*insert* (*f 0*) (*B*−{*y*}))
⟨*proof*⟩

lemma *swap-function-exists-y-in-B*:
assumes *indexing-A*: *indexing* (*A*,*f*)
and *indexing-B*: *indexing* (*B*,*g*)
and *B-in-V*: *B* ⊆ *carrier V*
and *A-not-empty*: *A*≠{}
and *li-A*: *linear-independent A*
and *sg-B*: *spanning-set B*
and *zero-notin-B*: $\mathbf{0}_V \notin B$
shows ∃ *y*∈*B*. *iset-to-set* (*snd*(*swap-function* ((*A*,*f*),(*B*,*g*))))=(*insert* (*f 0*) (*B*−{*y*}))
⟨*proof*⟩

From this we can obtain a corollary: $\mathbf{0}_V$ is not in the second term of the result of applying *swap-function* to a *spanning-set*.

corollary *zero-notin-snd-swap-function*:
assumes *indexing-A*: *indexing* (A,f)
and *indexing-B*: *indexing* (B,g)
and *B-in-V*: $B \subseteq$ *carrier* V
and *A-not-empty*: $A \neq \{\}$
and *li-A*: *linear-independent* A
and *sg-B*: *spanning-set* B
and *zero-notin-B*: $\mathbf{0}_V \notin B$
shows $\mathbf{0}_V \notin$ *iset-to-set* (*snd*(*swap-function* $((A,f),(B,g))$))
⟨*proof*⟩

The first term of the result of applying *swap-function* is an indexing.

lemma *fst-swap-function-indexing*:
assumes *indexing-A*: *indexing* (A,f)
and *A-in-V*: $A \subseteq$ *carrier* V
shows *indexing* (*fst*(*swap-function* $((A,f),(B,g))$))
⟨*proof*⟩

Similarly with the second term:

lemma *snd-swap-function-indexing*:
assumes *indexing-A*: *indexing* (A,f)
and *indexing-B*: *indexing* (B,g)
and *A-in-V*: $A \subseteq$ *carrier* V
and *B-in-V*: $B \subseteq$ *carrier* V
and *A-not-empty*: $A \neq \{\}$
and *li-A*: *linear-independent* A
and *sg-B*: *spanning-set* B
and *zero-notin-B*: $\mathbf{0}_V \notin B$
shows *indexing* (*snd*(*swap-function* $((A,f),(B,g))$))
⟨*proof*⟩

If the first argument is an empty iset, then *swap-function* will also return the empty set (in first component).

lemma *swap-function-empty*:
shows *iset-to-set*(*fst*(*swap-function* $((\{\},f),(B,g))) = \{\}$
⟨*proof*⟩

lemma *swap-function-empty2*:
assumes *A-empty*: $A = \{\}$
shows *iset-to-set*(*fst*(*swap-function* $((A,f),(B,g))) = \{\}$
⟨*proof*⟩

end

Up to now we have proved properties of *swap-function*. However, we want to iterate it a specific number of times (compose with itself several times). We

need to implement the power of a function because (surprisingly) it is not in the library. We are interpreting the power of a function as a composition with itself.

We will have to be careful with the types: we can not iterate (compose) every function: a function can be composed with itself if the result and the arguments are of the same type (and the number of arguments is the same as the number of arguments of the result).

We can do the instantiation out of our context, since it is more general:

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*⟩

end

Once we have finished the instatiation, we can prove some general properties about the power of a function.

For example: the power of the identity function is also the identity.

lemma *id-n*: **shows** *id ^ n = id*
  ⟨*proof*⟩

Any function power to zero is the identity.

lemma *power-zero-id*: *f^0=id*
  ⟨*proof*⟩

A corollary of this lemma will be indispensable for the proofs by induction.

lemma *fun-power-suc*: **shows** *f^*(*Suc n*)= *f* ∘ (*f^n*)
  ⟨*proof*⟩

corollary *fun-power-suc-eq*:
  shows (*f^*(*Suc n*)) *x* = *f* ((*f^n*) *x*)
  ⟨*proof*⟩

context *finite-dimensional-vector-space*
begin

Now we will begin with the proofs of properties that *swap-function* iterated

several times satisfies. In general, we have proved a property in the case $n = 1$ and now we are going to generalize it for any n by induction.

Most properties are invariants of the *swap-function*, so we will have proved a property in case $n = 1$. To generalize it we will apply induction: we suppose that a property is true for f^n and we want to prove it for $f^{(Suc(n))}$. By induction hypothesis, f^n satisfies the property and thanks to *fun-power-suc-eq* we can write $f^{Suc\ n}\ x = f\ (f^n\ x)$. As we have the property proved in case $n = 1$, we will obtain the result generalized.

For example, we have proved *swap-function-empty*: *iset-to-set* (*fst* (*swap-function* (({}, f), B, g))) = {} and now we will generalize it.

lemma *swap-function-power-empty*:
shows *iset-to-set*(*fst*((*swap-function*^*n*) (({},*f*),(*B*,*g*))))={}
⟨*proof*⟩

lemma *swap-function-power-empty2*:
assumes *A-empty*: *A*={}
shows *iset-to-set*(*fst*((*swap-function*^*n*) ((*A*,*f*),(*B*,*g*))))={}
⟨*proof*⟩

The generalized lemma for *swap-function-fst-in-carrier*.

lemma *swap-function-power-fst-in-carrier*:
assumes *A-in-V*: *A* ⊆ *carrier V*
shows *iset-to-set*(*fst*((*swap-function*^*n*) ((*A*,*f*),(*B*,*g*)))) ⊆ *carrier V*
⟨*proof*⟩

Iterating the function the independence (in first argument) is preserved.

lemma *fst-swap-function-power-preserves-li*:
assumes *li-A*: *linear-independent A*
shows *linear-independent* (*iset-to-set*(*fst*(((*swap-function*^(*n*))) ((*A*,*f*),(*B*,*g*)))))
⟨*proof*⟩

The first term is always an indexing. This is the generalization of *fst-swap-function-indexing*.

lemma *fst-swap-function-power-indexing*:
assumes *indexing-A*: *indexing* (*A*,*f*)
and *A-in-V*: *A* ⊆ *carrier V*
shows *indexing* (*fst*((*swap-function*^*n*) ((*A*,*f*),(*B*,*g*))))
⟨*proof*⟩

Now we can prove that if we compose n-times *swap-function*, the cardinality of the set of the first term will be decreased in n. Note that to use the induction hypothesis, we have to have proved previously *fst-swap-function-power-indexing* (and obviously also *fst-swap-function-decr-card*).

lemma *fst-swap-function-power-decr-card*:
assumes *indexing-A*: *indexing* (*A*, *f*)
and *A-in-V*: *A* ⊆ *carrier V*

shows *card* (*iset-to-set* (*fst* ((*swap-function*^*n*) ((*A*, *f*), *B*, *g*)))) = *card A* − *n*
⟨*proof*⟩

The generalization of *finite-fst-swap-function*:

lemma *finite-fst-swap-function-power*:
assumes *indexing-A*: *indexing* (*A*,*f*)
and *A-in-V*: *A* ⊆ *carrier V*
shows *finite* (*iset-to-set*(*fst*((*swap-function*^*n*) ((*A*,*f*),(*B*,*g*)))))
⟨*proof*⟩

If we iterate cardinality of A times the function, where A is the set of the first argument, then the first term of the result will be the empty set (we have removed card A elements in A).

corollary *swap-function-power-card-fst-empty*:
assumes *indexing-A*: *indexing* (*A*,*f*)
and *A-in-V*: *A* ⊆ *carrier V*
shows *iset-to-set*(*fst*((*swap-function*^(*card A*)) ((*A*,*f*),(*B*,*g*))))={}
⟨*proof*⟩

And if we iterate a number of times less than card A, then the (first) result set will not be empty:

corollary *swap-function-power-fst-not-empty-if-n-l-cardA*:
assumes *indexing-A*: *indexing* (*A*,*f*)
and *A-in-V*: *A* ⊆ *carrier V*
and *n-l-card*: *n* < *card A*
shows *iset-to-set*(*fst*((*swap-function*^*n*) ((*A*,*f*),(*B*,*g*))))≠{}
⟨*proof*⟩

This is a very important property which shows us how is the result of applying the function *remove-iset-0* a specific number of times.

lemma *remove-iset-0-eq*:
assumes *i*: *indexing* (*A*,*f*)
and *k-l-card*: *k*<*card A*
shows (*remove-iset-0*^*k*) (*A*,*f*)=(*f*‘{*k*..<*card A*},λ*n*. *f*(*n*+*k*))
⟨*proof*⟩

corollary *corollary-remove-iset-0-eq*:
assumes *i*: *indexing* (*A*,*f*)
and *n-l-card*: *n* < *card A*
shows *snd* ((*remove-iset-0*^*n*) (*A*,*f*)) *0* = *f n*
⟨*proof*⟩

In the next lemma we prove some properties at same the time. We have done like that because in the induction case the properties need each others. We can not prove one separately: for example, to prove that $\mathbf{0}_V \notin$ *iset-to-set* (*snd* (*swap-function*$^{Suc\ n}$ ((*A*, *f*), *B*, *g*))) we would write that *swap-function*$^{Suc\ n}$ ((*A*, *f*), *B*, *g*) = *swap-function* (*swap-function*^{n} ((*A*, *f*), *B*, *g*)) and we would apply the theorem *zero-notin-snd-swap-function*:

$[\![$*indexing* (A, f); *indexing* (B, g); $B \subseteq$ *carrier* V; $A \neq \{\}$; *linear-independent* A; *spanning-set* B; $\mathbf{0}_V \notin B]\!] \Longrightarrow \mathbf{0}_V \notin$ *iset-to-set* (*snd* (*swap-function* ((A, f), B, g)))

However, to apply this theorem we need that *spanning-set* (*iset-to-set* (*snd* (*swap-function*^{n} ((A, f), B, g))))). To prove that we would need to use *swap-function-preserves-sg*:

$[\![$*indexing* (A, f); *indexing* (B, g); $B \subseteq$ *carrier* V; $A \neq \{\}$; *linear-independent* A; *spanning-set* B; $\mathbf{0}_V \notin B]\!] \Longrightarrow$ *spanning-set* (*iset-to-set* (*snd* (*swap-function* ((A, f), B, g))))

And a premise would be that $\mathbf{0}_V \notin$ *iset-to-set* (*snd* (*swap-function*^{n} ((A, f), B, g)))...but this is what we want to prove. Bringing all together in the same theorem we will have everything we need like induction hypothesis, so we can prove it. Next we will separate the properties.

lemma *zeronotin-sg-carrier-indexing*:
assumes *indexing-A*: *indexing* (A,f)
and *indexing-B*: *indexing* (B,g)
and *A-in-V*: $A \subseteq$ *carrier* V
and *B-in-V*: $B \subseteq$ *carrier* V
and *A-not-empty*: $A \neq \{\}$
and *li-A*: *linear-independent* A
and *sg-B*: *spanning-set* B
and *zero-notin-B*: $\mathbf{0}_V \notin B$
and *n-l-cardA*: $n <$ *card* A
shows $\mathbf{0}_V \notin$ *iset-to-set* (*snd* ((*swap-function*^*n*) ((A, f), B, g)))
$\wedge$ *spanning-set*(*iset-to-set*(*snd*((*swap-function*^*n*)((A,f),(B,g)))))
$\wedge$ (*iset-to-set*(*snd*((*swap-function*^*n*) ((A,f),(B,g)))))
$\subseteq$ *carrier* V
$\wedge$ *indexing* (*snd*((*swap-function*^*n*) ((A,f),(B,g))))
⟨*proof*⟩

Now we can obtain the properties separately as corollaries.

corollary *zero-notin-snd-swap-function-power*:
assumes *indexing-A*: *indexing* (A,f)
and *indexing-B*: *indexing* (B,g)
and *A-in-V*: $A \subseteq$ *carrier* V
and *B-in-V*: $B \subseteq$ *carrier* V
and *A-not-empty*: $A \neq \{\}$
and *li-A*: *linear-independent* A
and *sg-B*: *spanning-set* B
and *zero-notin-B*: $\mathbf{0}_V \notin B$
and *n-l-cardA*: $n <$ *card* A
shows $\mathbf{0}_V \notin$ *iset-to-set* (*snd* ((*swap-function*^*n*) ((A, f), B, g)))
⟨*proof*⟩

corollary *swap-function-power-preserves-sg*:
assumes *indexing-A*: *indexing* (A,f)
and *indexing-B*: *indexing* (B,g)
and *A-in-V*: $A \subseteq$ *carrier* V
and *B-in-V*: $B \subseteq$ *carrier* V
and *A-not-empty*: $A \neq \{\}$
and *li-A*: *linear-independent* A
and *sg-B*: *spanning-set* B
and *zero-notin-B*: $\mathbf{0}_V \notin B$
and *n-l-cardA*: $n <$ *card* A
shows *spanning-set* (*iset-to-set* (*snd* ((*swap-function*^*n*) ((*A*, *f*), *B*, *g*))))
⟨*proof*⟩

corollary *swap-function-power-snd-in-carrier*:
assumes *indexing-A*: *indexing* (A,f)
and *indexing-B*: *indexing* (B,g)
and *A-in-V*: $A \subseteq$ *carrier* V
and *B-in-V*: $B \subseteq$ *carrier* V
and *A-not-empty*: $A \neq \{\}$
and *li-A*: *linear-independent* A
and *sg-B*: *spanning-set* B
and *zero-notin-B*: $\mathbf{0}_V \notin B$
and *n-l-cardA*: $n <$ *card* A
shows *iset-to-set* (*snd* ((*swap-function*^*n*) ((*A*, *f*), *B*, *g*))) $\subseteq$ *carrier* V
⟨*proof*⟩

corollary *snd-swap-function-power-indexing*:
assumes *indexing-A*: *indexing* (A,f)
and *indexing-B*: *indexing* (B,g)
and *A-in-V*: $A \subseteq$ *carrier* V
and *B-in-V*: $B \subseteq$ *carrier* V
and *A-not-empty*: $A \neq \{\}$
and *li-A*: *linear-independent* A
and *sg-B*: *spanning-set* B
and *zero-notin-B*: $\mathbf{0}_V \notin B$
and *n-l-cardA*: $n <$ *card* A
shows *indexing* (*snd* ((*swap-function* ^ *n*) ((*A*, *f*), *B*, *g*)))
⟨*proof*⟩

Swap-function preserves the cardinality of the second iset.

lemma *snd-swap-function-power-preserves-card*:
assumes *indexing-A*: *indexing* (A, f)
and *indexing-B*: *indexing* (B, g)
and *A-in-V*: $A \subseteq$ *carrier* V
and *B-in-V*: $B \subseteq$ *carrier* V
and *A-not-empty*: $A \neq \{\}$
and *li-A*: *linear-independent* A
and *sg-B*: *spanning-set* B

and *zero-notin-B*: $\mathbf{0}_V \notin B$
and *n-l-card*: $n<card\ A$
shows *card* (*iset-to-set* (*snd* ((*swap-function*^*n*) ((*A*, *f*), *B*, *g*)))) = *card B*
⟨*proof*⟩

The first term of *swap-function* iterated is the same than *remove-iset-0* iterated.

lemma *fst-swap-function-power-eq*:
fst ((*swap-function* ^ *n*) ((*A*, *f*), *B*, *g*)) = (*remove-iset-0* ^ *n*) (*A*, *f*)
⟨*proof*⟩

The first element of the result of the first term in the nth iteration is f(n).

lemma *snd-fst-swap-function-image-0*:
assumes *indexing-A*: *indexing* (*A*,*f*)
and *c*: *n* < *card A*
shows *snd* (*fst* ((*swap-function* ^ *n*) ((*A*, *f*), *B*, *g*))) *0* = *f* (*n*)
⟨*proof*⟩

If we compose n times the *swap-function*, the first term will be the first set minus the first n elements of it.

lemma *swap-function-fst-image-until-n*:
assumes *indexing-A*: *indexing* (*A*,*f*)
and *A-not-empty*: *A*≠{}
and *n-l-cardA*: *n*<*card A*
shows *iset-to-set* (*fst* ((*swap-function* ^ *n*) ((*A*, *f*), *B*, *g*))) = *f* ‘ {*n*..<*card A*}
⟨*proof*⟩

Now an auxiliar and ugly lemma which we will use to prove the swap theorem. It is very laborious and hard lemma, similar that *swap-function-exists-y-in-B* but much more precisse and difficult (over 400 lines). It represents properties that has the function during the process of iterating.

lemma *aux-swap-theorem1*:
assumes *indexing-A*: *indexing* (*A*,*f*) — In this set are the elements that we have not included in second term yet.
and *indexing-B*: *indexing* (*B*,*g*)
and *B-in-V*: *B* ⊆ *carrier V*
and *A-not-empty*: *A*≠{}
and *sg-B*: *spanning-set B*
and *zero-notin-B*: $\mathbf{0}_V \notin B$
and *li-Z*: *linear-independent Z* — Z is the first independent set, the set over we would apply our function the first time. A is the subset of Z where there are the elements of Z that we have not added to B yet. The elements that we have added to B are in C.
and *A-union-C*: *A*∪*C*=*Z* — Of course, the union of A and C is Z.
and *disjoint*: *A*∩*C*={} — The sets are disjoints.
and *surj-g-C*: *g*‘{..<*card C*}= *C* — In first positions of B there are elements of Z that we have already included. This set will be independent, so when we apply *remove-ld* we will delete an element of (B-C)

shows $\exists\, y{\in}B.$ *iset-to-set* (*snd*(*swap-function* ((*A*,*f*),(*B*,*g*))))
=(*insert* (*f 0*) (*B*−{*y*}))
∧ $y \notin C$
∧ *iset-to-index* (*snd*(*swap-function* ((*A*,*f*),(*B*,*g*))))
‘ {..<*card* (*C*) + *1*} = $C \cup \{f\ 0\}$
⟨*proof*⟩

Another important auxiliary lemma. Applying the swap function n-times (with $n < card(A)$) to $((A, f), B, g)$, where A is independent and B a spanning set, we will have that the first n elements of A will be in the first positions of the second component of the result. Of course, these elements come from A and thus they are independent. We make use of *aux-swap-theorem1* to prove this lemma.

lemma *aux-swap-theorem2*:
assumes *indexing-A*: *indexing* (*A*,*f*)
and *indexing-B*: *indexing* (*B*,*g*)
and *B-in-V*: $B \subseteq$ *carrier V*
and *A-not-empty*: $A{\neq}\{\}$
and *li-A*: *linear-independent A*
and *sg-B*: *spanning-set B*
and *zero-notin-B*: $\mathbf{0}_V \notin B$
and *n-l-cardA*: $n <$ *card A*
shows *f*‘{..<*n*}
= *iset-to-index*(*snd*((*swap-function*^(*n*)) ((*A*,*f*),(*B*,*g*)))‘{..<*n*}
∧ *iset-to-index*(*snd*((*swap-function*^(*n*)) ((*A*,*f*),(*B*,*g*)))‘{..<*n*}
⊂ *A*
∧ *linear-independent*
(*iset-to-index*(*snd*((*swap-function*^(*n*)) ((*A*,*f*),(*B*,*g*)))‘{..<*n*})
∧ *n* = (*card* (*iset-to-index*(*snd*((*swap-function*^(*n*))
((*A*,*f*),(*B*,*g*)))‘{..<*n*}))
⟨*proof*⟩

At last, we can prove the swap theorem. We separate it in cases, when A is empty and when it is not. We use the auxiliar lemma *aux-swap-theorem2*.

theorem *swap-theorem-not-empty*:
assumes *indexing-A*: *indexing* (*A*,*f*)
and *indexing-B*: *indexing* (*B*,*g*)
and *A-in-V*: $A \subseteq$ *carrier V*
and *B-in-V*: $B \subseteq$ *carrier V*
and *A-not-empty*: $A{\neq}\{\}$
and *li-A*: *linear-independent A*
and *sg-B*: *spanning-set B*
and *zero-notin-B*: $\mathbf{0}_V \notin B$
shows *card A* ≤ *card B*
⟨*proof*⟩

Finally the theorem (every independent set has cardinal less than or equal to every spanning set) and some corollaries:

theorem *swap-theorem*:
assumes *indexing-A*: *indexing* (*A*,*f*)
and *indexing-B*: *indexing* (*B*,*g*)
and *A-in-V*: $A \subseteq$ *carrier V*
and *B-in-V*: $B \subseteq$ *carrier V*
and *li-A*: *linear-independent A*
and *sg-B*: *spanning-set B*
and *zero-notin-B*: $\mathbf{0}_V \notin B$
shows *card A* $\leq$ *card B*
⟨*proof*⟩

The next corollary omits the need of indexing functions for A and B (these are obtained through auxiliary lemmas).

corollary *swap-theorem2*:
assumes *finite-B*: *finite B*
and *B-in-V*: $B \subseteq$ *carrier V*
and *A-in-V*: $A \subseteq$ *carrier V*
and *li-A*: *linear-independent A*
and *sg-B*: *spanning-set B*
and *zero-notin-B*: $\mathbf{0}_V \notin B$
shows *card A* $\leq$ *card B*
⟨*proof*⟩

Now we can prove that the number of elements in any (finite) basis (of a finite-dimensional vector space) is the same as in any other (finite) basis.

theorem *eq-cardinality-basis*:
assumes *basis-B*: *basis B*
and *finite-B*: *finite B*
shows *card X* = *card B*
⟨*proof*⟩

corollary *eq-cardinality-basis2*:
assumes *basis-A*: *basis A*
and *finite-A*: *finite A*
and *basis-B*: *basis B*
and *finite-B*: *finite B*
shows *card A* = *card B*
⟨*proof*⟩

We can make the definicion of dimension of a vector space and relationate the concept with above theorems.

The dimension of a vector space is the cardinality of one of its basis. We have fixed X as a basis, so the definition is trivial:

definition *dimension* :: *nat*
where *dimension* = *card X*

If we have another basis, the dimension is equal to its cardinality.

lemma *eq-dimension-basis*:
assumes *basis-A*: *basis A*
and *finite-A*: *finite A*
shows *dimension* = *card A*
⟨*proof*⟩

Whenever we have an independent set, we will know that its cardinality is less than the dimension of the vector space.

lemma *card-li-le-dim*:
assumes *li-A*: *linear-independent A*
shows *card A* ≤ *dimension*
⟨*proof*⟩

Whenever the cardinality of a set is greater (strictly) than the dimension of V then the set is dependent.

corollary *card-g-dim-implies-ld*:
assumes *card-g-dim*: *card A* > *dimension*
and *A-in-V*: *A* ⊆ *carrier V*
shows *linear-dependent A*
⟨*proof*⟩

The following lemma proves that the cardinality of any spanning set is greater than the dimension. In the infinite case (when A is not finite but is a *spanning-set-ext*) it would be trivial, but Isabelle assigns 0 as the cardinality of an infinite set.

We will use *swap-theorem*, so $\mathbf{0}_V$ must not be in the *spanning-set* over we apply it.

lemma *card-sg-ge-dim*:
assumes *sg-A*: *spanning-set A*
shows *card A* ≥ *dimension*
⟨*proof*⟩

There not exists a *spanning-set* with cardinality less than the dimension.

corollary *card-less-dim-implies-not-sg*:
assumes *cardA-l-dim*: *card A* < *dimension*
shows ¬ *spanning-set A*
⟨*proof*⟩

If we have a set which cardinality is equal to the dimension of a finite vector space, then it is a finite set. We have to assume that the basis is not empty: if X is empty, then $card(X) = 0 = card(A)$. However and due to the implementation of cardinality in Isabelle (giving 0 as the cardinality of an infinite set), we could only prove that either A is infinite or empty.

lemma *card-eq-not-empty-basis-implies-finite*:
assumes *cardA-dim*: *card A* = *dimension*
and *X-not-empty*: *X*≠{}

shows *finite A*
⟨*proof*⟩

Assuming that A is in V, the problem is solved.

lemma *card-eq-basis-implies-finite*:
assumes *cardA-dim*: *card A* = *dimension*
and *A-in-V*: $A \subseteq$ *carrier V*
shows *finite A*
⟨*proof*⟩

If a set has cardinality equal to the dimension, if it is a basis then is independent.

lemma *card-eq-basis-imp-li*:
assumes *cardA-dim*: *card A* = *dimension*
shows *basis A* ⟹ *linear-independent A*
⟨*proof*⟩

If we have an independent set with cardinality equal to the dimension, then this set is a basis.

lemma *card-li-set-eq-basis-imp-li*:
assumes *card-eq-dim*: *card A* = *dimension*
shows *linear-independent A* ⟹ *basis A*
⟨*proof*⟩

If a spanning set has cardinality equal to the dimension, then is independent (so a basis).

lemma *card-sg-set-eq-basis-imp-li*:
assumes *card-eq-dim*: *card A* = *dimension*
shows *spanning-set A* ⟹ *linear-independent A*
⟨*proof*⟩

corollary *card-sg-set-eq-basis-imp-basis*:
assumes *card-eq-dim*: *card A* = *dimension*
shows *spanning-set A* ⟹ *basis A*
⟨*proof*⟩

lemma *basis-iff-linear-independent*:
assumes *card-eq*: *card A* = *dimension*
shows *basis A* ⟷ *linear-independent A*
⟨*proof*⟩

We can remove from *eq-cardinality-basis2* the premises about finiteness: we can prove that in a finite dimensional vector space there not exist infinite bases.

lemma
not-finite-A-contains-empty-set:
assumes *A*: ¬ *finite A*

shows $\{\} \subseteq A$
⟨*proof*⟩

lemma *not-finite-diff*:
assumes *A*: $\neg$ *finite A*
shows $\neg$ *finite* $(A - \{x\})$
⟨*proof*⟩

lemma *not-finite-diff-set*:
assumes *A*: $\neg$ *finite A*
and *B*: *finite B*
shows $\neg$ *finite* $(A - B)$
⟨*proof*⟩

We can obtain a subset with the number of elements that we want from an infinite set:

lemma *subset-card-n*:
assumes *A*: $\neg$ *finite A*
shows $\forall k{::}nat.\ \exists B.\ B \subseteq A \wedge card\ B = k$
⟨*proof*⟩

Every basis of a finite dimensional vector space is finite (because each set of cardinality greater than the dimension is linearly dependent (*card-g-dim-implies-ld*), so we can not have an infinite basis).

lemma *basis-not-infinite*:
assumes *basis-A*: *basis A*
shows *finite A*
⟨*proof*⟩

Finally the theorem:

lemma *eq-cardinality-basis'*:
assumes *A*: *basis A* **and** *B*: *basis B*
shows *card A* = *card B*
⟨*proof*⟩

end
end

theory *Isomorphism*
imports *Dimension*
begin

11 Isomorphism

The *types* keyword seems to be replaced by *type-synonym* in the Isabelle 2011 release.

The following definition of *vector* has been obtained from the *AFP*, where a similar one is defined over *real*, instead of $'a$, for defining the Cauchy-Schwarz Inequality http://afp.sourceforge.net/entries/Cauchy.shtml.

22-07-2011: JE: For some time I thought that many of the proofs required the vector spaces to be non empty (not of dimension zero). This is why one can meet a lot of premises of the type $(0::'a) < n$ or about the dimension being non zero (all these premises are now enclosed between comments). After a closer look I could remove each of these premises and make everything general for every finite dimension.

types $'a\ vector = (nat => 'a) * nat$

definition
$ith :: 'a\ vector => nat => 'a$
where $ith\ v\ i = fst\ v\ i$

definition
$vlen :: 'a\ vector => nat$
where $vlen\ v = snd\ v$

Before getting into the definition of the vector space K_n, we introduce a generic lemma that states that the decomposition of an element $x \in carrier\ V$ as a linear combination of the elements of a given basis is unique.

The lemma requires the basis X to be finite, because otherwise there would be a linear combination of the infinite number of elements of the basis equal to zero, but the *finsum* of an infinite set is undefined, and thus we cannot complete the proof.

context *abelian-group*
begin

Some previous lemmas about addition in abelian monoids.

lemma
add-minus-add-minus:
assumes a: $a \in carrier\ G$
and b: $b \in carrier\ G$
and c: $c \in carrier\ G$
shows $a \oplus b \ominus c = a \oplus (b \ominus c)$
⟨*proof*⟩

lemma
minus-add-minus-minus:
assumes a: $a \in carrier\ G$
and b: $b \in carrier\ G$
and c: $c \in carrier\ G$
shows $a \ominus (b \oplus c) = a \ominus b \ominus c$
⟨*proof*⟩

thm *minus-add*
⟨*proof*⟩

lemma
add-minus-add-minus-add-minus:
assumes *a*: $a \in carrier\ G$
and *b*: $b \in carrier\ G$
and *c*: $c \in carrier\ G$
and *d*: $d \in carrier\ G$
shows $a \oplus b \ominus (c \oplus d) = a \ominus c \oplus (b \ominus d)$
⟨*proof*⟩

corollary *add-minus-add-minus-add-minus-comm*:
assumes *a*: $a \in carrier\ G$
and *b*: $b \in carrier\ G$
and *c*: $c \in carrier\ G$
and *d*: $d \in carrier\ G$
shows $a \oplus b \ominus (c \oplus d) = b \ominus d \oplus (a \ominus c)$
⟨*proof*⟩

lemma *finsum-minus-eq*:
assumes *fin-A*: *finite A*
and *f-PI*: $f \in A \to carrier\ G$
shows $\ominus\ finsum\ G\ f\ A = finsum\ G\ (\lambda x.\ \ominus f\ x)\ A$
⟨*proof*⟩

end

context *vector-space*
begin

The following function should replace to *coefficients-function*; the problem with *coefficients-function* is that it does not impose any condition over functions out of their domain, *carrier V*; thus, we cannot prove that two coefficient functions which are equal over their corresponding domain (the basis X) are equal. We have to impose an additional restriction that the function out of its domain is equal to **0**

end

11.1 Definition of $\mathbb{K}^n$

context *field*
begin

The following definition represents the carrier set of the vector space. Note that the type variable is now $'a$, so we define only the following concepts over the field of the coefficients.

— Seleccionamos un representante cannico para cada elemento, haciendo que todas las coordenadas sean cero por encima de la dimensin del espacio vectorial

— Adems, debemos asegurar que la dimensin del vector, o la longitud del mismo, sea igual al nmero de componenetes en el que estamos interesados; sino perderamos la inyectividad de algunas operaciones

— Hay que tener en cuenta que en una lista de 1 elemento (por ejemplo, los elementos del carrier de K1) nos interesa nicamente el elemento en la posicin 0, de ah que nos interesen los elementos con vlen = n - 1;

Para los elementos en *K-n-carrier A* ($0::'d$) debemos observar que su primera componente ser **0** y su segunda componente ser tambin *0*, lo que nos deja con un K_0 cuyo nico elemento es el $0::'c$ de la estructura correspondiente (K_n).

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*))}

lemma *ith-closed*:
  assumes *k*: *k* ∈ *K-n-carrier A n* **and** *i*: *i* ∈ {..<*n*}
  shows *ith k i* ∈ *A*
  ⟨*proof*⟩

lemma *K-n-carrier-zero*:
  K-n-carrier A 0 = {*v*. (*ith v 0* = **0**) ∧ (∀ *i*>*0*. *ith v i* = **0**) ∧ (*vlen v* = *0*)}
  ⟨*proof*⟩

lemma *K-n-carrier-zero-ext*: *K-n-carrier A 0* = {(λ*i*. **0**, *0*)}
  ⟨*proof*⟩

lemma *K-n-carrier-one*:
  K-n-carrier A 1 = {*v*. *ith v 0* ∈ *A* ∧ (∀ *i*≥*1*. *ith v i* = **0**) ∧ (*vlen v* = *0*)}
  ⟨*proof*⟩

definition
  K-n-add :: *nat => 'a vector => 'a vector => 'a vector* (**infixr** ⊕₁ *65*)
  where *K-n-add n* = (λ*v w*. ((λ*i*. *ith v i* ⊕$_R$ *ith w i*), *n* − *1*))

lemma *K-n-add-zero*:
  shows *K-n-add 0* = (λ*v w*. ((λ*i*. *ith v i* ⊕$_R$ *ith w i*), *0*))
  ⟨*proof*⟩

definition *K-n-mult* :: *nat => 'a vector => 'a vector => 'a vector*
  where *K-n-mult n* = (λ*v w*. ((λ*i*. *ith v i* ⊗$_R$ *ith w i*), *n* − *1*))

lemma *K-n-mult-zero*:
  shows *K-n-mult 0* = (λ*v w*. ((λ*i*. *ith v i* ⊗$_R$ *ith w i*), *0*))
  ⟨*proof*⟩

definition *K-n-zero* :: *nat => 'a vector*

where *K-n-zero n* = ((λ*i*. $\mathbf{0}_R$), *n* − *1*)

lemma *K-n-zero-zero*:
shows *K-n-zero 0* = ((λ*i*. $\mathbf{0}_R$), *0*)
⟨*proof*⟩

definition *K-n-one* :: *nat* => *'a vector*
where *K-n-one n* = ((λ*i*. $\mathbf{1}_R$), *n* − *1*)

Actually, in the following case, one should be equal to zero

lemma *K-n-one-zero*:
shows *K-n-one 0* = ((λ*i*. $\mathbf{1}_R$), *0*)
⟨*proof*⟩

We are now forced to define also operations *K-n-mult* and *K-n-one* for our *abelian group* K^n. This is due to the fact that the abelian group predicate in the Algebra Library is defined over rings, and even if we have no interest in using that operations (they are not required to prove that an algebraic structure is an abelian group), they must be defined somehow. In our case this is not a major problem, since they can be defined just following the previous definitions of *K-n-zero* and *K-n-add*.

definition *K-n* :: *nat* => *'a vector ring*
where
K-n n = (| *carrier* = *K-n-carrier* (*carrier R*) *n*,
mult = (λ*v w*. *K-n-mult n v w*),
one = *K-n-one n*,
zero = *K-n-zero n*,
add = (λ*v w*. *K-n-add n v w*)|)

lemma *abelian-group-K-n*:
shows *abelian-group* (*K-n n*)
⟨*proof*⟩

corollary *abelian-monoid-K-n*:
shows *abelian-monoid* (*K-n n*)
⟨*proof*⟩

We are later to consider *K-n* like one abelian group over which *R* gives place to a vector space. We must define first the scalar product between both structures.

definition
K-n-scalar-product :: *'a* => *'a vector* => *'a vector*
(**infixr** ⊙ *65*)
where *a* ⊙ *b* = (λ*n*::*nat*. *a* ⊗$_R$ *ith b n*, *vlen b*)

lemma *K-n-scalar-product-closed*:
assumes *a*: *a* ∈ *carrier R*
and *b*: *b* ∈ *carrier* (*K-n n*)

shows $a \odot b \in$ *carrier* (*K-n n*)
⟨*proof*⟩

lemma *field-R*: *field R*
⟨*proof*⟩

lemma
vector-space-K-n:
shows *vector-space R* (*K-n n*) (*op* $\odot$)
⟨*proof*⟩

11.2 Canonical basis of $\mathbb{K}^n$:

In the following section we introduce the elements that generate the canonical basis of the vector space *K-n n* and prove some properties of them.

The elements of the canonical basis of *K-n* are the following ones:

definition *x-i* :: *nat* => *nat* => *'a vector*
where *x-i j n* = ((λ*i*. *if i* = *j then* **1** *else* **0**), *n* − *1*)

The elements *x-i* are part of the *carrier* (*K-n n*).

lemma
x-i-closed:
assumes *j-l-n*: *j* < *n*
shows *x-i j n* ∈ *carrier* (*K-n n*)
⟨*proof*⟩

Any two elements of the basis are different:

lemma *x-i-ne-x-j*:
assumes *i-ne-j*: *i* ≠ *j*
shows *x-i i n* ≠ *x-i j n*
⟨*proof*⟩

In the following lemma we can even omit the premise of i being smaller than n, so the result is also true for vectors which are not part of the canonical basis. It claims that an element of the canonical basis is not equal to $\mathbf{0}_{K\text{-}n\ n}$

lemma *x-i-ne-zero*:
shows *x-i i n* ≠ $\mathbf{0}_{K\text{-}n\ n}$
⟨*proof*⟩

end

context *vector-space*
begin

lemma

coefficients-function-Pi:
assumes x: $x \in$ *carrier* V
and *cf-f*: $f \in$ *coefficients-function* A
shows $f\ x \in$ *carrier* K
⟨*proof*⟩

end

context *abelian-group*
begin

lemma
finsum-twice:
assumes f: $f \in \{i,j\} \to$ *carrier* G
and *i-ne-j*: $i \neq j$
shows *finsum* $G\ f\ \{i, j\} = f\ i \oplus f\ j$
⟨*proof*⟩

end

context *comm-monoid*
begin

lemma *mult-if*:
shows $(\lambda k.\ x \otimes (\textit{if}\ k = i\ \textit{then}\ y\ \textit{else}\ z)) = (\lambda k.\ \textit{if}\ k = i\ \textit{then}\ x \otimes y\ \textit{else}\ x \otimes z)$
⟨*proof*⟩

end

lemma
fun-eq-contr:
assumes *fg*: $f = g$
and x: $f\ x \neq g\ x$
shows *False* ⟨*proof*⟩

context *abelian-monoid*
begin

lemma
finsum-singleton-set:
assumes f: $f\ a \in$ *carrier* G
shows *finsum* $G\ f\ \{a\} = f\ a$
⟨*proof*⟩

end

context *field*
begin

lemma *comm-monoid-R*: *comm-monoid R* ⟨*proof*⟩

lemma *abelian-monoid-R*: *abelian-monoid R* ⟨*proof*⟩

Some previous about the linear independece of the elements of the canonical basis:

lemma *x-i-li*:
assumes *j-l-n*: $j < n$
shows *vector-space.linear-independent R* (*K-n n*) (*op* ⊙) {(*x-i j n*)}
⟨*proof*⟩

Any two different elements of the canonical basis are linearly independent:

lemma *x-i-x-j-li*:
assumes *j-l-n*: $j < n$
and *i-l-n*: $i < n$
and *i-ne-j*: $i \neq j$
shows *vector-space.linear-independent R* (*K-n n*) (*op* ⊙) {(*x-i i n*), (*x-i j n*)}
⟨*proof*⟩

We did not find a better way to define the elements of the canonical basis than accumulating them iteratively. In order to define them as a range, from *x-i 0 n* up to *x-i* (*n − 1*) *n*, the underlying type, in this case *'a vector*, should be of sort "order" (which in general is not, only the elements of the basis have some notion of order.)

The following fuction iteratively joins all the elements of the form *x-i k n* in order to create the canonical basis of *K-n n*.

We have considered as a special case the situation where both indexes are equal to *0*. This case will give us the basis of *K-n 0*, which is the empty set. Note that a linear combination over an empty set is equal to (λ*i*. $\mathbf{0}_K$, *0*), which is the only element in *carrier* (*K-n 0*).

fun *canonical-basis-acc* :: *nat* => *nat* => *'a vector set*
where
canonical-basis-acc 0 0 = {}
| *canonical-basis-acc 0 n* = {*x-i 0 n*}
| *canonical-basis-acc* (*Suc i*) *n*
= (*if* (*Suc i* < *n*) *then*
insert (*x-i* (*Suc i*) *n*) (*canonical-basis-acc i n*) *else* {})

We now prove some lemmas trying to establish the relation between the elements of the form *x-i i n* and the ones in *canonical-basis-acc*.

lemma
finite-canonical-basis-acc:
shows *finite* (*canonical-basis-acc k n*)
⟨*proof*⟩

lemma
canonical-basis-acc-closed:
assumes *i-l-j*: $i < j$
shows *canonical-basis-acc i j* $\subseteq$ *carrier* (*K-n j*)
⟨*proof*⟩

The canonical basis in dimension n is given by all elements ranging from *x-i 0 n* up to *x-i* $(n - 1)$ *n*

definition *canonical-basis-K-n* :: *nat* => *'a vector set* **where**
canonical-basis-K-n n = *canonical-basis-acc* $(n - 1)$ *n*

lemma
canonical-basis-acc-insert:
assumes *j-l-k*: $j < k$
and *k-l-n*: $k < n$
shows *x-i k n* $\notin$ *canonical-basis-acc j n*
⟨*proof*⟩

lemma
card-canonical-basis-acc:
assumes *k-le-n*: $k < n$
shows *card* (*canonical-basis-acc k n*) = *Suc k*
⟨*proof*⟩

end

lemma
n-minus-one-l-n:
assumes *n-g-0*: $0 < n$
shows $n - (1::nat) < n$
⟨*proof*⟩

context *field*
begin

The following lemma is true for dimension *0* thanks to the special case *canonical-basis-acc 0 0* = {} previously introduced:

lemma
canonical-basis-K-n-closed:

shows *canonical-basis-K-n n* $\subseteq$ *carrier* (*K-n n*)
⟨*proof*⟩

The following lemma is true for dimension *0* thanks to the special case *canonical-basis-acc 0 0* = {} previously introduced:

lemma
card-canonical-basis-K-n:

shows *card* (*canonical-basis-K-n n*) = *n*
⟨*proof*⟩

The following lemma does not even require to have a dimension greater than *0*.

lemma
finite-canonical-basis-K-n:

shows *finite* (*canonical-basis-K-n n*)
⟨*proof*⟩

lemma
canonical-basis-acc-insert2:
assumes *j-le-k*: $j \leq k$
and *k-l-n*: $k < n$
shows *x-i j n* ∈ *canonical-basis-acc k n*
⟨*proof*⟩

lemma
canonical-basis-K-n-elements:
assumes *j-in-n*: $j \in \{..<n\}$
shows *x-i j n* ∈ *canonical-basis-K-n n*
⟨*proof*⟩

lemma
canonical-basis-K-n-good-set:

shows *vector-space.good-set* (*K-n n*) (*canonical-basis-K-n n*)
⟨*proof*⟩

end

JE: I have moved this definition to *Finite-Vector-Space*, so I remove it from here. This is to be checked with the other files.

11.3 Theorem on bijection

context *abelian-monoid*
begin

We need to prove the following lemma which is a generic version of the theorem *finsum-cong*:

⟦*A* = *B*; (*f* ∈ *B* → *carrier G*) = *True*; ⋀*i*. *i* ∈ *B* =*simp*=> *f i* = *g i*⟧ ⟹ *finsum G f A* = *finsum G g B* in the case where finite sums are defined over sets of different type, but isomorphic (in *finsum-cong* only the case where both sets of both finite sums are equal is considered).

lemma *finsum-cong''*:
assumes *fB*: *finite B*
and *bb*: *bij-betw h B A*
and *f*: $f : A \to$ *carrier G* **and** *g*: $g : B \to$ *carrier G*
and *eq*: $(\bigwedge x.\ x \in B \Longrightarrow g\ x = f\ (h\ x))$
shows *finsum G f A* = *finsum G g B*
⟨*proof*⟩

end

lemma *n-notin-lessThan-n*: $(n::nat) \notin \{..<n\}$
⟨*proof*⟩

context *field*
begin

lemma
snd-in-carrier:
assumes *x*: $x \in$ *carrier* (*K-n n*)
shows *snd x* = *n* − *1*
⟨*proof*⟩

The following lemma gives a different representation of the elements of *K-n n*; this representation will be later used to prove that the elements of *K-n n* can be expressed as linear combinations of the elements of *canonical-basis-K-n n*.

lemma
x-in-carrier:
assumes *x*: $x \in$ *carrier* (*K-n n*)
shows $x = (\lambda i.$ *if* $i \in \{..<n\}$ *then fst x i else* **0**, *n* − *1*$)$
⟨*proof*⟩

The following lemma was later unused; every element can be "embedded" into a smaller dimension by means of "forgetful" function (we forget the last position of the vector).

lemma
K-n-carrier-embed:
assumes *x*: $x \in$ *carrier* (*K-n* (*Suc k*))
shows $((\lambda n.$ *if* $n \in \{..<k\}$ *then fst x n else* **0**$)$, *k* − *1*$) \in$ *carrier* (*K-n k*)
⟨*proof*⟩

Functions with only a single nonzero element can be expressed as scalar products of *x-i* elements.

lemma
singleton-function-x-i:
assumes *x*: $x \in$ *carrier R*
shows $(\lambda i.$ *if* $i = j$ *then x else* **0**, *n* − *1*$) = x \odot$ *x-i j n*
⟨*proof*⟩

The following lemma is rather important, since it shows how to express any element in *carrier* (*K-n k*) in a canonical way: it proves that any element in *carrier* (*K-n k*) can be expressed as a finite sum of the elements *x-i j k*.

It is important to note that in the proof we have introduced an extra natural variable n, with $n \leq k$, which permits to prove the result by induction in n over the field *K-n k*.

If we do not use the extra variable n and we apply induction directly over k, the induction step will produce two different algebraic structures, *K-n k*, where the property holds, and *K-n* (*Suc k*), where the property must be proved, but then the induction hypothesis cannot be used.

lemma
lambda-finsum:
assumes *cl*: $\forall\, i{\in}\{..{<}n\}.\ x\ i \in$ *carrier R*
and *n-le-k*: $n \leq k$
shows $(\lambda i.$ *if* $i \in \{..{<}n\}$ *then x i else* $\mathbf{0},\ k - 1) =$
finsum (*K-n k*) $(\lambda i.\ x\ i \odot$ *x-i i k*) $\{..{<}n\}$
⟨*proof*⟩

Now, as a corollary of the previous result, we obtain that any element of *K-n n* can be expressed as a finite sum of the elements of the form *x-i j n*.

lemma *lambda-finsum-n*:
assumes *cl*: $\forall\, i{\in}\{..{<}n\}.\ x\ i \in$ *carrier R*
shows $(\lambda i.$ *if* $i \in \{..{<}n\}$ *then x i else* $\mathbf{0},\ n - 1) =$
finsum (*K-n n*) $(\lambda i.\ x\ i \odot$ *x-i i n*) $\{..{<}n\}$
⟨*proof*⟩

Finally, we get the lemma that states tha any element of the set *K-n-carrier n* is a linear combination of elements of *canonical-basis-K-n n*:

lemma
K-n-carrier-finsum-x-i:
assumes *x*: $x \in$ *carrier* (*K-n n*)
shows $x =$ *finsum* (*K-n n*) $(\lambda j.$ *fst x j* $\odot$ *x-i j n*) $\{..{<}n\}$
⟨*proof*⟩

11.4 Bijection between basis:

In the following lemmas we try to establish an explicit bijection between the sets X, which is a basis of V, and the set *canonical-basis-K-n n*. This bijection will be later extended, by linearity, to a bijection between *carrier* V and *carrier* (*K-n n*)

lemma *canonical-basis-acc-eq-x-i*:
assumes *x*: $x \in$ *canonical-basis-acc k n*
and *k-l-n*: $k < n$
shows $\exists\, j{\in}\{..{<}$*Suc k*$\}.$ *x-i j n* $= x$
⟨*proof*⟩

corollary
canonical-basis-acc-isom-x-i:
assumes *x*: $x \in$ *canonical-basis-acc k n*
and *k-l-n*: $k < n$
shows $\exists!j{\in}\{..{<}Suc\ k\}.\ x = x\text{-}i\ j\ n$
⟨*proof*⟩

corollary
canonical-basis-acc-isom-x-i2:
assumes *x*: $x \in$ *canonical-basis-acc k n*
and *k-l-n*: $k < n$
shows $\exists!j{\in}\{..{<}n\}.\ x = x\text{-}i\ j\ n$
⟨*proof*⟩

lemma
canonical-basis-is-x-i:
assumes *x*: $x \in$ *canonical-basis-K-n n*

shows $\exists j{\in}\{..{<}n\}.\ x = x\text{-}i\ j\ n$
⟨*proof*⟩

corollary
canonical-basis-isom-x-i:
assumes *x*: $x \in$ *canonical-basis-K-n n*

shows $\exists!j{\in}\{..{<}n\}.\ x = x\text{-}i\ j\ n$
⟨*proof*⟩

The function *preim* maps vectors of the basis *canonical-basis-K-n n* to their index.

definition
preim :: *'a vector => nat => nat*
where $preim\ x\ n = (THE\ j.\ j \in \{..{<}n\} \wedge x = x\text{-}i\ j\ n)$

lemma
preim-x-i-x-eq-x:
assumes *x-l-n*: $x < n$

shows $preim\ (x\text{-}i\ x\ n)\ n = x$
⟨*proof*⟩

lemma
preim-eq-x-i-acc:
assumes *x*: $x \in$ *canonical-basis-acc k n*
and *k-l-n*: $k < n$
shows $x\text{-}i\ (preim\ x\ n)\ n = x$
⟨*proof*⟩

lemma
preim-eq-x-i:
assumes x: $x \in$ *canonical-basis-K-n n*

shows *x-i* (*preim x n*) $n = x$
⟨*proof*⟩

lemma
preim-lessThan:
assumes x: $x \in$ *canonical-basis-K-n n*

shows *preim x n* $\in \{..<n\}$
⟨*proof*⟩

11.5 Properties of *canonical-basis-K-n n*:

The following lemma proves that two different ways of writing down an element of *K-n n* as a linear combination of the elements of the basis *canonical-basis-K-n n* are equivalent.:

lemma
finsum-canonical-basis-acc-finsum-card:
assumes *k-l-n*: $k < n$
and f: $f \in$ *carrier* (*K-n n*) $\rightarrow$ *carrier R*
shows $(\bigoplus_{K\text{-}n\ n} x \in$ *canonical-basis-acc k n*. $f\ x \odot x)$
$= (\bigoplus_{K\text{-}n\ n} k \in \{..<$*Suc k*$\}$. f (*x-i k n*) $\odot$ *x-i k n*)
⟨*proof*⟩

lemma
finsum-canonical-basis-K-n-finsum-card:
assumes f: $f \in$ *carrier* (*K-n n*) $\rightarrow$ *carrier R*
shows $(\bigoplus_{K\text{-}n\ n} x \in$ (*canonical-basis-K-n n*). $f\ x \odot x)$
$= (\bigoplus_{K\text{-}n\ n} k \in \{..<n\}$. f (*x-i k n*) $\odot$ *x-i k n*)
⟨*proof*⟩

The space generated by the *vector-space.span* of *canonical-basis-K-n n* is equal to the vector space *K-n n*.

lemma
span-canonical-basis-K-n-carrier-K-n:

shows *vector-space.span R* (*K-n n*) (*op* $\odot$) (*canonical-basis-K-n n*) = *carrier* (*K-n n*)
⟨*proof*⟩

lemma
canonical-basis-K-n-spanning-set:

shows *vector-space.spanning-set R* (*K-n n*) (*op* $\odot$) (*canonical-basis-K-n n*)
⟨*proof*⟩

The elements of *canonical-basis-acc j n* are linearly independent.

lemma
canonical-basis-acc-linear-independent-ext:
assumes *j-l-n*: $j < n$
shows *vector-space.linear-independent-ext R* (*K-n n*) (*op* $\odot$) (*canonical-basis-acc j n*)
⟨*proof*⟩

end

context *vector-space*
begin

The following lemma should be moved to the place where *linear-independent-ext* has been defined, like a *simp* rule:

lemma *linear-independent-ext-empty* [*simp*]:
shows *linear-independent-ext* {}
⟨*proof*⟩

end

context *field*
begin

lemma
canonical-basis-K-n-linear-independent-ext:

shows *vector-space.linear-independent-ext R* (*K-n n*) (*op* $\odot$) (*canonical-basis-K-n n*)
⟨*proof*⟩

We finally prove that *canonical-basis-K-n n* is a basis for *K-n*.

lemma
canonical-basis-K-n-basis:

shows *vector-space.basis R* (*K-n n*) (*op* $\odot$) (*canonical-basis-K-n n*)
⟨*proof*⟩

corollary
canonical-basis-K-n-basis-card-n:

shows *vector-space.basis R* (*K-n n*) (*op* $\odot$) (*canonical-basis-K-n n*) $\wedge$
card (*canonical-basis-K-n n*) = *n*
⟨*proof*⟩

end

context *finite-dimensional-vector-space*

begin

After proving the most relevant properties of *field.K-n K n*, we fix one indexing of the basis elements (of X) that will allow us to define later the function which given any element of the carrier set decomposes it into the coefficients for each term if the indexation.

The theorem *obtain-indexing*: *finite A* $\Longrightarrow \exists f.$ *indexing* (A, f) and the premise that the vector space is finite, and so is it basis X, ensures that the following definition is sound.

definition *indexing-X* :: *nat* => *'c*
  where *indexing-X-def*: *indexing-X* = (*SOME f*. *indexing* (*X*, *f*))

Relying in the fact that at least one indexing of the basis X exists, we can prove that *indexing-X* satisfies the properties of every *indexing*.

lemma *indexing-X-is-indexing*:
  shows *indexing* (*X*, *indexing-X*)
  ⟨*proof*⟩

The following function is to be used as the inverse function of *field.preim*; this function and *field.preim* will be defined to prove an isomorphism between *field.canonical-basis-K-n K* (*card X*) and {..<*card X*}.

definition *iso-nat-can* :: *nat* => *'a vector*
  where *iso-nat-can n* = (*x-i n* (*dimension*))

The composition of the functions *field.preim K* and *iso-nat-can* over the set {..<*dimension*} is equal to the identity.

lemma *preim-iso-nat-can-id*:
  assumes *x*: *x* ∈ {..<*dimension*}
  shows *preim* (*iso-nat-can x*) (*dimension*) = *x*
  ⟨*proof*⟩

In a very similar way, the composition of *field.preim K* and *iso-nat-can* over the set *field.canonical-basis-K-n K dimension* is equal to the identity:

lemma *iso-nat-can-preim-id*:
  assumes *y*: *y* ∈ *canonical-basis-K-n* (*dimension*)
  shows *iso-nat-can* (*preim y* (*dimension*)) = *y*
  ⟨*proof*⟩

lemma
  bij-betw-iso-nat-can:
  shows *bij-betw iso-nat-can* {..<*dimension*}
  (*canonical-basis-K-n* (*dimension*))
⟨*proof*⟩

lemma
  bij-betw-preim:

shows *bij-betw* ($\lambda i.$ *preim i* (*dimension*))
(*canonical-basis-K-n* (*dimension*)) {..<*dimension*}
⟨*proof*⟩

The following function will be used to define an isomorphism between the sets {..<*dimension*} and X, which inverse will be the inverse of the indexing function *indexing-X*.

definition
iso-nat-X :: *nat* => *'c*
where *iso-nat-X n* = *indexing-X n*

The inverse function of the previous *iso-nat-X* is the following function, which properties we are to prove first:

definition
preim2 :: *'c* => *nat*
where *preim2 x* = (*THE j*. *j* ∈ {..<*dimension*} ∧ *x* = *indexing-X j*)

The *preim2* function needs to be completed, since otherwise we can not ensure for the elements out of the basis X that their value *preim2 x* is not in the set {..<*dimension*}. If the value *preim2 x* could be in {..<*dimension*} for elements out of X, then the function *fst x* (*preim2 y*), for $y \notin X$ could take values different from **0**.

The way to complete it is a bit artificial, since we can not use *0* to complete it, but some element a with *dimension* $\leq a$, which are the natural numbers that are mapped to **0** by *coefficients-function*. In particular, we have chosen a = *dimension*.

definition
preim2-comp :: *'c* => *nat*
where *preim2-comp x* = (*if x* ∈ *X then* (*THE j*. *j* ∈ {..<*dimension*} ∧ *x* = *indexing-X j*) *else dimension*)

lemma
indexing-X-bij:
shows *bij-betw indexing-X* {..<*dimension*} *X*
⟨*proof*⟩

lemma
indexing-X-preimage:
assumes *x*: *x* ∈ *X*
shows ∃*j*. *j* ∈ {..<*dimension*} ∧ *x* = *indexing-X j*
⟨*proof*⟩

corollary
indexing-X-preimage-unique:
assumes *x*: *x* ∈ *X*
shows ∃!*j*. *j* ∈ {..<*dimension*} ∧ *x* = *indexing-X j*
⟨*proof*⟩

lemma
preim2-in-dimension:
assumes x: $x \in X$
shows *preim2* $x \in \{..<dimension\}$
⟨*proof*⟩

lemma
preim2-comp-in-dimension:
assumes x: $x \in X$
shows *preim2-comp* $x \in \{..<dimension\}$
⟨*proof*⟩

lemma
preim2-is-indexing-X:
assumes x: $x \in X$
shows $x =$ *indexing-X* (*preim2* x)
⟨*proof*⟩

The functions *preim2-comp* and *iso-nat-X* are inverse of each other, over the sets X and $\{..<dimension\}$

lemma
preim2-comp-is-indexing-X:
assumes x: $x \in X$
shows $x =$ *indexing-X* (*preim2-comp* x)
⟨*proof*⟩

lemma *iso-nat-X-preim2-id*:
assumes x: $x \in X$
shows *iso-nat-X* (*preim2* x) $= x$
⟨*proof*⟩

lemma *iso-nat-X-preim2-comp-id*:
assumes x: $x \in X$
shows *iso-nat-X* (*preim2-comp* x) $= x$
⟨*proof*⟩

lemma *preim2-iso-nat-X-id*:
assumes n: $n \in \{..<dimension\}$
shows *preim2* (*iso-nat-X* n) $= n$
⟨*proof*⟩

lemma *preim2-comp-iso-nat-X-id*:
assumes n: $n \in \{..<dimension\}$
shows *preim2-comp* (*iso-nat-X* n) $= n$
⟨*proof*⟩

Therefore, we can prove that there exists a bijection between them:

lemma

bij-betw-iso-nat-X:
shows *bij-betw iso-nat-X* {..<*dimension*} *X*
⟨*proof*⟩

lemma
bij-betw-preim2:
shows *bij-betw preim2 X* {..<*dimension*}
⟨*proof*⟩

end

11.6 Linear maps.

In this section we are going to introduce the notion of linear map between vector spaces. This is a previous step for the definition of an isomorphism between vector spaces. Then, we will have to prove the existence of an isomorphism between the vector spaces *K-n dimension* and *V*.

The definition between comments would be the expected and desired one. Unfortunately, it introduces changes in the namespace that are really inconvenient. The second locale hides the names of constants in vector space, demanding long names for the first locale constanst. We do not know how to control this behaviour: thus, we preferred the long version, in which locale interpretation has to be done later by hand:

locale *linear-map* =
fixes *K* :: ('*a*, '*b*) *ring-scheme*
and *V* :: ('*c*, '*d*) *ring-scheme*
and *W* :: ('*e*, '*f*) *ring-scheme*
and *scalar-product1* :: '*a* => '*c* => '*c* (**infixr** $\cdot_V$ *70*)
and *scalar-product2* :: '*a* => '*e* => '*e* (**infixr** $\cdot_W$ *70*)
assumes *V*: *vector-space K V* (*op* $\cdot_V$)
and *W*: *vector-space K W* (*op* $\cdot_W$)

context *linear-map*
begin

Linear maps, as characterised in "Linear Algebra Done Right", have to satisfy the additivity and homogeneity properties:

definition *additivity* :: ('*c* => '*e*) => *bool*
where *additivity T* = (∀ *x*∈*carrier V*. ∀ *y* ∈ *carrier V*. *T* (*x* $\oplus_V$ *y*) = *T x* $\oplus_W$ *T y*)

definition *homogeneity* :: ('*c* => '*e*) => *bool*
where *homogeneity T* = (∀ *k*∈*carrier K*. ∀ *x*∈*carrier V*. *T* (*k* $\cdot_V$ *x*) = *k* $\cdot_W$ *T x*)

definition *linear-map* :: ('*c* => '*e*) => *bool*
where *linear-map T* = (*additivity T* ∧ *homogeneity T*)

end

We introduce a new locale for finite dimensional vector spaces, just imposing that there is a finite basis for one ot the vector spaces.

locale *linear-map-fin-dim* = *linear-map* +
  fixes *X*
  assumes *fin-dim*: *finite-dimensional-vector-space K V* (*op* $\cdot_V$) *X*

We produce two different sublocales, or interpretations, of the locale *linear-map-fin-dim* by means of the locale *finite-dimensional-vector-space*. They allow us to later define linear maps from V to *K-n* and also the opposite way, from *K-n* to V. The system forces us to make them *named* interpretations, just to avoid colliding names.

sublocale *finite-dimensional-vector-space* <
  V-K-n: *linear-map-fin-dim K V K-n dimension op* $\cdot$ *K-n-scalar-product X*
⟨*proof*⟩

sublocale *finite-dimensional-vector-space* < *K-n-V*: *linear-map-fin-dim K K-n dimension V*
  K-n-scalar-product op $\cdot$ *canonical-basis-K-n dimension*
⟨*proof*⟩

11.7 Defining the isomorphism between $\mathbb{K}^n$ and V.

context *finite-dimensional-vector-space*
begin

Some properties proving that there exists a unique function of coefficients for each element in the carrier set of V; this unique function is the one that decomposes any element into its linear combination over the elements of the basis:

lemma
  basis-implies-linear-combination:
  assumes *x*: (*x*::'*c*) ∈ *carrier V*
  shows ∃*f*. *f* ∈ *coefficients-function* (*carrier V*) ∧ *x* = *linear-combination f X*
  ⟨*proof*⟩

In order to ensure the uniqueness of the coefficients function we have to use *coefficients-function*, which is mapped to **0** out of its domain.

lemma
  basis-implies-coeff-function-comp-linear-combination:
  assumes *x*: (*x*::'*c*) ∈ *carrier V*
  shows ∃*f*. *f* ∈ *coefficients-function X* ∧ *x* = *linear-combination f X*
⟨*proof*⟩

Firstly we prove a theorem similar to *unique-coordenates*: ⟦*x* ∈ *carrier V*; *f* ∈ *coefficients-function* (*carrier V*); *x* = *linear-combination f X*; *g* ∈

coefficients-function (*carrier V*); x = *linear-combination g X*$\rrbracket \Longrightarrow \forall\, x{\in}X.$ $g\ x = f\ x$. It claims that the coordinates are unique in a basis.

lemma
linear-combination-unique:
assumes x: $x \in$ *carrier V*
shows $\exists!f.\ f \in$ *coefficients-function X* & *linear-combination f X* $= x$
⟨*proof*⟩

The previous lemma ensures the existence of only one function f satisfying to be a linear combination and a coefficients function which generates any x belonging to *carrier V*

definition *lin-comb* :: $'c$ => ($'c$ => $'a$)
where *lin-comb x* = (*THE f*. $f \in$ *coefficients-function X*
$\wedge$ *linear-combination f X* $= x$)

lemma
lin-comb-is-coefficients-function:
assumes x: $x \in$ *carrier V*
shows *lin-comb x* $\in$ *coefficients-function X*
⟨*proof*⟩

lemma
lin-comb-is-the-linear-combination:
assumes x: $x \in$ *carrier V*
shows x = *linear-combination* (*lin-comb x*) *X*
⟨*proof*⟩

lemma
indexing-X-n-in-X:
assumes *n-dimension*: $n <$ *dimension*
shows *indexing-X n* $\in$ *X*
⟨*proof*⟩

corollary
indexing-X-n-in-carrier-V:
assumes *n-dimension*: $n <$ *dimension*
shows *indexing-X n* $\in$ *carrier V*
⟨*proof*⟩

A lemma stating that every element of the carrier set can be expressed as a finite sum over the elements of the set {..<*dimension*} thanks to the function *lin-comb*.

lemma
lin-comb-is-the-linear-combination-indexing:
assumes x: $x \in$ *carrier V*
shows x = *finsum V* (λi. *lin-comb x* (*indexing-X i*) $\cdot$ *indexing-X i*) {..<*dimension*}

⟨*proof*⟩

A lemma on how the elements of the basis are mapped by *lin-comb*:

lemma
lin-comb-basis:
assumes *x*: $x \in X$
shows *lin-comb* $x = (\lambda i.$ *if* $i = x$ *then* **1** *else* **0**)
⟨*proof*⟩
thm *linear-combination-def*
⟨*proof*⟩

end

context *vector-space*
begin

The following lemma is a minor modification of ⟦*finite ?X*; *?X* ⊆ *carrier V*; *?a* ∈ *carrier K*; *?f* ∈ *?X* → *carrier K*⟧ ⟹ *?a* · (⨁ $_V$*y*∈*?X*. *?f y* · *y*) = (⨁ $_V$*y*∈*?X*. *?a* · *?f y* · *y*), but with a bit more general statement. In particular, it removes a premise stating that $X \subseteq$ *carrier V*, which is never used in the proof of ⟦*finite ?X*; *?X* ⊆ *carrier V*; *?a* ∈ *carrier K*; *?f* ∈ *?X* → *carrier K*⟧ ⟹ *?a* · (⨁ $_V$*y*∈*?X*. *?f y* · *y*) = (⨁ $_V$*y*∈*?X*. *?a* · *?f y* · *y*) and also generalizes the inner expression of the finite sum. It may either replace ⟦*finite ?X*; *?X* ⊆ *carrier V*; *?a* ∈ *carrier K*; *?f* ∈ *?X* → *carrier K*⟧ ⟹ *?a* · (⨁ $_V$*y*∈*?X*. *?f y* · *y*) = (⨁ $_V$*y*∈*?X*. *?a* · *?f y* · *y*) in the file *Vector-Space* or added besides it in the same file.

lemma *finsum-aux2*:
⟦*finite X*; *a* ∈ *carrier K*; *f* ∈ *X* → *carrier K*; *g* ∈ *X* → *carrier V*⟧
⟹ *a* · (⨁ $_V$*y*∈*X*. *f y* · *g y*)=(⨁ $_V$*y*∈*X*. *a* · (*f y* · *g y*))
⟨*proof*⟩

end

context *finite-dimensional-vector-space*
begin

The following functions are the candidates to be proved to define the isomorphism between the vector spaces *V* and *field.K-n K dimension*. They have to be proved to be linear maps between the vector spaces, and inverse one of each other.

definition *iso-K-n-V* :: *'a vector* => *'c*
where *iso-K-n-V x* = *finsum V* (λ*i*. *fst x i* · *indexing-X i*) {..<*dimension*}

definition *iso-V-K-n* :: *'c* => *'a vector*
where *iso-V-K-n x* =
finsum (*K-n dimension*) (λ*i*. (*K-n-scalar-product* (*lin-comb* (*x*) (*indexing-X i*)) (*x-i i dimension*))) {..<*dimension*}

We prove that *iso-K-n-V* is a linear map, this means both additive and homogeneous:

lemma *linear-map-iso-K-n-V*: *K-n-V.linear-map iso-K-n-V*
⟨*proof*⟩
  find-theorems *ith ?x ?i* ∈ -
  ⟨*proof*⟩

The following lemma states that the function *lin-comb* satisfies the additivity condition. It will be later used to prove that the function *iso-V-K-n* is also an additive function.

lemma
  lin-comb-additivity:
  assumes *x*: $x \in$ *carrier V*
  and *y*: $y \in$ *carrier V*
  shows *lin-comb* $(x \oplus_V y) = (\lambda i.$ *lin-comb x i* $\oplus$ *lin-comb y i*$)$
  ⟨*proof*⟩

end

context *vector-space*
begin

lemma
  finsum-mult-assocf:
  assumes *x1*: $X \subseteq$ *carrier V*
  and *x2*: *finite X*
  and *k*: $k \in$ *carrier K*
  and *f*: $f \in X \to$ *carrier K*
  and *g*: $g \in X \to$ *carrier V*
  shows $(\bigoplus_V y \in X.\ (k \otimes f\ y) \cdot g\ y) = k \cdot (\bigoplus_V y \in X.\ f\ y \cdot g\ y)$
  ⟨*proof*⟩ **thm** *insert.hyps* (*2*)
    ⟨*proof*⟩

lemma
  finsum-mult-assoc:
  assumes *k*: $k \in$ *carrier K*
  and *f*: $f \in \{..n\} \to$ *carrier K*
  and *g*: $g \in \{..n\} \to$ *carrier V*
  shows $(\bigoplus_V y \in \{..n::nat\}.\ (k \otimes f\ y) \cdot g\ y) = k \cdot (\bigoplus_V y \in \{..n\}.\ f\ y \cdot g\ y)$
  ⟨*proof*⟩

lemma
  finsum-mult-assoc-le:
  assumes *k*: $k \in$ *carrier K*
  and *f*: $f \in \{..<n\} \to$ *carrier K*
  and *g*: $g \in \{..<n\} \to$ *carrier V*
  shows $(\bigoplus_V y \in \{..<n::nat\}.\ (k \otimes f\ y) \cdot g\ y) = k \cdot (\bigoplus_V y \in \{..<n\}.\ f\ y \cdot g\ y)$
⟨*proof*⟩

end

context *finite-dimensional-vector-space*
begin

The following lemma states that the function *lin-comb* satisfies the homogeneous property. It will be later used to prove that the function *iso-V-K-n* is homogeneous:

lemma
lin-comb-homogeneity:
assumes k: $k \in$ *carrier* K
and x: $x \in$ *carrier* V
shows *lin-comb* $(k \cdot x) = (\lambda i.\ k \otimes$ *lin-comb* $x\ i)$
⟨*proof*⟩

end

context *abelian-monoid*
begin

lemma *finsum-add'*:
assumes f: $f \in \{..<n\} \to$ *carrier* G
and g: $g \in \{..<n\} \to$ *carrier* G
shows $(\bigoplus i{\in}\{..<n{::}nat\}.\ f\ i \oplus g\ i) =$ *finsum* $G\ f\ \{..<n\} \oplus$ *finsum* $G\ g\ \{..<n\}$
⟨*proof*⟩

end

context *finite-dimensional-vector-space*
begin

The following lemma proves that the application *iso-V-K-n* is a linear map between V and *field.K-n K dimension*.

lemma *linear-map-iso-V-K-n*: *V-K-n.linear-map iso-V-K-n*
⟨*proof*⟩

end

lemma
lessThan-remove:
assumes i: $(i{::}nat) \in \{..<k\}$
shows $\{..<k\} = (\{..<k\} - \{i\}) \cup \{i\}$
⟨*proof*⟩

context *finite-dimensional-vector-space*
begin

The functions *iso-K-n-V* and *iso-V-K-n* behave correctly in their respective domains:

lemma *iso-V-K-n-Pi*: *iso-V-K-n* ∈ *carrier V* → *carrier* (*K-n dimension*)
⟨*proof*⟩

lemma *iso-K-n-V-Pi*: **shows** *iso-K-n-V* ∈ *carrier* (*K-n dimension*) → *carrier V*
⟨*proof*⟩

lemma
lin-comb-fimsum-candidate:
assumes *x*: *x* ∈ *carrier* (*K-n dimension*)
shows (⨁ *V y*∈*X*. *fst x* (*preim2-comp y*) · *y*) = (⨁ *V i*∈{..<*dimension*}. *fst x i* · *indexing-X i*)
⟨*proof*⟩

The following lemma expresses how to write down the *lin-comb* of a finite sum of the elements of the basis:

lemma
lin-comb-linear-combination-candidate:
assumes *x*: *x* ∈ *carrier* (*K-n dimension*)
shows *lin-comb* (⨁ *V i*∈{..<*dimension*}. *fst x i* · *indexing-X i*) = (λ*y*. *fst x* (*preim2-comp y*))
⟨*proof*⟩

With the previous lemmas, we can now prove that *iso-V-K-n* is a bijection between the correspoding carrier sets:

lemma *iso-V-K-n-bij*: **shows** *bij-betw iso-V-K-n* (*carrier V*) (*carrier* (*K-n dimension*))
⟨*proof*⟩

lemma *iso-K-n-V-bij*: **shows** *bij-betw iso-K-n-V* (*carrier* (*K-n dimension*)) (*carrier V*)
⟨*proof*⟩

end

context *linear-map*
begin

definition *vector-space-isomorphism* :: ('*c* => '*e*) => *bool*
where *vector-space-isomorphism f* == *bij-betw f* (*carrier V*) (*carrier W*) ∧ *linear-map f*

end

context *finite-dimensional-vector-space*
begin

Finally, the two following lemmas state the isomorphism (in both directions actually) between *field.K-n K dimension* and *V*:

lemma *V-K-n.vector-space-isomorphism iso-V-K-n*

⟨*proof*⟩

lemma *vector-space-isomorphism iso-K-n-V*
⟨*proof*⟩

end

end
theory *Subspaces*
imports *Isomorphism*
begin

12 Subspaces

context *vector-space*
begin

definition *subspace* :: *'b set => bool*
where *subspace M* == (($M \subseteq$ *carrier V*) $\wedge$ $M \neq \{\}$
$\wedge$ ($\forall \alpha \in$ *carrier K*. $\forall \beta \in$ *carrier K*. $\forall x \in M$. $\forall y \in M$.
$\alpha \cdot x \oplus_V \beta \cdot y \in M$))

lemma
zero-in-subspace:
assumes *s*: *subspace M*
shows $\mathbf{0}_V \in M$
⟨*proof*⟩

In the following statement we can observe the operation of field updating for records:

lemma
subspace-is-vector-space:
assumes *s*: *subspace M*
shows *vector-space K* (*V*(|*carrier*:= *M*|)) (*op* ·)
⟨*proof*⟩

lemma
subspace-zero:
shows *subspace* $\{\mathbf{0}_V\}$
⟨*proof*⟩

lemma *subspace-V*:
shows *subspace* (*carrier V*)
⟨*proof*⟩

As one would expect, a subspace is closed under addition:

lemma *subspace-add-closed*:
assumes *s*: *subspace S*

and x: $x \in S$ **and** y: $y \in S$
shows $x \oplus_V y \in S$
⟨*proof*⟩

The definition of *finsum* (see *finsum ?G = finprod* (|*carrier = carrier ?G*, *mult = op* $\oplus_{?G}$, *one* = $\mathbf{0}_{?G}$|)) is done in such a way hat for any infinite set it returns *undefined* and otherwise the result of a folding operator over the finite set. Under these circumstances it seems rather hard to prove properties of subspaces considering infinite sums:

lemma *subspace-finsum-closed*:
assumes s: *subspace S*
and f: *finite S*
and y: $Y \subseteq S$
and c: $f \in Y \to$ *carrier K*
shows *finsum V* $(\lambda i.\ f\ i \cdot i)$ $Y \in S$
⟨*proof*⟩

lemma *subspace-finsum-closed′*:
assumes s: *subspace S*
and f: *finite Y*
and y: $Y \subseteq S$
and c: $f \in Y \to$ *carrier K*
shows *finsum V* $(\lambda i.\ f\ i \cdot i)$ $Y \in S$
⟨*proof*⟩

corollary *subspace-linear-combination-closed*:
assumes s: *subspace S*
and f: *finite Y*
and y: $Y \subseteq S$
and c: $f \in$ *coefficients-function Y*
shows *linear-combination f Y* $\in S$
⟨*proof*⟩

end

end

theory *Calculus-of-Subspaces*
imports *Subspaces Ideal*
begin

13 Calculus of Subspaces

The theory Ideal is imported in order to use the definition of the sum of two sets, given by the operation *set-add′*

context *vector-space*
begin

lemma
subspace-inter-closed:
assumes *s*: *subspace M*
and *sm*: *subspace M′*
shows *subspace* (*M* ∩ *M′*)
⟨*proof*⟩

13.1 Theorem 1

In the following result we have to avoid empty intersections, since the empty intersection is defined to be equal to *UNIV*. *UNIV* is not a subspace, since it is not (in general, it could be in some cases) a subset of *carrier V*.

Nevertheless, this does not mean any limitation in practice, since any set will be always a subset of the subspace *carrier V* (see *subspace* (*carrier V*))

We need to prove that intersection of subspaces is a subspace to define later the subspace spanned by any set as the intersection of every subspace in which the set is contained. Thus, assuming that the intersection will be not empty (*carrier V* will be always a member of such intersection) is natural.

lemma *subspace-finite-inter-closed*:
assumes *a*: *finite A*
and *ne*: *A* ≠ {}
and *kj*: ∀ *j*∈*A*. *subspace* (*P j*)
shows *subspace* (⋂*j*∈*A*. *P j*)
⟨*proof*⟩

The same lemma than ⟦*finite ?A*; *?A* ≠ {}; ∀ *j*∈*?A*. *subspace* (*?P j*)⟧ ⟹ *subspace* (⋂ *j*∈*?A* *?P j*) but for collections indexed by the natural numbers:

lemma *subspace-finite-inter-index-closed*:
assumes *smn*: ∀ *j*∈{..(*n*::*nat*)}. *subspace* (*M j*)
shows *subspace* (⋂*j*∈{..*n*}. *M j*)
⟨*proof*⟩

We now remove the requisite of the collection of subspaces being finite. Thus, the proof cannot be longer carried out by induction in the structure of the set.

lemma *subspace-infinite-inter-closed*:
assumes *ne*: *A* ≠ {}
and *kj*: ∀ *j*∈*A*. *subspace* (*P j*)
shows *subspace* (⋂*j*∈*A*. *P j*)
⟨*proof*⟩

It is now clear than the previous results for finite intersections ∀ *j*∈{..*?n*}. *subspace* (*?M j*) ⟹ *subspace* (⋂ *j* ≤ *?n* *?M j*) and ⟦*finite ?A*; *?A* ≠ {};

$\forall j{\in}?A.\ subspace\ (?P\ j)\rrbracket \Longrightarrow subspace\ (\bigcap_{j\in ?A}\ ?P\ j)$ can be proved as a corollary of $\llbracket ?A \neq \{\};\ \forall j{\in}?A.\ subspace\ (?P\ j)\rrbracket \Longrightarrow subspace\ (\bigcap_{j\in ?A}\ ?P\ j)$, but we prefer to leave their induction proofs since they illustrate different ways of proving similar results depending on the context or the premises.

Here Halmos introduces the definition of the span of a set $S \subseteq carrier\ V$ as the interection of all the subsets in which S is contained. We already have a notion of the *span* of a set in our setting, as the set of all the elements which are equal to the linear combinations of the elements of this set. We will name this new notion *subspace-span*, and then prove that they both are equal:

We introduce an auxiliar definition of the set of subspaces in which one set is enclosed:

definition *subspace-encloser* :: $('b => bool) => ('b => bool)\ set$
  where $subspace\text{-}encloser\ A = \{M.\ subspace\ M \wedge A \subseteq M\}$

A trivial lemma stating that a set is always enclosed in the subspace *carrier* V:

lemma
  assumes m: $M \subseteq carrier\ V$
  shows $carrier\ V \in subspace\text{-}encloser\ M$
  ⟨*proof*⟩

The definition of the subspace spanned by a set, following Halmos:

definition *subspace-span* :: $('b \Rightarrow bool) \Rightarrow 'b \Rightarrow bool$
  where $subspace\text{-}span\ A = (\bigcap B{\in}(subspace\text{-}encloser\ A).\ B)$

The previous lemma $\llbracket finite\ ?A;\ ?A \neq \{\};\ \forall j{\in}?A.\ subspace\ (?P\ j)\rrbracket \Longrightarrow subspace\ (\bigcap_{j\in ?A}\ ?P\ j)$ is now used to prove that *subspace-span* is a subspace itself.

lemma
  subspace-span-monotone:
  assumes s: $S \subseteq carrier\ V$
  shows $S \subseteq subspace\text{-}span\ S$
  ⟨*proof*⟩

lemma
  subspace-subspace-span:
  assumes s: $S \subseteq carrier\ V$
  shows $subspace\ (subspace\text{-}span\ S)$
  ⟨*proof*⟩

13.2 Theorem 2.

The definition of *finsum* in Isabelle relies on the notion of finiteness of the set which elements are added up. Working in a finite dimensional vector

space does not mean that every subset is finite, and thus the elements in the span of such a set cannot be written as finite sums of its elements.

The previous point is not explicit is Halmos, where it is never explained how to deal with infinite sums (or sums over not finite sets).

lemma
subspace-span-empty:
subspace-span $\{\} = \{\mathbf{0}_V\}$
⟨*proof*⟩

lemma *theorem-2*:
assumes *f*: *finite S*
and *s*: $S \subseteq$ *carrier V*
shows *span S* = *subspace-span S*
⟨*proof*⟩

13.3 Theorem 3.

The following theorem appears in Halmos as an easy consequence of the previous one; probably it should be proved based on the fact that any linear combination can be written down as the sum of two elements, being one in the first set and the other in the second one.

term $I <+>_R J$
find-theorems - $<+>_{?F}$ -
lemma *theorem-3*:
assumes *I*: *subspace I*
and *J*: *subspace J*
shows *subspace-span* $(I \cup J) = I <+>_V J$
⟨*proof*⟩

The following definition is simply a rewriting rule, it may be skipped; note also that produces ambiguous parse trees when parsing deducing types from expressions, so it could be avoided if it produces any clashes:

definition *set-add2* :: *'b set => 'b set => 'b set* (**infixl** + *60*)
where *set-add2 A B* = *subspace-span* $(A \cup B)$

corollary *set-add2-set-add'*:
assumes *I*: *subspace I*
and *J*: *subspace J*
shows $I + J = I <+>_V J$
⟨*proof*⟩

The following definition is applied only to subspaces:

definition *complement* :: *'b set => 'b set => bool*
where *complement I J* = $((I \cap J = \{\mathbf{0}_V\}) \wedge (I + J =$ *carrier V*$))$

end

end
theory *Dimension-of-a-Subspace*
imports *Calculus-of-Subspaces*
begin

14 Dimension of a Subspace

context *finite-dimensional-vector-space*
begin

14.1 Theorem 1.

The theorem states that the subspace is itself a vector space and that its dimesion is less than or equal to the one of V. We split both conclusions in two different lemmas that later will be merged.

The first part of the theorem has been already proved:

lemma *theorem-1-part-1*:
assumes *m*: *subspace M*
shows *vector-space K* (*V*(|*carrier*:=*M*|)) (*op* ·)
⟨*proof*⟩

The second part of the theorem requires a definition of dimension. The dimension of a (finite) vector space should be defined as the cardinal of any of its basis, once we have proved that every basis has the same cardinal (file *Finite-Vector-Space.thy*). In the meanwhile, I use *dim*

Its proof should be direct by reduction ad absurdum, following the one in Halmos.

lemma *theorem-1-part-2*:
assumes *m*: *subspace M*
shows *dim* (*V*(|*carrier*:=*M*|)) ≤ *dimension*
⟨*proof*⟩

14.2 Theorem 2.

The notation in the following statement might be a bit confusing. The indexing f is just necessary to later select the first m elements of a base, with m being the dimension of the subspace M. These m elements can be completed up to a basis of V.

The proof should be done using that M is a vector space of dimension less or equal to the one of V. Therefore we can find a basis of it which cardinal is less than or equal to *dimension*. This basis is a collection of linearly independent vectors, and therefore can be completed up to a basis of V, thanks to one of the lemmas proved in *Finite-Vector-Space.thy*.

lemma *theorem-2*:
assumes *m*: *subspace M*
shows (∃ *B f*. (*basis B*) ∧ *indexing* (*B*, *f*) ∧
(*vector-space.basis K* (*V*(|*carrier*:=*M*|)) (*op* ·) (*f* ‘ {..*dim* (*V*(|*carrier*:=*M*|)}))))
⟨*proof*⟩

end

end
theory *Dual-Spaces*
imports *Dimension-of-a-Subspace*
begin

15 Dual Spaces

context *vector-space*
begin

This definition can be found also on Bauer's development, taking as the scalar field the set of real numbers, and with the name of linear form. We follow linear functional as Halmos' text

We split the definition of linear form into its multiplicative and additive components:

definition *additive-functional* :: ('*b* => '*a*) => *bool*
where *additive-functional f*
≡ (∀ *x*∈*carrier V*. ∀ *y*∈*carrier V*. *f* (*x* ⊕$_V$ *y*) = *f x* ⊕$_K$ *f y*)

definition *multiplicative-functional* :: ('*b* => '*a*) => *bool*
where *multiplicative-functional f*
≡ (∀ *k*∈*carrier K*. ∀ *x*∈*carrier V*. *f* (*k* · *x*) = *k* ⊗$_K$ (*f x*))

definition *linear-functional* :: ('*b* => '*a*) => *bool*
where *linear-functional f* ≡ *additive-functional f*
∧ *multiplicative-functional f*

The following lemma appears in Halmos (as the homogeneous property) and also in Bauer's files; in Bauer there are also some properties about the difference anf lineal functionals.

lemma *linear-functional-zero*:
assumes *linear-functional f*
shows *f* **0**$_V$ = **0**
⟨*proof*⟩

We introduce the definition of the dual space of the vector space *V*. We have to provide a carrier set, a zero operation and an addition. As the definition of abelian groups in Isabelle is done omver the ring type, we also have to provide some definition of unit and multiplication, that will be useless.

The dual space is also denoted in Halmos V'

definition *dual-space* :: $('b \Rightarrow 'a)$ *ring* (V')
  where *dual-space* = ⦇ *carrier* = *linear-functional*,
  mult = *undefined*,
  one = *undefined*,
  zero = $(\lambda x.\ \mathbf{0})$,
  add = $(\lambda y1.\lambda y2.\lambda x.\ y1\ x \oplus y2\ x)$⦈

We create a synonim for the previous definition to ease readability:

lemmas V'*-def* = *dual-space-def*

term *vector-space* $K\ V'$

term $(\lambda x\ f\ y.\ x \otimes f\ y)$

I guess it is not necessary to go down to finite dimensional vector spaces to prove the following lemma. If it is necessary, the context should be changed accordingly:

lemma *vector-space-*V': *vector-space* $K\ V'\ (\lambda x\ f\ y.\ x \otimes f\ y)$

  ⟨*proof*⟩

end

end
theory *Brackets*
  imports *Dual-Spaces*
begin

16 Brackets

context *vector-space*
begin

The following notation is not working properly: 1. I do not know how to invert the order of the parameters, in such a a way that <x,f> denotes $f\ x$; 2. Even in the right order, where <f,x> denotes $f\ x$, the notation <f,x> produces problems when trying to use it.

A couple of notes on the following notation; it is done trying to mimic the similar ideas in Halmos. First of all, we have chosen the symbols < - > instead of [-] since brackets would produce ambiguous inputs with lists, forcing us to write explicitly in a lot of scenarios the type of each of the components of the pair.

Second, the *input* annotation of *abbreviation* makes the special syntax proposed to work only in the input mode, *i.e.*, when we write something. Wihtout this annotation, the output would be also changed, but that would

affect to every function application in our setting, which is not our intention and apparently makes the pretty printer loop. For more details see https://lists.cam.ac.uk/pipermail/cl-isabelle-users/2011-August/msg00007.html

abbreviation (*input*)
  app :: *'b* => (*'b* => *'a*) => *'a* (<(-),(-)> *90*)
  where <*x*, *f*> == *f x*

term <*x*, *f*> ⊕ <*y*, *f*>

end

end
theory *Dual-Bases*
  imports *Brackets*
begin

17 Dual Bases

context *finite-dimensional-vector-space*
begin

17.1 Theorem 1.

We recall here that X is a basis for the vector space V and *indexing-X* is a way to provide the basis with coordinates.

The definition of *indexing* is polymorphic, and in this lemma will be used both for the basis X and also for the set of scalars.

In this lemma will be useful the results in file *Vector-Space-K-n.thy*, for instance *?x* ∈ *carrier V* ⟹ ∃!*f*. *f* ∈ *coefficients-function X* ∧ *linear-combination f X* = *?x* and *?x* ∈ *carrier V* ⟹ *?x* = (⨁ *V*)*i*∈{..<*dimension*}. *lin-comb ?x* (*indexing-X i*) · *indexing-X i*), where it is proved that any element in *carrier V* can be expressed in a unique way as a linear combination of the elements in X.

thm *lin-comb-is-the-linear-combination-indexing*
find-theorems (∃!*f*. -)

lemma *theorem-1*:
  assumes *ia*: *indexing* ((*A*::*'a set*), *fA*)
  and *c*: *card A* = *dimension*
  shows (∃!*y*. *linear-functional y* ∧ (∀ *i*∈{..<*dimension*}. <*indexing-X i*, *y*> = *fA i*))
⟨*proof*⟩

17.2 Theorem 2.

term *linear-functional*

definition *delta* :: *nat* => *nat* => *'a*
  where *delta i j* = (*if i* = *j then* **1** *else* **0**)

definition *linear-functional-basis* :: *nat* => (*'c* => *'a*)
  where *linear-functional-basis n* = (λ*x*. *delta* (*preim2 x*) *n*)

definition *linear-functional-basis-set* :: (*'c* => *'a*) *set*
  where *linear-functional-basis-set* = {(λ*x*. *delta* (*preim2 x*) *n*) | *n*. *n* ∈ {..<*dimension*}}

lemma *theorem-2*:
  shows *vector-space.basis K V'* (λ*x f y*. *x* ⊗ *f y*) *linear-functional-basis-set*
⟨*proof*⟩

17.3 Theorem 3.

lemma *theorem-3*:
  assumes *x-ne-0*: *x* ≠ **0**$_V$
  shows ∃ *y*. *linear-functional y* ∧ <*x*,*y*> ≠ **0**$_K$
  ⟨*proof*⟩

corollary *theorem-3-c*:
  assumes *x-ne-0*: *u* ≠ *v*
  shows ∃ *y*. *linear-functional y* ∧ <*u*,*y*> ≠ <*v*,*y*>
  ⟨*proof*⟩

end

end