VectorSpace

Holden Lee[*]

March 17, 2025

Abstract

I present a formalisation of basic linear algebra based completely on locales, building off HOL-Algebra. It includes the following:

1. basic definitions: linear combinations, span, linear independence
2. linear transformations
3. interpretation of function spaces as vector spaces
4. direct sum of vector spaces, sum of subspaces
5. the replacement theorem
6. existence of bases in finite-dimensional vector spaces, definition of dimension
7. rank-nullity theorem.

Note that some concepts are actually defined and proved for modules as they also apply there.

In the process, I also prove some basic facts about rings, modules, and fields, as well as finite sums in monoids/modules.

Note that infinite-dimensional vector spaces are supported, but dimension is only supported for finite-dimensional vector spaces.

The proofs are standard; the proofs of the replacement theorem and rank-nullity theorem roughly follow the presentation in [**?**]. The rank-nullity theorem generalises the existing development in [**?**] (originally using type classes, now using a mix of type classes and locales).

Contents

[*]This work was funded by the Post-Masters Consultancy and the Computer Laboratory at the University of Cambridge.

1 Basic facts about rings and modules

theory *RingModuleFacts*
imports *Main*
  HOL−Algebra.Module
  HOL−Algebra.Coset

begin

1.1 Basic facts

In a field, every nonzero element has an inverse.

lemma (**in** *field*) *inverse-exists* [*simp*, *intro*]:
  assumes *h1*: $a \in carrier\ R$ **and** *h2*: $a \neq \mathbf{0}_R$
  shows $inv_R\ a \in carrier\ R$
proof −
  have *1*: $Units\ R = carrier\ R - \{\mathbf{0}_R\}$ **by** (*rule field-Units*)
  from *h1 h2 1* **show** *?thesis* **by** *auto*
qed

Multiplication by 0 in R gives 0. (Note that this fact encompasses smult-l-null as this is for module while that is for algebra, so smult-l-null is redundant.)

lemma (**in** *module*) *lmult-0* [*simp*]:
  assumes *1*: $m \in carrier\ M$
  shows $\mathbf{0}_R \odot_M m = \mathbf{0}_M$
proof −
  from *1* **have** *0*: $\mathbf{0}_R \odot_M m \in carrier\ M$ **by** *simp*
  from *1* **have** *2*: $\mathbf{0}_R \odot_M m = (\mathbf{0}_R \oplus_R \mathbf{0}_R) \odot_M m$ **by** *simp*
  from *1* **have** *3*: $(\mathbf{0}_R \oplus_R \mathbf{0}_R) \odot_M m = (\mathbf{0}_R \odot_M m) \oplus_M (\mathbf{0}_R \odot_M m)$
using [[*simp-trace*, *simp-trace-depth-limit=3*]]
    by (*simp add*: *smult-l-distr del*: *R.add.r-one R.add.l-one*)

from *2 3* **have** *4*: $\mathbf{0}_R \odot_M m = (\mathbf{0}_R \odot_M m) \oplus_M (\mathbf{0}_R \odot_M m)$ **by** *auto*
from *0 4* **show** *?thesis*
using *M.l-neg M.r-neg1* **by** *fastforce*
qed

Multiplication by 0 in M gives 0.

lemma (**in** *module*) *rmult-0* [*simp*]:
assumes *0*: $r \in carrier\ R$
shows $r \odot_M \mathbf{0}_M = \mathbf{0}_M$
by (*metis M.zero-closed R.zero-closed assms lmult-0 r-null smult-assoc1*)

Multiplication by -1 is the same as negation. May be useful as a simp rule.

lemma (**in** *module*) *smult-minus-1*:
fixes *v*
assumes *0*: $v \in carrier\ M$
shows $(\ominus_R \mathbf{1}_R) \odot_M v = (\ominus_M v)$

proof −
from *0* **have** *a0*: $\mathbf{1}_R \odot_M v = v$ **by** *simp*
from *0* **have** *1*: $((\ominus_R \mathbf{1}_R) \oplus_R \mathbf{1}_R) \odot_M v = \mathbf{0}_M$
by (*simp add*:*R.l-neg*)
from *0* **have** *2*: $((\ominus_R \mathbf{1}_R) \oplus_R \mathbf{1}_R) \odot_M v = (\ominus_R \mathbf{1}_R) \odot_M v \oplus_M \mathbf{1}_R \odot_M v$
by (*simp add*: *smult-l-distr*)
from *1 2* **show** *?thesis* **by** (*metis M.minus-equality R.add.inv-closed*

a0 assms one-closed smult-closed)
qed

The version with equality reversed.

lemmas (**in** *module*) *smult-minus-1-back* = *smult-minus-1*[*THEN sym*]

-1 is not 0

lemma (**in** *field*) *neg-1-not-0* [*simp*]: $\ominus_R \mathbf{1}_R \neq \mathbf{0}_R$
by (*metis minus-minus minus-zero one-closed zero-not-one*)

Note smult-assoc1 is the wrong way around for simplification. This is the reverse of smult-assoc1.

lemma (**in** *module*) *smult-assoc-simp*:
$[\mid a \in carrier\ R;\ b \in carrier\ R;\ x \in carrier\ M \mid] ==>$
$a \odot_M (b \odot_M x) = (a \otimes b) \odot_M x$
by (*auto simp add*: *smult-assoc1*)

lemmas (**in** *abelian-group*) *show-r-zero*= *add.l-cancel-one*
lemmas (**in** *abelian-group*) *show-l-zero*= *add.r-cancel-one*

A nontrivial ring has $0 \neq 1$.

lemma (**in** *ring*) *nontrivial-ring* [*simp*]:
  assumes *carrier* $R \neq \{\mathbf{0}_R\}$
  shows $\mathbf{0}_R \neq \mathbf{1}_R$
proof (*rule ccontr*)
  assume *1*: $\neg(\mathbf{0}_R \neq \mathbf{1}_R)$
  {
    fix *r*
    assume *2*: $r \in$ *carrier R*
    from *1 2* **have** *3*: $\mathbf{1}_R \otimes_R r = \mathbf{0}_R \otimes_R r$ **by** *auto*
    from *2 3* **have** $r = \mathbf{0}_R$ **by** *auto*
  }
  from *this assms* **show** *False* **by** *auto*
qed

Use as simp rule. To show $a - b = 0$, it suffices to show $a = b$.

lemma (**in** *abelian-group*) *minus-other-side* [*simp*]:
  $[\![a \in carrier\ G;\ b \in carrier\ G]\!] \Longrightarrow (a \ominus_G b = \mathbf{0}_G) = (a = b)$
  by (*metis a-minus-def add.inv-closed add.m-comm r-neg r-neg2*)

1.2 Units group

Define the units group $R^{\times}$ and show it is actually a group.

definition *units-group*::('*a*,'*b*) *ring-scheme* ⇒ '*a monoid*
  where *units-group* $R = (\!|carrier = Units\ R,\ mult = (\lambda x\ y.\ x \otimes_R y),$
$one = \mathbf{1}_R|\!)$

The units form a group.

lemma (**in** *ring*) *units-form-group*: *group* (*units-group R*)
  apply (*intro groupI*)
  apply (*unfold units-group-def*, *auto*)
  apply (*intro m-assoc*)
  apply *auto*
  apply (*unfold Units-def*)
  apply *auto*
  done

The units of a *cring* form a commutative group.

lemma (**in** *cring*) *units-form-cgroup*: *comm-group* (*units-group R*)
  apply (*intro comm-groupI*)
  apply (*unfold units-group-def*) **apply** *auto*
  apply (*intro m-assoc*) **apply** *auto*
  apply (*unfold Units-def*) **apply** *auto*
  apply (*rule m-comm*) **apply** *auto*
  done

end

2 Basic lemmas about functions

theory *FunctionLemmas*

imports *Main*
HOL−Library.FuncSet
begin

These are used in simplification. Note that the difference from Pi-mem is that the statement about the function comes first, so Isabelle can more easily figure out what S is.

lemma *PiE-mem2*: $f \in S \rightarrow_E T \Longrightarrow x \in S \Longrightarrow f\, x \in T$
unfolding *PiE-def* **by** *auto*
lemma *Pi-mem2*: $f \in S \rightarrow T \Longrightarrow x \in S \Longrightarrow f\, x \in T$
unfolding *Pi-def* **by** *auto*

end

3 Sums in monoids

theory *MonoidSums*

imports *Main*
HOL−Algebra.Module
RingModuleFacts
FunctionLemmas
begin

We build on the finite product simplifications in FiniteProduct.thy and the analogous ones for finite sums (see "lemmas" in Ring.thy).

Use as an intro rule

lemma (**in** *comm-monoid*) *factors-equal*:
$[\![a=b;\ c=d]\!] \Longrightarrow a \otimes_G c = b \otimes_G d$
by *simp*

lemma (**in** *comm-monoid*) *extend-prod*:
fixes $a\ A\ S$
assumes *fin*: *finite* S **and** *subset*: $A \subseteq S$ **and** a: $a \in A \rightarrow carrier\ G$
shows $(\bigotimes_G x \in S.\ (if\ x \in A\ then\ a\ x\ else\ \mathbf{1}_G)) = (\bigotimes_G x \in A.\ a\ x)$
(**is** $(\bigotimes_G x \in S.\ ?b\ x) = (\bigotimes_G x \in A.\ a\ x)$)
proof −
from *subset* **have** *uni*:$S = A \cup (S - A)$ **by** *auto*
from *assms subset* **show** *?thesis*
apply (*subst uni*)
apply (*subst finprod-Un-disjoint*, *auto*)

by (*auto cong*: *finprod-cong if-cong elim*: *finite-subset simp add*:*Pi-def finite-subset*)

qed

Scalar multiplication distributes over scalar multiplication (on left).

lemma (**in** *module*) *finsum-smult*:
[| $c \in$ *carrier R*; $g \in A \to$ *carrier M* |] ==>
($c \odot_M$ *finsum M g A*) = *finsum M* (%x. $c \odot_M g\ x$) A
proof (*induct A rule*: *infinite-finite-induct*)
case (*insert a A*)
from *insert.hyps insert.prems* **have** *1*: *finsum M g* (*insert a A*) = $g\ a \oplus_M$ *finsum M g A*
by (*intro finsum-insert*, *auto*)
from *insert.hyps insert.prems* **have** *2*: ($\bigoplus_M x \in$*insert a A*. $c \odot_M g\ x$) = $c \odot_M g\ a \oplus_M$($\bigoplus_M x \in A$. $c \odot_M g\ x$)
by (*intro finsum-insert*, *auto*)
from *insert.hyps insert.prems* **show** *?case*
by (*auto simp add*:*1 2 smult-r-distr*)
qed *auto*

Scalar multiplication distributes over scalar multiplication (on right).

lemma (**in** *module*) *finsum-smult-r*:
[| $v \in$ *carrier M*; $f \in A \to$ *carrier R* |] ==>
(*finsum R f A* $\odot_M v$) = *finsum M* (%x. $f\ x \odot_M v$) A
proof (*induct A rule*: *infinite-finite-induct*)
case (*insert a A*)
from *insert.hyps insert.prems* **have** *1*: *finsum R f* (*insert a A*) = $f\ a \oplus_R$ *finsum R f A*
by (*intro R.finsum-insert*, *auto*)
from *insert.hyps insert.prems* **have** *2*: ($\bigoplus_M x \in$*insert a A*. $f\ x \odot_M v$) = $f\ a \odot_M v \oplus_M$($\bigoplus_M x \in A$. $f\ x \odot_M v$)
by (*intro finsum-insert*, *auto*)
from *insert.hyps insert.prems* **show** *?case*
by (*auto simp add*:*1 2 smult-l-distr*)
qed *auto*

A sequence of lemmas that shows that the product does not depend on the ambient group. Note I had to dig back into the definitions of foldSet to show this.

lemma *foldSet-not-depend*:
fixes *A E*
assumes *h1*: $D \subseteq E$
shows *foldSetD D f e* $\subseteq$*foldSetD E f e*
proof −
from *h1* **have** *1*: $\bigwedge x1\ x2$. ($x1$,$x2$) $\in$ *foldSetD D f e* $\Longrightarrow$ ($x1$, $x2$) $\in$ *foldSetD E f e*

proof −
fix *x1 x2*
assume *2*: $(x1,x2) \in$ *foldSetD D f e*
from *h1 2* **show** *?thesis x1 x2*
apply (*intro foldSetD.induct*[**where** *?D=D* **and** *?f=f* **and** *?e=e* **and** *?x1.0=x1* **and** *?x2.0=x2*
and $?P = \lambda x1\ x2.\ ((x1,\ x2) \in$ *foldSetD E f e*)])
apply *auto*
apply (*intro emptyI*, *auto*)
by (*intro insertI*, *auto*)
qed
from *1* **show** *?thesis* **by** *auto*
qed

lemma *foldD-not-depend*:
fixes *D E B f e A*
assumes *h1*: *LCD B D f* **and** *h2*: *LCD B E f* **and** *h3*: $D \subseteq E$ **and** *h4*: $e \in D$ **and** *h5*: $A \subseteq B$ **and** *h6*: *finite B*
shows *foldD D f e A = foldD E f e A*
proof −
from *assms* **have** *1*: $\exists\, y.\ (A,y) \in$ *foldSetD D f e*
apply (*intro finite-imp-foldSetD*, *auto*)
apply (*metis finite-subset*)
by (*unfold LCD-def*, *auto*)
from *1* **obtain** *y* **where** *2*: $(A,y) \in$ *foldSetD D f e* **by** *auto*
from *assms 2* **have** *3*: *foldD D f e A = y* **by** (*intro LCD.foldD-equality*[*of B*], *auto*)
from *h3* **have** *4*: *foldSetD D f e* $\subseteq$ *foldSetD E f e* **by** (*rule foldSet-not-depend*)
from *2 4* **have** *5*: $(A,y) \in$ *foldSetD E f e* **by** *auto*
from *assms 5* **have** *6*: *foldD E f e A = y* **by** (*intro LCD.foldD-equality*[*of B*], *auto*)

from *3 6* **show** *?thesis* **by** *auto*
qed

lemma (**in** *comm-monoid*) *finprod-all1* [*simp*]:
assumes *all1*: $\bigwedge a.\ a \in A \Longrightarrow f\ a = \mathbf{1}_G$
shows $(\bigotimes_G\ a \in A.\ f\ a) = \mathbf{1}_G$

proof −
from *assms* **show** *?thesis*
by (*simp cong*: *finprod-cong*)
qed

context *abelian-monoid*
begin
lemmas *summands-equal = add.factors-equal*
lemmas *extend-sum = add.extend-prod*

lemmas *finsum-all0* = *add.finprod-all1*
end

end

4 Linear Combinations

theory *LinearCombinations*
imports *Main*
  HOL−Algebra.Module
  HOL−Algebra.Coset
  RingModuleFacts
  MonoidSums
  FunctionLemmas
begin

4.1 Lemmas for simplification

The following are helpful in certain simplifications (esp. congruence rules). Warning: arbitrary use leads to looping.

lemma (**in** *ring*) *coeff-in-ring*:
  $[\![a{\in}A{\to}carrier\ R;\ x{\in}A]\!] \Longrightarrow a\ x\ {\in}carrier\ R$
by (*rule Pi-mem*)

lemma (**in** *ring*) *coeff-in-ring2*:
  $[\![\ x{\in}A;a{\in}A{\to}carrier\ R]\!] \Longrightarrow a\ x\ {\in}carrier\ R$
by (*metis Pi-mem*)

lemma *ring-subset-carrier*:
  $[\![x\ {\in}A;\ A{\subseteq}carrier\ R]\!] \Longrightarrow x\ {\in}carrier\ R$
by *auto*

lemma *disj-if*:
  $[\![A{\cap}B{=}\{\};\ x{\in}\ B]\!] \Longrightarrow (if\ x{\in}A\ then\ f\ x\ else\ g\ x) = g\ x$
by *auto*

lemmas (**in** *module*) *sum-simp* = *ring-subset-carrier*

4.2 Linear combinations

A linear combination is $\sum_{v \in A} a_v v$. $(a_v)_{v \in S}$ is a function $A \to K$, where $A \subseteq K$.

definition (**in** *module*) *lincomb*::$['c \Rightarrow 'a,\ 'c\ set] \Rightarrow 'c$
where *lincomb* $a\ A = (\bigoplus_M\ v{\in}A.\ (a\ v\ \odot_M\ v))$

lemma (**in** *module*) *summands-valid*:

fixes *A a*
assumes *h2*: *A*⊆ *carrier M* **and** *h3*: *a*∈(*A*→*carrier R*)
shows ∀ *v*∈ *A*. (((*a v*) $\odot_M$ *v*)∈ *carrier M*)
proof −
from *assms* **show** *?thesis* **by** *auto*
qed

lemma (**in** *module*) *lincomb-closed* [*simp*, *intro*]:
fixes *S a*
assumes *h2*: *S*⊆ *carrier M* **and** *h3*: *a*∈(*S*→*carrier R*)
shows *lincomb a S* ∈ *carrier M*
proof −
from *h2 h3* **show** *?thesis* **by** (*unfold lincomb-def*, *auto intro*:*finsum-closed*)

qed

lemma (**in** *comm-monoid*) *finprod-cong2*:
[| *A* = *B*;
!!*i*. *i* ∈ *B* ==> *f i* = *g i*; *f* ∈ *B* → *carrier G*|] ==>
finprod G f A = *finprod G g B*
by (*intro finprod-cong*, *auto*)

lemmas (**in** *abelian-monoid*) *finsum-cong2* = *add.finprod-cong2*

lemma (**in** *module*) *lincomb-cong*:
assumes *h2*: *A*=*B* **and** *h3*: *A* ⊆ *carrier M*
and *h4*: ⋀*v*. *v*∈*A* ⟹ *a v* = *b v* **and** *h5*: *b*∈ *B*→*carrier R*
shows *lincomb a A* = *lincomb b B*
using *assms*
by (*simp cong*: *finsum-cong2 add*: *lincomb-def summands-valid ring-subset-carrier*)

lemma (**in** *module*) *lincomb-union*:
fixes *a A B*
assumes *h1*: *finite* (*A*∪ *B*) **and** *h3*: *A*∪*B* ⊆ *carrier M*
and *h4*: *A*∩*B*={} **and** *h5*: *a*∈(*A*∪*B*→*carrier R*)
shows *lincomb a* (*A*∪*B*) = *lincomb a A* $\oplus_M$ *lincomb a B*
using *assms*
by (*auto cong*: *finsum-cong2 simp add*: *lincomb-def finsum-Un-disjoint summands-valid ring-subset-carrier*)

This is useful as a simp rule sometimes, for combining linear combinations.

lemma (**in** *module*) *lincomb-union2*:
fixes *a b A B*
assumes *h1*: *finite* (*A*∪ *B*) **and** *h3*: *A*∪*B* ⊆ *carrier M*
and *h4*: *A*∩*B*={} **and** *h5*: *a*∈*A*→*carrier R* **and** *h6*: *b*∈*B*→*carrier R*
shows *lincomb a A* $\oplus_M$ *lincomb b B* = *lincomb* (λ*v*. *if* (*v*∈*A*) *then*

a v else b v) ($A \cup B$)
  (**is** *lincomb a A* $\oplus_M$ *lincomb b B* = *lincomb ?c* ($A \cup B$))
using *assms*
 by (*auto cong*: *finsum-cong2*
   simp add: *lincomb-def finsum-Un-disjoint summands-valid ring-subset-carrier disj-if*)

lemma (**in** *module*) *lincomb-del2*:
 fixes *S a v*
 assumes *h1*: *finite S* **and** *h2*: $S \subseteq$ *carrier M* **and** *h3*: $a \in (S \rightarrow$ *carrier R*) **and** *h4*: $v \in S$
 shows *lincomb a S* = ((*a v*) $\odot_M$ *v*) $\oplus_M$ *lincomb a* ($S - \{v\}$)
proof −
 from *h4* **have** *1*: $S = \{v\} \cup (S - \{v\})$ **by** (*metis insert-Diff insert-is-Un*)

 from *assms* **show** *?thesis*
  apply (*subst 1*)
  apply (*subst lincomb-union*, *auto*)
  by (*unfold lincomb-def*, *auto simp add*: *coeff-in-ring*)
qed

lemma (**in** *module*) *lincomb-insert*:
 fixes *S a v*
 assumes *h1*: *finite S* **and** *h2*: $S \subseteq$ *carrier M* **and** *h3*: $a \in (S \cup \{v\} \rightarrow$ *carrier R*) **and** *h4*: $v \notin S$ **and** *h5*: $v \in$ *carrier M*
 shows *lincomb a* ($S \cup \{v\}$) = ((*a v*) $\odot_M$ *v*) $\oplus_M$ *lincomb a S*
using *assms*
 by (*auto cong*: *finsum-cong2*
   simp add: *lincomb-def finsum-Un-disjoint summands-valid ring-subset-carrier disj-if*)

lemma (**in** *module*) *lincomb-elim-if* [*simp*]:
 fixes *b c S*
 assumes *h1*: $S \subseteq$ *carrier M* **and** *h2*: $\bigwedge v.\ v \in S \Longrightarrow \neg P\ v$ **and** *h3*: $c \in S \rightarrow$ *carrier R*
 shows *lincomb* (λw. *if P w then b w else c w*) *S* = *lincomb c S*
using *assms*
 by (*auto cong*: *finsum-cong2*
   simp add: *lincomb-def finsum-Un-disjoint summands-valid ring-subset-carrier disj-if*)

lemma (**in** *module*) *lincomb-smult*:
 fixes *A c*
 assumes *h2*: $A \subseteq$ *carrier M* **and** *h3*: $a \in A \rightarrow$ *carrier R* **and** *h4*: $c \in$ *carrier R*
 shows *lincomb* (λw. $c \otimes_R$ *a w*) *A* = $c \odot_M$ (*lincomb a A*)
using *assms*

by (*auto cong*: *finsum-cong2*
simp add: *lincomb-def finsum-Un-disjoint finsum-smult ring-subset-carrier disj-if smult-assoc1 coeff-in-ring*)

4.3 Linear dependence and independence.

A set S in a module/vectorspace is linearly dependent if there is a finite set $A \subseteq S$ and coefficients $(a_v)_{v\in A}$ such that $sum_{v\in A} a_v v = 0$ and for some v, $a_v \neq 0$.

definition (**in** *module*) *lin-dep* **where**
$lin\text{-}dep\ S = (\exists A\ a\ v.\ (finite\ A \wedge A\subseteq S \wedge (a\in (A\to carrier\ R)) \wedge (lincomb\ a\ A = \mathbf{0}_M) \wedge (v\in A) \wedge (a\ v\neq \mathbf{0}_R)))$

abbreviation (**in** *module*) $lin\text{-}indpt::'c\ set \Rightarrow bool$
where $lin\text{-}indpt\ S \equiv \neg lin\text{-}dep\ S$

In the finite case, we can take $A = S$. This may be more convenient (e.g., when adding two linear combinations.

lemma (**in** *module*) *finite-lin-dep*:
fixes S
assumes *finS*:$finite\ S$ **and** *ld*: $lin\text{-}dep\ S$ **and** *inC*: $S\subseteq carrier\ M$
shows $\exists a\ v.\ (a\in (S\to carrier\ R)) \wedge (lincomb\ a\ S = \mathbf{0}_M) \wedge (v\in S) \wedge (a\ v\neq \mathbf{0}_R)$
proof −
from *ld* **obtain** $A\ a\ v$ **where** A: $(A\subseteq S \wedge (a\in (A\to carrier\ R)) \wedge (lincomb\ a\ A = \mathbf{0}_M) \wedge (v\in A) \wedge (a\ v\neq \mathbf{0}_R))$
by (*unfold lin-dep-def*, *auto*)
let $?b=\lambda w.\ if\ w\in A\ then\ a\ w\ else\ \mathbf{0}_R$
from *finS inC A* **have** *if-in*: $(\bigoplus_M v\in S.\ (if\ v \in A\ then\ a\ v\ else\ \mathbf{0}) \odot_M v) = (\bigoplus_M v\in S.\ (if\ v \in A\ then\ a\ v \odot_M v\ else\ \mathbf{0}_M))$
apply *auto*
apply (*intro finsum-cong'*)
by (*auto simp add*: *coeff-in-ring*)
from *finS inC A* **have** *b*: $lincomb\ ?b\ S = \mathbf{0}_M$
apply (*unfold lincomb-def*)
apply (*subst if-in*)
by (*subst extend-sum*, *auto*)
from *A b* **show** *?thesis*
apply (*rule-tac* $x=?b$ **in** *exI*)
apply (*rule-tac* $x=v$ **in** *exI*)
by *auto*
qed

Criteria of linear dependency in a easy format to apply: apply (rule lin-dep-crit)

lemma (**in** *module*) *lin-dep-crit*:
fixes $A\ S\ a\ v$

assumes *fin*: *finite A* **and** *subset*: $A \subseteq S$ **and** *h1*: $(a \in (A \to carrier\ R))$ **and** *h2*: $v \in A$
and *h3*: $a\ v \neq \mathbf{0}_R$ **and** *h4*: $(lincomb\ a\ A = \mathbf{0}_M)$
shows *lin-dep S*
proof –
from *assms* **show** *?thesis*
by (*unfold lin-dep-def*, *auto*)
qed

If $\sum_{v \in A} a_v v = 0$ implies $a_v = 0$ for all $v \in S$, then A is linearly independent.

lemma (**in** *module*) *finite-lin-indpt2*:
fixes *A*
assumes *A-fin*: *finite A* **and** *AinC*: $A \subseteq carrier\ M$ **and**
lc0: $\bigwedge a.\ a \in (A \to carrier\ R) \Longrightarrow (lincomb\ a\ A = \mathbf{0}_M) \Longrightarrow (\forall\ v \in A.\ a\ v = \mathbf{0}_R)$
shows *lin-indpt A*
proof (*rule ccontr*)
assume $\neg$*lin-indpt A*
from *A-fin AinC this* **obtain** *a v* **where** *av*:
$(a \in (A \to carrier\ R)) \wedge (lincomb\ a\ A = \mathbf{0}_M) \wedge (v \in A) \wedge (a\ v \neq \mathbf{0}_R)$
by (*metis finite-lin-dep*)
from *av lc0* **show** *False* **by** *auto*
qed

Any set containing 0 is linearly dependent.

lemma (**in** *module*) *zero-lin-dep*:
assumes *0*: $\mathbf{0}_M \in S$ **and** *nonzero*: $carrier\ R \neq \{\mathbf{0}_R\}$
shows *lin-dep S*
proof –
from *nonzero* **have** *zero-not-one*: $\mathbf{0}_R \neq \mathbf{1}_R$ **by** (*rule nontrivial-ring*)
from *0 zero-not-one* **show** *?thesis*
apply (*unfold lin-dep-def*)
apply (*rule-tac* $x=\{\mathbf{0}_M\}$ **in** *exI*)
apply (*rule-tac* $x=(\lambda v.\ \mathbf{1}_R)$ **in** *exI*)
apply (*rule-tac* $x=\mathbf{0}_M$ **in** *exI*)
by (*unfold lincomb-def*, *auto*)
qed

lemma (**in** *module*) *zero-nin-lin-indpt*:
assumes *h2*: $S \subseteq carrier\ M$ **and** *li*: $\neg(lin\text{-}dep\ S)$ **and** *nonzero*: $carrier\ R \neq \{\mathbf{0}_R\}$
shows $\mathbf{0}_M \notin S$
proof (*rule ccontr*)
assume *a1*: $\neg(\mathbf{0}_M \notin S)$
from *a1* **have** *a2*: $\mathbf{0}_M \in S$ **by** *auto*
from *a2 nonzero* **have** *ld*: *lin-dep S* **by** (*rule zero-lin-dep*)
from *li ld* **show** *False* **by** *auto*
qed

The *span* of S is the set of linear combinations with $A \subseteq S$.

definition (**in** *module*) *span*::$'c$ *set* $\Rightarrow 'c$ *set*
where *span* $S = \{$*lincomb* a A $\mid$ a A. *finite* $A \wedge A \subseteq S \wedge a \in (A \to$ *carrier* $R)\}$

The *span* interpreted as a module or vectorspace.

abbreviation (**in** *module*) *span-vs*::$'c$ *set* $\Rightarrow ('a,'c,'d)$ *module-scheme*

where *span-vs* $S \equiv M$ ⦇*carrier* := *span* S⦈

In the finite case, we can take $A = S$ without loss of generality.

lemma (**in** *module*) *finite-span*:
assumes *fin*: *finite* S **and** *inC*: $S \subseteq$ *carrier* M
shows *span* $S = \{$*lincomb* a S $\mid$ a. $a \in (S \to$ *carrier* $R)\}$
proof (*rule equalityI*)
{
fix A a
assume *subset*: $A \subseteq S$ **and** a: $a \in A \to$ *carrier* R
let *?b*$=(\lambda v.$ *if* $v \in A$ *then* a v *else* $\mathbf{0})$
from *fin inC subset a* **have** *if-in*: $(\bigoplus_M v \in S.$ *?b* $v \odot_M v) = (\bigoplus_M v \in S.$ (*if* $v \in A$ *then* a $v \odot_M v$ *else* $\mathbf{0}_M))$
apply (*intro finsum-cong'*)
by (*auto simp add*: *coeff-in-ring*)
from *fin inC subset a* **have** $\exists$ b. *lincomb* a A = *lincomb* b $S \wedge b \in S \to$ *carrier* R
apply (*rule-tac x=?b* **in** *exI*)
apply (*unfold lincomb-def*, *auto*)
apply (*subst if-in*)
by (*subst extend-sum*, *auto*)
}
from *this* **show** *span* $S \subseteq \{$*lincomb* a S $|a.$ $a \in S \to$ *carrier* $R\}$
by (*unfold span-def*, *auto*)
next
from *fin* **show** $\{$*lincomb* a S $|a.$ $a \in S \to$ *carrier* $R\} \subseteq$ *span* S
by (*unfold span-def*, *auto*)
qed

If $v \in$ span S, then we can find a linear combination. This is in an easy to apply format (e.g. obtain a A where. . .)

lemma (**in** *module*) *in-span*:
fixes S v
assumes *h2*: $S \subseteq$ *carrier* V **and** *h3*: $v \in$ *span* S
shows $\exists$ a A. $(A \subseteq S \wedge (a \in A \to$ *carrier* $R) \wedge ($*lincomb* a $A = v))$
proof −
from *h2 h3* **show** *?thesis*
apply (*unfold span-def*)
by *auto*
qed

In the finite case, we can take $A = S$.

lemma (**in** *module*) *finite-in-span*:
 fixes *S v*
 assumes *fin*: *finite S* **and** *h2*: *S⊆carrier M* **and** *h3*: *v∈span S*
 shows ∃ *a*. (*a∈S→carrier R*) ∧ (*lincomb a S=v*)
proof −
 from *fin h2* **have** *fin-span*: *span S* = {*lincomb a S* |*a*. *a* ∈ *S* → *carrier R*} **by** (*rule finite-span*)
 from *h3 fin-span* **show** *?thesis* **by** *auto*
qed

If a subset is linearly independent, then any linear combination that is 0 must have a nonzero coefficient outside that set.

lemma (**in** *module*) *lincomb-must-include*:
 fixes *A S T b v*
 assumes *inC*: *T⊆carrier M* **and** *li*: *lin-indpt S* **and** *Ssub*: *S⊆T* **and** *Ssub*: *A⊆T*
  and *fin*: *finite A*
  and *b*: *b∈A→carrier R* **and** *lc*: *lincomb b A*=$\mathbf{0}_M$ **and** *v-in*: *v∈A*
  and *nz-coeff*: *b v*≠$\mathbf{0}_R$
 shows ∃ *w∈A−S*. *b w*≠$\mathbf{0}_R$
proof (*rule ccontr*)

 assume *0*: ¬(∃ *w∈A−S*. *b w*≠$\mathbf{0}_R$)
 from *0* **have** *1*: ⋀*w*. *w∈A−S*⟹ *b w*=$\mathbf{0}_R$ **by** *auto*
 have *Auni*: *A*=(*S∩A*) ∪(*A−S*) **by** *auto*
 from *fin b Ssub inC 1* **have** *2*: *lincomb b A* = *lincomb b* (*S∩A*)
  apply (*subst Auni*)
  apply (*subst lincomb-union*, *auto*)

  apply (*unfold lincomb-def*)
  apply (*subst* (*2*) *finsum-all0*, *auto*)
  by (*subst show-r-zero*, *auto intro*!: *finsum-closed*)
 from *1 2 assms* **have** *ld*: *lin-dep S*
  apply (*intro lin-dep-crit*[**where** *?A*=*S∩A* **and** *?a*=*b* **and** *?v*=*v*])
  by *auto*
 from *ld li* **show** *False* **by** *auto*
qed

A generating set is a set such that the span of S is all of M.

abbreviation (**in** *module*) *gen-set*::′*c set* ⇒ *bool*
 where *gen-set S* ≡ (*span S* = *carrier M*)

4.4 Submodules

lemma *module-criteria*:
 fixes *R* **and** *M*
 assumes *cring*: *cring R*
  and *zero*: $\mathbf{0}_M$∈ *carrier M*

and *add*: $\forall\, v\ w.\ v\in carrier\ M \wedge w\in carrier\ M \longrightarrow v\oplus_M w\in\ carrier\ M$

and *neg*: $\forall\, v\in carrier\ M.\ (\exists\, neg\text{-}v\in carrier\ M.\ v\oplus_M neg\text{-}v=\mathbf{0}_M)$

and *smult*: $\forall\, c\ v.\ c\in\ carrier\ R \wedge v\in carrier\ M \longrightarrow c\odot_M\ v \in carrier\ M$

and *comm*: $\forall\, v\ w.\ v\in carrier\ M \wedge w\in carrier\ M \longrightarrow v\oplus_M w=w\oplus_M v$

and *assoc*: $\forall\, v\ w\ x.\ v\in carrier\ M \wedge w\in carrier\ M \wedge x\in carrier\ M \longrightarrow (v\oplus_M w)\oplus_M x= v\oplus_M (w\oplus_M x)$

and *add-id*: $\forall\, v\in carrier\ M.\ (v\oplus_M \mathbf{0}_M =v)$

and *compat*: $\forall\, a\ b\ v.\ a\in\ carrier\ R \wedge b\in\ carrier\ R \wedge v\in carrier\ M \longrightarrow (a\otimes_R\ b)\odot_M\ v =a\odot_M (b\odot_M v)$

and *smult-id*: $\forall\, v\in carrier\ M.\ (\mathbf{1}_R \odot_M v =v)$

and *dist-f*: $\forall\, a\ b\ v.\ a\in\ carrier\ R \wedge b\in\ carrier\ R \wedge v\in carrier\ M \longrightarrow (a\oplus_R\ b)\odot_M\ v =(a\odot_M v) \oplus_M (b\odot_M v)$

and *dist-add*: $\forall\, a\ v\ w.\ a\in\ carrier\ R \wedge v\in carrier\ M \wedge w\in carrier\ M \longrightarrow a\odot_M (v\oplus_M w) =(a\odot_M v) \oplus_M (a\odot_M w)$

shows *module R M*
proof –
from *assms* **have** *2*: *abelian-group M*
by (*intro abelian-groupI*, *auto*)
from *assms* **have** *3*: *module-axioms R M*
by (*unfold module-axioms-def*, *auto*)
from *2 3 cring* **show** *?thesis*
by (*unfold module-def module-def*, *auto*)
qed

A submodule is $N \subseteq M$ that is closed under addition and scalar multiplication, and contains 0 (so is not empty).

locale *submodule* =
fixes *R* **and** *N* **and** *M* (**structure**)
assumes *module*: *module R M*
and *subset*: $N \subseteq carrier\ M$
and *m-closed* [*intro*, *simp*]: $[\![v \in N;\ w \in N]\!] \Longrightarrow v \oplus w \in N$
and *zero-closed* [*simp*]: $\mathbf{0} \in N$
and *smult-closed* [*intro*, *simp*]: $[\![c \in carrier\ R;\ v \in N]\!] \Longrightarrow c\odot v \in N$

abbreviation (**in** *module*) $md{::}'c\ set \Rightarrow ('a,\ 'c,\ 'd)\ module\text{-}scheme$
where $md\ N \equiv M(\!|carrier := N|\!)$

lemma (**in** *module*) *carrier-vs-is-self* [*simp*]:
$carrier\ (md\ N) = N$
by *auto*

lemma (**in** *module*) *submodule-is-module*:
fixes $N{::}'c\ set$
assumes *0*: *submodule R N M*
shows *module R (md N)*

proof (*unfold module-def*, *auto*)
  show *1*: *cring R*..
next
  from *assms* **show** *2*: *abelian-group* (*md N*)
    apply (*unfold submodule-def*)
    apply (*intro abelian-groupI*, *auto*)
        apply (*metis* (*no-types*, *opaque-lifting*) *M.add.m-assoc contra-subsetD*)
    apply (*metis* (*no-types*, *opaque-lifting*) *M.add.m-comm contra-subsetD*)
    apply (*rename-tac v*)

The inverse of v under addition is $-v$

    apply (*rule-tac x*=$\ominus_M v$ **in** *bexI*)
    apply (*metis M.l-neg contra-subsetD*)
    by (*metis R.add.inv-closed one-closed smult-minus-1 subset-iff*)
next
  from *assms* **show** *3*: *module-axioms R* (*md N*)
    apply (*unfold module-axioms-def submodule-def*, *auto*)
    apply (*metis* (*no-types*, *opaque-lifting*) *smult-l-distr contra-subsetD*)
    apply (*metis* (*no-types*, *opaque-lifting*) *smult-r-distr contra-subsetD*)
    by (*metis* (*no-types*, *opaque-lifting*) *smult-assoc1 contra-subsetD*)
qed

$N_1 + N_2 = \{x + y | x \in N_1, y \in N_2\}$

definition (**in** *module*) *submodule-sum*:: [*'c set*, *'c set*] ⇒ *'c set*
  where *submodule-sum N1 N2* = (λ (*x*,*y*). *x* $\oplus_M$ *y*) ‘{(*x*,*y*). *x*∈ *N1* ∧ *y*∈ *N2*}

A module homomorphism $M \to N$ preserves addition and scalar multiplication.

definition *module-hom*:: [(*'a*, *'c0*) *ring-scheme*,
 (*'a*,*'b1*,*'c1*) *module-scheme*, (*'a*,*'b2*,*'c2*) *module-scheme*] ⇒(*'b1*⇒*'b2*) *set*
  where *module-hom R M N* = {*f*.
    ((*f*∈ *carrier M* → *carrier N*)
    ∧ (∀ *m1 m2*. *m1*∈*carrier M*∧ *m2*∈*carrier M* ⟶ *f* (*m1* $\oplus_M$ *m2*) = (*f m1*) $\oplus_N$ (*f m2*))
    ∧ (∀ *r m*. *r*∈*carrier R*∧ *m*∈*carrier M* ⟶*f* (*r* $\odot_M$ *m*) = *r* $\odot_N$ (*f m*)))}

lemma *module-hom-closed*: *f*∈ *module-hom R M N* ⟹ *f*∈ *carrier M* → *carrier N*
by (*unfold module-hom-def*, *auto*)

lemma *module-hom-add*: ⟦*f*∈ *module-hom R M N*; *m1*∈*carrier M*; *m2*∈*carrier M* ⟧ ⟹ *f* (*m1* $\oplus_M$ *m2*) = (*f m1*) $\oplus_N$ (*f m2*)
by (*unfold module-hom-def*, *auto*)

lemma *module-hom-smult*: $[\![f\in$ *module-hom* R M N; $r\in$*carrier* R; $m\in$*carrier* M $]\!] \Longrightarrow f\ (r \odot_M m) = r \odot_N (f\ m)$
by (*unfold module-hom-def*, *auto*)

locale *mod-hom* =
  M?: *module* R M + *N?*: *module* R N
    for R **and** M **and** N +
  fixes f
  assumes *f-hom*: $f \in$ *module-hom* R M N
  notes *f-add* [*simp*] = *module-hom-add* [*OF f-hom*]
    and *f-smult* [*simp*] = *module-hom-smult* [*OF f-hom*]

Some basic simplification rules for module homomorphisms.

context *mod-hom*
begin

lemma *f-im* [*simp*, *intro*]:
assumes $v \in$ *carrier* M
shows $f\ v \in$ *carrier* N
proof −
  have *0*: *mod-hom* R M N f..
  from *0 assms* **show** *?thesis*
    apply (*unfold mod-hom-def module-hom-def mod-hom-axioms-def Pi-def*)
    by *auto*
qed

definition *im*:: *'e set*
  where $im = f`(carrier\ M)$

definition *ker*:: *'c set*
  where $ker = \{v.\ v \in carrier\ M\ \&\ f\ v = \mathbf{0}_N\}$

lemma *f0-is-0*[*simp*]: $f\ \mathbf{0}_M = \mathbf{0}_N$
proof −
  have *1*: $f\ \mathbf{0}_M = f\ (\mathbf{0}_R \odot_M \mathbf{0}_M)$ **by** *simp*
  have *2*: $f\ (\mathbf{0}_R \odot_M \mathbf{0}_M) = \mathbf{0}_N$
    using *M.M.zero-closed N.lmult-0 R.zero-closed f-im f-smult* **by** *presburger*
  from *1 2* **show** *?thesis* **by** *auto*
qed

lemma *f-neg* [*simp*]: $v \in carrier\ M \Longrightarrow f\ (\ominus_M v) = \ominus_N f\ v$
  by (*simp flip*: *M.smult-minus-1 N.smult-minus-1*)

lemma *f-minus* [*simp*]: $[\![v\in carrier\ M;\ w\in carrier\ M]\!] \Longrightarrow f\ (v \ominus_M w) = f\ v \ominus_N f\ w$
  by (*simp add*: *a-minus-def*)

lemma *ker-is-submodule*: *submodule R ker M*
proof −
  have *0*: *mod-hom R M N f*..
  from *0* **have** *1*: *module R M* **by** (*unfold mod-hom-def*, *auto*)
  show *?thesis*
    by (*rule submodule.intro*, *auto simp add*: *ker-def*, *rule 1*)
qed

lemma *im-is-submodule*: *submodule R im N*
proof −
  have *1*: *im* $\subseteq$ *carrier N* **by** (*auto simp add*: *im-def image-def mod-hom-def module-hom-def f-im*)
  have *2*: $\bigwedge$*w1 w2*.$[\![$*w1* $\in$ *im*; *w2* $\in$ *im*$]\!]$ $\Longrightarrow$ *w1* $\oplus_N$ *w2* $\in$ *im*
  proof −
    fix *w1 w2*
    assume *w1*: *w1* $\in$ *im* **and** *w2*: *w2*$\in$ *im*
    from *w1* **obtain** *v1* **where** *3*: *v1*$\in$ *carrier M* $\wedge$ *f v1* = *w1* **by** (*unfold im-def*, *auto*)
    from *w2* **obtain** *v2* **where** *4*: *v2*$\in$ *carrier M* $\wedge$ *f v2* = *w2* **by** (*unfold im-def*, *auto*)
    from *3 4* **have** *5*: *f* (*v1*$\oplus_M$*v2*) = *w1* $\oplus_N$ *w2* **by** *simp*
    from *3 4* **have** *6*: *v1*$\oplus_M$*v2*$\in$ *carrier M* **by** *simp*
    from *5 6* **have** *7*: $\exists$ *x*$\in$*carrier M*. *w1* $\oplus_N$ *w2* = *f x* **by** *metis*
    from *7* **show** *?thesis w1 w2* **by** (*unfold im-def image-def*, *auto*)
  qed
  have *3*: $\mathbf{0}_N$ $\in$ *im*
  proof −
    have *8*: *f* $\mathbf{0}_M$ = $\mathbf{0}_N$ $\wedge$ $\mathbf{0}_M$$\in$*carrier M* **by** *auto*
    from *8* **have** *9*: $\exists$ *x*$\in$*carrier M*. $\mathbf{0}_N$ = *f x* **by** *metis*
    from *9* **show** *?thesis* **by** (*unfold im-def image-def*, *auto*)
  qed
  have *4*: $\bigwedge$*c w*. $[\![$*c* $\in$ *carrier R*; *w* $\in$ *im*$]\!]$ $\Longrightarrow$ *c*$\odot_N$ *w* $\in$ *im*
  proof −
    fix *c w*
    assume *c*: *c* $\in$ *carrier R* **and** *w*: *w* $\in$ *im*
    from *w* **obtain** *v* **where** *10*: *v*$\in$ *carrier M* $\wedge$ *f v* = *w* **by** (*unfold im-def*, *auto*)
    from *c 10* **have** *11*: *f* (*c*$\odot_M$ *v*) = *c*$\odot_N$ *w*$\wedge$ (*c* $\odot_M$ *v*$\in$*carrier M*) **by** *auto*
    from *11* **have** *12*: $\exists$ *v1*$\in$*carrier M*. *c*$\odot_N$ *w*=*f v1* **by** *metis*
    from *12* **show** *?thesis c w* **by** (*unfold im-def image-def*, *auto*)
  qed
  from *1 2 3 4* **show** *?thesis* **by** (*unfold-locales*, *auto*)
qed

lemma (**in** *mod-hom*) *f-ker*:
  v$\in$*ker* $\Longrightarrow$ *f v*=$\mathbf{0}_N$
by (*unfold ker-def*, *auto*)
end

We will show that for any set S, the space of functions $S \to K$ forms a vector space.

```
definition (in ring) func-space:: 'z set⇒('a,('z ⇒ 'a)) module
  where func-space S = (|carrier = S→E carrier R,
                 mult = (λ f g. restrict (λv. 0R) S),
                 one =  restrict (λv. 0R) S,
                 zero = restrict (λv. 0R) S,
                 add = (λ f g. restrict (λv. f v ⊕R g v) S),
                 smult = (λ c f. restrict (λv. c ⊗R f v) S)|)

lemma (in cring) func-space-is-module:
  fixes S
  shows module R (func-space S)
proof −
have 0: cring R..
from 0 show ?thesis
  apply (auto intro!: module-criteria simp add: func-space-def)
         apply (auto simp add: module-def)
       apply (rename-tac f)
       apply (rule-tac x=restrict (λv'. ⊖R (f v')) S in bexI)
     apply (auto simp add:restrict-def cong: if-cong split: if-split-asm,
auto)
       apply (auto simp add: a-ac PiE-mem2 r-neg)
    apply (unfold PiE-def extensional-def Pi-def)
    by (auto simp add: m-assoc l-distr r-distr)
qed
```

Note: one can define M^n from this.

A linear combination is a module homomorphism from the space of coefficients to the module, $(a_v) \mapsto \sum_{v\in S} a_v v$.

```
lemma (in module) lincomb-is-mod-hom:
  fixes S
  assumes h: finite S and h2: S⊆carrier M
  shows mod-hom R (func-space S) M (λa. lincomb a S)
proof −
  have 0: module R M..
  {
   fix m1 m2
   assume m1: m1 ∈ S →E carrier R and m2: m2 ∈ S →E carrier
R
   from h h2 m1 m2 have a1: (⨁Mv∈S. (λv∈S. m1 v ⊕R m2 v) v
⊙M v) =
    (⨁Mv∈S. m1 v ⊙M v ⊕M m2 v ⊙M v)
    by (intro finsum-cong', auto simp add: smult-l-distr PiE-mem2)
   from h h2 m1 m2 have a2: (⨁Mv∈S. m1 v ⊙M v ⊕M m2 v ⊙M
v) =
    (⨁Mv∈S. m1 v ⊙M v) ⊕M (⨁Mv∈S. m2 v ⊙M v)
    by (intro finsum-addf, auto)
```

from *a1 a2* **have** $(\bigoplus_M v{\in}S.\ (\lambda v{\in}S.\ m1\ v \oplus m2\ v)\ v \odot_M v) =$
$(\bigoplus_M v{\in}S.\ m1\ v \odot_M v) \oplus_M (\bigoplus_M v{\in}S.\ m2\ v \odot_M v)$ **by** *auto*
}
hence *1*: $\bigwedge m1\ m2.$
$m1 \in S \rightarrow_E carrier\ R \Longrightarrow$
$m2 \in S \rightarrow_E carrier\ R \Longrightarrow (\bigoplus_M v{\in}S.\ (\lambda v{\in}S.\ m1\ v \oplus m2\ v)\ v$
$\odot_M v) =$
$(\bigoplus_M v{\in}S.\ m1\ v \odot_M v) \oplus_M (\bigoplus_M v{\in}S.\ m2\ v \odot_M v)$ **by** *auto*
{
fix *r m*
assume *r*: $r \in carrier\ R$ **and** *m*: $m \in S \rightarrow_E carrier\ R$
from *h h2 r m* **have** *b1*: $r \odot_M (\bigoplus_M v{\in}S.\ m\ v \odot_M v) = (\bigoplus_M v{\in}S.$
$r \odot_M (m\ v \odot_M v))$
by (*intro finsum-smult*, *auto*)
from *h h2 r m* **have** *b2*: $(\bigoplus_M v{\in}S.\ (\lambda v{\in}S.\ r \otimes m\ v)\ v \odot_M v) =$
$r \odot_M (\bigoplus_M v{\in}S.\ m\ v \odot_M v)$
apply (*subst b1*)
apply (*intro finsum-cong'*, *auto*)
by (*subst smult-assoc1*, *auto*)
}
hence *2*: $\bigwedge r\ m.\ r \in carrier\ R \Longrightarrow$
$m \in S \rightarrow_E carrier\ R \Longrightarrow (\bigoplus_M v{\in}S.\ (\lambda v{\in}S.\ r \otimes m\ v)\ v \odot_M$
$v) = r \odot_M (\bigoplus_M v{\in}S.\ m\ v \odot_M v)$
by *auto*
from *h h2 0 1 2* **show** *?thesis*
apply (*unfold mod-hom-def*, *auto*)
apply (*rule func-space-is-module*)
apply (*unfold mod-hom-axioms-def module-hom-def*, *auto*)
apply (*rule lincomb-closed*, *unfold func-space-def*, *auto*)
apply (*unfold lincomb-def*)
by *auto*
qed

lemma (**in** *module*) *lincomb-sum*:
assumes *A-fin*: *finite A* **and** *AinC*: $A \subseteq carrier\ M$ **and** *a-fun*: $a \in A \rightarrow carrier$
R **and**
b-fun: $b \in A \rightarrow carrier\ R$
shows *lincomb* $(\lambda v.\ a\ v \oplus_R b\ v)\ A = lincomb\ a\ A \oplus_M lincomb\ b\ A$
proof −
from *A-fin AinC* **interpret** *mh*: *mod-hom R func-space A M* $(\lambda a.$
$lincomb\ a\ A)$ **by** (*rule*
lincomb-is-mod-hom)
let *?a=restrict a A*
let *?b=restrict b A*
from *a-fun b-fun A-fin AinC*
have *1*: *LinearCombinations.module.lincomb M* $(?a \oplus_{(LinearCombinations.ring.func\text{-}space\ R\ A)}$
$?b)\ A$
$=$ *LinearCombinations.module.lincomb M* $(\lambda x.\ a\ x \oplus_R b\ x)\ A$

by (*auto simp add*: *func-space-def Pi-iff restrict-apply′ cong*: *lincomb-cong*)
from *a-fun b-fun A-fin AinC*
have *2*: *LinearCombinations.module.lincomb M ?a A* $\oplus_M$ *LinearCombinations.module.lincomb M ?b A* = *LinearCombinations.module.lincomb M a A* $\oplus_M$ *LinearCombinations.module.lincomb M b A*
by (*simp-all add*: *sum-simp cong*: *lincomb-cong*)
from *a-fun b-fun* **have** *ainC*: *?a*∈*carrier* (*LinearCombinations.ring.func-space R A*)
and *binC*: *?b*∈*carrier* (*LinearCombinations.ring.func-space R A*)
by (*unfold func-space-def*, *auto*)
from *ainC binC* **have** *LinearCombinations.module.lincomb M* (*?a*$\oplus_{(LinearCombinations.ring.func\text{-}space}$ *?b*) *A*
= *LinearCombinations.module.lincomb M ?a A* $\oplus_M$ *LinearCombinations.module.lincomb M ?b A*
by (*simp cong*: *lincomb-cong*)
with *1 2* **show** *?thesis* **by** *auto*
qed

The negative of a function is just pointwise negation.

lemma (**in** *cring*) *func-space-neg*:
fixes *f*
assumes *f*∈ *carrier* (*func-space S*)
shows $\ominus_{func\text{-}space\ S}$ *f* = (λ *v*. *if* (*v*∈*S*) *then* $\ominus_R$ *f v else undefined*)
proof −
interpret *fs*: *module R func-space S* **by** (*rule func-space-is-module*)
from *assms* **show** *?thesis*
apply (*intro fs.minus-equality*)
apply (*unfold func-space-def PiE-def extensional-def*)
apply *auto*
apply (*intro restrict-ext*, *auto*)
by (*simp add*: *l-neg coeff-in-ring*)
qed

Ditto for subtraction. Note the above is really a special case, when a is the 0 function.

lemma (**in** *module*) *lincomb-diff*:
assumes *A-fin*: *finite A* **and** *AinC*: *A*⊆*carrier M* **and** *a-fun*: *a*∈*A*→*carrier R* **and**
b-fun: *b*∈*A*→*carrier R*
shows *lincomb* (λ*v*. *a v* $\ominus_R$ *b v*) *A* = *lincomb a A* $\ominus_M$ *lincomb b A*
proof −
from *A-fin AinC* **interpret** *mh*: *mod-hom R func-space A M* (λ*a*. *lincomb a A*) **by** (*rule lincomb-is-mod-hom*)
let *?a*=*restrict a A*
let *?b*=*restrict b A*

from *a-fun b-fun* **have** *ainC*: *?a*∈*carrier* (*LinearCombinations.ring.func-space R A*)
and *binC*: *?b*∈*carrier* (*LinearCombinations.ring.func-space R A*)
by (*unfold func-space-def*, *auto*)
from *a-fun b-fun ainC binC A-fin AinC*
have *1*: *LinearCombinations.module.lincomb M* (*?a*$\ominus_{(\textit{func-space } A)}$ *?b*) *A*
= *LinearCombinations.module.lincomb M* ($\lambda x.$ *a x* $\ominus_R$ *b x*) *A*
apply (*subst mh.M.M.minus-eq*)
apply (*intro lincomb-cong*, *auto*)
apply (*subst func-space-neg*, *auto*)
apply (*simp add*: *restrict-def func-space-def*)
by (*subst R.minus-eq*, *auto*)
from *a-fun b-fun A-fin AinC*
have *2*: *LinearCombinations.module.lincomb M ?a A* $\ominus_M$
LinearCombinations.module.lincomb M ?b A = *LinearCombinations.module.lincomb M a A* $\ominus_M$
LinearCombinations.module.lincomb M b A
by (*simp cong*: *lincomb-cong*)
from *ainC binC* **have** *LinearCombinations.module.lincomb M* (*?a*$\ominus_{(\textit{LinearCombinations.ring.func-space}}$ *?b*) *A*
= *LinearCombinations.module.lincomb M ?a A* $\ominus_M$
LinearCombinations.module.lincomb M ?b A
by (*simp cong*: *lincomb-cong*)
with *1 2* **show** *?thesis* **by** *auto*
qed

The union of nested submodules is a submodule. We will use this to show that span of any set is a submodule.

lemma (**in** *module*) *nested-union-vs*:
fixes *I N N'*
assumes *subm*: $\bigwedge i.\ i{\in}I \Longrightarrow$ *submodule R* (*N i*) *M*
and *max-exists*: $\bigwedge i\ j.\ i{\in}I \Longrightarrow j{\in}I \Longrightarrow (\exists k.\ k{\in}I \wedge N\ i{\subseteq}N\ k \wedge N\ j \subseteq N\ k)$
and *uni*: $N'{=}(\bigcup\ i{\in}I.\ N\ i)$
and *ne*: $I{\neq}\{\}$
shows *submodule R N' M*
proof −
have *1*: *module R M*..
from *subm* **have** *all-in*: $\bigwedge i.\ i{\in}I \Longrightarrow N\ i \subseteq$ *carrier M*
by (*unfold submodule-def*, *auto*)
from *uni all-in* **have** *2*: $\bigwedge x.\ x \in N' \Longrightarrow x \in$ *carrier M*
by *auto*
from *uni* **have** *3*: $\bigwedge v\ w.\ v \in N' \Longrightarrow w \in N' \Longrightarrow v \oplus_M w \in N'$
proof −
fix *v w*
assume *v*: $v \in N'$ **and** *w*: $w \in N'$
from *uni v w* **obtain** *i j* **where** *i*: $i{\in}I \wedge v{\in} N\ i$ **and** *j*: $j{\in}I \wedge w{\in} N\ j$ **by** *auto*

from *max-exists i j* **obtain** *k* **where** *k*: $k \in I \wedge N\ i \subseteq N\ k \wedge N\ j \subseteq N\ k$ **by** *presburger*
from *v w i j k* **have** *v2*: $v \in N\ k$ **and** *w2*: $w \in N\ k$ **by** *auto*
from *v2 w2 k subm*[*of k*] **have** *vw*: $v \oplus_M w \in N\ k$ **apply** (*unfold submodule-def*) **by** *auto*
from *k vw uni* **show** *?thesis v w* **by** *auto*
qed
have *4*: $\mathbf{0}_M \in N'$
proof −
from *ne* **obtain** *i* **where** *i*: $i \in I$ **by** *auto*
from *i subm* **have** *zi*: $\mathbf{0}_M \in N\ i$ **by** (*unfold submodule-def*, *auto*)
from *i zi uni* **show** *?thesis* **by** *auto*
qed
from *uni subm* **have** *5*: $\bigwedge c\ v.\ c \in carrier\ R \Longrightarrow v \in N' \Longrightarrow c \odot_M v \in N'$
by (*unfold submodule-def*, *auto*)
from *1 2 3 4 5* **show** *?thesis* **by** (*unfold submodule-def*, *auto*)
qed

lemma (**in** *module*) *span-is-monotone*:
fixes *S T*
assumes *subs*: $S \subseteq T$
shows *span S* $\subseteq$ *span T*
proof −
from *subs* **show** *?thesis*
by (*unfold span-def*, *auto*)
qed

lemma (**in** *module*) *span-is-submodule*:
fixes *S*
assumes *h2*: $S \subseteq carrier\ M$
shows *submodule R* (*span S*) *M*
proof (*cases* $S=\{\}$)
case *True*
moreover have *module R M*..
ultimately show *?thesis* **apply** (*unfold submodule-def span-def lincomb-def*, *auto*) **done**
next
case *False*
show *?thesis*
proof (*rule nested-union-vs*[**where** $?I=\{F.\ F \subseteq S \wedge finite\ F\}$ **and** $?N=\lambda F.\ span\ F$ **and** $?N'=span\ S$])
show $\bigwedge F.\ F \in \{F.\ F \subseteq S \wedge finite\ F\} \Longrightarrow submodule\ R\ (span\ F)\ M$
proof −
fix *F*
assume *F*: $F \in \{F.\ F \subseteq S \wedge finite\ F\}$
from *F* **have** *h1*: *finite F* **by** *auto*

```
      from F h2 have inC: F⊆carrier M by auto
      from h1 inC interpret mh: mod-hom R (func-space F) M (λa.
lincomb a F)
        by (rule lincomb-is-mod-hom)
      from h1 inC have 1: mh.im = span F
        apply (unfold mh.im-def)
        apply (unfold func-space-def, simp)
        apply (subst finite-span, auto)
        apply (unfold image-def, auto)
        apply (rule-tac x=restrict a F in bexI)
         by (auto intro!: lincomb-cong)
    from 1 show submodule R (span F) M by (metis mh.im-is-submodule)
    qed
  next
    show ⋀i j. i ∈ {F. F ⊆ S ∧ finite F} ⟹
           j ∈ {F. F ⊆ S ∧ finite F} ⟹
           ∃k. k ∈ {F. F ⊆ S ∧ finite F} ∧ span i ⊆ span k ∧ span j
⊆ span k
    proof -
      fix i j
      assume i: i ∈ {F. F ⊆ S ∧ finite F} and j: j ∈ {F. F ⊆ S ∧
finite F}
      from i j show ?thesis i j
        apply (rule-tac x=i∪j in exI)
        apply (auto del: subsetI)
         by (intro span-is-monotone, auto del: subsetI)+
    qed
  next
    show span S=(⋃ i∈{F. F ⊆ S ∧ finite F}. span i)
      by (unfold span-def,auto)
  next
    have ne: S≠{} by fact
    from ne show {F. F ⊆ S ∧ finite F} ≠ {} by auto
  qed
qed
```

A finite sum does not depend on the ambient module. This can be done for monoid, but "submonoid" isn't currently defined. (It can be copied, however, for groups...) This lemma requires a somewhat annoying lemma foldD-not-depend. Then we show that linear combinations, linear independence, span do not depend on the ambient module.

```
lemma (in module) finsum-not-depend:
  fixes a A N
   assumes h1: finite A and h2: A⊆N and h3: submodule R N M
and h4: f:A→N
  shows (⨁(md N) v∈A. f v) = (⨁M v∈A. f v)
proof -
```

```
  from h1 h2 h3 h4 show ?thesis
    apply (unfold finsum-def finprod-def)
    apply simp
    apply (intro foldD-not-depend[where ?B=A])
        apply (unfold submodule-def LCD-def, auto)
    apply (meson M.add.m-lcomm PiE subsetCE)+
    done
qed

lemma (in module) lincomb-not-depend:
  fixes a A N
   assumes h1: finite A and h2: A⊆N and h3: submodule R N M
and h4: a:A→carrier R
  shows lincomb a A = module.lincomb (md N) a A
proof -
 from h3 interpret N: module R (md N) by (rule submodule-is-module)
  have 3: N=carrier (md N) by auto
  have 4: (smult M ) = (smult (md N)) by auto
  from h1 h2 h3 h4 have (⨁(md N)v∈A. a v ⊙M v) = (⨁Mv∈A. a
v ⊙M v)
    apply (intro finsum-not-depend)
    using N.summands-valid by auto
 from this show ?thesis by (unfold lincomb-def N.lincomb-def, simp)
qed

lemma (in module) span-li-not-depend:
  fixes S N
  assumes h2: S⊆N and  h3: submodule R N M
  shows module.span R (md N) S = module.span R M S
    and module.lin-dep R (md N) S = module.lin-dep R M S
proof -
 from h3 interpret w: module R (md N) by (rule submodule-is-module)
  from h2 have 1:submodule R (module.span R (md N) S) (md N)
    by (intro w.span-is-submodule, simp)
  have 3: ⋀a A. (finite A ∧ A⊆S ∧ a ∈ A → carrier R ⟹
    module.lincomb M a A = module.lincomb (md N) a A)
  proof -
    fix a A
    assume 31: finite A ∧ A⊆S ∧ a ∈ A → carrier R
    from assms 31 show ?thesis a A
      by (intro lincomb-not-depend, auto)
  qed
  from 3 show 4: module.span R (md N) S = module.span R M S
    apply (unfold span-def w.span-def)
    apply auto
    by (metis)
  have zeros: 0md N=0M by auto
 from assms 3 show 5: module.lin-dep R (md N) S = module.lin-dep
R M S
```

apply (*unfold lin-dep-def w.lin-dep-def*)
apply (*subst zeros*)
by *metis*
qed

lemma (**in** *module*) *span-is-subset*:
fixes *S N*
assumes *h2*: *S*⊆*N* **and** *h3*: *submodule R N M*
shows *span S* ⊆ *N*
proof −
from *h3* **interpret** *w*: *module R* (*md N*) **by** (*rule submodule-is-module*)
from *h2* **have** *1*:*submodule R* (*module.span R* (*md N*) *S*) (*md N*)
by (*intro w.span-is-submodule*, *simp*)
from *assms* **have** *4*: *module.span R* (*md N*) *S* = *module.span R M S*
by (*rule span-li-not-depend*)
from *1 4* **have** *5*: *submodule R* (*module.span R M S*) (*md N*) **by** *auto*
from *5* **show** *?thesis* **by** (*unfold submodule-def*, *simp*)
qed

lemma (**in** *module*) *span-is-subset2*:
fixes *S*
assumes *h2*: *S*⊆*carrier M*
shows *span S* ⊆ *carrier M*
proof −
have *0*: *module R M*..
from *0* **have** *h3*: *submodule R* (*carrier M*) *M* **by** (*unfold submodule-def*, *auto*)
from *h2 h3* **show** *?thesis* **by** (*rule span-is-subset*)
qed

lemma (**in** *module*) *in-own-span*:
fixes *S*
assumes *inC*:*S*⊆*carrier M*
shows *S* ⊆ *span S*
proof −
from *inC* **show** *?thesis*
apply (*unfold span-def*, *auto*)
apply (*rename-tac v*)
apply (*rule-tac x*=(λ *w. if* (*w*=*v*) *then* $\mathbf{1}_R$ *else* $\mathbf{0}_R$) **in** *exI*)
apply (*rule-tac x*={*v*} **in** *exI*)
apply (*unfold lincomb-def*)
by *auto*
qed

lemma (**in** *module*) *supset-ld-is-ld*:
fixes *A B*

assumes *ld*: *lin-dep A* **and** *sub*: $A \subseteq B$
shows *lin-dep B*
proof –
from *ld* **obtain** A' *a v* **where** *1*: (*finite* $A' \wedge A' \subseteq A \wedge (a \in (A' \rightarrow$ *carrier* $R)) \wedge$ (*lincomb a* $A' = \mathbf{0}_M) \wedge (v \in A') \wedge (a\ v \neq \mathbf{0}_R)$)
by (*unfold lin-dep-def*, *auto*)
from *1 sub* **show** *?thesis*
apply (*unfold lin-dep-def*)
apply (*rule-tac* $x=A'$ **in** *exI*)
apply (*rule-tac* $x=a$ **in** *exI*)
apply (*rule-tac* $x=v$ **in** *exI*)
by *auto*
qed

lemma (**in** *module*) *subset-li-is-li*:
fixes *A B*
assumes *li*: *lin-indpt A* **and** *sub*: $B \subseteq A$
shows *lin-indpt B*
proof (*rule ccontr*)
assume *ld*: $\neg$*lin-indpt B*
from *ld sub* **have** *ldA*: *lin-dep A* **by** (*metis supset-ld-is-ld*)
from *li ldA* **show** *False* **by** *auto*
qed

lemma (**in** *mod-hom*) *hom-sum*:
fixes *A B g*
assumes *h2*: $A \subseteq$ *carrier M* **and** *h3*: g:$A \rightarrow$ *carrier M*
shows $f\ (\bigoplus_M a \in A.\ g\ a) = (\bigoplus_N a \in A.\ f\ (g\ a))$
proof –
from *h2 h3* **show** *?thesis*
proof (*induct A rule*: *infinite-finite-induct*)
case (*insert a A*)
then have $(\bigoplus_N a \in$ *insert a A*. $f\ (g\ a)) = f\ (g\ a) \oplus_N (\bigoplus_N a \in A.\ f\ (g\ a))$
by (*intro finsum-insert*, *auto*)
with *insert.prems insert.hyps* **show** *?case*
by *simp*
qed *auto*
**qed*

end

5 The direct sum of modules.

theory *SumSpaces*
imports *Main*
HOL–Algebra.Module
HOL–Algebra.Coset

RingModuleFacts
MonoidSums
FunctionLemmas
LinearCombinations
begin

We define the direct sum $M_1 \oplus M_2$ of 2 vector spaces as the set $M_1 \times M_2$ under componentwise addition and scalar multiplication.

definition *direct-sum*:: $('a,'b,\ 'd)$ *module-scheme* $\Rightarrow$ $('a,\ 'c,\ 'e)$ *module-scheme* $\Rightarrow$$('a,\ ('b\times'c))$ *module*
where *direct-sum M1 M2* = $(\!|$*carrier* = *carrier M1* $\times$ *carrier M2*,
mult = $(\lambda\ v\ w.\ (\mathbf{0}_{M1},\ \mathbf{0}_{M2}))$,
one = $(\mathbf{0}_{M1},\ \mathbf{0}_{M2})$,
zero = $(\mathbf{0}_{M1},\ \mathbf{0}_{M2})$,
add = $(\lambda\ v\ w.\ (fst\ v \oplus_{M1} fst\ w,\ snd\ v \oplus_{M2} snd\ w))$,
smult = $(\lambda\ c\ v.\ (c \odot_{M1} fst\ v,\ c \odot_{M2} snd\ v))|\!)$

lemma *direct-sum-is-module*:
fixes *R M1 M2*
assumes *h1*: *module R M1* **and** *h2*: *module R M2*
shows *module R* (*direct-sum M1 M2*)
proof −
from *h1* **have** *1*: *cring R* **by** (*unfold module-def*, *auto*)
from *h1* **interpret** *v1*: *module R M1* **by** *auto*
from *h2* **interpret** *v2*: *module R M2* **by** *auto*
from *h1 h2* **have** *2*: *abelian-group* (*direct-sum M1 M2*)
apply (*intro abelian-groupI*, *auto*)
apply (*unfold direct-sum-def*, *auto*)
by (*auto simp add*: *v1.a-ac v2.a-ac*)
from *h1 h2 assms* **have** *3*: *module-axioms R* (*direct-sum M1 M2*)
apply (*unfold module-axioms-def*, *auto*)
apply (*unfold direct-sum-def*, *auto*)
by (*auto simp add*: *v1.smult-l-distr v2.smult-l-distr v1.smult-r-distr v2.smult-r-distr*
v1.smult-assoc1 v2.smult-assoc1)
from *1 2 3* **show** *?thesis* **by** (*unfold module-def*, *auto*)
qed

definition *inj1*:: $('a,'b)$ *module* $\Rightarrow$ $('a,\ 'c)$ *module* $\Rightarrow$$('b\Rightarrow('b\times'c))$
where *inj1 M1 M2* = $(\lambda v.\ (v,\ \mathbf{0}_{M2}))$

definition *inj2*:: $('a,'b)$ *module* $\Rightarrow$ $('a,\ 'c)$ *module* $\Rightarrow$$('c\Rightarrow('b\times'c))$
where *inj2 M1 M2* = $(\lambda v.\ (\mathbf{0}_{M1},\ v))$

lemma *inj1-hom*:
fixes *R M1 M2*
assumes *h1*: *module R M1* **and** *h2*: *module R M2*
shows *mod-hom R M1* (*direct-sum M1 M2*) (*inj1 M1 M2*)

proof −
  from *h1* **interpret** *v1*:*module R M1* **by** *auto*
  from *h2* **interpret** *v2*:*module R M2* **by** *auto*
  from *h1 h2* **show** *?thesis*
    apply (*unfold mod-hom-def module-hom-def mod-hom-axioms-def inj1-def*, *auto*)
      apply (*rule direct-sum-is-module*, *auto*)
     by (*unfold direct-sum-def*, *auto*)
qed

lemma *inj2-hom*:
  fixes *R M1 M2*
  assumes *h1*: *module R M1* **and** *h2*: *module R M2*
  shows *mod-hom R M2* (*direct-sum M1 M2*) (*inj2 M1 M2*)
proof −
  from *h1* **interpret** *v1*:*module R M1* **by** *auto*
  from *h2* **interpret** *v2*:*module R M2* **by** *auto*
  from *h1 h2* **show** *?thesis*
    apply (*unfold mod-hom-def module-hom-def mod-hom-axioms-def inj2-def*, *auto*)
      apply (*rule direct-sum-is-module*, *auto*)
     by (*unfold direct-sum-def*, *auto*)
qed

For submodules $M_1, M_2 \subseteq M$, the map $M_1 \oplus M_2 \to M$ given by $(m_1, m_2) \mapsto m_1 + m_2$ is linear.

lemma (**in** *module*) *sum-map-hom*:
  fixes *M1 M2*
  assumes *h1*: *submodule R M1 M* **and** *h2*: *submodule R M2 M*
  shows *mod-hom R* (*direct-sum* (*md M1*) (*md M2*)) *M* (λ *v*. (*fst v*) $\oplus_M$ (*snd v*))
proof −
  have *0*: *module R M*..
  from *h1* **have** *1*: *module R* (*md M1*) **by** (*rule submodule-is-module*)
  from *h2* **have** *2*: *module R* (*md M2*) **by** (*rule submodule-is-module*)
  from *h1* **interpret** *w1*: *module R* (*md M1*) **by** (*rule submodule-is-module*)
  from *h2* **interpret** *w2*: *module R* (*md M2*) **by** (*rule submodule-is-module*)
  from *0 h1 h2 1 2* **show** *?thesis*
    apply (*unfold mod-hom-def mod-hom-axioms-def module-hom-def*, *auto*)
      apply (*rule direct-sum-is-module*, *auto*)
     apply (*unfold direct-sum-def*, *auto*)
     apply (*unfold submodule-def*, *auto*)
    by (*auto simp add*: *a-ac smult-r-distr ring-subset-carrier*)

qed

lemma (**in** *module*) *sum-is-submodule*:
  fixes *N1 N2*

assumes *h1*: *submodule R N1 M* **and** *h2*: *submodule R N2 M*
shows *submodule R* (*submodule-sum N1 N2*) *M*
proof −
from *h1 h2* **interpret** *l*: *mod-hom R* (*direct-sum* (*md N1*) (*md N2*)) *M* (λ *v*. (*fst v*) $\oplus_M$ (*snd v*))
by (*rule sum-map-hom*)
have *1*: *l.im* =*submodule-sum N1 N2*
apply (*unfold l.im-def submodule-sum-def*)
apply (*unfold direct-sum-def*, *auto*)
by (*unfold image-def*, *auto*)
have *2*: *submodule R* (*l.im*) *M* **by** (*rule l.im-is-submodule*)
from *1 2* **show** *?thesis* **by** *auto*
qed

lemma (**in** *module*) *in-sum*:
fixes *N1 N2*
assumes *h1*: *submodule R N1 M* **and** *h2*: *submodule R N2 M*
shows *N1* $\subseteq$ *submodule-sum N1 N2*
proof −
from *h1 h2* **show** *?thesis*
apply *auto*
apply (*unfold submodule-sum-def image-def*, *auto*)
apply (*rename-tac v*)
apply (*rule-tac x=v* **in** *bexI*)
apply (*rule-tac x=*$\mathbf{0}_M$ **in** *bexI*)
by (*unfold submodule-def*, *auto*)
qed

lemma (**in** *module*) *msum-comm*:
fixes *N1 N2*
assumes *h1*: *submodule R N1 M* **and** *h2*: *submodule R N2 M*
shows (*submodule-sum N1 N2*) = (*submodule-sum N2 N1*)
proof −

from *h1 h2* **show** *?thesis*
apply (*unfold submodule-sum-def image-def*, *auto*)
apply (*unfold submodule-def*)
apply (*rename-tac v w*)
by (*metis* (*full-types*) *M.add.m-comm subsetD*)+

qed

If $M_1, M_2 \subseteq M$ are submodules, then $M_1 + M_2$ is the minimal subspace such that both $M_1 \subseteq M$ and $M_2 \subseteq M$.

lemma (**in** *module*) *sum-is-minimal*:
fixes *N N1 N2*
assumes *h1*: *submodule R N1 M* **and** *h2*: *submodule R N2 M* **and** *h3*: *submodule R N M*
shows (*submodule-sum N1 N2*) $\subseteq$ *N* $\longleftrightarrow$ *N1* $\subseteq$ *N* $\wedge$ *N2* $\subseteq$ *N*

proof –
have *1*: (*submodule-sum N1 N2*) $\subseteq$ *N* $\Longrightarrow$ *N1* $\subseteq$ *N* $\wedge$ *N2* $\subseteq$ *N*
proof –
assume *10*: (*submodule-sum N1 N2*) $\subseteq$ *N*
from *h1 h2* **have** *11*: *N1*$\subseteq$*submodule-sum N1 N2* **by** (*rule in-sum*)
from *h2 h1* **have** *12*: *N2*$\subseteq$*submodule-sum N2 N1* **by** (*rule in-sum*)
from *12 h1 h2* **have** *13*: *N2*$\subseteq$*submodule-sum N1 N2* **by** (*metis msum-comm*)
from *10 11 13* **show** *?thesis* **by** *auto*
qed
have *2*: *N1* $\subseteq$ *N* $\wedge$ *N2* $\subseteq$ *N* $\Longrightarrow$ (*submodule-sum N1 N2*) $\subseteq$ *N*
proof –
assume *19*: *N1* $\subseteq$ *N* $\wedge$ *N2* $\subseteq$ *N*
{
fix *v*
assume *20*: *v*$\in$*submodule-sum N1 N2*
from *20* **obtain** *w1 w2* **where** *21*: *w1*$\in$*N1* **and** *22*: *w2*$\in$*N2* **and** *23*: *v*=*w1*$\oplus_M$ *w2*
by (*unfold submodule-sum-def image-def*, *auto*)
from *19 21 22 23 h3* **have** *v* $\in$ *N*
apply (*unfold submodule-def*, *auto*)
by (*metis* (*poly-guards-query*) *contra-subsetD*)

}
thus *?thesis*
by (*metis subset-iff*)
qed
from *1 2* **show** *?thesis* **by** *metis*
qed

$\text{span}A \cup B = \text{span}A + \text{span}B$

lemma (**in** *module*) *span-union-is-sum*:
fixes *A B*
assumes *h2*: *A*$\subseteq$*carrier M* **and** *h3*: *B*$\subseteq$*carrier M*
shows *span* (*A*$\cup$ *B*) = *submodule-sum* (*span A*) (*span B*)
proof–
let *?AplusB*=*submodule-sum* (*span A*) (*span B*)
from *h2* **have** *s0*: *submodule R* (*span A*) *M* **by** (*rule span-is-submodule*)
from *h3* **have** *s1*: *submodule R* (*span B*) *M* **by** (*rule span-is-submodule*)
from *s0* **have** *s0-1*: (*span A*)$\subseteq$*carrier M* **by** (*unfold submodule-def*, *auto*)
from *s1* **have** *s1-1*: (*span B*)$\subseteq$*carrier M* **by** (*unfold submodule-def*, *auto*)
from *h2 h3* **have** *1*: *A*$\cup$*B*$\subseteq$*carrier M* **by** *auto*
from *1* **have** *2*: *submodule R* (*span* (*A*$\cup$*B*)) *M* **by** (*rule span-is-submodule*)
from *s0 s1* **have** *3*: *submodule R ?AplusB M* **by** (*rule sum-is-submodule*)
have *c1*: *span* (*A*$\cup$*B*) $\subseteq$ *?AplusB*

proof –

```
    from h2 have a1: A⊆span A by (rule in-own-span)
    from s0 s1 have a2: span A ⊆ ?AplusB by (rule in-sum)
    from a1 a2 have a3: A⊆ ?AplusB by auto

    from h3 have b1: B⊆span B by (rule in-own-span)
   from s1 s0 have b2: span B ⊆ ?AplusB by (metis in-sum msum-comm)

    from b1 b2 have b3: B⊆ ?AplusB by auto
    from a3 b3 have 5: A∪B⊆ ?AplusB by auto

    from 5 3 show ?thesis by (rule span-is-subset)
  qed
  have c2: ?AplusB ⊆ span (A∪B)
  proof −
    have 11: A⊆A∪B by auto
    have 12: B⊆A∪B by auto
   from 11 have 21:span A ⊆span (A∪B) by (rule span-is-monotone)
   from 12 have 22:span B ⊆span (A∪B) by (rule span-is-monotone)
   from s0 s1 2 21 22 show ?thesis by (auto simp add: sum-is-minimal)
  qed
  from c1 c2 show ?thesis by auto
qed

end
```

6 Basic theory of vector spaces, using locales

```
theory VectorSpace
imports Main
  HOL−Algebra.Module
  HOL−Algebra.Coset
  RingModuleFacts
  MonoidSums
  LinearCombinations
  SumSpaces
begin
```

6.1 Basic definitions and facts carried over from modules

A *vectorspace* is a module where the ring is a field. Note that we switch notation from (R, M) to (K, V).

```
locale vectorspace =
  module?: module K V + field?: field K
  for K and V
```

A *subspace* of a vectorspace is a nonempty subset that is closed under addition and scalar multiplication. These properties have already been defined in submodule. Caution: W is a set, while V is a module record. To get W as a vectorspace, write vs W.

```
locale subspace =
  fixes K and W and V (structure)
  assumes vs: vectorspace K V
     and submod: submodule K W V


lemma (in vectorspace) is-module[simp]:
  subspace K W V⟹submodule K W V
by (unfold subspace-def, auto)
```

We introduce some basic facts and definitions copied from module. We introduce some abbreviations, to match convention.

```
abbreviation (in vectorspace) vs::'c set ⇒ ('a, 'c, 'd) module-scheme
  where vs W ≡ V(|carrier := W|)

lemma (in vectorspace) carrier-vs-is-self [simp]:
  carrier (vs W) = W
  by auto

lemma (in vectorspace) subspace-is-vs:
  fixes W::'c set
  assumes 0: subspace K W V
  shows vectorspace K (vs W)
proof −
  from 0 show ?thesis
    apply (unfold vectorspace-def subspace-def, auto)
    by (intro submodule-is-module, auto)
qed

abbreviation (in module) subspace-sum:: ['c set, 'c set] ⇒ 'c set
  where subspace-sum W1 W2 ≡submodule-sum W1 W2

lemma (in vectorspace) vs-zero-lin-dep:
  assumes h2: S⊆carrier V and h3: lin-indpt S
  shows 0_V ∉ S
proof −
  have vs: vectorspace K V..
  from vs have nonzero: carrier K ≠{0_K}
    by (metis one-zeroI zero-not-one)
  from h2 h3 nonzero show ?thesis by (rule zero-nin-lin-indpt)
qed
```

A *linear-map* is a module homomorphism between 2 vectorspaces over the same field.

```
locale linear-map =
```

V?: *vectorspace K V* + *W?*: *vectorspace K W*
+ *mod-hom?*: *mod-hom K V W T*
for *K* **and** *V* **and** *W* **and** *T*

context *linear-map*
begin
lemmas *T-hom* = *f-hom*
lemmas *T-add* = *f-add*
lemmas *T-smult* = *f-smult*
lemmas *T-im* = *f-im*
lemmas *T-neg* = *f-neg*
lemmas *T-minus* = *f-minus*
lemmas *T-ker* = *f-ker*

abbreviation *imT*:: *'e set*
where *imT* ≡ *mod-hom.im*

abbreviation *kerT*:: *'c set*
where *kerT* ≡ *mod-hom.ker*

lemmas *T0-is-0*[*simp*] = *f0-is-0*

lemma *kerT-is-subspace*: *subspace K ker V*
proof −
have *vs*: *vectorspace K V*..
from *vs* **show** *?thesis*
apply (*unfold subspace-def*, *auto*)
by (*rule ker-is-submodule*)
qed

lemma *imT-is-subspace*: *subspace K imT W*
proof −
have *vs*: *vectorspace K W*..
from *vs* **show** *?thesis*
apply (*unfold subspace-def*, *auto*)
by (*rule im-is-submodule*)
qed
end

lemma *vs-criteria*:
fixes *K* **and** *V*
assumes *field*: *field K*
and *zero*: $\mathbf{0}_V$∈ *carrier V*
and *add*: ∀ *v w*. *v*∈*carrier V* ∧ *w*∈*carrier V* ⟶ *v*$\oplus_V$ *w*∈ *carrier V*
and *neg*: ∀ *v*∈*carrier V*. (∃ *neg-v*∈*carrier V*. *v*$\oplus_V$*neg-v*=$\mathbf{0}_V$)
and *smult*: ∀ *c v*. *c*∈ *carrier K* ∧ *v*∈*carrier V* ⟶ *c*$\odot_V$ *v* ∈ *carrier V*
and *comm*: ∀ *v w*. *v*∈*carrier V* ∧ *w*∈*carrier V* ⟶ *v*$\oplus_V$ *w*=*w*$\oplus_V$

v
and *assoc*: ∀ *v w x*. *v*∈*carrier V* ∧ *w*∈*carrier V* ∧ *x*∈*carrier V* ⟶ (*v*⊕$_V$ *w*)⊕$_V$ *x*= *v*⊕$_V$ (*w*⊕$_V$ *x*)
and *add-id*: ∀ *v*∈*carrier V*. (*v*⊕$_V$ **0**$_V$ =*v*)
and *compat*: ∀ *a b v*. *a*∈ *carrier K* ∧ *b*∈ *carrier K* ∧ *v*∈*carrier V* ⟶ (*a*⊗$_K$ *b*)⊙$_V$ *v* =*a*⊙$_V$ (*b*⊙$_V$ *v*)
and *smult-id*: ∀ *v*∈*carrier V*. (**1**$_K$ ⊙$_V$ *v* =*v*)
and *dist-f*: ∀ *a b v*. *a*∈ *carrier K* ∧ *b*∈ *carrier K* ∧ *v*∈*carrier V* ⟶ (*a*⊕$_K$ *b*)⊙$_V$ *v* =(*a*⊙$_V$ *v*) ⊕$_V$ (*b*⊙$_V$ *v*)
and *dist-add*: ∀ *a v w*. *a*∈ *carrier K* ∧ *v*∈*carrier V* ∧ *w*∈*carrier V* ⟶ *a*⊙$_V$ (*v*⊕$_V$ *w*) =(*a*⊙$_V$ *v*) ⊕$_V$ (*a*⊙$_V$ *w*)
shows *vectorspace K V*
proof −
from *field* **have** *1*: *cring K* **by** (*unfold field-def domain-def*, *auto*)
from *assms 1* **have** *2*: *module K V* **by** (*intro module-criteria*, *auto*)
from *field 2* **show** *?thesis* **by** (*unfold vectorspace-def module-def*, *auto*)
qed

For any set S, the space of functions $S \to K$ forms a vector space.

lemma (**in** *vectorspace*) *func-space-is-vs*:
fixes *S*
shows *vectorspace K* (*func-space S*)
proof −
have *0*: *field K*..
have *1*: *module K* (*func-space S*) **by** (*rule func-space-is-module*)
from *0 1* **show** *?thesis* **by** (*unfold vectorspace-def module-def*, *auto*)
qed

lemma *direct-sum-is-vs*:
fixes *K V1 V2*
assumes *h1*: *vectorspace K V1* **and** *h2*: *vectorspace K V2*
shows *vectorspace K* (*direct-sum V1 V2*)
proof −
from *h1 h2* **have** *mod*: *module K* (*direct-sum V1 V2*) **by** (*unfold vectorspace-def*, *intro direct-sum-is-module*, *auto*)
from *mod h1* **show** *?thesis* **by** (*unfold vectorspace-def*, *auto*)
qed

lemma *inj1-linear*:
fixes *K V1 V2*
assumes *h1*: *vectorspace K V1* **and** *h2*: *vectorspace K V2*
shows *linear-map K V1* (*direct-sum V1 V2*) (*inj1 V1 V2*)
proof −
from *h1 h2* **have** *mod*: *mod-hom K V1* (*direct-sum V1 V2*) (*inj1 V1 V2*) **by** (*unfold vectorspace-def*, *intro inj1-hom*, *auto*)
from *mod h1 h2* **show** *?thesis*

by (*unfold linear-map-def vectorspace-def* , *auto*, *intro direct-sum-is-module*, *auto*)
qed

lemma *inj2-linear*:
fixes *K V1 V2*
assumes *h1*: *vectorspace K V1* **and** *h2*: *vectorspace K V2*
shows *linear-map K V2* (*direct-sum V1 V2*) (*inj2 V1 V2*)
proof −
from *h1 h2* **have** *mod*: *mod-hom K V2* (*direct-sum V1 V2*) (*inj2 V1 V2*) **by** (*unfold vectorspace-def*, *intro inj2-hom*, *auto*)
from *mod h1 h2* **show** *?thesis*
by (*unfold linear-map-def vectorspace-def* , *auto*, *intro direct-sum-is-module*, *auto*)
qed

For subspaces $V_1, V_2 \subseteq V$, the map $V_1 \oplus V_2 \to V$ given by $(v_1, v_2) \mapsto v_1 + v_2$ is linear.

lemma (**in** *vectorspace*) *sum-map-linear*:
fixes *V1 V2*
assumes *h1*: *subspace K V1 V* **and** *h2*: *subspace K V2 V*
shows *linear-map K* (*direct-sum* (*vs V1*) (*vs V2*)) *V* (λ *v*. (*fst v*) $\oplus_V$ (*snd v*))
proof −
from *h1 h2* **have** *mod*: *mod-hom K* (*direct-sum* (*vs V1*) (*vs V2*)) *V* (λ *v*. (*fst v*) $\oplus_V$ (*snd v*))
by (*intro sum-map-hom*, *unfold subspace-def*, *auto*)
from *mod h1 h2* **show** *?thesis*
apply (*unfold linear-map-def*, *auto*) **apply** (*intro direct-sum-is-vs subspace-is-vs*, *auto*)**..**
qed

lemma (**in** *vectorspace*) *sum-is-subspace*:
fixes *W1 W2*
assumes *h1*: *subspace K W1 V* **and** *h2*: *subspace K W2 V*
shows *subspace K* (*subspace-sum W1 W2*) *V*
proof −
from *h1 h2* **have** *mod*: *submodule K* (*submodule-sum W1 W2*) *V*
by (*intro sum-is-submodule*, *unfold subspace-def*, *auto*)
from *mod h1 h2* **show** *?thesis*
by (*unfold subspace-def*, *auto*)
qed

If $W_1, W_2 \subseteq V$ are subspaces, $W_1 \subseteq W_1 + W_2$

lemma (**in** *vectorspace*) *in-sum-vs*:
fixes *W1 W2*
assumes *h1*: *subspace K W1 V* **and** *h2*: *subspace K W2 V*
shows *W1* $\subseteq$ *subspace-sum W1 W2*
proof −

from *h1 h2* **show** *?thesis* **by** (*intro in-sum*, *unfold subspace-def*, *auto*)
qed

lemma (**in** *vectorspace*) *vsum-comm*:
fixes *W1 W2*
assumes *h1*: *subspace K W1 V* **and** *h2*: *subspace K W2 V*
shows (*subspace-sum W1 W2*) = (*subspace-sum W2 W1*)
proof –
from *h1 h2* **show** *?thesis* **by** (*intro msum-comm*, *unfold subspace-def*, *auto*)
qed

If $W_1, W_2 \subseteq V$ are subspaces, then $W_1 + W_2$ is the minimal subspace such that both $W_1 \subseteq W$ and $W_2 \subseteq W$.

lemma (**in** *vectorspace*) *vsum-is-minimal*:
fixes *W W1 W2*
assumes *h1*: *subspace K W1 V* **and** *h2*: *subspace K W2 V* **and** *h3*: *subspace K W V*
shows (*subspace-sum W1 W2*) ⊆ *W* ⟷ *W1* ⊆ *W* ∧ *W2* ⊆ *W*
proof –
from *h1 h2 h3* **show** *?thesis* **by** (*intro sum-is-minimal*, *unfold subspace-def*, *auto*)
qed

lemma (**in** *vectorspace*) *span-is-subspace*:
fixes *S*
assumes *h2*: *S*⊆*carrier V*
shows *subspace K* (*span S*) *V*
proof –
have *0*: *vectorspace K V*..
from *h2* **have** *1*: *submodule K* (*span S*) *V* **by** (*rule span-is-submodule*)
from *0 1* **show** *?thesis* **by** (*unfold subspace-def mod-hom-def linear-map-def*, *auto*)
qed

6.1.1 Facts specific to vector spaces

If $av = w$ and $a \neq 0$, $v = a^{-1}w$.

lemma (**in** *vectorspace*) *mult-inverse*:
assumes *h1*: *a*∈*carrier K* **and** *h2*: *v*∈*carrier V* **and** *h3*: $a \odot_V v = w$ **and** *h4*: $a \neq \mathbf{0}_K$
shows $v = (inv_K\ a) \odot_V w$
proof –
from *h1 h2 h3* **have** *1*: *w*∈*carrier V* **by** *auto*
from *h3 1* **have** *2*: $(inv_K\ a) \odot_V (a \odot_V v) = (inv_K\ a) \odot_V w$ **by** *auto*
from *h1 h4* **have** *3*: $inv_K\ a$∈*carrier K* **by** *auto*
interpret *g*: *group* (*units-group K*) **by** (*rule units-form-group*)

have *f*: *field K*..
from *f h1 h4* **have** *4*: *a*∈*Units K*
by (*unfold field-def field-axioms-def*, *simp*)
from *4 h1 h4* **have** *5*: $((inv_K\ a) \otimes_K a) = \mathbf{1}_K$
by (*intro Units-l-inv*, *auto*)
from *5* **have** *6*: $(inv_K\ a\ ) \odot_V (a \odot_V v) = v$
proof −
from *h1 h2 h4* **have** *7*: $(inv_K\ a\ ) \odot_V (a \odot_V v) = (inv_K\ a \otimes_K a) \odot_V v$ **by** (*auto simp add*: *smult-assoc1*)
from *5 h2* **have** *8*: $(inv_K\ a \otimes_K a) \odot_V v = v$ **by** *auto*
from *7 8* **show** *?thesis* **by** *auto*
qed
from *2 6* **show** *?thesis* **by** *auto*
qed

If $w \in S$ and $\sum_{w \in S} a_w w = 0$, we have $v = \sum_{w \notin S} a_v^{-1} a_w w$

lemma (**in** *vectorspace*) *lincomb-isolate*:
fixes *A v*
assumes *h1*: *finite A* **and** *h2*: *A*⊆*carrier V* **and** *h3*: *a*∈*A*→*carrier K* **and** *h4*: *v*∈*A*
and *h5*: $a\ v \neq \mathbf{0}_K$ **and** *h6*: *lincomb a A*=$\mathbf{0}_V$
shows $v = lincomb\ (\lambda w.\ \ominus_K (inv_K\ (a\ v)) \otimes_K a\ w)\ (A - \{v\})$ **and** $v \in span\ (A - \{v\})$
proof −
from *h1 h2 h3 h4* **have** *1*: $lincomb\ a\ A = ((a\ v) \odot_V v) \oplus_V lincomb\ a\ (A - \{v\})$
by (*rule lincomb-del2*)
from *1* **have** *2*: $\mathbf{0}_V = ((a\ v) \odot_V v) \oplus_V lincomb\ a\ (A - \{v\})$ **by** (*simp add*: *h6*)
from *h1 h2 h3* **have** *5*: $lincomb\ a\ (A - \{v\}) \in carrier\ V$ **by** *auto*
from *2 h1 h2 h3 h4* **have** *3*: $\ominus_V lincomb\ a\ (A - \{v\}) = ((a\ v) \odot_V v)$
by (*auto intro*!: *M.minus-equality*)
have *6*: $v = (\ominus_K (inv_K\ (a\ v))) \odot_V lincomb\ a\ (A - \{v\})$
proof −
from *h2 h3 h4 h5 3* **have** *7*: $v = inv_K\ (a\ v) \odot_V (\ominus_V lincomb\ a\ (A - \{v\}))$
by (*intro mult-inverse*, *auto*)
from *assms* **have** *8*: $inv_K\ (a\ v) \in carrier\ K$ **by** *auto*
from *assms 5 8* **have** *9*: $inv_K\ (a\ v) \odot_V (\ominus_V lincomb\ a\ (A - \{v\}))$
$= (\ominus_K (inv_K\ (a\ v))) \odot_V lincomb\ a\ (A - \{v\})$
by (*simp add*: *smult-assoc-simp smult-minus-1-back r-minus*)
from *7 9* **show** *?thesis* **by** *auto*
qed
from *h1* **have** *10*: *finite* $(A - \{v\})$ **by** *auto*
from *assms* **have** *11* : $(\ominus_K (inv_K\ (a\ v))) \in carrier\ K$ **by** *auto*
from *assms* **have** *12*: $lincomb\ (\lambda w.\ \ominus_K (inv_K\ (a\ v)) \otimes_K a\ w)\ (A - \{v\}) =$
$(\ominus_K (inv_K\ (a\ v))) \odot_V lincomb\ a\ (A - \{v\})$

by (*intro lincomb-smult*, *auto*)
from *6 12* **show** v=*lincomb* (λw. $\ominus_K$(*inv*$_K$ (*a v*)) $\otimes_K$ *a w*) ($A-\{v\}$) **by** *auto*
with *assms* **show** $v\in$ *span* ($A-\{v\}$)
unfolding *span-def*
by (*force simp add*: *11 ring-subset-carrier*)
qed

The map $(S \to K) \mapsto V$ given by $(a_v)_{v\in S} \mapsto \sum_{v\in S} a_v v$ is linear.

lemma (**in** *vectorspace*) *lincomb-is-linear*:
fixes *S*
assumes *h*: *finite S* **and** *h2*: $S\subseteq$*carrier V*
shows *linear-map K* (*func-space S*) *V* (λa. *lincomb a S*)
proof −
have *0*: *vectorspace K V*..
from *h h2* **have** *1*: *mod-hom K* (*func-space S*) *V* (λa. *lincomb a S*) **by** (*rule lincomb-is-mod-hom*)
from *0 1* **show** *?thesis* **by** (*unfold vectorspace-def mod-hom-def linear-map-def*, *auto*)
qed

6.2 Basic facts about span and linear independence

If S is linearly independent, then $v \in \mathrm{span}S$ iff $S \cup \{v\}$ is linearly dependent.

theorem (**in** *vectorspace*) *lin-dep-iff-in-span*:
fixes *A v S*
assumes *h1*: $S \subseteq$ *carrier V* **and** *h2*: *lin-indpt S* **and** *h3*: $v\in$ *carrier V* **and** *h4*: $v\notin S$
shows $v\in$ *span S* $\longleftrightarrow$ *lin-dep* ($S \cup \{v\}$)
proof −
let *?T* = $S \cup \{v\}$
have *0*: $v\in$*?T* **by** *auto*
from *h1 h3* **have** *h1-1*: *?T* $\subseteq$ *carrier V* **by** *auto*
have *a1*:*lin-dep ?T* $\Longrightarrow$ $v\in$ *span S*
proof −
assume *a11*: *lin-dep ?T*
from *a11* **obtain** *a w A* **where** *a*: (*finite A* $\wedge$ $A\subseteq$*?T* $\wedge$ ($a\in$ ($A\rightarrow$*carrier K*)) $\wedge$ (*lincomb a A* = $\mathbf{0}_V$) $\wedge$ ($w\in A$) $\wedge$ (*a w*$\neq$ $\mathbf{0}_K$))
by (*metis lin-dep-def*)
from *assms a* **have** *nz2*: $\exists\, v\in A-S$. *a v*$\neq\mathbf{0}_K$
by (*intro lincomb-must-include*[**where** *?v*=*w* **and** *?T*=$S\cup\{v\}$], *auto*)
from *a nz2* **have** *singleton*: $\{v\}$=$A-S$ **by** *auto*
from *singleton nz2* **have** *nz3*: *a v*$\neq\mathbf{0}_K$ **by** *auto*

let *?b*=(λw. $\ominus_K$ (*inv*$_K$ (*a v*)) $\otimes_K$ (*a w*))
from *singleton* **have** *Ains*: ($A\cap S$) = $A-\{v\}$ **by** *auto*

from *assms a singleton nz3* **have** *a31*: *v= lincomb ?b* (*A*∩*S*)
apply (*subst Ains*)
by (*intro lincomb-isolate*(*1*), *auto*)
from *a a31 nz3 singleton* **show** *?thesis*
apply (*unfold span-def*, *auto*)
apply (*rule-tac x=?b* **in** *exI*)
apply (*rule-tac x=A*∩*S* **in** *exI*)
by (*auto intro*!: *m-closed*)
qed
have *a2*: *v*∈ (*span S*) ⟹ *lin-dep ?T*
proof −
assume *inspan*: *v*∈ (*span S*)
from *inspan* **obtain** *a A* **where** *a*: *A*⊆*S* ∧ *finite A* ∧ (*v* = *lincomb a A*)∧ *a*∈*A*→*carrier K* **by** (*simp add*: *span-def*, *auto*)
let *?b* = λ *w*. *if* (*w=v*) *then* ($\ominus_K$ $\mathbf{1}_K$) *else a w*
have *lc0*: *lincomb ?b* (*A*∪{*v*})=$\mathbf{0}_V$
proof −
from *assms a* **have** *lc-ins*: *lincomb ?b* (*A*∪{*v*}) = ((*?b v*) $\odot_V$ *v*) $\oplus_V$ *lincomb ?b A*
by (*intro lincomb-insert*, *auto*)
from *assms a* **have** *lc-elim*: *lincomb ?b A=lincomb a A* **by** (*intro lincomb-elim-if*, *auto*)
from *assms lc-ins lc-elim a* **show** *?thesis* **by** (*simp add*: *M.l-neg smult-minus-1*)
qed
from *a lc0* **show** *?thesis*
apply (*unfold lin-dep-def*)
apply (*rule-tac x=A*∪{*v*} **in** *exI*)
apply (*rule-tac x=?b* **in** *exI*)
apply (*rule-tac x=v* **in** *exI*)
by *auto*
qed
from *a1 a2* **show** *?thesis* **by** *auto*
qed

If $v \in \text{span}A$ then $\text{span}A = \text{span}(A \cup \{v\})$

lemma (**in** *vectorspace*) *already-in-span*:
fixes *v A*
assumes *inC*: *A*⊆*carrier V* **and** *inspan*: *v*∈*span A*
shows *span A*= *span* (*A*∪{*v*})
proof −
from *inC inspan* **have** *dir1*: *span A* ⊆ *span* (*A*∪{*v*}) **by** (*intro span-is-monotone*, *auto*)

from *inC* **have** *inown*: *A*⊆*span A* **by** (*rule in-own-span*)
from *inC* **have** *subm*: *submodule K* (*span A*) *V* **by** (*rule span-is-submodule*)
from *inown inspan subm* **have** *dir2*: *span* (*A* ∪ {*v*}) ⊆ *span A* **by** (*intro span-is-subset*, *auto*)

  from *dir1 dir2* **show** *?thesis* **by** *auto*
qed

6.3 The Replacement Theorem

If $A, B \subseteq V$ are finite, A is linearly independent, B generates W, and $A \subseteq W$, then there exists $C \subseteq V$ disjoint from A such that $\text{span}(A \cup C) = W$ and $|C| \leq |B| - |A|$. In other words, we can complete any linearly independent set to a generating set of W by adding at most $|B| - |A|$ more elements.

theorem (**in** *vectorspace*) *replacement*:
  fixes *A B*
  assumes *h1*: *finite A*
      and *h2*: *finite B*
      and *h3*: *B⊆carrier V*
      and *h4*: *lin-indpt A*
      and *h5*: *A⊆span B*
  shows ∃ *C*. *finite C* ∧ *C⊆carrier V* ∧ *C⊆span B* ∧ *C∩A={}* ∧ *int* (*card C*) ≤ (*int* (*card B*)) − (*int* (*card A*)) ∧ (*span* (*A* ∪ *C*) = *span B*)
  (**is** ∃ *C*. *?P A B C*)

using *h1 h2 h3 h4 h5*
proof (*induct card A arbitrary*: *A B*)
  case *0*
  from *0.prems*(*1*) *0.hyps* **have** *a0*: *A={}* **by** *auto*
  from *0.prems*(*3*) **have** *a3*: *B⊆span B* **by** (*rule in-own-span*)
  from *a0 a3 0.prems* **show** *?case* **by** (*rule-tac x=B* **in** *exI*, *auto*)
next
  case (*Suc m*)
  let *?W=span B*
  from *Suc.prems*(*3*) **have** *BinC*: *span B⊆carrier V* **by** (*rule span-is-subset2*)

  from *Suc.prems Suc.hyps BinC* **have** *A*: *finite A lin-indpt A A⊆span B Suc m = card A A⊆carrier V*
    by *auto*

  from *Suc.prems* **have** *B*: *finite B B⊆carrier V* **by** *auto*

  from *Suc.hyps*(*2*) **obtain** *v* **where** *v*: *v∈A* **by** *fastforce*
  let *?A′=A−{v}*

  from *A*(*2*) **have** *liA′*: *lin-indpt ?A′*
    apply (*intro subset-li-is-li*[*of A ?A′*])
     by *auto*
  from *v liA′ Suc.prems Suc.hyps*(*2*) **have** ∃ *C′*. *?P ?A′ B C′*
    apply (*intro Suc.hyps*(*1*))
        by *auto*

from *this* **obtain** C' **where** C': *?P ?A' B C'* **by** *auto*

show *?case*
proof (*cases* $v\in C'$)
case *True*
have *vinC'*: $v\in C'$ **by** *fact*
from *vinC' v* **have** *seteq*: $A - \{v\} \cup C' = A \cup (C' - \{v\})$ **by** *auto*
from C' *seteq* **have** *spaneq*: *span* $(A \cup (C' - \{v\}))$ = *span* (B) **by** *algebra*
from *Suc.prems Suc.hyps C' vinC' v spaneq* **show** *?thesis*
apply (*rule-tac* $x=C'-\{v\}$ **in** *exI*)
apply (*subgoal-tac card* $C' > 0$)
by *auto*
next
case *False*
have *f*: $v\notin C'$ **by** *fact*
from *A v C'* **have** $\exists a.\ a\in(?A'\cup C')\rightarrow$*carrier K* $\wedge$ *lincomb a* $(?A' \cup C') = v$
by (*intro finite-in-span*, *auto*)
from *this* **obtain** *a* **where** *a*: $a\in(?A'\cup C')\rightarrow$*carrier K* $\wedge$ $v=$ *lincomb a* $(?A' \cup C')$ **by** *metis*
let $?b=(\lambda\ w.$ *if* $(w=v)$ *then* $\ominus_K \mathbf{1}_K$ *else a w*)
from *a* **have** *b*: $?b\in A\cup C'\rightarrow$*carrier K* **by** *auto*
from *v* **have** *rewrite-ins*: $A\cup C'=(?A'\cup C')\cup\{v\}$ **by** *auto*
from *f* **have** $v\notin ?A'\cup C'$ **by** *auto*
from *this A C' v a f* **have** *lcb*: *lincomb ?b* $(A \cup C') = \mathbf{0}_V$
apply (*subst rewrite-ins*)
apply (*subst lincomb-insert*)
apply (*simp-all add*: *ring-subset-carrier coeff-in-ring*)
apply (*auto split*: *if-split-asm*)
apply (*subst lincomb-elim-if*)
by (*auto simp add*: *smult-minus-1 l-neg ring-subset-carrier*)

from C' *f* **have** *rewrite-minus*: $C'=(A\cup C')-A$ **by** *auto*
from *A C' b lcb v* **have** *exw*: $\exists w\in C'.\ ?b\ w\neq\mathbf{0}_K$
apply (*subst rewrite-minus*)
apply (*intro lincomb-must-include*[**where** $?T=A\cup C'$ **and** $?v=v$])
by *auto*
from *exw* **obtain** *w* **where** *w*: $w\in C'$ $?b\ w\neq\mathbf{0}_K$ **by** *auto*
from *A C' w f b lcb* **have** *w-in*: $w\in$*span* $((A\cup C') - \{w\})$
apply (*intro lincomb-isolate*[**where** $a=?b$])
by *auto*
have *spaneq2*: *span* $(A\cup(C'-\{w\}))$ = *span B*
proof −
have *1*: *span* $(?A'\cup C')$ = *span* $(A\cup C')$
proof −
from *A C' v* **have** *m1*: *span* $(?A'\cup C')$ = *span* $((?A'\cup C')\cup\{v\})$
apply (*intro already-in-span*)

by *auto*
from *f m1* **show** *?thesis* **by** (*metis rewrite-ins*)
qed
have *2*: *span* $(A \cup (C' - \{w\})) =$ *span* $(A \cup C')$
proof $-$
from C' *w*(*1*) *f* **have** *b60*: $A \cup (C' - \{w\}) = (A \cup C') - \{w\}$ **by** *auto*
from *w*(*1*) **have** *b61*: $A \cup C' = (A \cup C' - \{w\}) \cup \{w\}$ **by** *auto*
from *A* C' *w-in* **show** *?thesis*
apply (*subst b61*)
apply (*subst b60*)
apply (*intro already-in-span*)
by *auto*
qed
from C' *1 2* **show** *?thesis* **by** *auto*
qed
from *A* C' *w f v spaneq2* **show** *?thesis*
apply (*rule-tac* $x = C' - \{w\}$ **in** *exI*)
apply (*subgoal-tac card* $C' > 0$)
by *auto*
qed
qed

6.4 Defining dimension and bases.

Finite dimensional is defined as having a finite generating set.

definition (**in** *vectorspace*) *fin-dim*:: *bool*
where *fin-dim* = ($\exists$ *A*. ((*finite A*) $\wedge$ ($A \subseteq$ *carrier V*) $\wedge$ (*gen-set A*)))

The dimension is the size of the smallest generating set. For equivalent characterizations see below.

definition (**in** *vectorspace*) *dim*:: *nat*
where *dim* = (*LEAST n*. ($\exists$ *A*. ((*finite A*) $\wedge$ (*card A* = *n*) $\wedge$ ($A \subseteq$ *carrier V*) $\wedge$ (*gen-set A*))))

A *basis* is a linearly independent generating set.

definition (**in** *vectorspace*) *basis*:: *'c set* $\Rightarrow$ *bool*
where *basis A* = ((*lin-indpt A*) $\wedge$ (*gen-set A*) $\wedge$ ($A \subseteq$ *carrier V*))

From the replacement theorem, any linearly independent set is smaller than any generating set.

lemma (**in** *vectorspace*) *li-smaller-than-gen*:
fixes *A B*
assumes *h1*: *finite A* **and** *h2*: *finite B* **and** *h3*: $A \subseteq$ *carrier V* **and** *h4*: $B \subseteq$ *carrier V*
and *h5*: *lin-indpt A* **and** *h6*: *gen-set B*
shows *card A* $\leq$ *card B*

proof −
  from *h3 h6* **have** *1*: $A \subseteq span\ B$ **by** *auto*
  from *h1 h2 h4 h5 1* **obtain** *C* **where**
    2: $finite\ C \wedge C \subseteq carrier\ V \wedge C \subseteq span\ B \wedge C \cap A = \{\} \wedge int\ (card\ C) \leq int\ (card\ B) - int\ (card\ A) \wedge (span\ (A \cup C) = span\ B)$
    by (*metis replacement*)
  from *2* **show** *?thesis* **by** *arith*
qed

The dimension is the cardinality of any basis. (In particular, all bases are the same size.)

lemma (**in** *vectorspace*) *dim-basis*:
  fixes *A*
  assumes *fin*: *finite A* **and** *h2*: *basis A*
  shows $dim = card\ A$
proof −
  have *0*: $\bigwedge B\ m.\ ((finite\ B) \wedge (card\ B = m) \wedge (B \subseteq carrier\ V) \wedge (gen\text{-}set\ B)) \Longrightarrow card\ A \leq m$
  proof −
    fix *B m*
    assume *1*: $((finite\ B) \wedge (card\ B = m) \wedge (B \subseteq carrier\ V) \wedge (gen\text{-}set\ B))$
    from *1 fin h2* **have** *2*: $card\ A \leq card\ B$
      apply (*unfold basis-def*)
      apply (*intro li-smaller-than-gen*)
        by *auto*
    from *1 2* **show** *?thesis B m* **by** *auto*
  qed
  from *fin h2 0* **show** *?thesis*
    apply (*unfold dim-def basis-def*)
    apply (*intro Least-equality*)
     apply (*rule-tac* $x=A$ **in** *exI*)
     by *auto*
qed

A *maximal* set with respect to P is such that if $B \supseteq A$ and P is also satisfied for B, then $B = A$.

definition $maximal::'a\ set \Rightarrow ('a\ set \Rightarrow bool) \Rightarrow bool$
  where $maximal\ A\ P = ((P\ A) \wedge (\forall B.\ B \supseteq A \wedge P\ B \longrightarrow B = A))$

A *minimal* set with respect to P is such that if $B \subseteq A$ and P is also satisfied for B, then $B = A$.

definition $minimal::'a\ set \Rightarrow ('a\ set \Rightarrow bool) \Rightarrow bool$
  where $minimal\ A\ P = ((P\ A) \wedge (\forall B.\ B \subseteq A \wedge P\ B \longrightarrow B = A))$

A maximal linearly independent set is a generating set.

lemma (**in** *vectorspace*) *max-li-is-gen*:
  fixes *A*

assumes *h1*: *maximal A* (λS. *S*⊆*carrier V* ∧ *lin-indpt S*)
shows *gen-set A*
proof (*rule ccontr*)
assume *0*: ¬(*gen-set A*)
from *h1* **have** *1*: *A*⊆ *carrier V* ∧ *lin-indpt A* **by** (*unfold maximal-def*, *auto*)
from *1* **have** *2*: *span A* ⊆ *carrier V* **by** (*intro span-is-subset2*, *auto*)
from *0 1 2* **have** *3*: ∃ *v*. *v*∈*carrier V* ∧ *v* ∉ (*span A*)
by *auto*
from *3* **obtain** *v* **where** *4*: *v*∈*carrier V* ∧ *v* ∉ (*span A*) **by** *auto*
have *5*: *v*∉*A*
proof −
from *h1 1* **have** *51*: *A*⊆*span A* **apply** (*intro in-own-span*) **by** *auto*
from *4 51* **show** *?thesis* **by** *auto*
qed
from *lin-dep-iff-in-span* **have** *6*: ⋀*S v*. *S* ⊆ *carrier V*∧ *lin-indpt S* ∧ *v*∈ *carrier V* ∧ *v*∉*S*
∧ *v*∉ *span S* ⟹ (*lin-indpt* (*S* ∪ {*v*})) **by** *auto*
from *1 4 5* **have** *7*: *lin-indpt* (*A* ∪ {*v*}) **apply** (*intro 6*) **by** *auto*

have *9*: ¬(*maximal A* (λS. *S*⊆*carrier V* ∧ *lin-indpt S*))
proof −
from *1 4 5 7* **have** *8*: (∃ *B*. *A* ⊆ *B* ∧ *B* ⊆ *carrier V* ∧ *lin-indpt B* ∧ *B* ≠ *A*)
apply (*rule-tac x*=*A*∪{*v*} **in** *exI*)
by *auto*
from *8* **show** *?thesis*
apply (*unfold maximal-def*)
by *simp*
qed
from *h1 9* **show** *False* **by** *auto*
qed

A minimal generating set is linearly independent.

lemma (**in** *vectorspace*) *min-gen-is-li*:
fixes *A*
assumes *h1*: *minimal A* (λS. *S*⊆*carrier V* ∧ *gen-set S*)
shows *lin-indpt A*
proof (*rule ccontr*)
assume *0*: ¬*lin-indpt A*
from *h1* **have** *1*: *A*⊆ *carrier V* ∧ *gen-set A* **by** (*unfold minimal-def*, *auto*)
from *1* **have** *2*: *span A* = *carrier V* **by** *auto*
from *0 1* **obtain** *a v A'* **where**
3: *finite A'* ∧ *A'*⊆*A* ∧ *a* ∈ *A'* → *carrier K* ∧ *LinearCombinations.module.lincomb V a A'* = $\mathbf{0}_V$ ∧ *v* ∈ *A'* ∧ *a v* ≠ $\mathbf{0}_K$
by (*unfold lin-dep-def*, *auto*)
have *4*: *gen-set* (*A*−{*v*})
proof −

from *1 3* **have** *5*: *v*∈*span* (*A'*−{*v*})
apply (*intro lincomb-isolate*[**where** *a*=*a* **and** *v*=*v*])
by *auto*
from *3 5* **have** *51*: *v*∈*span* (*A*−{*v*})
apply (*intro subsetD*[**where** *?A*=*span* (*A'*−{*v*}) **and** *?B*=*span* (*A*−{*v*}) **and** *?c*=*v*])
by (*intro span-is-monotone*, *auto*)
from *1* **have** *6*: *A*⊆*span A* **apply** (*intro in-own-span*) **by** *auto*
from *1 51* **have** *7*: *span* (*A*−{*v*}) = *span* ((*A*−{*v*})∪{*v*}) **apply** (*intro already-in-span*) **by** *auto*
from *3* **have** *8*: *A* = ((*A*−{*v*})∪{*v*}) **by** *auto*
from *2 7 8* **have** *9*:*span* (*A*−{*v*}) = *carrier V* **by** *auto*
from *9* **show** *?thesis* **by** *auto*
qed
have *10*: ¬(*minimal A* (λ*S*. *S*⊆*carrier V* ∧ *gen-set S*))
proof −
from *1 3 4* **have** *11*: (∃*B*. *A* ⊇ *B* ∧ *B* ⊆ *carrier V* ∧ *gen-set B* ∧ *B* ≠ *A*)
apply (*rule-tac x*=*A*−{*v*} **in** *exI*)
by *auto*
from *11* **show** *?thesis*
apply (*unfold minimal-def*)
by *auto*
qed
from *h1 10* **show** *False* **by** *auto*
qed

Given that some finite set satisfies P, there is a minimal set that satisfies P.

lemma *minimal-exists*:
fixes *A P*
assumes *h1*: *finite A* **and** *h2*: *P A*
shows ∃*B*. *B*⊆*A* ∧ *minimal B P*
using *h1 h2*
proof (*induct card A arbitrary*: *A rule*: *less-induct*)
case (*less A*)
show *?case*
proof (*cases card A* = *0*)
case *True*
from *True less.hyps less.prems* **show** *?thesis*
apply (*rule-tac x*={} **in** *exI*)
apply (*unfold minimal-def*)
by *auto*
next
case *False*
show *?thesis*
proof (*cases minimal A P*)
case *True*
then show *?thesis*

apply (*rule-tac x=A* **in** *exI*)
by *auto*
next
case *False*
have *2*: ¬*minimal A P* **by** *fact*
from *less.prems 2* **have** *3*: $\exists B.\ P\ B \wedge B \subseteq A \wedge B \neq A$
apply (*unfold minimal-def*)
by *auto*
from *3* **obtain** *B* **where** *4*: $P\ B \wedge B \subset A \wedge B \neq A$ **by** *auto*
from *4* **have** *5*: *card B* < *card A* **by** (*metis less.prems*(*1*) *psubset-card-mono*)
from *less.hyps less.prems 3 4 5* **have** *6*: $\exists\ C \subseteq B.$ *minimal C P*
apply (*intro less.hyps*)
apply *auto*
by (*metis rev-finite-subset*)
from *6* **obtain** *C* **where** *7*: $C \subseteq B \wedge$ *minimal C P* **by** *auto*
from *4 7* **show** *?thesis*
apply (*rule-tac x=C* **in** *exI*)
apply (*unfold minimal-def*)
by *auto*
qed
qed
qed

If V is finite-dimensional, then any linearly independent set is finite.

lemma (**in** *vectorspace*) *fin-dim-li-fin*:
assumes *fd*: *fin-dim* **and** *li*: *lin-indpt A* **and** *inC*: $A \subseteq$ *carrier V*
shows *fin*: *finite A*
proof (*rule ccontr*)
assume *A*: ¬*finite A*
from *fd* **obtain** *C* **where** *C*: *finite C* ∧ $C \subseteq$ *carrier V* ∧ *gen-set C* **by** (*unfold fin-dim-def*, *auto*)
from *A* **obtain** *B* **where** *B*: $B \subseteq A \wedge$ *finite B* ∧ *card B* = *card C* + *1*
by (*metis infinite-arbitrarily-large*)
from *B li* **have** *liB*: *lin-indpt B*
by (*intro subset-li-is-li*[**where** *?A=A* **and** *?B=B*], *auto*)
from *B C liB inC* **have** *card B* ≤ *card C* **by** (*intro li-smaller-than-gen*, *auto*)
from *this B* **show** *False* **by** *auto*
qed

If V is finite-dimensional (has a finite generating set), then a finite basis exists.

lemma (**in** *vectorspace*) *finite-basis-exists*:
assumes *h1*: *fin-dim*
shows $\exists\beta.$ *finite* β ∧ *basis* β
proof −

from *h1* **obtain** *A* **where** *1*: *finite* $A \wedge A \subseteq$ *carrier* $V \wedge$ *gen-set* A **by** (*metis fin-dim-def*)
hence *2*: $\exists \beta.\ \beta \subseteq A \wedge$ *minimal* β $(\lambda S.\ S \subseteq$ *carrier* $V \wedge$ *gen-set* $S)$
apply (*intro minimal-exists*)
by *auto*
then obtain β **where** *3*: $\beta \subseteq A \wedge$ *minimal* β $(\lambda S.\ S \subseteq$ *carrier* $V \wedge$ *gen-set* $S)$ **by** *auto*
hence *4*: *lin-indpt* β **apply** (*intro min-gen-is-li*) **by** *auto*
moreover from *3* **have** *5*: *gen-set* $\beta \wedge \beta \subseteq$ *carrier* V **apply** (*unfold minimal-def*) **by** *auto*
moreover from *1 3* **have** *6*: *finite* β **by** (*auto simp add*: *finite-subset*)
ultimately show *?thesis* **apply** (*unfold basis-def*) **by** *auto*
qed

The proof is as follows.

1. Because V is finite-dimensional, there is a finite generating set (we took this as our definition of finite-dimensional).
2. Hence, there is a minimal $\beta \subseteq A$ such that β generates V.
3. β is linearly independent because a minimal generating set is linearly independent.

Finally, β is a basis because it is both generating and linearly independent.

Any linearly independent set has cardinality at most equal to the dimension.

lemma (**in** *vectorspace*) *li-le-dim*:
fixes *A*
assumes *fd*: *fin-dim* **and** *c*: $A \subseteq$ *carrier* V **and** *l*: *lin-indpt* A
shows *finite* A *card* $A \leq$ *dim*
proof −
from *fd c l* **show** *fa*: *finite* A **by** (*intro fin-dim-li-fin*, *auto*)
from *fd* **obtain** β **where** *1*: *finite* $\beta \wedge$ *basis* β
by (*metis finite-basis-exists*)
from *assms fa 1* **have** *2*: *card* $A \leq$ *card* β
apply (*intro li-smaller-than-gen*, *auto*)
by (*unfold basis-def*, *auto*)
from *assms 1* **have** *3*: *dim* $=$ *card* β **by** (*intro dim-basis*, *auto*)
from *2 3* **show** *card* $A \leq$ *dim* **by** *auto*
qed

Any generating set has cardinality at least equal to the dimension.

lemma (**in** *vectorspace*) *gen-ge-dim*:
fixes *A*
assumes *fa*: *finite* A **and** *c*: $A \subseteq$ *carrier* V **and** *l*: *gen-set* A
shows *card* $A \geq$ *dim*

proof −
  from *assms* **have** *fd*: *fin-dim* **by** (*unfold fin-dim-def*, *auto*)
  from *fd* **obtain** β **where** *1*: *finite* β $\wedge$ *basis* β **by** (*metis finite-basis-exists*)
  from *assms 1* **have** *2*: *card A* $\geq$ *card* β
    apply (*intro li-smaller-than-gen*, *auto*)
     by (*unfold basis-def*, *auto*)
  from *assms 1* **have** *3*: *dim* = *card* β **by** (*intro dim-basis*, *auto*)
  from *2 3* **show** *?thesis* **by** *auto*
qed

If there is an upper bound on the cardinality of sets satisfying P, then there is a maximal set satisfying P.

lemma *maximal-exists*:
  fixes *P B N*
  assumes *maxc*: $\bigwedge$*A*. *P A* $\Longrightarrow$ *finite A* $\wedge$ (*card A* $\leq$*N*) **and** *b*: *P B*
  shows $\exists$ *A*. *finite A* $\wedge$ *maximal A P*
proof −

  let *?S*={*card A*| *A*. *P A*}
  let *?n*=*Max ?S*
  from *maxc* **have** *1*:*finite ?S*
    apply (*simp add*: *finite-nat-set-iff-bounded-le*) **by** *auto*
  from *1* **have** *2*: *?n*$\in$*?S*
    by (*metis* (*mono-tags*, *lifting*) *Collect-empty-eq Max-in b*)
  from *assms 2* **have** *3*: $\exists$ *A*. *P A* $\wedge$ *finite A* $\wedge$ *card A* = *?n*
    by *auto*
  from *3* **obtain** *A* **where** *4*: *P A* $\wedge$ *finite A* $\wedge$ *card A* = *?n* **by** *auto*
  from *1 maxc* **have** *5*: $\bigwedge$*A*. *P A* $\Longrightarrow$ *finite A* $\wedge$ (*card A* $\leq$*?n*)
    by (*metis* (*mono-tags*, *lifting*) *Max.coboundedI mem-Collect-eq*)
  from *4 5* **have** *6*: *maximal A P*
    apply (*unfold maximal-def*)
    by (*metis card-seteq*)
  from *4 6* **show** *?thesis* **by** *auto*
qed

Any maximal linearly independent set is a basis.

lemma (**in** *vectorspace*) *max-li-is-basis*:
  fixes *A*
  assumes *h1*: *maximal A* (λ*S*. *S*$\subseteq$*carrier V* $\wedge$ *lin-indpt S*)
  shows *basis A*
proof −
  from *h1* **have** *1*: *gen-set A* **by** (*rule max-li-is-gen*)
  from *assms 1* **show** *?thesis* **by** (*unfold basis-def maximal-def*, *auto*)
qed

Any minimal linearly independent set is a generating set.

lemma (**in** *vectorspace*) *min-gen-is-basis*:
  fixes *A*
  assumes *h1*: *minimal A* (λ*S*. *S*$\subseteq$*carrier V* $\wedge$ *gen-set S*)

shows *basis A*
proof −
from *h1* **have** *1*: *lin-indpt A* **by** (*rule min-gen-is-li*)
from *assms 1* **show** *?thesis* **by** (*unfold basis-def minimal-def*, *auto*)
qed

Any linearly independent set with cardinality at least the dimension is a basis.

lemma (**in** *vectorspace*) *dim-li-is-basis*:
fixes *A*
assumes *fd*: *fin-dim* **and** *fa*: *finite A* **and** *ca*: $A \subseteq$ *carrier V* **and** *li*: *lin-indpt A*
and *d*: *card A* $\geq$ *dim*
shows *basis A*
proof −
from *fd* **have** *0*: $\bigwedge S$. $S \subseteq$ *carrier V* $\wedge$ *lin-indpt S* $\Longrightarrow$ *finite S* $\wedge$ *card S* $\leq$ *dim*
by (*auto intro*: *li-le-dim*)

from *0 assms* **have** *h1*: *finite A* $\wedge$ *maximal A* (λS. $S \subseteq$ *carrier V* $\wedge$ *lin-indpt S*)
apply (*unfold maximal-def*)
apply *auto*
by (*metis card-seteq eq-iff*)
from *h1* **show** *?thesis* **by** (*auto intro*: *max-li-is-basis*)
qed

Any generating set with cardinality at most the dimension is a basis.

lemma (**in** *vectorspace*) *dim-gen-is-basis*:
fixes *A*
assumes *fa*: *finite A* **and** *ca*: $A \subseteq$ *carrier V* **and** *li*: *gen-set A*
and *d*: *card A* $\leq$ *dim*
shows *basis A*
proof −
have *0*: $\bigwedge S$. *finite S* $\wedge$ $S \subseteq$ *carrier V* $\wedge$ *gen-set S* $\Longrightarrow$ *card S* $\geq$ *dim*
by (*intro gen-ge-dim*, *auto*)

from *0 assms* **have** *h1*: *minimal A* (λS. *finite S* $\wedge$ $S \subseteq$ *carrier V* $\wedge$ *gen-set S*)
apply (*unfold minimal-def*)
apply *auto*
by (*metis card-seteq eq-iff*)

from *h1* **have** *h*: $\bigwedge B$. $B \subseteq A \wedge B \subseteq$ *carrier V* $\wedge$ *LinearCombinations.module.gen-set K V B* $\Longrightarrow B = A$
proof −
fix *B*

assume *asm*: $B \subseteq A \wedge B \subseteq$ *carrier V* $\wedge$ *LinearCombinations.module.gen-set K V B*
from *asm h1* **have** *finite B*
apply (*unfold minimal-def*)
apply (*intro finite-subset*[**where** *?A=B* **and** *?B=A*])
by *auto*
from *h1 asm this* **show** *?thesis B* **apply** (*unfold minimal-def*) **by** *simp*
qed
from *h1 h* **have** *h2*: *minimal A* ($\lambda S.$ $S \subseteq$*carrier V* $\wedge$ *gen-set S*)
apply (*unfold minimal-def*)
by *presburger*
from *h2* **show** *?thesis* **by** (*rule min-gen-is-basis*)
qed

β is a basis iff for all $v \in V$, there exists a unique $(a_v)_{v\in S}$ such that $\sum_{v\in S} a_v v = v$.

lemma (**in** *vectorspace*) *basis-criterion*:
assumes *A-fin*: *finite A* **and** *AinC*: $A \subseteq$*carrier V*
shows *basis A* $\longleftrightarrow$ ($\forall v.$ $v\in$ *carrier V* $\longrightarrow$($\exists!$ *a*. $a \in A \rightarrow_E$ *carrier K* $\wedge$ *lincomb a A* = *v*))
proof −
have *1*: $\neg$($\forall v.$ $v\in$ *carrier V* $\longrightarrow$($\exists!$ *a*. $a \in A \rightarrow_E$ *carrier K* $\wedge$ *lincomb a A* = *v*)) $\Longrightarrow$ $\neg$*basis A*
proof −
assume $\neg$($\forall v.$ $v\in$ *carrier V* $\longrightarrow$($\exists!$ *a*. $a \in A \rightarrow_E$ *carrier K* $\wedge$ *lincomb a A* = *v*))
then obtain *v* **where** *v*: $v\in$ *carrier V* $\wedge$ $\neg$($\exists!$ *a*. $a \in A \rightarrow_E$ *carrier K* $\wedge$ *lincomb a A* = *v*) **by** *metis*

from *v* **have** *vinC*: $v\in$*carrier V* **by** *auto*
from *v* **have** $\neg$($\exists$ *a*. $a \in A \rightarrow_E$ *carrier K* $\wedge$ *lincomb a A* = *v*) $\vee$ ($\exists$ *a b*.
$a \in A \rightarrow_E$ *carrier K* $\wedge$ *lincomb a A* = *v* $\wedge$ $b \in A \rightarrow_E$ *carrier K* $\wedge$ *lincomb b A* = *v*
$\wedge$ $a \neq b$) **by** *metis*
then show *?thesis*
proof
assume *a*: $\neg$($\exists$ *a*. $a \in A \rightarrow_E$ *carrier K* $\wedge$ *lincomb a A* = *v*)
from *A-fin AinC* **have** $\bigwedge a.$ $a \in A \rightarrow$ *carrier K* $\Longrightarrow$ *lincomb a A* = *lincomb* (*restrict a A*) *A*
unfolding *lincomb-def restrict-def*
by (*simp cong*: *finsum-cong add*: *ring-subset-carrier coeff-in-ring*)
with *a* **have** $\neg$($\exists$ *a*. $a \in A \rightarrow$ *carrier K* $\wedge$ *lincomb a A* = *v*) **by** *auto*
with *A-fin AinC* **have** $v \notin$*span A*
using *finite-in-span* **by** *blast*
with *AinC v* **show** $\neg$(*basis A*) **by** (*unfold basis-def*, *auto*)
next

assume *a2*: ($\exists$ *a b*.
$a \in A \rightarrow_E$ *carrier K* $\wedge$ *lincomb a A* = *v* $\wedge$ $b \in A \rightarrow_E$ *carrier K* $\wedge$ *lincomb b A* = *v*
$\wedge$ $a \neq b$)
then obtain *a b* **where** *ab*: $a \in A \rightarrow_E$ *carrier K* $\wedge$ *lincomb a A* = *v* $\wedge$ $b \in A \rightarrow_E$ *carrier K* $\wedge$ *lincomb b A* = *v*
$\wedge$ $a \neq b$ **by** *metis*
from *ab* **obtain** *w* **where** *w*: $w \in A$ $\wedge$ *a w* $\neq$ *b w* **apply** (*unfold PiE-def*, *auto*)
by (*metis extensionalityI*)
let *?c*=λ *x*. (*if* $x \in A$ *then* ((*a x*) $\ominus_K$ (*b x*)) *else undefined*)
from *ab* **have** *a-fun*: $a \in A \rightarrow$ *carrier K*
and *b-fun*: $b \in A \rightarrow$ *carrier K*
by (*unfold PiE-def*, *auto*)
from *w a-fun b-fun* **have** *abinC*: *a w* $\in$*carrier K b w* $\in$*carrier K* **by** *auto*

from *abinC w* **have** *nz*: *a w* $\ominus_K$ *b w* $\neq$ $\mathbf{0}_K$
by *auto*
from *A-fin AinC a-fun b-fun ab vinC* **have** *a-b*:
LinearCombinations.module.lincomb V (λx. *if* $x \in A$ *then a x* $\ominus_K$ *b x else undefined*) *A* = $\mathbf{0}_V$
by (*simp cong*: *lincomb-cong add*: *coeff-in-ring lincomb-diff*)
from *A-fin AinC ab w v nz a-b* **have** *lin-dep A*
apply (*intro lin-dep-crit*[**where** *?A*=*A* **and** *?a*=*?c* **and** *?v*=*w*])
apply (*auto simp add*: *PiE-def*)
by *auto*
thus $\neg$*basis A* **by** (*unfold basis-def*, *auto*)
qed
qed
have *2*: ($\forall$ *v*. $v \in$ *carrier V* $\longrightarrow$($\exists$! *a*. $a \in A \rightarrow_E$ *carrier K* $\wedge$ *lincomb a A* = *v*)) $\Longrightarrow$ *basis A*
proof –
assume *b1*: ($\forall$ *v*. $v \in$ *carrier V* $\longrightarrow$($\exists$! *a*. $a \in A \rightarrow_E$ *carrier K* $\wedge$ *lincomb a A* = *v*))
(**is** ($\forall$ *v*. $v \in$ *carrier V* $\longrightarrow$($\exists$! *a*. *?Q a v*)))
from *b1* **have** *b2*: ($\forall$ *v*. $v \in$ *carrier V* $\longrightarrow$($\exists$ *a*. $a \in A \rightarrow$ *carrier K* $\wedge$ *lincomb a A* = *v*))
apply (*unfold PiE-def*)
by *blast*
from *A-fin AinC b2* **have** *gen-set A*
apply (*unfold span-def*)
by *blast*
from *b1* **have** *A-li*: *lin-indpt A*
proof –
let *?z*=λ *x*. (*if* ($x \in A$) *then* $\mathbf{0}_K$ *else undefined*)
from *A-fin AinC* **have** *zero*: *?Q ?z* $\mathbf{0}_V$
by (*unfold PiE-def extensional-def lincomb-def*, *auto simp add*: *ring-subset-carrier*)

```
      from A-fin AinC show ?thesis
      proof (rule finite-lin-indpt2)
        fix a
        assume a-fun: a ∈ A → carrier K and
          lc-a: LinearCombinations.module.lincomb V a A = 0_V
       from a-fun have a-res: restrict a A ∈ A →_E carrier K by auto
        from a-fun A-fin AinC lc-a have
          lc-a-res: LinearCombinations.module.lincomb V (restrict a A)
A = 0_V
          apply (unfold lincomb-def restrict-def)
       by (simp cong: finsum-cong2 add: coeff-in-ring ring-subset-carrier)
         from a-fun a-res lc-a-res zero b1 have restrict a A = ?z by
auto
        from this show ∀ v∈A. a v = 0_K
          apply (unfold restrict-def)
          by meson
      qed
    qed
    have A-gen: gen-set A
    proof −
    from AinC have dir1: span A⊆carrier V by (rule span-is-subset2)
      have dir2: carrier V⊆span A
      proof (auto)
        fix v
        assume v: v∈carrier V
         from v b2 obtain a where a∈A → carrier K ∧ lincomb a A
= v by auto
         from this A-fin AinC show v∈span A by (subst finite-span,
auto)
      qed
      from dir1 dir2 show ?thesis by auto
    qed
    from A-li A-gen AinC show basis A by (unfold basis-def, auto)
  qed
  from 1 2 show ?thesis by satx
qed

lemma (in linear-map) surj-imp-imT-carrier:
  assumes surj: T‘ (carrier V) = carrier W
  shows (imT) = carrier W
  by (simp add: surj im-def)
```

6.5 The rank-nullity (dimension) theorem

If V is finite-dimensional and $T : V \to W$ is a linear map, then $\dim(\mathrm{im}(T)) + \dim(\ker(T)) = \dim V$. Moreover, we prove that if T is surjective linear-map between V and W, where V is finite-dimensional, then also W is finite-dimensional.

theorem (**in** *linear-map*) *rank-nullity-main*:
assumes *fd*: *V.fin-dim*
shows (*vectorspace.dim K* (*W.vs imT*)) + (*vectorspace.dim K* (*V.vs kerT*)) = *V.dim*
T ‘ (*carrier V*) = *carrier W* $\Longrightarrow$ *W.fin-dim*
proof −
— First interpret kerT, imT as vectorspaces
have *subs-ker*: *subspace K kerT V* **by** (*intro kerT-is-subspace*)
from *subs-ker* **have** *vs-ker*: *vectorspace K* (*V.vs kerT*) **by** (*rule V.subspace-is-vs*)
from *vs-ker* **interpret** *ker*: *vectorspace K* (*V.vs kerT*) **by** *auto*
have *kerInC*: *kerT*$\subseteq$*carrier V* **by** (*unfold ker-def*, *auto*)

have *subs-im*: *subspace K imT W* **by** (*intro imT-is-subspace*)
from *subs-im* **have** *vs-im*: *vectorspace K* (*W.vs imT*) **by** (*rule W.subspace-is-vs*)
from *vs-im* **interpret** *im*: *vectorspace K* (*W.vs imT*) **by** *auto*
have *imInC*: *imT*$\subseteq$*carrier W* **by** (*unfold im-def*, *auto*)

have *zero-same*[*simp*]: $\mathbf{0}_{V.vs\ kerT} = \mathbf{0}_V$ **apply** (*unfold ker-def*) **by** *auto*
— Show ker T has a finite basis. This is not obvious. Show that any linearly independent set has size at most that of V. There exists a maximal linearly independent set, which is the basis.
have *every-li-small*: $\bigwedge A$. ($A \subseteq$ *kerT*)$\wedge$ *ker.lin-indpt A* $\Longrightarrow$
finite A $\wedge$ *card A* $\leq$ *V.dim*
proof −
fix *A*
assume *eli-asm*: ($A \subseteq$ *kerT*)$\wedge$ *ker.lin-indpt A*

note *V.module.span-li-not-depend*(*2*)[**where** *?N*=*kerT* **and** *?S*=*A*]

from *this subs-ker fd eli-asm kerInC* **show** *?thesis A*
apply (*intro conjI*)
by (*auto intro*!: *V.li-le-dim*)
qed
from *every-li-small* **have** *exA*:
$\exists$ *A. finite A* $\wedge$ *maximal A* (λS. *S*$\subseteq$*carrier* (*V.vs kerT*) $\wedge$ *ker.lin-indpt S*)
apply (*intro maximal-exists*[**where** *?N*=*V.dim* **and** *?B*={}])
apply *auto*
by (*unfold ker.lin-dep-def*, *auto*)
from *exA* **obtain** *A* **where** *A*: *finite A* $\wedge$ *maximal A* (λS. *S*$\subseteq$*carrier* (*V.vs kerT*) $\wedge$ *ker.lin-indpt S*)
by *blast*
hence *finA*: *finite A* **and** *Ainker*: *A*$\subseteq$*carrier* (*V.vs kerT*) **and** *AinC*: *A*$\subseteq$*carrier V*
by (*unfold maximal-def ker-def*, *auto*)
— We obtain the basis A of kerT. It is also linearly independent when

considered in V rather than kerT

from *A* **have** *Abasis*: *ker.basis A*
by (*intro ker.max-li-is-basis*, *auto*)
from *subs-ker Abasis* **have** *spanA*: *V.module.span A* = *kerT*
apply (*unfold ker.basis-def*)
by (*subst sym*[*OF V.module.span-li-not-depend*(*1*)[**where** *?N*=*kerT*]], *auto*)
from *Abasis* **have** *Akerli*: *ker.lin-indpt A*
apply (*unfold ker.basis-def*)
by *auto*
from *subs-ker Ainker Akerli* **have** *Ali*: *V.module.lin-indpt A*
by (*auto simp add*: *V.module.span-li-not-depend*(*2*))

Use the replacement theorem to find C such that $A \cup C$ is a basis of V.

from *fd* **obtain** *B* **where** *B*: *finite B*∧ *V.basis B* **by** (*metis V.finite-basis-exists*)
from *B* **have** *Bfin*: *finite B* **and** *Bbasis*:*V.basis B* **by** *auto*
from *B* **have** *Bcard*: *V.dim* = *card B* **by** (*intro V.dim-basis*, *auto*)
from *Bbasis* **have** *62*: *V.module.span B* = *carrier V*
by (*unfold V.basis-def*, *auto*)
from *A Abasis Ali B vs-ker* **have** ∃ *C*. *finite C* ∧ *C*⊆*carrier V* ∧ *C*⊆ *V.module.span B* ∧ *C*∩*A*={}
∧ *int* (*card C*) ≤ (*int* (*card B*)) − (*int* (*card A*)) ∧ (*V.module.span* (*A* ∪ *C*) = *V.module.span B*)
apply (*intro V.replacement*)
apply (*unfold vectorspace.basis-def V.basis-def*)
by (*unfold ker-def*, *auto*)

From replacement we got $|C| \le |B| - |A|$. Equality must actually hold, because no generating set can be smaller than B. Now $A \cup C$ is a maximal generating set, hence a basis; its cardinality equals the dimension.

We claim that $T(C)$ is basis for $\mathrm{im}(T)$.

then obtain *C* **where** *C*: *finite C* ∧ *C*⊆*carrier V* ∧ *C*⊆ *V.module.span B* ∧ *C*∩*A*={}
∧ *int* (*card C*) ≤ (*int* (*card B*)) − (*int* (*card A*)) ∧ (*V.module.span* (*A* ∪ *C*) = *V.module.span B*) **by** *auto*
hence *Cfin*: *finite C* **and** *CinC*: *C*⊆*carrier V* **and** *CinspanB*: *C*⊆*V.module.span B* **and** *CAdis*: *C*∩*A*={}
and *Ccard*: *int* (*card C*) ≤ (*int* (*card B*)) − (*int* (*card A*))
and *ACspanB*: (*V.module.span* (*A* ∪ *C*) = *V.module.span B*) **by** *auto*
from *C* **have** *cardLe*: *card A* + *card C* ≤ *card B* **by** *auto*
from *B C* **have** *ACgen*: *V.module.gen-set* (*A*∪*C*) **apply** (*unfold V.basis-def*) **by** *auto*
from *finA C ACgen AinC B* **have** *cardGe*: *card* (*A*∪*C*) ≥ *card B* **by** (*intro V.li-smaller-than-gen*, *unfold V.basis-def*, *auto*)
from *finA C* **have** *cardUn*: *card* (*A*∪*C*)≤ *card A* + *card C*
by (*metis Int-commute card-Un-disjoint le-refl*)
from *cardLe cardUn cardGe Bcard* **have** *cardEq*:

card $(A \cup C)$ = *card* A + *card* C
card $(A \cup C)$ = *card* B
card $(A \cup C)$ = V.*dim*
by *auto*
from *Abasis C cardEq* **have** *disj*: $A \cap C = \{\}$ **by** *auto*
from *finA AinC C cardEq 62* **have** *ACfin*: *finite* $(A \cup C)$ **and** *ACbasis*: V.*basis* $(A \cup C)$
by (*auto intro*!: V.*dim-gen-is-basis*)
have *lm*: *linear-map K V W T*..

Let C' be the image of C under T. We will show C' is a basis for $\text{im}(T)$.

let $?C' = T`C$
from *Cfin* **have** *C'fin*: *finite* $?C'$ **by** *auto*
from *AinC C* **have** *cim*: $?C' \subseteq imT$ **by** (*unfold im-def*, *auto*)

"There is a subtle detail: we first have to show T is injective on C.

We establish that no nontrivial linear combination of C can have image 0 under T, because that would mean it is a linear combination of A, giving that $A \cup C$ is linearly dependent, contradiction. We use this result in 2 ways: (1) if T is not injective on C, then we obtain $v, w \in C$ such that $v - w$ is in the kernel, contradiction, (2) if $T(C)$ is linearly dependent, taking the inverse image of that linear combination gives a linear combination of C in the kernel, contradiction. Hence T is injective on C and $T(C)$ is linearly independent.

have *lc-in-ker*: $\bigwedge d\ D\ v.\ [\![D \subseteq C;\ d \in D \rightarrow carrier\ K;\ T\ (V.module.lincomb\ d\ D) = \mathbf{0}_W;$
$v \in D;\ d\ v \neq \mathbf{0}_K]\!] \Longrightarrow False$
proof –
fix *d D v*
assume *D*: $D \subseteq C$ **and** *d*: $d \in D \rightarrow carrier\ K$ **and** *T0*: $T\ (V.module.lincomb\ d\ D) = \mathbf{0}_W$
and *v*: $v \in D$ **and** *dvnz*: $d\ v \neq \mathbf{0}_K$
from *D Cfin* **have** *Dfin*: *finite D* **by** (*auto intro*: *finite-subset*)
from *D CinC* **have** *DinC*: $D \subseteq carrier\ V$ **by** *auto*
from *T0 d Dfin DinC* **have** *lc-d*: $V.module.lincomb\ d\ D \in kerT$
by (*unfold ker-def*, *auto*)
from *lc-d spanA AinC* **have** $\exists\, a'\ A'.\ A' \subseteq A \land a' \in A' \rightarrow carrier\ K$
$\land$
$V.module.lincomb\ a'\ A' = V.module.lincomb\ d\ D$
by (*intro* V.*module.in-span*, *auto*)
then obtain a' A' **where** a': $A' \subseteq A \land a' \in A' \rightarrow carrier\ K \land$
$V.module.lincomb\ d\ D = V.module.lincomb\ a'\ A'$
by *metis*
hence *A'sub*: $A' \subseteq A$ **and** *a'fun*: $a' \in A' \rightarrow carrier\ K$
and *a'-lc*: $V.module.lincomb\ d\ D = V.module.lincomb\ a'\ A'$ **by** *auto*
from a' *finA Dfin* **have** *A'fin*: *finite* (A') **by** (*auto intro*: *finite-subset*)

from *AinC A′sub* **have** *A′inC*: $A' \subseteq$ *carrier V* **by** *auto*
let *?e* = (λv. *if* $v \in A'$ *then* a' *v else* $\ominus_K \mathbf{1}_K \otimes_K$ *d v*)
from *a′fun d* **have** *e-fun*: *?e* $\in A' \cup D \rightarrow$ *carrier K*
apply (*unfold Pi-def*)
by *auto*
from
A′fin Dfin
A′inC DinC
a′fun d e-fun
disj D A′sub
have *lccomp1*:
V.module.lincomb a' A' $\oplus_V$ $\ominus_K \mathbf{1}_K \odot_V$ *V.module.lincomb d D* =
V.module.lincomb (λv. *if* $v \in A'$ *then* a' *v else* $\ominus_K \mathbf{1}_K \otimes_K$ *d v*)
($A' \cup D$)
apply (*subst sym*[*OF V.module.lincomb-smult*])
apply (*simp-all*)
apply (*subst V.module.lincomb-union2*)
by (*auto*)
from
A′fin
A′inC
a′fun
have *lccomp2*:
V.module.lincomb a' A' $\oplus_V$ $\ominus_K \mathbf{1}_K \odot_V$ *V.module.lincomb d D* =
$\mathbf{0}_V$
by (*simp add*: *a′-lc*
V.module.smult-minus-1 V.module.M.r-neg)
from *lccomp1 lccomp2* **have** *lc0*: *V.module.lincomb* (λv. *if* $v \in A'$
then a' *v else* $\ominus_K \mathbf{1}_K \otimes_K$ *d v*) ($A' \cup D$)
$=\mathbf{0}_V$ **by** *auto*
from *disj a′ v D* **have** *v-nin*: $v \notin A'$ **by** *auto*
from *A′fin Dfin*
A′inC DinC
e-fun d
A′sub D disj
v dvnz
lc0
have *AC-ld*: *V.module.lin-dep* ($A \cup C$)
apply (*intro V.module.lin-dep-crit*[**where** *?A*=$A' \cup D$ **and**
?S=$A \cup C$ **and** *?a*=λv. *if* $v \in A'$ *then* a' *v else* $\ominus_K \mathbf{1}_K \otimes_K$ *d v* **and**
?v=*v*])
by (*auto dest*: *integral*)
from *AC-ld ACbasis* **show** *False* **by** (*unfold V.basis-def*, *auto*)
qed
have *C′-card*: *inj-on T C card C* = *card ?C′*
proof −
show *inj-on T C*
proof (*rule ccontr*)
assume ¬*inj-on T C*

then obtain *v w* **where** $v{\in}C$ $w{\in}C$ $v{\neq}w$ $T\ v = T\ w$ **by** (*unfold inj-on-def*, *auto*)
from *this CinC* **show** *False*
apply (*intro lc-in-ker*[**where** $?D1{=}\{v,w\}$ **and** $?d1{=}\lambda x.$ *if* $x{=}v$ *then* $\mathbf{1}_K$ *else* $\ominus_K \mathbf{1}_K$
and $?v1{=}v$])
by (*auto simp add*: *V.module.lincomb-def hom-sum ring-subset-carrier*

W.module.smult-minus-1 r-neg T-im)
qed
from *this Cfin* **show** *card C* = *card ?C′*
by (*metis card-image*)
qed
let *?f*=*the-inv-into C T*
have *f*: $\bigwedge x.\ x{\in}C \Longrightarrow ?f\ (T\ x) = x$ $\bigwedge y.\ y{\in}?C' \Longrightarrow T\ (?f\ y) = y$
apply (*insert C′-card*(*1*))
apply (*metis the-inv-into-f-f*)
by (*metis f-the-inv-into-f*)

have *C′-li*: *im.lin-indpt ?C′*
proof (*rule ccontr*)
assume *Cld*: ¬*im.lin-indpt ?C′*
from *Cld cim subs-im* **have** *CldW*: *W.module.lin-dep ?C′*
apply (*subst sym*[*OF W.module.span-li-not-depend*(*2*)[**where** *?S*=*T‘C* **and** *?N*=*imT*]])
by *auto*
from *C CldW* **have** $\exists\, c'\ v'.\ (c'{\in}\ (?C' {\to} carrier\ K)) \land (W.module.lincomb\ c'\ ?C' = \mathbf{0}_W)$
$\land\ (v'{\in}?C') \land (c'\ v' {\neq}\ \mathbf{0}_K)$ **by** (*intro W.module.finite-lin-dep*, *auto*)
then obtain *c′ v′* **where** *c′*: $(c'{\in}\ (?C' {\to} carrier\ K)) \land (W.module.lincomb\ c'\ ?C' = \mathbf{0}_W)$
$\land\ (v'{\in}?C') \land (c'\ v' {\neq}\ \mathbf{0}_K)$ **by** *auto*
hence *c′fun*: $(c'{\in}\ (?C' {\to} carrier\ K))$ **and** *c′lc*: $(W.module.lincomb\ c'\ ?C' = \mathbf{0}_W)$ **and**
v′:$(v'{\in}?C')$ **and** *cvnz*: $(c'\ v' {\neq}\ \mathbf{0}_K)$ **by** *auto*

We take the inverse image of C' under T to get a linear combination of C that is in the kernel and hence a linear combination of A. This contradicts $A \cup C$ being linearly independent.

let *?c*=$\lambda v.\ c'\ (T\ v)$
from *c′fun* **have** *c-fun*: $?c{\in}\ C {\to} carrier\ K$ **by** *auto*
from *Cfin*
c-fun c′fun
C′-card
CinC
f
c′lc
have $T\ (V.module.lincomb\ ?c\ C) = \mathbf{0}_W$

apply (*unfold V.module.lincomb-def W.module.lincomb-def*)
apply (*subst hom-sum*, *auto*)
apply (*simp cong*: *finsum-cong add*: *ring-subset-carrier coeff-in-ring*)
apply (*subst finsum-reindex*[**where** *?f*=$\lambda w.\ c'\ w \odot_W w$ **and** *?h*=*T* **and** *?A*=*C*, *THEN sym*])
by *auto*
with *f c'fun cvnz v'* **show** *False*
by (*intro lc-in-ker*[**where** *?D1*=*C* **and** *?d1*=*?c* **and** *?v1*=*?f v'*], *auto*)
qed
have *C'-gen*: *im.gen-set ?C'*
proof −
have *C'-span*: *span ?C'* = *imT*
proof (*rule equalityI*)
from *cim subs-im* **show** *W.module.span ?C'* ⊆ *imT*
by (*intro span-is-subset*, *unfold subspace-def*, *auto*)
next
show *imT*⊆*W.module.span ?C'*
proof (*auto*)
fix *w*
assume *w*: *w*∈*imT*
from *this finA Cfin AinC CinC* **obtain** *v* **where** *v-inC*: *v*∈*carrier V* **and** *w-eq-T-v*: *w*= *T v*
by (*unfold im-def image-def*, *auto*)
from *finA Cfin AinC CinC v-inC ACgen* **have** ∃ *a*. *a* ∈ *A*∪*C* → *carrier K*∧ *V.module.lincomb a* (*A*∪*C*) = *v*
by (*intro V.module.finite-in-span*, *auto*)
then obtain *a* **where**
a-fun: *a* ∈ *A*∪*C* → *carrier K* **and**
lc-a-v: *v*= *V.module.lincomb a* (*A*∪*C*)
by *auto*
let *?a'*=λ*v*. *a* (*?f v*)
from *finA Cfin AinC CinC a-fun disj Ainker f C'-card* **have** *Tv*: *T v* = *W.module.lincomb ?a' ?C'*
apply (*subst lc-a-v*)
apply (*subst V.module.lincomb-union*, *simp-all*)
apply (*unfold lincomb-def V.module.lincomb-def*)
apply (*subst hom-sum*, *auto*)
apply (*simp add*: *subsetD coeff-in-ring*
hom-sum
T-ker
)
apply (*subst finsum-reindex*[**where** *?h*=*T* **and** *?f*=$\lambda v.\ ?a'\ v \odot_W v$], *auto*)
by (*auto cong*: *finsum-cong simp add*: *coeff-in-ring ring-subset-carrier*)
from *a-fun f* **have** *a'-fun*: *?a'*∈*?C'* → *carrier K* **by** *auto*
from *C'fin CinC this w-eq-T-v a'-fun Tv* **show** *w* ∈ *LinearCombinations.module.span K W* (*T* ‘ *C*)
by (*subst finite-span*, *auto*)

qed
qed
from *this subs-im CinC* **show** *?thesis*
apply (*subst span-li-not-depend*(*1*))
by (*unfold im-def subspace-def*, *auto*)
qed
from *C′-li C′-gen C cim* **have** *C′-basis*: *im.basis* (*T‘C*)
by (*unfold im.basis-def*, *auto*)
have *C-card-im*: *card C* = (*vectorspace.dim K* (*W.vs imT*))
using *C′-basis C′-card*(*2*) *C′fin im.dim-basis* **by** *auto*
from *finA Abasis* **have** *ker.dim* = *card A* **by** (*rule ker.dim-basis*)
note ∗ = *this C-card-im cardEq*
show (*vectorspace.dim K* (*W.vs imT*)) + (*vectorspace.dim K* (*V.vs kerT*)) = *V.dim* **using** ∗ **by** *auto*
assume *T‘* (*carrier V*) = *carrier W*
from ∗ *surj-imp-imT-carrier*[*OF this*]
show *W.fin-dim* **using** *C′-basis C′fin* **unfolding** *W.fin-dim-def im.basis-def* **by** *auto*
qed

theorem (**in** *linear-map*) *rank-nullity*:
assumes *fd*: *V.fin-dim*
shows (*vectorspace.dim K* (*W.vs imT*)) + (*vectorspace.dim K* (*V.vs kerT*)) = *V.dim*
by (*rule rank-nullity-main*[*OF fd*])

end