A verified LLL algorithm*

Jose Divasón Sebastiaan Joosten René Thiemann
Akihisa Yamada

February 3, 2018

Abstract

The Lenstra–Lenstra–Lovász basis reduction algorithm, also known as LLL algorithm, is an algorithm to find a basis with short, nearly orthogonal vectors of an integer lattice. Thereby, it can also be seen as an approximation to solve the shortest vector problem (SVP), which is an NP-hard problem, where the approximation quality solely depends on the dimension of the lattice, but not the lattice itself. The algorithm also possesses many applications in diverse fields of computer science, from cryptanalysis to number theory, but it is specially well-known since it was used to implement the first polynomial-time algorithm to factor polynomials. In this work we present the first mechanized soundness proof of the LLL algorithm to compute short vectors in lattices. The formalization follows a textbook by von zur Gathen and Gerhard [1].

Contents

*Supported by FWF (Austrian Science Fund) project Y757. Jose Divasón is partially funded by the Spanish project MTM2017-88804-P.

1 Introduction

The LLL basis reduction algorithm by Lenstra, Lenstra and Lovász [2] is a remarkable algorithm with numerous applications in diverse fields. For instance, it can be used for finding the minimal polynomial of an algebraic number given to a good enough approximation, for finding integer relations, for integer programming and even for breaking knapsack based cryptographic protocols. Its most famous application is a polynomial-time algorithm to factor integer polynomials. Moreover, the LLL algorithm is used as part of the best known polynomial factorization algorithm that is used in today's computer algebra systems.

In this work we implement it in Isabelle/HOL and fully formalize the correctness of the implementation. The algorithm is parametric by some $\alpha > \frac{4}{3}$, and given *fs* a list of m-linearly independent vectors $fs_0, \ldots, fs_{m-1} \in \mathbb{Z}^n$, it computes a short vector whose norm is at most $\alpha^{\frac{m-1}{2}}$ larger than the norm of any nonzero vector in the lattice generated by the vectors of the list *fs*. The soundness theorem follows.

Theorem 1 (Soundness of LLL algorithm)

lemma *short_vector* :
assumes $\alpha \geq 4/3$
and *lin_indpt_list* $(RAT\ fs)$
and *short_vector* $\alpha\ fs = v$
and *length* $fs = m$
and $m \neq 0$
shows $v \in lattice\_of\ fs - \{0_v\ n\}$
and $h \in lattice\_of\ fs - \{0_v\ n\} \longrightarrow \|v\|^2 \leq \alpha^{m-1} \cdot \|h\|^2$

To this end, we have performed the following tasks:

- We firstly have to improve some AFP entries, as well as generalize several concepts from the standard library.
- We have to develop a library about norms of vectors and their properties.

- We formalize the Gram–Schmidt orthogonalization procedure, which is a crucial sub-routine of the LLL algorithm. Indeed, we already formalized this procedure in Isabelle as a function *gram_schmidt* when proving the existence of Jordan normal forms [3]. Unfortunately, lemma *gram_schmidt* does not suffice for verifying the LLL algorithm and we have had to extend such a formalization.

- We prove the termination of the algorithm and its soundness.

Regarding the complexity of the LLL algorithm, we did not include a formal statement which would have required an instrumentation of the algorithm by some instruction counter. However, from the termination proof of our Isabelle implementation of the LLL algorithm, one can easily infer a polynomial bound on the number of arithmetic operations. To our knowledge, this is the first formalization of the LLL algorithm in any theorem prover.

2 List representation

theory *List-Representation*
  imports *Main*
begin

lemma *rev-take-Suc*: **assumes** *j*: $j < length\ xs$
  shows $rev\ (take\ (Suc\ j)\ xs) = xs\ !\ j\ \#\ rev\ (take\ j\ xs)$
proof –
 from *j* **have** *xs*: $xs = take\ j\ xs\ @\ xs\ !\ j\ \#\ drop\ (Suc\ j)\ xs$ **by** (*rule id-take-nth-drop*)
  show *?thesis* **unfolding** *arg-cong*[*OF xs*, *of* λ *xs*. *rev* (*take* (*Suc j*) *xs*)]
    by (*simp add*: *min-def*)
qed

type-synonym $'a\ list\text{-}repr = 'a\ list \times 'a\ list$

definition *list-repr* :: $nat \Rightarrow 'a\ list\text{-}repr \Rightarrow 'a\ list \Rightarrow bool$ **where**
  $list\text{-}repr\ i\ ba\ xs = (i \leq length\ xs \wedge fst\ ba = rev\ (take\ i\ xs) \wedge snd\ ba = drop\ i\ xs)$

definition *of-list-repr* :: $'a\ list\text{-}repr \Rightarrow 'a\ list$ **where**
  $of\text{-}list\text{-}repr\ ba = (rev\ (fst\ ba)\ @\ snd\ ba)$

lemma *of-list-repr*: $list\text{-}repr\ i\ ba\ xs \Longrightarrow of\text{-}list\text{-}repr\ ba = xs$
  unfolding *of-list-repr-def list-repr-def* **by** *auto*

definition *get-nth-i* :: $'a\ list\text{-}repr \Rightarrow 'a$ **where**
  $get\text{-}nth\text{-}i\ ba = hd\ (snd\ ba)$

definition *get-nth-im1* :: $'a\ list\text{-}repr \Rightarrow 'a$ **where**
  $get\text{-}nth\text{-}im1\ ba = hd\ (fst\ ba)$

lemma *get-nth-i*: *list-repr i ba xs* $\Longrightarrow$ *i* < *length xs* $\Longrightarrow$ *get-nth-i ba* = *xs* ! *i*
  unfolding *list-repr-def get-nth-i-def*
  by (*auto simp*: *hd-drop-conv-nth*)

lemma *get-nth-im1*: *list-repr i ba xs* $\Longrightarrow$ *i* $\neq$ *0* $\Longrightarrow$ *get-nth-im1 ba* = *xs* ! (*i* − *1*)

  unfolding *list-repr-def get-nth-im1-def*
  by (*cases i*, *auto simp*: *rev-take-Suc*)

definition *update-i* :: *'a list-repr* ⇒ *'a* ⇒ *'a list-repr* **where**
  update-i ba x = (*fst ba*, *x* # *tl* (*snd ba*))

lemma *Cons-tl-drop-update*: *i* < *length xs* $\Longrightarrow$ *x* # *tl* (*drop i xs*) = *drop i* (*xs*[*i* := *x*])
proof (*induct i arbitrary*: *xs*)
  case (*0 xs*)
  thus *?case* **by** (*cases xs*, *auto*)
next
  case (*Suc i xs*)
  thus *?case* **by** (*cases xs*, *auto*)
qed

lemma *update-i*: *list-repr i ba xs* $\Longrightarrow$ *i* < *length xs* $\Longrightarrow$ *list-repr i* (*update-i ba x*) (*xs* [*i* := *x*])
  unfolding *update-i-def list-repr-def*
  by (*auto simp*: *Cons-tl-drop-update*)

definition *update-im1* :: *'a list-repr* ⇒ *'a* ⇒ *'a list-repr* **where**
  update-im1 ba x = (*x* # *tl* (*fst ba*), *snd ba*)

lemma *update-im1*: *list-repr i ba xs* $\Longrightarrow$ *i* $\neq$ *0* $\Longrightarrow$ *list-repr i* (*update-im1 ba x*) (*xs* [*i* − *1* := *x*])
  unfolding *update-im1-def list-repr-def*
  by (*cases i*, *auto simp*: *rev-take-Suc*)

lemma *tl-drop-Suc*: *tl* (*drop i xs*) = *drop* (*Suc i*) *xs*
proof (*induct i arbitrary*: *xs*)
  case (*0 xs*) **thus** *?case* **by** (*cases xs*, *auto*)
next
  case (*Suc i xs*) **thus** *?case* **by** (*cases xs*, *auto*)
qed

definition *inc-i* :: *'a list-repr* ⇒ *'a list-repr* **where**
  inc-i ba = (*case ba of* (*b*,*a*) ⇒ (*hd a* # *b*, *tl a*))

lemma *inc-i*: *list-repr i ba xs* $\Longrightarrow$ *i* < *length xs* $\Longrightarrow$ *list-repr* (*Suc i*) (*inc-i ba*) *xs*
  unfolding *list-repr-def inc-i-def* **by** (*cases ba*, *auto simp*: *rev-take-Suc hd-drop-conv-nth tl-drop-Suc*)

definition *dec-i* :: *'a list-repr* ⇒ *'a list-repr* **where**
dec-i ba = (*case ba of* (*b*,*a*) ⇒ (*tl b*, *hd b* # *a*))

lemma *dec-i*: *list-repr i ba xs* ⟹ *i* ≠ *0* ⟹ *list-repr* (*i* − *1*) (*dec-i ba*) *xs*
unfolding *list-repr-def dec-i-def*
by (*cases ba*; *cases i*, *auto simp*: *rev-take-Suc hd-drop-conv-nth Cons-nth-drop-Suc*)

lemma *dec-i-Suc*: *list-repr* (*Suc i*) *ba xs* ⟹ *list-repr i* (*dec-i ba*) *xs*
using *dec-i*[*of Suc i ba xs*] **by** *auto*

end

3 Missing lemmas

This theory contains many results that are important but not specific for our development. They could be moved to the stardard library and some other AFP entries.

theory *Missing-Lemmas*
imports
Berlekamp-Zassenhaus.Sublist-Iteration
Berlekamp-Zassenhaus.Square-Free-Int-To-Square-Free-GFp
Algebraic-Numbers.Resultant
Jordan-Normal-Form.Conjugate
Jordan-Normal-Form.Missing-VectorSpace
VS-Connect
Berlekamp-Zassenhaus.Finite-Field-Factorization-Record-Based
Berlekamp-Zassenhaus.Berlekamp-Hensel
begin

hide-const(**open**) *module.smult up-ring.monom up-ring.coeff*

locale *comp-fun-commute-on* =
fixes *f* :: *'a* ⇒ *'a* ⇒ *'a* **and** *A*::*'a set*
assumes *comp-fun-commute-restrict*: ∀ *y*∈*A*. ∀ *x*∈*A*. ∀ *z*∈*A*. *f y* (*f x z*) = *f x* (*f y z*)
and *f*: *f* : *A* → *A* → *A*
begin

lemma *comp-fun-commute-on-UNIV*:
assumes *A* = (*UNIV* :: *'a set*)
shows *comp-fun-commute f*
unfolding *comp-fun-commute-def*
using *assms comp-fun-commute-restrict f* **by** *auto*

lemma *fun-left-comm*:
  assumes $y \in A$ **and** $x \in A$ **and** $z \in A$ **shows** $f\ y\ (f\ x\ z) = f\ x\ (f\ y\ z)$
  using *comp-fun-commute-restrict assms* **by** *auto*

lemma *commute-left-comp*:
  assumes $y \in A$ **and** $x{\in}A$ **and** $z{\in}A$ **and** $g \in A \to A$
  shows $f\ y\ (f\ x\ (g\ z)) = f\ x\ (f\ y\ (g\ z))$
  using *assms* **by** (*auto simp add*: *Pi-def o-assoc comp-fun-commute-restrict*)

lemma *fold-graph-finite*:
  assumes *fold-graph f z B y*
  shows *finite B*
  using *assms* **by** *induct simp-all*

lemma *fold-graph-closed*:
  assumes *fold-graph f z B y* **and** $B \subseteq A$ **and** $z \in A$
  shows $y \in A$
  using *assms*
proof (*induct set*: *fold-graph*)
  case *emptyI*
  then show *?case* **by** *auto*
next
  case (*insertI x B y*)
  then show *?case* **using** *insertI f* **by** *auto*
qed

lemma *fold-graph-insertE-aux*:
  fold-graph f z B y $\Longrightarrow a \in B \Longrightarrow z{\in}A$
  $\Longrightarrow B \subseteq A$
  $\Longrightarrow \exists\, y'.\ y = f\ a\ y' \wedge$ *fold-graph f z* $(B - \{a\})\ y' \wedge y' \in A$
proof (*induct set*: *fold-graph*)
  case *emptyI*
  then show *?case* **by** *auto*
next
  case (*insertI x B y*)
  show *?case*
  proof (*cases* $x = a$)
    case *True*
    show *?thesis*
    proof (*rule exI*[*of - y*])
      have *B*: (*insert x B* $- \{a\}) = B$ **using** *True insertI* **by** *auto*
      have $f\ x\ y = f\ a\ y$ **by** (*simp add*: *True*)
      moreover have *fold-graph f z* (*insert x B* $- \{a\})\ y$ **by** (*simp add*: *B insertI*)
      moreover have $y \in A$ **using** *insertI fold-graph-closed*[*of z B*] **by** *auto*
      ultimately show $f\ x\ y = f\ a\ y \wedge$ *fold-graph f z* (*insert x B* $- \{a\})\ y \wedge y \in$
A **by** *simp*
    qed
  next

case *False*
then obtain y' **where** y: $y = f\ a\ y'$ **and** y': *fold-graph f z* $(B - \{a\})\ y'$ **and** y'*-in-A*: $y' \in A$
using *insertI f* **by** *auto*
have $f\ x\ y = f\ a\ (f\ x\ y')$
unfolding y
proof (*rule fun-left-comm*)
show $x \in A$ **using** *insertI* **by** *auto*
show $a \in A$ **using** *insertI* **by** *auto*
show $y' \in A$ **using** y'*-in-A* **by** *auto*
qed
moreover have *fold-graph f z* (*insert x B* $- \{a\}$) $(f\ x\ y')$
using y' **and** ‹$x \neq a$› **and** ‹$x \notin B$›
by (*simp add*: *insert-Diff-if fold-graph.insertI*)
moreover have $(f\ x\ y') \in A$ **using** *insertI f* y'*-in-A* **by** *auto*
ultimately show *?thesis* **using** y'*-in-A*
by *auto*
qed
qed

lemma *fold-graph-insertE*:
assumes *fold-graph f z* (*insert x B*) v **and** $x \notin B$ **and** *insert x B* $\subseteq A$ **and** $z \in A$
obtains y **where** $v = f\ x\ y$ **and** *fold-graph f z B y*
using *assms* **by** (*auto dest*: *fold-graph-insertE-aux* [*OF - insertI1*])

lemma *fold-graph-determ*: *fold-graph f z B x* $\Longrightarrow$ *fold-graph f z B y* $\Longrightarrow$ $B \subseteq A$ $\Longrightarrow$ $z \in A$ $\Longrightarrow$ $y = x$
proof (*induct arbitrary*: *y set*: *fold-graph*)
case *emptyI*
then show *?case*
by (*meson empty-fold-graphE*)
next
case (*insertI x B y v*)
from ‹*fold-graph f z* (*insert x B*) v› **and** ‹$x \notin B$› **and** ‹*insert x B* $\subseteq A$› **and** ‹$z \in A$›
obtain y' **where** $v = f\ x\ y'$ **and** *fold-graph f z B* y'
by (*rule fold-graph-insertE*)
from ‹*fold-graph f z B* y'› **and** ‹*insert x B* $\subseteq A$› **have** $y' = y$ **using** *insertI* **by** *auto*
with ‹$v = f\ x\ y'$› **show** $v = f\ x\ y$
by *simp*
qed

lemma *fold-equality*: *fold-graph f z B y* $\Longrightarrow$ $B \subseteq A$ $\Longrightarrow$ $z \in A$ $\Longrightarrow$ *Finite-Set.fold f z B* $= y$
by (*cases finite B*)
(*auto simp add*: *Finite-Set.fold-def intro*: *fold-graph-determ dest*: *fold-graph-finite*)

lemma *fold-graph-fold*:

assumes *f*: *finite B* **and** *BA*: *B*⊆*A* **and** *z*: *z* ∈ *A*
shows *fold-graph f z B* (*Finite-Set.fold f z B*)
proof −
have ∃ *x*. *fold-graph f z B x*
by (*rule finite-imp-fold-graph*[*OF f*])
moreover note *fold-graph-determ*
ultimately have ∃!*x*. *fold-graph f z B x* **using** *f BA z* **by** *auto*
then have *fold-graph f z B* (*The* (*fold-graph f z B*))
by (*rule theI′*)
with *assms* **show** *?thesis*
by (*simp add*: *Finite-Set.fold-def*)
qed

lemma *fold-insert* [*simp*]:
assumes *finite B* **and** *x* ∉ *B* **and** *BA*: *insert x B* ⊆ *A* **and** *z*: *z* ∈ *A*
shows *Finite-Set.fold f z* (*insert x B*) = *f x* (*Finite-Set.fold f z B*)
proof (*rule fold-equality*[*OF - BA z*])
from ‹*finite B*› **have** *fold-graph f z B* (*Finite-Set.fold f z B*)
using *BA fold-graph-fold z* **by** *auto*
hence *fold-graph f z* (*insert x B*) (*f x* (*Finite-Set.fold f z B*))
using *BA fold-graph.insertI assms* **by** *auto*
then show *fold-graph f z* (*insert x B*) (*f x* (*Finite-Set.fold f z B*))
by *simp*
qed
end

lemma *fold-cong*:
assumes *f*: *comp-fun-commute-on f A* **and** *g*: *comp-fun-commute-on g A*
and *finite S*
and *cong*: ⋀*x*. *x* ∈ *S* ⟹ *f x* = *g x*
and *s* = *t* **and** *S* = *T*
and *SA*: *S* ⊆ *A* **and** *s*: *s*∈*A*
shows *Finite-Set.fold f s S* = *Finite-Set.fold g t T*
proof −
have *Finite-Set.fold f s S* = *Finite-Set.fold g s S*
using ‹*finite S*› *cong SA s*
proof (*induct S*)
case *empty*
then show *?case* **by** *simp*
next
case (*insert x F*)
interpret *f*: *comp-fun-commute-on f A* **by** (*fact f*)
interpret *g*: *comp-fun-commute-on g A* **by** (*fact g*)
show *?case* **using** *insert* **by** *auto*
qed
with *assms* **show** *?thesis* **by** *simp*
qed

context *comp-fun-commute-on*
begin

lemma *comp-fun-Pi*: $(\lambda x.\ f\ x\ \hat{}\hat{}\ g\ x) \in A \to A \to A$
proof $-$
  have $(f\ x\ \hat{}\hat{}\ g\ x)\ y \in A$ **if** y: $y \in A$ **and** x: $x \in A$ **for** $x\ y$
    using $x\ y$
   proof $(induct\ g\ x\ arbitrary{:}\ g)$
     case *0*
     then show *?case* **by** *auto*
   next
     case $(Suc\ n\ g)$
     define h **where** $h\ z = g\ z - 1$ **for** z
     have *hyp*: $(f\ x\ \hat{}\hat{}\ h\ x)\ y \in A$
       using *h-def Suc.prems Suc.hyps diff-Suc-1* **by** *metis*
     have $g\ x = Suc\ (h\ x)$ **unfolding** *h-def*
       using *Suc.hyps*(*2*) **by** *auto*
     then show *?case* **using** *f x hyp* **unfolding** *Pi-def* **by** *auto*
   qed
  thus *?thesis* **by** (*auto simp add*: *Pi-def*)
qed

lemma *comp-fun-commute-funpow*: *comp-fun-commute-on* $(\lambda x.\ f\ x\ \hat{}\hat{}\ g\ x)\ A$
proof $-$
  have f: $(f\ y\ \hat{}\hat{}\ g\ y)\ ((f\ x\ \hat{}\hat{}\ g\ x)\ z) = (f\ x\ \hat{}\hat{}\ g\ x)\ ((f\ y\ \hat{}\hat{}\ g\ y)\ z)$
    if x: $x{\in}A$ **and** y: $y \in A$ **and** z: $z \in A$ **for** $x\ y\ z$
  proof $(cases\ x = y)$
    case *False*
    show *?thesis*
    proof $(induct\ g\ x\ arbitrary{:}\ g)$
      case $(Suc\ n\ g)$
      have *hyp1*: $(f\ y\ \hat{}\hat{}\ g\ y)\ (f\ x\ k) = f\ x\ ((f\ y\ \hat{}\hat{}\ g\ y)\ k)$ **if** k: $k \in A$ **for** k
      proof $(induct\ g\ y\ arbitrary{:}\ g)$
        case *0*
        then show *?case* **by** *simp*
      next
        case $(Suc\ n\ g)$
        define h **where** $h\ z = g\ z - 1$ **for** z
        with *Suc* **have** $n = h\ y$
          by *simp*
        with *Suc* **have** *hyp*: $(f\ y\ \hat{}\hat{}\ h\ y)\ (f\ x\ k) = f\ x\ ((f\ y\ \hat{}\hat{}\ h\ y)\ k)$
          by *auto*
        from *Suc h-def* **have** g: $g\ y = Suc\ (h\ y)$
          by *simp*
         have $((f\ y\ \hat{}\hat{}\ h\ y)\ k) \in A$ **using** $y\ k$ *comp-fun-Pi*[*of h*] **unfolding** *Pi-def*
by *auto*
        then show *?case*

by (*simp add*: *comp-assoc g hyp*) (*auto simp add*: *o-assoc comp-fun-commute-restrict x y k*)
qed
define *h* **where** *h a* = (*if a* = *x then g x* − *1 else g a*) **for** *a*
with *Suc* **have** *n* = *h x*
by *simp*
with *Suc* **have** (*f y* ^^ *h y*) ((*f x* ^^ *h x*) *z*) = (*f x* ^^ *h x*) ((*f y* ^^ *h y*) *z*)
by *auto*
with *False* **have** *Suc2*: (*f x* ^^ *h x*) ((*f y* ^^ *g y*) *z*) = (*f y* ^^ *g y*) ((*f x* ^^ *h x*) *z*)
using *h-def* **by** *auto*
from *Suc h-def* **have** *g*: *g x* = *Suc* (*h x*)
by *simp*
have (*f x* ^^ *h x*) *z* ∈*A* **using** *comp-fun-Pi*[*of h*] *x z* **unfolding** *Pi-def* **by** *auto*
hence ∗: (*f y* ^^ *g y*) (*f x* ((*f x* ^^ *h x*) *z*)) = *f x* ((*f y* ^^ *g y*) ((*f x* ^^ *h x*) *z*))
using *hyp1* **by** *auto*
thus *?case* **using** *g Suc2* **by** *auto*
qed *simp*
qed *simp*
thus *?thesis* **by** (*auto simp add*: *comp-fun-commute-on-def comp-fun-Pi o-def*)
qed

lemma *fold-mset-add-mset*:
assumes *MA*: *set-mset M* ⊆ *A* **and** *s*: *s* ∈ *A* **and** *x*: *x* ∈ *A*
shows *fold-mset f s* (*add-mset x M*) = *f x* (*fold-mset f s M*)
proof −
interpret *mset*: *comp-fun-commute-on* λ*y*. *f y* ^^ *count M y A*
by (*fact comp-fun-commute-funpow*)
interpret *mset-union*: *comp-fun-commute-on* λ*y*. *f y* ^^ *count* (*add-mset x M*) *y A*
by (*fact comp-fun-commute-funpow*)
show *?thesis*
proof (*cases x* ∈ *set-mset M*)
case *False*
then have ∗: *count* (*add-mset x M*) *x* = *1*
by (*simp add*: *not-in-iff*)
have *Finite-Set.fold* (λ*y*. *f y* ^^ *count* (*add-mset x M*) *y*) *s* (*set-mset M*) =
Finite-Set.fold (λ*y*. *f y* ^^ *count M y*) *s* (*set-mset M*)
by (*rule fold-cong*[*of - A*], *auto simp add*: *assms False comp-fun-commute-funpow*)
with *False* ∗ *s MA x* **show** *?thesis*
by (*simp add*: *fold-mset-def del*: *count-add-mset*)
next
case *True*
let *?f* = (λ*xa*. *f xa* ^^ *count* (*add-mset x M*) *xa*)
let *?f2* = (λ*x*. *f x* ^^ *count M x*)
define *N* **where** *N* = *set-mset M* − {*x*}

have *F*: *Finite-Set.fold ?f s* (*insert x N*) = *?f x* (*Finite-Set.fold ?f s N*)
by (*rule mset-union.fold-insert*, *auto simp add*: *assms N-def*)
have *F2*: *Finite-Set.fold ?f2 s* (*insert x N*) = *?f2 x* (*Finite-Set.fold ?f2 s N*)
by (*rule mset.fold-insert*, *auto simp add*: *assms N-def*)
from *N-def True* **have** ∗: *set-mset M* = *insert x N x* ∉ *N finite N* **by** *auto*
then have *Finite-Set.fold* (*λy. f y* ^^ *count* (*add-mset x M*) *y*) *s N* =
Finite-Set.fold (*λy. f y* ^^ *count M y*) *s N*
using *MA N-def s*
by (*auto intro*!: *fold-cong comp-fun-commute-funpow*)
with ∗ **show** *?thesis* **by** (*simp add*: *fold-mset-def del*: *count-add-mset*, *unfold F F2*, *auto*)
qed
qed
end

context *abelian-monoid* **begin**

definition *sumlist*
where *sumlist xs* ≡ *foldr* (*op* ⊕) *xs* **0**

lemma [*simp*]:
shows *sumlist-Cons*: *sumlist* (*x#xs*) = *x* ⊕ *sumlist xs*
and *sumlist-Nil*: *sumlist* [] = **0**
by (*simp-all add*: *sumlist-def*)

lemma *sumlist-carrier* [*simp*]:
assumes *set xs* ⊆ *carrier G* **shows** *sumlist xs* ∈ *carrier G*
using *assms* **by** (*induct xs*, *auto*)

lemma *sumlist-neutral*:
assumes *set xs* ⊆ {**0**} **shows** *sumlist xs* = **0**
proof (*insert assms*, *induct xs*)
case (*Cons x xs*)
then have *x* = **0** **and** *set xs* ⊆ {**0**} **by** *auto*
with *Cons.hyps* **show** *?case* **by** *auto*
qed *simp*

lemma *sumlist-append*:
assumes *set xs* ⊆ *carrier G* **and** *set ys* ⊆ *carrier G*
shows *sumlist* (*xs* @ *ys*) = *sumlist xs* ⊕ *sumlist ys*
proof (*insert assms*, *induct xs arbitrary*: *ys*)
case (*Cons x xs*)
have *sumlist* (*xs* @ *ys*) = *sumlist xs* ⊕ *sumlist ys*
using *Cons.prems* **by** (*auto intro*: *Cons.hyps*)

with *Cons.prems* **show** *?case* **by** (*auto intro*!: *a-assoc*[*symmetric*])
qed *auto*

lemma *sumlist-snoc*:
assumes *set xs* $\subseteq$ *carrier G* **and** $x \in$ *carrier G*
shows *sumlist* (*xs* @ [*x*]) = *sumlist xs* $\oplus$ *x*
by (*subst sumlist-append*, *insert assms*, *auto*)

lemma *sumlist-as-finsum*:
assumes *set xs* $\subseteq$ *carrier G* **and** *distinct xs* **shows** *sumlist xs* = ($\bigoplus x{\in}$*set xs*. *x*)
using *assms* **by** (*induct xs*, *auto intro*:*finsum-insert*[*symmetric*])

lemma *sumlist-map-as-finsum*:
assumes *f* : *set xs* $\rightarrow$ *carrier G* **and** *distinct xs*
shows *sumlist* (*map f xs*) = ($\bigoplus x \in$ *set xs*. *f x*)
using *assms* **by** (*induct xs*, *auto*)

definition *summset* **where** *summset M* $\equiv$ *fold-mset* (*op* $\oplus$) **0** *M*

lemma *summset-empty* [*simp*]: *summset* {#} = **0** **by** (*simp add*: *summset-def*)

lemma *fold-mset-add-carrier*: $a \in$ *carrier G* $\Longrightarrow$ *set-mset M* $\subseteq$ *carrier G* $\Longrightarrow$ *fold-mset op* $\oplus$ *a M* $\in$ *carrier G*
proof (*induct M arbitrary*: *a*)
case (*add x M*)
thus *?case* **by**
(*subst comp-fun-commute-on.fold-mset-add-mset*[*of - carrier G*], *unfold-locales*, *auto simp*: *a-lcomm*)
qed *simp*

lemma *summset-carrier*[*intro*]: *set-mset M* $\subseteq$ *carrier G* $\Longrightarrow$ *summset M* $\in$ *carrier G*
unfolding *summset-def* **by** (*rule fold-mset-add-carrier*, *auto*)

lemma *summset-add-mset*[*simp*]:
assumes *a*: $a \in$ *carrier G* **and** *MG*: *set-mset M* $\subseteq$ *carrier G*
shows *summset* (*add-mset a M*) = *a* $\oplus$ *summset M*
using *assms*
by (*auto simp add*: *summset-def*)
(*rule comp-fun-commute-on.fold-mset-add-mset*, *unfold-locales*, *auto simp add*: *a-lcomm*)

lemma *sumlist-as-summset*:
assumes *set xs* $\subseteq$ *carrier G* **shows** *sumlist xs* = *summset* (*mset xs*)
by (*insert assms*, *induct xs*, *auto*)

lemma *sumlist-rev*:
assumes *set xs* $\subseteq$ *carrier G*

shows *sumlist* (*rev xs*) = *sumlist xs*
using *assms* **by** (*simp add*: *sumlist-as-summset*)

lemma *sumlist-as-fold*:
assumes *set xs* ⊆ *carrier G*
shows *sumlist xs* = *fold* (*op* ⊕) *xs* **0**
by (*fold sumlist-rev*[*OF assms*], *simp add*: *sumlist-def foldr-conv-fold*)

end

lemma (**in** *zero-less-one*) *zero-le-one* [*simp*]: *0* ≤ *1* **by** (*rule less-imp-le*, *simp*)
subclass (**in** *zero-less-one*) *zero-neq-one* **by** (*unfold-locales*, *simp add*: *less-imp-neq*)

class *ordered-semiring-1* = *Rings.ordered-semiring-0* + *monoid-mult* + *zero-less-one*
begin

subclass *semiring-1* **..**

lemma *of-nat-ge-zero*[*intro*!]: *of-nat n* ≥ *0*
using *add-right-mono*[*of - - 1*] **by** (*induct n*, *auto*)

lemma *zero-le-power* [*simp*]: *0* ≤ *a* ⟹ *0* ≤ *a* ˆ *n*
by (*induct n*) *simp-all*

lemma *power-mono*: *a* ≤ *b* ⟹ *0* ≤ *a* ⟹ *a* ˆ *n* ≤ *b* ˆ *n*
by (*induct n*) (*auto intro*: *mult-mono order-trans* [*of 0 a b*])

lemma *one-le-power* [*simp*]: *1* ≤ *a* ⟹ *1* ≤ *a* ˆ *n*
using *power-mono* [*of 1 a n*] **by** *simp*

lemma *power-le-one*: *0* ≤ *a* ⟹ *a* ≤ *1* ⟹ *a* ˆ *n* ≤ *1*
using *power-mono* [*of a 1 n*] **by** *simp*

lemma *power-gt1-lemma*:
assumes *gt1*: *1* < *a*
shows *1* < *a* ∗ *a* ˆ *n*
proof −
from *gt1* **have** *0* ≤ *a*
by (*fact order-trans* [*OF zero-le-one less-imp-le*])
from *gt1* **have** *1* ∗ *1* < *a* ∗ *1* **by** *simp*
also from *gt1* **have** ... ≤ *a* ∗ *a* ˆ *n*
by (*simp only*: *mult-mono* ‹*0* ≤ *a*› *one-le-power order-less-imp-le zero-le-one order-refl*)
finally show *?thesis* **by** *simp*
qed

```
lemma power-gt1: 1 < a ⟹ 1 < a ^ Suc n
  by (simp add: power-gt1-lemma)

lemma one-less-power [simp]: 1 < a ⟹ 0 < n ⟹ 1 < a ^ n
  by (cases n) (simp-all add: power-gt1-lemma)
```

Proof resembles that of *power-strict-decreasing*.

```
lemma power-decreasing: n ≤ N ⟹ 0 ≤ a ⟹ a ≤ 1 ⟹ a ^ N ≤ a ^ n
proof (induct N)
  case 0
  then show ?case by simp
next
  case (Suc N)
  then show ?case
    apply (auto simp add: le-Suc-eq)
    apply (subgoal-tac a * a^N ≤ 1 * a^n)
     apply simp
    apply (rule mult-mono)
       apply auto
    done
qed
```

Proof again resembles that of *power-strict-decreasing*.

```
lemma power-increasing: n ≤ N ⟹ 1 ≤ a ⟹ a ^ n ≤ a ^ N
proof (induct N)
  case 0
  then show ?case by simp
next
  case (Suc N)
  then show ?case
    apply (auto simp add: le-Suc-eq)
    apply (subgoal-tac 1 * a^n ≤ a * a^N)
     apply simp
    apply (rule mult-mono)
       apply (auto simp add: order-trans [OF zero-le-one])
    done
qed

lemma power-Suc-le-self: 0 ≤ a ⟹ a ≤ 1 ⟹ a ^ Suc n ≤ a
  using power-decreasing [of 1 Suc n a] by simp

end

lemma prod-list-nonneg: (⋀ x. (x :: 'a :: ordered-semiring-1) ∈ set xs ⟹ x ≥ 0)
⟹ prod-list xs ≥ 0
  by (induct xs, auto)

subclass (in ordered-idom) ordered-semiring-1 by unfold-locales auto
```

lemma *log-prod*: **assumes** $0 < a$ $a \neq 1$ $\bigwedge x.\ x \in X \Longrightarrow 0 < f\ x$
  shows *log a* (*prod f X*) = *sum* (*log a o f*) *X*
  using *assms*(*3*)
proof (*induct X rule*: *infinite-finite-induct*)
  case (*insert x F*)
  have *log a* (*prod f* (*insert x F*)) = *log a* (*f x* ∗ *prod f F*) **using** *insert* **by** *simp*
  also have … = *log a* (*f x*) + *log a* (*prod f F*)
    by (*rule log-mult*[*OF assms*(*1*−*2*) *insert*(*4*) *prod-pos*], *insert insert*, *auto*)
  finally show *?case* **using** *insert* **by** *auto*
qed *auto*

subclass (**in** *ordered-idom*) *zero-less-one* **by** (*unfold-locales*, *auto*)
hide-fact *Missing-Ring.zero-less-one*

instance *real* :: *ordered-semiring-strict* **by** (*intro-classes*, *auto*)
instance *real* :: *linordered-idom*..

lemma *less-1-mult'*:
  fixes *a*::*'a*::*linordered-semidom*
  shows $1 < a \Longrightarrow 1 \leq b \Longrightarrow 1 < a * b$
  by (*metis le-less less-1-mult mult.right-neutral*)

lemma *upt-minus-eq-append*: $i \leq j \Longrightarrow i \leq j-k \Longrightarrow [i..<j] = [i..<j-k]$ @ $[j-k..<j]$
proof (*induct k*)
  case (*Suc k*)
  have *hyp*: $[i..<j] = [i..<j - k]$ @ $[j - k..<j]$ **using** *Suc.hyps Suc.prems* **by** *auto*
  then show *?case*
    by (*metis Suc.prems*(*2*) *append.simps*(*1*) *diff-Suc-less nat-less-le neq0-conv upt-append upt-rec zero-diff*)
qed *auto*

lemma *list-trisect*: $x <$ *length lst* $\Longrightarrow$ [*0*..<*length lst*] = [*0*..<*x*]@*x*#[*Suc x*..<*length lst*]
  by (*induct lst*, *force*, *rename-tac a lst*, *case-tac x* = *length lst*, *auto*)

lemma *nth-map-out-of-bound*: $i \geq$ *length xs* $\Longrightarrow$ *map f xs* ! *i* = [] ! (*i* − *length xs*)
  by (*induct xs arbitrary*:*i*, *auto*)

lemma *filter-mset-inequality*: *filter-mset f xs* ≠ *xs* ⟹ ∃ *x* ∈# *xs*. ¬ *f x*
  by (*induct xs*, *auto*)

```
lemma id-imp-bij-betw:
  assumes f: f : A → A
      and ff: ⋀a. a ∈ A ⟹ f (f a) = a
  shows bij-betw f A A
  by (intro bij-betwI[OF f f], simp-all add: ff)

lemma if-distrib-ap:
  (if x then y else z) u = (if x then y u else z u) by auto

lemma range-subsetI:
  assumes ⋀x. f x = g (h x) shows range f ⊆ range g
  using assms by auto

lemma Gcd-uminus:
  fixes A::int set
  assumes finite A
  shows Gcd A = Gcd (uminus ‘ A)
  using assms
  by (induct A, auto)

lemma aux-abs-int: fixes c :: int
  assumes c ≠ 0
  shows |x| ≤ |x ∗ c|
proof –
  have abs x = abs x ∗ 1 by simp
  also have ... ≤ abs x ∗ abs c
    by (rule mult-left-mono, insert assms, auto)
  finally show ?thesis unfolding abs-mult by auto
qed

lemma sqrt-int-ceiling-bound: 0 ≤ x ⟹ x ≤ (sqrt-int-ceiling x)^2
 unfolding sqrt-int-ceiling using le-of-int-ceiling of-int-le-iff sqrt-le-D by fastforce

lemma mod-0-abs-less-imp-0:
  fixes a::int
  assumes a1: [a = 0] (mod m)
  and a2: abs(a)<m
  shows a = 0
proof –
  have m≥0 using assms by auto
  thus ?thesis
    using assms unfolding cong-int-def
    using int-mod-pos-eq large-mod-0 zless-imp-add1-zle
    by (metis abs-of-nonneg le-less not-less zabs-less-one-iff zmod-trival-iff)
qed


lemma sum-list-zero:
  assumes set xs ⊆ {0} shows sum-list xs = 0
```

using *assms* **by** (*induct xs*, *auto*)

lemma *max-idem* [*simp*]: **shows** *max a a* = *a* **by** (*simp add*: *max-def*)

lemma *hom-max*:
 assumes $a \leq b \longleftrightarrow f\ a \leq f\ b$
 shows *f* (*max a b*) = *max* (*f a*) (*f b*) **using** *assms* **by** (*auto simp*: *max-def*)

lemma *le-max-self*:
 fixes *a b* :: *'a* :: *preorder*
 assumes $a \leq b \lor b \leq a$ **shows** $a \leq$ *max a b* **and** $b \leq$ *max a b*
 using *assms* **by** (*auto simp*: *max-def*)

lemma *le-max*:
 fixes *a b* :: *'a* :: *preorder*
 assumes $c \leq a \lor c \leq b$ **and** $a \leq b \lor b \leq a$ **shows** $c \leq$ *max a b*
 using *assms*(*1*) *le-max-self*[*OF assms*(*2*)] **by** (*auto dest*: *order-trans*)

fun *max-list* **where**
 max-list [] = (*THE x*. *False*)
| *max-list* [*x*] = *x*
| *max-list* (*x* # *y* # *xs*) = *max x* (*max-list* (*y* # *xs*))

declare *max-list.simps*(*1*) [*simp del*]
declare *max-list.simps*(*2*−*3*)[*code*]

lemma *max-list-Cons*: *max-list* (*x*#*xs*) = (*if xs* = [] *then x else max x* (*max-list xs*))
 by (*cases xs*, *auto*)

lemma *max-list-mem*: *xs* ≠ [] ⟹ *max-list xs* ∈ *set xs*
 by (*induct xs*, *auto simp*: *max-list-Cons max-def*)

lemma *mem-set-imp-le-max-list*:
 fixes *xs* :: *'a* :: *preorder list*
 assumes ⋀*a b*. *a* ∈ *set xs* ⟹ *b* ∈ *set xs* ⟹ $a \leq b \lor b \leq a$
  and *a* ∈ *set xs*
 shows $a \leq$ *max-list xs*
proof (*insert assms*, *induct xs arbitrary*:*a*)
 case *Nil*
 with *assms* **show** *?case* **by** *auto*
next
 case (*Cons x xs*)
 show *?case*
 proof (*cases xs* = [])
  case *False*
  have $x \leq$ *max-list xs* ∨ *max-list xs* $\leq x$
   apply (*rule Cons*(*2*)) **using** *max-list-mem*[*of xs*] *False* **by** *auto*

```
    note 1 = le-max-self[OF this]
    from Cons have a = x ∨ a ∈ set xs by auto
    then show ?thesis
    proof (elim disjE)
      assume a: a = x
      show ?thesis by (unfold a max-list-Cons, auto simp: False intro!: 1)
    next
      assume a ∈ set xs
      then have a ≤ max-list xs by (intro Cons, auto)
      with 1 have a ≤ max x (max-list xs) by (auto dest: order-trans)
      then show ?thesis by (unfold max-list-Cons, auto simp: False)
    qed
  qed (insert Cons, auto)
qed

lemma le-max-list:
  fixes xs :: 'a :: preorder list
  assumes ord: ⋀a b. a ∈ set xs ⟹ b ∈ set xs ⟹ a ≤ b ∨ b ≤ a
      and ab: a ≤ b
      and b: b ∈ set xs
  shows a ≤ max-list xs
proof−
  note ab
  also have b ≤ max-list xs
    by (rule mem-set-imp-le-max-list, fact ord, fact b)
  finally show ?thesis.
qed

lemma max-list-le:
  fixes xs :: 'a :: preorder list
  assumes a: ⋀x. x ∈ set xs ⟹ x ≤ a
      and xs: xs ≠ []
  shows max-list xs ≤ a
  using max-list-mem[OF xs] a by auto

lemma max-list-as-Greatest:
  assumes ⋀x y. x ∈ set xs ⟹ y ∈ set xs ⟹ x ≤ y ∨ y ≤ x
  shows max-list xs = (GREATEST a. a ∈ set xs)
proof (cases xs = [])
  case True
  then show ?thesis by (unfold Greatest-def, auto simp: max-list.simps(1))
next
  case False
  from assms have 1: x ∈ set xs ⟹ x ≤ max-list xs for x
    by (auto intro: le-max-list)
  have 2: max-list xs ∈ set xs by (fact max-list-mem[OF False])
  have ∃!x. x ∈ set xs ∧ (∀ y. y ∈ set xs ⟶ y ≤ x) (is ∃!x. ?P x)
  proof (intro ex1I)
    from 1 2
```

show *?P* (*max-list xs*) **by** *auto*
next
fix *x* **assume** *3*: *?P x*
with *1* **have** $x \leq$ *max-list xs* **by** *auto*
moreover from *2 3* **have** *max-list xs* $\leq x$ **by** *auto*
ultimately show $x =$ *max-list xs* **by** *auto*
qed
note *3* = *theI-unique*[*OF this*,*symmetric*]
from *1 2* **show** *?thesis*
by (*unfold Greatest-def Cons 3*, *auto*)
qed

lemma *hom-max-list-commute*:
assumes *xs* $\neq$ []
and $\bigwedge x\ y.\ x \in set\ xs \Longrightarrow y \in set\ xs \Longrightarrow h\ (max\ x\ y) = max\ (h\ x)\ (h\ y)$
shows *h* (*max-list xs*) = *max-list* (*map h xs*)
by (*insert assms*, *induct xs*, *auto simp*: *max-list-Cons max-list-mem*)

primrec *rev-upt* :: *nat* ⇒ *nat* ⇒ *nat list* ((*1*[->..-])) **where**
rev-upt-0: [*0*>..*j*] = [] |
rev-upt-Suc: [(*Suc i*)>..*j*] = (*if i* ≥ *j then i* # [*i*>..*j*] *else* [])

lemma *rev-upt-rec*: [*i*>..*j*] = (*if i*>*j then* [*i*>..*Suc j*] @ [*j*] *else* [])
by (*induct i*, *auto*)

definition *rev-upt-aux* :: *nat* ⇒ *nat* ⇒ *nat list* ⇒ *nat list* **where**
rev-upt-aux i j js = [*i*>..*j*] @ *js*

lemma *upt-aux-rec* [*code*]:
rev-upt-aux i j js = (*if j*≥*i then js else rev-upt-aux i* (*Suc j*) (*j*#*js*))
by (*induct j*, *auto simp add*: *rev-upt-aux-def rev-upt-rec*)

lemma *rev-upt-code*[*code*]: [*i*>..*j*] = *rev-upt-aux i j* []
by(*simp add*: *rev-upt-aux-def*)

lemma *upt-rev-upt*:
rev [*j*>..*i*] = [*i*..<*j*]
by (*induct j*, *auto*)

lemma *rev-upt-upt*:
rev [*i*..<*j*] = [*j*>..*i*]
by (*induct j*, *auto*)

lemma *length-rev-upt* [*simp*]: *length* [*i*>..*j*] = *i* − *j*
by (*induct i*) (*auto simp add*: *Suc-diff-le*)

lemma *nth-rev-upt* [*simp*]: *j* + *k* < *i* ⟹ [*i*>..*j*] ! *k* = *i* − *1* − *k*

```
proof −
  assume jk-i: j + k < i
  have [i>..j] = rev [j..<i] using rev-upt-upt by simp
  also have ... ! k = [j..<i] ! (length [j..<i] − 1 − k)
    by (rule nth-rev, insert jk-i, auto)
  also have ... = [j..<i] ! (i − j − 1 − k) by auto
  also have ... = j + (i − j − 1 − k) by (rule nth-upt, insert jk-i, auto)
  finally show ?thesis using jk-i by auto
qed

lemma nth-map-rev-upt:
  assumes i: i < m−n
  shows (map f [m>..n]) ! i = f (m − 1 − i)
proof −
  have (map f [m>..n]) ! i = f ([m>..n] ! i) by (rule nth-map, auto simp add: i)
  also have ... = f (m − 1 − i)
  proof (rule arg-cong[of - - f], rule nth-rev-upt)
    show n + i < m using i by linarith
  qed
  finally show ?thesis .
qed

lemma coeff-mult-monom:
 coeff (p ∗ monom a d) i = (if d ≤ i then a ∗ coeff p (i − d) else 0)
  using coeff-monom-mult[of a d p] by (simp add: ac-simps)


lemma smult-sum2: smult m (∑ i ∈ S. f i) = (∑ i ∈ S. smult m (f i))
  by (induct S rule: infinite-finite-induct, auto simp add: smult-add-right)

lemma deg-not-zero-imp-not-unit:
  fixes f:: 'a::{idom-divide,semidom-divide-unit-factor} poly
  assumes deg-f: degree f > 0
  shows ¬ is-unit f
proof −
  have degree (normalize f) > 0
    using deg-f degree-normalize by auto
  hence normalize f ≠ 1
    by fastforce
  thus ¬ is-unit f using normalize-1-iff by auto
qed
```

lemma *conjugate-square-eq-0* [*simp*]:
fixes *x* :: *'a* :: {*conjugatable-ring*,*semiring-no-zero-divisors*}
shows $x * conjugate\ x = 0 \longleftrightarrow x = 0$
by *simp*

lemma *conjugate-square-greater-0* [*simp*]:
fixes *x* :: *'a* :: {*conjugatable-ordered-ring*,*ring-no-zero-divisors*}
shows $x * conjugate\ x > 0 \longleftrightarrow x \neq 0$
using *conjugate-square-positive*[*of x*]
by (*auto simp*: *le-less*)

lemma *set-rows-carrier*:
assumes $A \in carrier\text{-}mat\ m\ n$ **and** $v \in set\ (rows\ A)$ **shows** $v \in carrier\text{-}vec\ n$
using *assms* **by** (*auto simp*: *set-conv-nth*)

abbreviation *vNil* **where** $vNil \equiv vec\ 0\ undefined$
definition *vCons* **where** $vCons\ a\ v \equiv vec\ (Suc\ (dim\text{-}vec\ v))\ (\lambda i.\ case\ i\ of\ 0 \Rightarrow a \mid Suc\ i \Rightarrow v\ \$\ i)$

lemma *vec-index-vCons-0* [*simp*]: $vCons\ a\ v\ \$\ 0 = a$
by (*simp add*: *vCons-def*)

lemma *vec-index-vCons-Suc* [*simp*]:
fixes *v* :: *'a vec*
shows $vCons\ a\ v\ \$\ Suc\ n = v\ \$\ n$
proof−
have *1*: $vec\ (Suc\ d)\ f\ \$\ Suc\ n = vec\ d\ (f \circ Suc)\ \$\ n$ **for** *d* **and** $f :: nat \Rightarrow 'a$
by (*transfer*, *auto simp*: *mk-vec-def*)
show *?thesis*
apply (*auto simp*: *1 vCons-def o-def*) **apply** *transfer* **apply** (*auto simp*: *mk-vec-def*)
done
qed

lemma *vec-index-vCons*: $vCons\ a\ v\ \$\ n = (if\ n = 0\ then\ a\ else\ v\ \$\ (n - 1))$
by (*cases n*, *auto*)

lemma *dim-vec-vCons* [*simp*]: $dim\text{-}vec\ (vCons\ a\ v) = Suc\ (dim\text{-}vec\ v)$
by (*simp add*: *vCons-def*)

lemma *vCons-carrier-vec*[*simp*]: $vCons\ a\ v \in carrier\text{-}vec\ (Suc\ n) \longleftrightarrow v \in carrier\text{-}vec\ n$
by (*auto dest*!: *carrier-vecD intro*: *carrier-vecI*)

lemma *vec-Suc*: *vec* (*Suc n*) *f* = *vCons* (*f 0*) (*vec n* (*f* ∘ *Suc*)) (**is** *?l* = *?r*)
proof (*unfold vec-eq-iff*, *intro conjI allI impI*)
  fix *i* **assume** *i* < *dim-vec ?r*
  then show *?l* $ *i* = *?r* $ *i* **by** (*cases i*, *auto*)
qed *simp*

declare *Abs-vec-cases*[*cases del*]

lemma *vec-cases* [*case-names vNil vCons*, *cases type*: *vec*]:
  assumes *v* = *vNil* ⟹ *thesis* **and** ⋀*a w*. *v* = *vCons a w* ⟹ *thesis*
  shows *thesis*
proof (*cases dim-vec v*)
  case *0* **then show** *thesis* **by** (*intro assms*(*1*), *auto*)
next
  case (*Suc n*)
  show *thesis*
  proof (*rule assms*(*2*))
    show *v*: *v* = *vCons* (*v* $ *0*) (*vec n* (λ*i*. *v* $ *Suc i*)) (**is** *v* = *?r*)
    proof (*rule eq-vecI*, *unfold dim-vec-vCons dim-vec Suc*)
      fix *i*
      assume *i* < *Suc n*
      then show *v* $ *i* = *?r* $ *i* **by** (*cases i*, *auto simp*: *vCons-def*)
    qed *simp*
  qed
qed

lemma *vec-induct* [*case-names vNil vCons*, *induct type*: *vec*]:
  assumes *P vNil* **and** ⋀*a v*. *P v* ⟹ *P* (*vCons a v*)
  shows *P v*
proof (*induct dim-vec v arbitrary*:*v*)
  case *0* **then show** *?case* **by** (*cases v*, *auto intro*: *assms*(*1*))
next
  case (*Suc n*) **then show** *?case* **by** (*cases v*, *auto intro*: *assms*(*2*))
qed

lemma *carrier-vec-induct* [*consumes 1*, *case-names 0 Suc*, *induct set*:*carrier-vec*]:
  assumes *v*: *v* ∈ *carrier-vec n*
    and *1*: *P 0 vNil* **and** *2*: ⋀*n a v*. *v* ∈ *carrier-vec n* ⟹ *P n v* ⟹ *P* (*Suc n*) (*vCons a v*)
  shows *P n v*
proof (*insert v*, *induct n arbitrary*: *v*)
  case *0* **then have** *v* = *vec 0 undefined* **by** *auto*
  with *1* **show** *?case* **by** *auto*
next
  case (*Suc n*) **then show** *?case* **by** (*cases v*, *auto dest*!: *carrier-vecD intro*:*2*)
qed

lemma *vec-of-list-Cons*[*simp*]: *vec-of-list* (*a*#*as*) = *vCons a* (*vec-of-list as*)
  by (*unfold vCons-def*, *transfer*, *auto simp*:*mk-vec-def split*:*nat.split*)

lemma *vec-of-list-Nil*[*simp*]: *vec-of-list* [] = *vNil*
  by *transfer auto*

lemma *scalar-prod-vCons*[*simp*]:
  vCons a v · *vCons b w* = *a* ∗ *b* + *v* · *w*
  apply (*unfold scalar-prod-def atLeast0-lessThan-Suc-eq-insert-0 dim-vec-vCons*)
  apply (*subst sum.insert*) **apply** (*simp*,*simp*)
  apply (*subst sum.reindex*) **apply** *force*
  apply *simp*
  done

lemma *zero-vec-Suc*: 0_v (*Suc n*) = *vCons 0* (0_v *n*)
  by (*auto simp*: *zero-vec-def vec-Suc o-def*)

lemma *zero-vec-zero*[*simp*]: 0_v *0* = *vNil* **by** *auto*

lemma *vCons-eq-vCons*[*simp*]: *vCons a v* = *vCons b w* ⟷ *a* = *b* ∧ *v* = *w* (**is** *?l* ⟷ *?r*)
proof
  assume *?l*
  note *arg-cong*[*OF this*]
  from *this*[*of dim-vec*] *this*[*of* λ*x*. *x*\$*0*] *this*[*of* λ*x*. *x*\$*Suc* -]
  show *?r* **by** (*auto simp*: *vec-eq-iff*)
qed *simp*

instantiation *vec* :: (*conjugate*) *conjugate*
begin

definition *conjugate-vec* :: ′*a* :: *conjugate vec* ⇒ ′*a vec*
  where *conjugate v* = *vec* (*dim-vec v*) (λ*i*. *conjugate* (*v* \$ *i*))

lemma *conjugate-vCons* [*simp*]:
  conjugate (*vCons a v*) = *vCons* (*conjugate a*) (*conjugate v*)
  by (*auto simp*: *vec-Suc conjugate-vec-def*)

lemma *dim-vec-conjugate*[*simp*]: *dim-vec* (*conjugate v*) = *dim-vec v*
  unfolding *conjugate-vec-def* **by** *auto*

lemma *carrier-vec-conjugate*[*simp*]: *v* ∈ *carrier-vec n* ⟹ *conjugate v* ∈ *carrier-vec n*
  by (*auto intro*!: *carrier-vecI*)

lemma *vec-index-conjugate*[*simp*]:
  shows *i* < *dim-vec v* ⟹ *conjugate v* \$ *i* = *conjugate* (*v* \$ *i*)
  unfolding *conjugate-vec-def* **by** *auto*

instance

```
proof
  fix v w :: 'a vec
  show conjugate (conjugate v) = v by (induct v, auto simp: conjugate-vec-def)
  let ?v = conjugate v
  let ?w = conjugate w
  show conjugate v = conjugate w ⟷ v = w
  proof(rule iffI)
    assume cvw: ?v = ?w show v = w
    proof(rule)
      have dim-vec ?v = dim-vec ?w using cvw by auto
      then show dim: dim-vec v = dim-vec w by simp
      fix i assume i: i < dim-vec w
      then have conjugate v $ i = conjugate w $ i using cvw by auto
      then have conjugate (v$i) = conjugate (w $ i) using i dim by auto
      then show v $ i = w $ i by auto
    qed
  qed auto
qed

end

lemma conjugate-add-vec:
  fixes v w :: 'a :: conjugatable-ring vec
  assumes dim: v : carrier-vec n w : carrier-vec n
  shows conjugate (v + w) = conjugate v + conjugate w
  by (rule, insert dim, auto simp: conjugate-dist-add)

lemma uminus-conjugate-vec:
  fixes v w :: 'a :: conjugatable-ring vec
  shows − (conjugate v) = conjugate (− v)
  by (rule, auto simp:conjugate-neg)

lemma conjugate-zero-vec[simp]:
  conjugate (0_v n :: 'a :: conjugatable-ring vec) = 0_v n by auto

lemma conjugate-vec-0[simp]:
  conjugate (vec 0 f) = vec 0 f by auto

lemma sprod-vec-0[simp]: v · vec 0 f = 0
  by(auto simp: scalar-prod-def)

lemma conjugate-zero-iff-vec[simp]:
  fixes v :: 'a :: conjugatable-ring vec
  shows conjugate v = 0_v n ⟷ v = 0_v n
  using conjugate-cancel-iff[of - 0_v n :: 'a vec] by auto

lemma conjugate-smult-vec:
  fixes k :: 'a :: conjugatable-ring
  shows conjugate (k ·_v v) = conjugate k ·_v conjugate v
```

using *conjugate-dist-mul* **by** (*intro eq-vecI*, *auto*)

lemma *conjugate-sprod-vec*:
fixes *v w* :: *'a* :: *conjugatable-ring vec*
assumes *v*: *v* : *carrier-vec n* **and** *w*: *w* : *carrier-vec n*
shows *conjugate* (*v* · *w*) = *conjugate v* · *conjugate w*
proof (*insert w v*, *induct w arbitrary*: *v rule*:*carrier-vec-induct*)
case *0* **then show** *?case* **by** (*cases v*, *auto*)
next
case (*Suc n b w*) **then show** *?case*
by (*cases v*, *auto dest*: *carrier-vecD simp*:*conjugate-dist-add conjugate-dist-mul*)
qed

abbreviation *cscalar-prod* :: *'a vec* ⇒ *'a vec* ⇒ *'a* :: *conjugatable-ring* (**infix** ·*c* *70*)
where *op* ·*c* ≡ λ*v w*. *v* · *conjugate w*

lemma *conjugate-conjugate-sprod*[*simp*]:
assumes *v*[*simp*]: *v* : *carrier-vec n* **and** *w*[*simp*]: *w* : *carrier-vec n*
shows *conjugate* (*conjugate v* · *w*) = *v* ·*c* *w*
apply (*subst conjugate-sprod-vec*[*of - n*]) **by** *auto*

lemma *conjugate-vec-sprod-comm*:
fixes *v w* :: *'a* :: {*conjugatable-ring*, *comm-ring*} *vec*
assumes *v* : *carrier-vec n* **and** *w* : *carrier-vec n*
shows *v* ·*c* *w* = (*conjugate w* · *v*)
unfolding *scalar-prod-def* **using** *assms* **by**(*subst sum-ivl-cong*, *auto simp*: *ac-simps*)

lemma *vec-carrier-vec*[*simp*]: *vec n f* ∈ *carrier-vec m* ⟷ *n* = *m*
unfolding *carrier-vec-def* **by** *auto*

lemma *conjugate-square-ge-0-vec*[*intro*!]:
fixes *v* :: *'a* :: *conjugatable-ordered-ring vec*
shows *v* ·*c* *v* ≥ *0*
proof (*induct v*)
case *vNil*
then show *?case* **by** *auto*
next
case (*vCons a v*)
then show *?case* **using** *conjugate-square-positive*[*of a*] **by** *auto*
qed

lemma *conjugate-square-eq-0-vec*[*simp*]:
fixes *v* :: *'a* :: {*conjugatable-ordered-ring*,*semiring-no-zero-divisors*} *vec*
assumes *v* ∈ *carrier-vec n*
shows *v* ·*c* *v* = *0* ⟷ *v* = 0_v *n*
proof (*insert assms*, *induct rule*: *carrier-vec-induct*)
case *0*
then show *?case* **by** *auto*

next
  case (*Suc n a v*)
  then show *?case*
    using *conjugate-square-positive*[*of a*] *conjugate-square-ge-0-vec*[*of v*]
    by (*auto simp*: *le-less add-nonneg-eq-0-iff zero-vec-Suc*)
qed

lemma *conjugate-square-greater-0-vec*[*simp*]:
  fixes $v :: 'a :: \{conjugatable\text{-}ordered\text{-}ring, semiring\text{-}no\text{-}zero\text{-}divisors\}\ vec$
  assumes $v \in carrier\text{-}vec\ n$
  shows $v \cdot c\ v > 0 \longleftrightarrow v \neq 0_v\ n$
  using *assms* **by** (*auto simp*: *less-le*)

lemma *vec-conjugate-rat*[*simp*]: $(conjugate :: rat\ vec \Rightarrow rat\ vec) = (\lambda x.\ x)$ **by** *force*
lemma *vec-conjugate-real*[*simp*]: $(conjugate :: real\ vec \Rightarrow real\ vec) = (\lambda x.\ x)$ **by** *force*

notation *transpose-mat* $((\text{-}^T)\ [1000])$

lemma *cols-transpose*[*simp*]: $cols\ A^T = rows\ A$ **unfolding** *cols-def rows-def* **by** *auto*
lemma *rows-transpose*[*simp*]: $rows\ A^T = cols\ A$ **unfolding** *cols-def rows-def* **by** *auto*
lemma *list-of-vec-vec* [*simp*]: $list\text{-}of\text{-}vec\ (vec\ n\ f) = map\ f\ [0..<n]$
  by (*transfer*, *auto simp*: *mk-vec-def*)

lemma *list-of-vec-0* [*simp*]: $list\text{-}of\text{-}vec\ (0_v\ n) = replicate\ n\ 0$
  by (*simp add*: *zero-vec-def map-replicate-trivial*)

lemma *diag-mat-map*:
  assumes *M-carrier*: $M \in carrier\text{-}mat\ n\ n$
  shows $diag\text{-}mat\ (map\text{-}mat\ f\ M) = map\ f\ (diag\text{-}mat\ M)$
proof −
  have *dim-eq*: $dim\text{-}row\ M = dim\text{-}col\ M$ **using** *M-carrier* **by** *auto*
  have *m*: $map\text{-}mat\ f\ M\ \$\$\ (i, i) = f\ (M\ \$\$\ (i, i))$ **if** i: $i < dim\text{-}row\ M$ **for** i
    using *dim-eq i* **by** *auto*
  show *?thesis*
    by (*rule nth-equalityI*, *insert m*, *auto simp add*: *diag-mat-def M-carrier*)
qed

lemma *mat-of-rows-map* [*simp*]:
  assumes *x*: $set\ vs \subseteq carrier\text{-}vec\ n$
  shows $mat\text{-}of\text{-}rows\ n\ (map\ (map\text{-}vec\ f)\ vs) = map\text{-}mat\ f\ (mat\text{-}of\text{-}rows\ n\ vs)$
proof−
  have $\forall x \in set\ vs.\ dim\text{-}vec\ x = n$ **using** *x* **by** *auto*
  then show *?thesis* **by** (*auto simp add*: *mat-eq-iff map-vec-def mat-of-rows-def*)
qed

lemma *mat-of-cols-map* [*simp*]:

assumes x: *set vs* $\subseteq$ *carrier-vec n*
shows *mat-of-cols n* (*map* (*map-vec f*) *vs*) = *map-mat f* (*mat-of-cols n vs*)
proof−
have $\forall x \in$*set vs*. *dim-vec x* = *n* **using** *x* **by** *auto*
then show *?thesis* **by** (*auto simp add*: *mat-eq-iff map-vec-def mat-of-cols-def*)
qed

lemma *vec-of-list-map* [*simp*]: *vec-of-list* (*map f xs*) = *map-vec f* (*vec-of-list xs*)
unfolding *map-vec-def* **by** (*transfer*, *auto simp add*: *mk-vec-def*)

lemma *map-vec*: *map-vec f* (*vec n g*) = *vec n* (*f o g*) **by** *auto*

lemma *mat-of-cols-Cons-index-0*: *i* < *n* $\Longrightarrow$ *mat-of-cols n* (*w* # *ws*) \$\$ (*i*, *0*) = *w* \$ *i*
by (*unfold mat-of-cols-def*, *transfer'*, *auto simp*: *mk-mat-def*)

lemma *mat-of-cols-Cons-index-Suc*:
i < *n* $\Longrightarrow$ *mat-of-cols n* (*w* # *ws*) \$\$ (*i*, *Suc j*) = *mat-of-cols n ws* \$\$ (*i*,*j*)
by (*unfold mat-of-cols-def*, *transfer*, *auto simp*: *mk-mat-def undef-mat-def nth-append nth-map-out-of-bound*)

lemma *mat-of-cols-index*: *i* < *n* $\Longrightarrow$ *j* < *length ws* $\Longrightarrow$ *mat-of-cols n ws* \$\$ (*i*,*j*) = *ws* ! *j* \$ *i*
by (*unfold mat-of-cols-def*, *auto*)

lemma *mat-of-rows-index*: *i* < *length rs* $\Longrightarrow$ *j* < *n* $\Longrightarrow$ *mat-of-rows n rs* \$\$ (*i*,*j*) = *rs* ! *i* \$ *j*
by (*unfold mat-of-rows-def*, *auto*)

lemma *transpose-mat-of-rows*: (*mat-of-rows n vs*)^T = *mat-of-cols n vs*
by (*auto intro*!: *eq-matI simp*: *mat-of-rows-index mat-of-cols-index*)

lemma *transpose-mat-of-cols*: (*mat-of-cols n vs*)^T = *mat-of-rows n vs*
by (*auto intro*!: *eq-matI simp*: *mat-of-rows-index mat-of-cols-index*)

lemma *vec-of-poly-0* [*simp*]: *vec-of-poly 0* = 0_v *1* **by** (*auto simp*: *vec-of-poly-def*)

lemma *nth-list-of-vec* [*simp*]:
assumes *i* < *dim-vec v* **shows** *list-of-vec v* ! *i* = *v* \$ *i*
using *assms* **by** (*transfer*, *auto*)

lemma *length-list-of-vec* [*simp*]:
length (*list-of-vec v*) = *dim-vec v* **by** (*transfer*, *auto*)

lemma *vec-eq-0-iff*:
v = 0_v *n* $\longleftrightarrow$ *n* = *dim-vec v* $\wedge$ (*n* = *0* $\vee$ *set* (*list-of-vec v*) = {*0*}) (**is** *?l* $\longleftrightarrow$ *?r*)
proof

show *?l* $\Longrightarrow$ *?r* **by** *auto*
show *?r* $\Longrightarrow$ *?l* **by** (*intro iffI eq-vecI*, *force simp*: *set-conv-nth*, *force*)
qed

lemma *list-of-vec-vCons*[*simp*]: *list-of-vec* (*vCons a v*) = *a* # *list-of-vec v* (**is** *?l* = *?r*)
proof (*intro nth-equalityI allI impI*)
fix *i*
assume $i <$ *length ?l*
then show *?l* ! *i* = *?r* ! *i* **by** (*cases i*, *auto*)
qed *simp*

lemma *append-vec-vCons*[*simp*]: *vCons a v* $@_v$ *w* = *vCons a* (*v* $@_v$ *w*) (**is** *?l* = *?r*)
proof (*unfold vec-eq-iff*, *intro conjI allI impI*)
fix *i* **assume** $i <$ *dim-vec ?r*
then show *?l* \$ *i* = *?r* \$ *i* **by** (*cases i*; *subst index-append-vec*, *auto*)
qed *simp*

lemma *append-vec-vNil*[*simp*]: *vNil* $@_v$ *v* = *v*
by (*unfold vec-eq-iff*, *auto*)

lemma *list-of-vec-append*[*simp*]: *list-of-vec* (*v* $@_v$ *w*) = *list-of-vec v* @ *list-of-vec w*
by (*induct v*, *auto*)

lemma *transpose-mat-eq*[*simp*]: $A^T = B^T \longleftrightarrow A = B$
using *transpose-transpose* **by** *metis*

lemma *mat-col-eqI*: **assumes** *cols*: $\bigwedge$ *i*. $i <$ *dim-col B* $\Longrightarrow$ *col A i* = *col B i*
and *dims*: *dim-row A* = *dim-row B* *dim-col A* = *dim-col B*
shows $A = B$
by(*subst transpose-mat-eq*[*symmetric*], *rule eq-rowI*,*insert assms*,*auto*)

lemma *upper-triangular-imp-det-eq-0-iff*:
fixes *A* :: *'a* :: *idom mat*
assumes $A \in$ *carrier-mat n n* **and** *upper-triangular A*
shows *det A* = *0* $\longleftrightarrow$ *0* $\in$ *set* (*diag-mat A*)
using *assms* **by** (*auto simp*: *det-upper-triangular*)

lemma *upper-triangular-imp-distinct*:
fixes *u* :: *'a* :: {*zero-neq-one*} *poly*
assumes *A*: $A \in$ *carrier-mat n n*
and *tri*: *upper-triangular A*
and *diag*: *0* $\notin$ *set* (*diag-mat A*)
shows *distinct* (*rows A*)
proof−
{ **fix** *i* **and** *j*
assume *eq*: *rows A* ! *i* = *rows A* ! *j* **and** *ij*: $i < j$ **and** *jn*: $j < n$

from *tri A ij jn* **have** *rows A ! j $ i = 0* **by** (*auto dest*!:*upper-triangularD*)
with *eq* **have** *rows A ! i $ i = 0* **by** *auto*
with *diag ij jn A* **have** *False* **by** (*auto simp*: *diag-mat-def*)
}
with *A* **show** *?thesis* **by** (*force simp*: *distinct-conv-nth nat-neq-iff*)
qed

lemma *vec-index-vec-of-poly* [*simp*]: *i* ≤ *degree p* ⟹ *vec-of-poly p $ i* = *coeff p* (*degree p* − *i*)
by (*simp add*: *vec-of-poly-def Let-def*)

lemma *poly-of-vec-vec*: *poly-of-vec* (*vec n f*) = *Poly* (*rev* (*map f* [*0*..<*n*]))
proof (*induct n arbitrary*:*f*)
case *0*
then show *?case* **by** *auto*
next
case (*Suc n*)
have *map f* [*0*..<*Suc n*] = *f 0* # *map* (*f* ∘ *Suc*) [*0*..<*n*] **by** (*simp add*: *map-upt-Suc del*: *upt-Suc*)
also have *Poly* (*rev* ...) = *Poly* (*rev* (*map* (*f* ∘ *Suc*) [*0*..<*n*])) + *monom* (*f 0*) *n*
by (*simp add*: *Poly-snoc smult-monom*)
also have ... = *poly-of-vec* (*vec n* (*f* ∘ *Suc*)) + *monom* (*f 0*) *n*
by (*fold Suc*, *simp*)
also have ... = *poly-of-vec* (*vec* (*Suc n*) *f*)
apply (*unfold poly-of-vec-def Let-def dim-vec sum-lessThan-Suc*)
by (*auto simp add*: *Suc-diff-Suc*)
finally show *?case*..
qed

lemma *sum-list-map-dropWhile0*:
assumes *f0*: *f 0 = 0*
shows *sum-list* (*map f* (*dropWhile* (*op* = *0*) *xs*)) = *sum-list* (*map f xs*)
by (*induct xs*, *auto simp add*: *f0*)

lemma *coeffs-poly-of-vec*:
coeffs (*poly-of-vec v*) = *rev* (*dropWhile* (*op* = *0*) (*list-of-vec v*))
proof−
obtain *n f* **where** *v*: *v* = *vec n f* **by** *transfer auto*
show *?thesis* **by** (*simp add*: *v poly-of-vec-vec*)
qed

lemma *poly-of-vec-vCons*:
poly-of-vec (*vCons a v*) = *monom a* (*dim-vec v*) + *poly-of-vec v* (**is** *?l* = *?r*)
by (*auto intro*: *poly-eqI simp*: *coeff-poly-of-vec vec-index-vCons*)

```
lemma poly-of-vec-as-Poly: poly-of-vec v = Poly (rev (list-of-vec v))
  by (induct v, auto simp:poly-of-vec-vCons Poly-snoc ac-simps)

lemma poly-of-vec-add:
  assumes dim-vec a = dim-vec b
  shows poly-of-vec (a + b) = poly-of-vec a + poly-of-vec b
  using assms
  by (auto simp add: poly-eq-iff coeff-poly-of-vec)


lemma degree-poly-of-vec-less:
  assumes 0 < dim-vec v and dim-vec v ≤ n shows degree (poly-of-vec v) < n
  using degree-poly-of-vec-less assms by (auto dest: less-le-trans)

lemma (in vec-module) poly-of-vec-finsum:
  assumes f ∈ X → carrier-vec n
  shows poly-of-vec (finsum V f X) = (∑ i∈X. poly-of-vec (f i))
proof (cases finite X)
  case False then show ?thesis by auto
next
  case True show ?thesis
  proof (insert True assms, induct X rule: finite-induct)
    case IH: (insert a X)
    have [simp]: f x ∈ carrier-vec n if x: x ∈ X for x
      using x IH.prems unfolding Pi-def by auto
    have [simp]: f a ∈ carrier-vec n using IH.prems unfolding Pi-def by auto
    have [simp]: dim-vec (finsum V f X) = n by simp
    have [simp]: dim-vec (f a) = n by simp
    show ?case
    proof (cases a ∈ X)
      case True then show ?thesis by (auto simp: insert-absorb IH)
    next
      case False
      then have (finsum V f (insert a X)) = f a + (finsum V f X)
        by (auto intro: finsum-insert IH)
      also have poly-of-vec ... = poly-of-vec (f a) + poly-of-vec (finsum V f X)
        by (rule poly-of-vec-add, simp)
      also have ... = (∑ i∈insert a X. poly-of-vec (f i))
        using IH False by (subst sum.insert, auto)
      finally show ?thesis .
    qed
  qed auto
qed


definition vec-of-poly-n p n =
  vec n (λi. if i < n − degree p − 1 then 0 else coeff p (n − i − 1))
```

lemma *vec-of-poly-as*: *vec-of-poly-n p* (*Suc* (*degree p*)) = *vec-of-poly p*
  by (*induct p*, *auto simp*: *vec-of-poly-def vec-of-poly-n-def*)

lemma *vec-of-poly-n-0* [*simp*]: *vec-of-poly-n p 0* = *vNil*
  by (*auto simp*: *vec-of-poly-n-def*)

lemma *vec-dim-vec-of-poly-n* [*simp*]:
  dim-vec (*vec-of-poly-n p n*) = *n*
  vec-of-poly-n p n ∈ *carrier-vec n*
  unfolding *vec-of-poly-n-def* **by** *auto*

lemma *dim-vec-of-poly* [*simp*]: *dim-vec* (*vec-of-poly f*) = *degree f* + *1*
  by (*simp add*: *vec-of-poly-as*[*symmetric*])

lemma *vec-index-of-poly-n*:
  assumes *i* < *n*
  shows *vec-of-poly-n p n* $ *i* =
    (*if i* < *n* − *Suc* (*degree p*) *then 0 else coeff p* (*n* − *i* − *1*))
  using *assms* **by** (*auto simp*: *vec-of-poly-n-def Let-def*)

lemma *vec-of-poly-n-pCons*[*simp*]:
  shows *vec-of-poly-n* (*pCons a p*) (*Suc n*) = *vec-of-poly-n p n* $@_v$ *vec-of-list* [*a*]
(**is** *?l* = *?r*)
proof (*unfold vec-eq-iff*, *intro conjI allI impI*)
  show *dim-vec ?l* = *dim-vec ?r* **by** *auto*
  show *i* < *dim-vec ?r* ⟹ *?l* $ *i* = *?r* $ *i* **for** *i*
    by (*cases n* − *i*, *auto simp*: *coeff-pCons less-Suc-eq-le vec-index-of-poly-n*)
qed

lemma *vec-of-poly-pCons*:
  shows *vec-of-poly* (*pCons a p*) =
   (*if p* = *0 then vec-of-list* [*a*] *else vec-of-poly p* $@_v$ *vec-of-list* [*a*])
  by (*cases degree p*, *auto simp*: *vec-of-poly-as*[*symmetric*])

lemma *list-of-vec-of-poly* [*simp*]:
  list-of-vec (*vec-of-poly p*) = (*if p* = *0 then* [*0*] *else rev* (*coeffs p*))
  by (*induct p*, *auto simp*: *vec-of-poly-pCons*)

lemma *poly-of-vec-of-poly-n*:
  assumes *p*: *degree p*<*n*
  shows *poly-of-vec* (*vec-of-poly-n p n*) = *p*
proof −
  have *vec-of-poly-n p n* $ (*n* − *Suc i*) = *coeff p i* **if** *i*: *i* < *n* **for** *i*
  proof −
    have *n*: *n* − *Suc i* < *n* **using** *i* **by** *auto*
    have *vec-of-poly-n p n* $ (*n* − *Suc i*) =
      (*if n* − *Suc i* < *n* − *Suc* (*degree p*) *then 0 else coeff p* (*n* − (*n* − *Suc i*) −
1))

```
      using vec-index-of-poly-n[OF n, of p] .
    also have ... = coeff p i using i n le-degree by fastforce
    finally show ?thesis .
  qed
  moreover have coeff p i = 0 if i2: i ≥ n for i
    by (rule coeff-eq-0, insert i2 p, simp)
  ultimately show ?thesis
  using assms
  unfolding poly-eq-iff
  unfolding coeff-poly-of-vec by auto
qed

lemma vec-of-poly-n0[simp]: vec-of-poly-n 0 n = 0_v n
  unfolding vec-of-poly-n-def by auto

lemma vec-of-poly-n-add: vec-of-poly-n (a + b) n = vec-of-poly-n a n + vec-of-poly-n
b n
proof (induct n arbitrary: a b)
  case 0
  then show ?case by auto
next
  case (Suc n)
  then show ?case by (cases a, cases b, auto)
qed

lemma vec-of-poly-n-poly-of-vec:
  assumes n: dim-vec g = n
  shows vec-of-poly-n (poly-of-vec g) n = g
proof (auto simp add: poly-of-vec-def vec-of-poly-n-def assms vec-eq-iff Let-def)
  have d: degree (∑ i<n. monom (g $ (n − Suc i)) i) = degree (poly-of-vec g)
    unfolding poly-of-vec-def Let-def n by auto
  fix i assume i1: i < n − Suc (degree (∑ i<n. monom (g $ (n − Suc i)) i))
    and i2: i < n
  have i3: i < n − Suc (degree (poly-of-vec g))
    using i1 unfolding d by auto
  hence dim-vec g − Suc i > degree (poly-of-vec g)
    using n by linarith
  then show g $ i = 0 using i1 i2 i3
    by (metis (no-types, lifting) Suc-diff-Suc coeff-poly-of-vec diff-Suc-less
        diff-diff-cancel leD le-degree less-imp-le-nat n neq0-conv)
next
  fix i assume i < n
  thus coeff (∑ i<n. monom (g $ (n − Suc i)) i) (n − Suc i) = g $ i
    by (metis (no-types) Suc-diff-Suc coeff-poly-of-vec diff-diff-cancel
        diff-less-Suc less-imp-le-nat n not-less-eq poly-of-vec-def)
qed

lemma poly-of-vec-scalar-mult:
  assumes degree b<n
```

shows *poly-of-vec* $(a \cdot_v (vec\text{-}of\text{-}poly\text{-}n\ b\ n)) = smult\ a\ b$
using *assms*
by (*auto simp add*: *poly-eq-iff coeff-poly-of-vec vec-of-poly-n-def coeff-eq-0*)

definition *vec-of-poly-rev-shifted* **where**
vec-of-poly-rev-shifted p n s j ≡
vec n ($\lambda i.$ *if* $i \le j \land j \le s + i$ *then coeff p* $(s + i - j)$ *else 0*)

lemma *vec-of-poly-rev-shifted-dim*[*simp*]: *dim-vec* (*vec-of-poly-rev-shifted p n s j*) $= n$
unfolding *vec-of-poly-rev-shifted-def* **by** *auto*

lemma *col-sylvester-sub*:
assumes *j*: $j < m + n$
shows *col* (*sylvester-mat-sub m n p q*) $j =$
vec-of-poly-rev-shifted p n m j $@_v$ *vec-of-poly-rev-shifted q m n j* (**is** *?l* = *?r*)
proof
show *dim-vec ?l* = *dim-vec ?r* **by** *simp*
fix *i* **assume** $i <$ *dim-vec ?r* **then have** *i*: $i < m+n$ **by** *auto*
show *?l* \$ *i* = *?r* \$ *i*
unfolding *vec-of-poly-rev-shifted-def*
apply (*subst index-col*) **using** *i* **apply** *simp* **using** *j* **apply** *simp*
apply (*subst sylvester-mat-sub-index*) **using** *i* **apply** *simp* **using** *j* **apply** *simp*
apply (*cases* $i < n$) **using** *i* **apply** *force* **using** *i*
apply (*auto simp*: *not-less not-le intro*!: *coeff-eq-0*)
done
qed

lemma *vec-of-poly-rev-shifted-scalar-prod*:
fixes *p v*
defines *q* ≡ *poly-of-vec v*
assumes *m*: *degree p* $\le m$ **and** *n*: *dim-vec v* $= n$
assumes *j*: $j < m+n$
shows *vec-of-poly-rev-shifted p n m* $(n+m-Suc\ j) \cdot v =$ *coeff* $(p * q)\ j$ (**is** *?l* = *?r*)
proof −
have *id1*: $\bigwedge i.\ m + i - (n + m - Suc\ j) = i + Suc\ j - n$
using *j* **by** *auto*
let *?g* = λ *i*. *if* $i \le n + m - Suc\ j \land n - Suc\ j \le i$ *then coeff p* $(i + Suc\ j - n) *\ v$ \$ *i else 0*
have *?thesis* = $((\sum i = 0..<n.\ ?g\ i) =$
$(\sum i \le j.$ *coeff p i* $*$ (*if* $j - i < n$ *then v* \$ $(n - Suc\ (j - i))$ *else 0*))) (**is** - = (*?l* = *?r*))
unfolding *vec-of-poly-rev-shifted-def coeff-mult m scalar-prod-def n q-def coeff-poly-of-vec*
by (*subst sum.cong*, *insert id1*, *auto*)
also have ...
proof −

have $?r = (\sum i{\le}j.\ (\mathit{if}\ j - i < n\ \mathit{then}\ \mathit{coeff}\ p\ i * v\ \$\ (n - \mathit{Suc}\ (j - i))\ \mathit{else}\ 0))$ (**is** - = *sum ?f* -)
by (*rule sum.cong*, *auto*)
also have $\mathit{sum}\ ?f\ \{..j\} = \mathit{sum}\ ?f\ (\{i.\ i \le j \wedge j - i < n\} \cup \{i.\ i \le j \wedge \neg\ j - i < n\})$
(**is** - = *sum* - (*?R1* ∪ *?R2*))
by (*rule sum.cong*, *auto*)
also have ... $= \mathit{sum}\ ?f\ ?R1 + \mathit{sum}\ ?f\ ?R2$
by (*subst sum.union-disjoint*, *auto*)
also have $\mathit{sum}\ ?f\ ?R2 = 0$
by (*rule sum.neutral*, *auto*)
also have $\mathit{sum}\ ?f\ ?R1 + 0 = \mathit{sum}\ (\lambda\ i.\ \mathit{coeff}\ p\ i * v\ \$\ (i + n - \mathit{Suc}\ j))\ ?R1$
(**is** - = *sum ?F* -)
by (*subst sum.cong*, *auto simp*: *ac-simps*)
also have ... $= \mathit{sum}\ ?F\ ((?R1 \cap \{..m\}) \cup (?R1 - \{..m\}))$
(**is** - = *sum* - (*?R* ∪ *?R′*))
by (*rule sum.cong*, *auto*)
also have ... $= \mathit{sum}\ ?F\ ?R + \mathit{sum}\ ?F\ ?R'$
by (*subst sum.union-disjoint*, *auto*)
also have $\mathit{sum}\ ?F\ ?R' = 0$
proof −
{
fix x
assume $x > m$
with m
have $?F\ x = 0$ **by** (*subst coeff-eq-0*, *auto*)
}
thus *?thesis*
by (*subst sum.neutral*, *auto*)
qed
finally have *r*: $?r = \mathit{sum}\ ?F\ ?R$ **by** *simp*

have $?l = \mathit{sum}\ ?g\ (\{i.\ i < n \wedge i \le n + m - \mathit{Suc}\ j \wedge n - \mathit{Suc}\ j \le i\}$
$\cup \{i.\ i < n \wedge \neg\ (i \le n + m - \mathit{Suc}\ j \wedge n - \mathit{Suc}\ j \le i)\})$
(**is** - = *sum* - (*?L1* ∪ *?L2*))
by (*rule sum.cong*, *auto*)
also have ... $= \mathit{sum}\ ?g\ ?L1 + \mathit{sum}\ ?g\ ?L2$
by (*subst sum.union-disjoint*, *auto*)
also have $\mathit{sum}\ ?g\ ?L2 = 0$
by (*rule sum.neutral*, *auto*)
also have $\mathit{sum}\ ?g\ ?L1 + 0 = \mathit{sum}\ (\lambda\ i.\ \mathit{coeff}\ p\ (i + \mathit{Suc}\ j - n) * v\ \$\ i)\ ?L1$
(**is** - = *sum ?G* -)
by (*subst sum.cong*, *auto*)
also have ... $= \mathit{sum}\ ?G\ (?L1 \cap \{i.\ i + \mathit{Suc}\ j - n \le m\} \cup (?L1 - \{i.\ i + \mathit{Suc}\ j - n \le m\}))$
(**is** - = *sum* - (*?L* ∪ *?L′*))
by (*subst sum.cong*, *auto*)
also have ... $= \mathit{sum}\ ?G\ ?L + \mathit{sum}\ ?G\ ?L'$
by (*subst sum.union-disjoint*, *auto*)

```
    also have sum ?G ?L' = 0
    proof -
      {
        fix x
        assume x + Suc j - n > m
        with m
        have ?G x = 0 by (subst coeff-eq-0, auto)
      }
      thus ?thesis
        by (subst sum.neutral, auto)
    qed
    finally have l: ?l = sum ?G ?L by simp

    let ?bij = λ i. i + n - Suc j
    {
      fix x
      assume x: j < m + n Suc (x + j) - n ≤ m x < n n - Suc j ≤ x
      define y where y = x + Suc j - n
      from x have x + Suc j ≥ n by auto
      with x have xy: x = ?bij y unfolding y-def by auto
      from x have y: y ∈ ?R unfolding y-def by auto
      have x ∈ ?bij ‘ ?R unfolding xy using y by blast
    } note tedious = this
    show ?thesis unfolding l r
      by (rule sum.reindex-cong[of ?bij], insert j, auto simp: inj-on-def tedious)
  qed
  finally show ?thesis by simp
qed

lemma sylvester-sub-poly:
  fixes p q :: 'a :: comm-semiring-0 poly
  assumes m: degree p ≤ m
  assumes n: degree q ≤ n
  assumes v: v ∈ carrier-vec (m+n)
  shows poly-of-vec ((sylvester-mat-sub m n p q)^T *_v v) =
    poly-of-vec (vec-first v n) * p + poly-of-vec (vec-last v m) * q (is ?l = ?r)
proof (rule poly-eqI)
  fix i
  let ?Tv = (sylvester-mat-sub m n p q)^T *_v v
  have dim: dim-vec (vec-first v n) = n dim-vec (vec-last v m) = m dim-vec ?Tv
= n + m
    using v by auto
  have if-distrib: ⋀ x y z. (if x then y else (0 :: 'a)) * z = (if x then y * z else 0)
    by auto
  show coeff ?l i = coeff ?r i
  proof (cases i < m+n)
    case False
      hence i-mn: i ≥ m+n
        and i-n: ⋀x. x ≤ i ∧ x < n ⟷ x < n
```

and *i-m*: $\bigwedge x.\ x \le i \wedge x < m \longleftrightarrow x < m$ **by** *auto*
have *coeff ?r i* =
($\sum$ $x < n$. *vec-first v n* \$ $(n - Suc\ x) * coeff\ p\ (i - x)$) +
($\sum$ $x < m$. *vec-last v m* \$ $(m - Suc\ x) * coeff\ q\ (i - x)$)
(**is** - = *sum ?f* - + *sum ?g* -)
unfolding *coeff-add coeff-mult Let-def*
unfolding *coeff-poly-of-vec dim if-distrib*
unfolding *atMost-def*
apply(*subst sum.inter-filter*[*symmetric*],*simp*)
apply(*subst sum.inter-filter*[*symmetric*],*simp*)
unfolding *mem-Collect-eq*
unfolding *i-n i-m*
unfolding *lessThan-def* **by** *simp*
also { **fix** *x* **assume** *x*: $x < n$
have *coeff p* $(i-x) = 0$
apply(*rule coeff-eq-0*) **using** *i-mn x m* **by** *auto*
hence *?f x* = *0* **by** *auto*
} **hence** *sum ?f* {..<*n*} = *0* **by** *auto*
also { **fix** *x* **assume** *x*: $x < m$
have *coeff q* $(i-x) = 0$
apply(*rule coeff-eq-0*) **using** *i-mn x n* **by** *auto*
hence *?g x* = *0* **by** *auto*
} **hence** *sum ?g* {..<*m*} = *0* **by** *auto*
finally have *coeff ?r i* = *0* **by** *auto*
also from *False* **have** *0* = *coeff ?l i*
unfolding *coeff-poly-of-vec dim sum.distrib*[*symmetric*] **by** *auto*
finally show *?thesis* **by** *auto*
next case *True*
hence *coeff ?l i* = $((sylvester\text{-}mat\text{-}sub\ m\ n\ p\ q)^T *_v v)$ \$ $(n + m - Suc\ i)$
unfolding *coeff-poly-of-vec dim sum.distrib*[*symmetric*] **by** *auto*
also have ... = *coeff* (*p* $*$ *poly-of-vec* (*vec-first v n*) + *q* $*$ *poly-of-vec* (*vec-last v m*)) *i*
apply(*subst index-mult-mat-vec*) **using** *True* **apply** *simp*
apply(*subst row-transpose*) **using** *True* **apply** *simp*
apply(*subst col-sylvester-sub*)
using *True* **apply** *simp*
apply(*subst vec-first-last-append*[*of v n m*,*symmetric*]) **using** *v* **apply**(*simp add*: *add.commute*)
apply(*subst scalar-prod-append*)
apply (*rule carrier-vecI*,*simp*)+
apply (*subst vec-of-poly-rev-shifted-scalar-prod*[*OF m*],*simp*) **using** *True* **apply** *simp*
apply (*subst add.commute*[*of n m*])
apply (*subst vec-of-poly-rev-shifted-scalar-prod*[*OF n*]) **apply** *simp* **using** *True* **apply** *simp*
by *simp*
also have ... =
($\sum x \le i$. (*if* $x < n$ *then vec-first v n* \$ $(n - Suc\ x)$ *else 0*) $*$ *coeff p* $(i - x)$) +

($\sum x \le i$. (*if* $x < m$ *then vec-last v m* \$ $(m - Suc\ x)$ *else 0*) $*$ *coeff q* $(i - x)$)
unfolding *coeff-poly-of-vec*[*of vec-first v n*,*unfolded dim-vec-first*,*symmetric*]
unfolding *coeff-poly-of-vec*[*of vec-last v m*,*unfolded dim-vec-last*,*symmetric*]
unfolding *coeff-mult*[*symmetric*] **by** (*simp add*: *mult.commute*)
also have ... = *coeff ?r i*
unfolding *coeff-add coeff-mult Let-def*
unfolding *coeff-poly-of-vec dim*..
finally show *?thesis*.
qed
qed

lemma *normalize-field* [*simp*]: *normalize* (*a* :: *'a* :: {*field*, *semiring-gcd*}) = (*if a* = *0 then 0 else 1*)
using *unit-factor-normalize* **by** *fastforce*

lemma *content-field* [*simp*]: *content* (*p* :: *'a* :: {*field*,*semiring-gcd*} *poly*) = (*if p* = *0 then 0 else 1*)
by (*induct p*, *auto simp*: *content-def*)

lemma *primitive-part-field* [*simp*]: *primitive-part* (*p* :: *'a* :: {*field*,*semiring-gcd*} *poly*) = *p*
by (*cases p* = *0*, *auto intro*!: *primitive-part-prim*)

lemma *primitive-part-dvd*: *primitive-part a dvd a*
by (*metis content-times-primitive-part dvd-def dvd-refl mult-smult-right*)

lemma *degree-abs* [*simp*]:
degree $|p|$ = *degree p* **by** (*auto simp*: *abs-poly-def*)

lemma *degree-gcd1*:
assumes *a-not0*: $a \neq 0$
shows *degree* (*gcd a b*) $\le$ *degree a*
proof −
let *?g* = *gcd a b*
have *gcd-dvd-b*: *?g dvd a* **by** *simp*
from *this* **obtain** *c* **where** *a-gc*: *a* = *?g* $*$ *c* **unfolding** *dvd-def* **by** *auto*
have *g-not0*: *?g* $\neq$*0* **using** *a-not0 a-gc* **by** *auto*
have *c0*: $c \neq 0$ **using** *a-not0 a-gc* **by** *auto*
have *degree ?g* $\le$ *degree* (*?g* $*$ *c*) **by** (*rule degree-mult-right-le*[*OF c0*])
also have ... = *degree a* **using** *a-gc* **by** *auto*
finally show *?thesis* .
qed

lemma *primitive-part-neg* [*simp*]:

fixes a::'a :: *factorial-ring-gcd poly*
shows *primitive-part* $(-a)$ = − *primitive-part a*
proof −
have *primitive-part* $(-a)$ = *primitive-part* (*smult* (−*1*) a) **by** *auto*
then show *?thesis* **unfolding** *primitive-part-smult*
by (*simp add*: *is-unit-unit-factor*)
qed

lemma *content-uminus*[*simp*]:
fixes f::*int poly*
shows *content* $(-f)$ = *content f*
proof −
have $-f = -$ (*smult 1 f*) **by** *auto*
also have ... = *smult* (−*1*) f **using** *smult-minus-left* **by** *auto*
finally have *content* $(-f)$ = *content* (*smult* (−*1*) f) **by** *auto*
also have ... = *normalize* (− *1*) ∗ *content f* **unfolding** *content-smult* **..**
finally show *?thesis* **by** *auto*
qed

lemma *pseudo-mod-monic*:
fixes f g :: 'a::{*comm-ring-1*,*semiring-1-no-zero-divisors*} *poly*
defines r ≡ *pseudo-mod f g*
assumes *monic-g*: *monic g*
shows ∃ q. $f = g * q + r$ $r = 0$ ∨ *degree r* < *degree g*
proof −
let *?cg* = *coeff g* (*degree g*)
let *?cge* = *?cg* ˆ (*Suc* (*degree f*) − *degree g*)
define a **where** a = *?cge*
from *r-def*[*unfolded pseudo-mod-def*] **obtain** q **where** *pdm*: *pseudo-divmod f g* = (q, r)
by (*cases pseudo-divmod f g*) *auto*
have g: $g \neq 0$ **using** *monic-g* **by** *auto*
from *pseudo-divmod*[*OF g pdm*] **have** *id*: *smult a f* = $g * q + r$ **and** $r = 0$ ∨ *degree r* < *degree g*
by (*auto simp*: *a-def*)
have *a1*: $a = 1$ **unfolding** *a-def* **using** *monic-g* **by** *auto*
hence *id2*: $f = g * q + r$ **using** *id* **by** *auto*
show $r = 0$ ∨ *degree r* < *degree g* **by** *fact*
from g **have** $a \neq 0$
by (*auto simp*: *a-def*)
with *id2* **show** ∃ q. $f = g * q + r$
by *auto*
qed

lemma *monic-imp-div-mod-int-poly-degree*:
fixes p :: 'a::{*comm-ring-1*,*semiring-1-no-zero-divisors*} *poly*
assumes m: *monic u*
shows ∃ q r. $p = q{*}u + r$ ∧ ($r = 0$ ∨ *degree r* < *degree u*)
using *pseudo-mod-monic*[*OF m*] **using** *mult.commute* **by** *metis*

corollary *monic-imp-div-mod-int-poly-degree2*:
  fixes p :: $'a$::{*comm-ring-1*,*semiring-1-no-zero-divisors*} *poly*
  assumes m: *monic* u **and** *deg-u*: *degree* $u > 0$
  shows $\exists\, q\ r.\ p = q{*}u + r \wedge$ (*degree* $r <$ *degree* u)
proof −
  obtain $q\ r$ **where** $p = q * u + r$ **and** r: ($r = 0 \vee$ *degree* $r <$ *degree* u)
      using *monic-imp-div-mod-int-poly-degree*[*OF* m, *of* p] **by** *auto*
    moreover have *degree* $r <$ *degree* u **using** *deg-u* r **by** *auto*
  ultimately show *?thesis* **by** *auto*
qed

lemma *det-identical-columns*:
  assumes A: $A \in$ *carrier-mat* n n
    and *ij*: $i \neq j$
    and i: $i < n$ **and** j: $j < n$
    and r: *col* A i = *col* A j
  shows *det* $A = 0$
proof−
  have *det* A = *det* A^T **using** *det-transpose*[*OF* A] **..**
  also have ... = *0*
  proof (*rule det-identical-rows*[*of* - n i j])
     show *row* (*transpose-mat* A) i = *row* (*transpose-mat* A) j
       using A i j r **by** *auto*
  qed (*auto simp add*: *assms*)
  finally show *?thesis* **.**
qed

lemma *irreducible-uminus* [*simp*]:
  fixes a::$'a$::*idom*
  shows *irreducible* $(-a) \longleftrightarrow$ *irreducible* a
  using *irreducible-mult-unit-left*[*of* -1::$'a$] **by** *auto*

context *poly-mod*
begin

lemma *dvd-imp-dvdm*:
  assumes a *dvd* b **shows** a *dvdm* b
  by (*metis assms dvd-def dvdm-def*)

```
lemma dvdm-add:
  assumes a: u dvdm a
  and b: u dvdm b
  shows u dvdm (a+b)
proof −
  obtain a' where a: a =m u*a' using a unfolding dvdm-def by auto
  obtain b' where b: b =m u*b' using b unfolding dvdm-def by auto
  have Mp (a + b) = Mp (u*a'+u*b') using a b
    by (metis poly-mod.plus-Mp(1) poly-mod.plus-Mp(2))
  also have ... = Mp (u * (a'+ b'))
    by (simp add: distrib-left)
  finally show ?thesis unfolding dvdm-def by auto
qed


lemma monic-dvdm-constant:
  assumes uk: u dvdm [:k:]
  and u1: monic u and u2: degree u > 0
  shows k mod m = 0
proof −
  have d1: degree-m [:k:] = degree [:k:]
    by (metis degree-pCons-0 le-zero-eq poly-mod.degree-m-le)
  obtain h where h: Mp [:k:] = Mp (u * h)
    using uk unfolding dvdm-def by auto
  have d2: degree-m [:k:] = degree-m (u*h) using h by metis
  have d3: degree (map-poly M (u * map-poly M h)) = degree (u * map-poly M
h)
    by (rule degree-map-poly)
       (metis coeff-degree-mult leading-coeff-0-iff mult.right-neutral M-M Mp-coeff
Mp-def u1)
  thus ?thesis using assms d1 d2 d3
    by (auto, metis M-def map-poly-pCons degree-mult-right-le h leD map-poly-0
       mult-poly-0-right pCons-eq-0-iff M-0 Mp-def mult-Mp(2))
qed

lemma dvdm-imp-div-mod:
  assumes u dvdm g
  shows ∃ q r. g = q*u + smult m r
proof −
  obtain q where q: Mp g = Mp (u*q)
    using assms unfolding dvdm-def by fast
  have (u*q) = Mp (u*q) + smult m (Dp (u*q))
    by (simp add: poly-mod.Dp-Mp-eq[of u*q])
  hence uq: Mp (u*q) = (u*q) − smult m (Dp (u*q))
    by auto
  have g: g = Mp g + smult m (Dp g)
    by (simp add: poly-mod.Dp-Mp-eq[of g])
  also have ... = poly-mod.Mp m (u*q) + smult m (Dp g) using q by simp
```

```
  also have ... = u ∗ q − smult m (Dp (u ∗ q)) + smult m (Dp g)
    unfolding uq by auto
  also have ... = u ∗ q + smult m (−Dp (u∗q)) + smult m (Dp g) by auto
  also have ... = u ∗ q + smult m (−Dp (u∗q) + Dp g)
    unfolding smult-add-right by auto
  also have ... = q ∗ u + smult m (−Dp (u∗q) + Dp g) by auto
  finally show ?thesis by auto
qed

lemma div-mod-imp-dvdm:
  assumes ∃ q r. b = q ∗ a + Polynomial.smult m r
  shows a dvdm b
proof −
  from assms obtain q r where b:b = a ∗ q + smult m r
    by (metis mult.commute)
  have a: Mp (Polynomial.smult m r) = 0 by auto
  show ?thesis
  proof (unfold dvdm-def, rule exI[of - q])
    have Mp (a ∗ q + smult m r) = Mp (a ∗ q + Mp (smult m r))
      using plus-Mp(2)[of a∗q smult m r] by auto
    also have ... = Mp (a∗q) by auto
    finally show eq-m b (a ∗ q) using b by auto
  qed
qed

corollary div-mod-iff-dvdm:
  shows a dvdm b = (∃ q r. b = q ∗ a + Polynomial.smult m r)
  using div-mod-imp-dvdm dvdm-imp-div-mod by blast

lemma dvdmE:
  assumes p dvdm q and ⋀r. q =m p ∗ Mp r ⟹ thesis
  shows thesis
  using assms by (auto simp: dvdm-def)

lemma lead-coeff-monic-mult:
  fixes p :: 'a :: {comm-semiring-1,semiring-no-zero-divisors} poly
  assumes monic p shows lead-coeff (p ∗ q) = lead-coeff q
  using assms by (simp add: lead-coeff-mult)

lemma degree-m-mult-eq:
  assumes p: monic p and q: lead-coeff q mod m ≠ 0 and m1: m > 1
  shows degree (Mp (p ∗ q)) = degree p + degree q
proof−
  have lead-coeff (p ∗ q) mod m ≠ 0
    using q p by (auto simp: lead-coeff-monic-mult)
  with m1 show ?thesis
    by (auto simp: degree-m-eq intro!: degree-mult-eq)
qed
```

lemma *dvdm-imp-degree-le*:
  assumes *pq*: *p dvdm q* **and** *p*: *monic p* **and** *q0*: *Mp q* $\neq$ *0* **and** *m1*: *m* > *1*
  shows *degree p* $\leq$ *degree q*
proof−
  from *q0*
  have *q*: *lead-coeff* (*Mp q*) *mod m* $\neq$ *0*
    by (*metis Mp-Mp Mp-coeff leading-coeff-neq-0 M-def*)
  from *pq* **obtain** *r* **where** *Mpq*: *Mp q* = *Mp* (*p* ∗ *Mp r*) **by** (*auto elim*: *dvdmE*)
  with *p q* **have** *lead-coeff* (*Mp r*) *mod m* $\neq$ *0*
    by (*metis Mp-Mp Mp-coeff leading-coeff-0-iff mult-poly-0-right M-def*)
  from *degree-m-mult-eq*[*OF p this m1*] *Mpq*
  have *degree p* $\leq$ *degree-m q* **by** *simp*
  thus *?thesis* **using** *degree-m-le le-trans* **by** *blast*
qed

lemma *dvdm-uminus* [*simp*]:
  p dvdm −*q* $\longleftrightarrow$ *p dvdm q*
  by (*metis add.inverse-inverse dvdm-smult smult-1-left smult-minus-left*)

lemma *Mp-const-poly*: *Mp* [:*a*:] = [:*a mod m*:]
  by (*simp add*: *Mp-def M-def Polynomial.map-poly-pCons*)

end

context *poly-mod-2*
begin
lemma *factorization-m-mem-dvdm*: **assumes** *fact*: *factorization-m f* (*c*,*fs*)
  and *mem*: *Mp g* ∈# *image-mset Mp fs*
shows *g dvdm f*
proof −
  from *fact* **have** *factorization-m f* (*Mf* (*c*, *fs*)) **by** *auto*
  then obtain *l* **where** *f*: *factorization-m f* (*l*, *image-mset Mp fs*) **by** (*auto simp*: *Mf-def*)
  from *multi-member-split*[*OF mem*] **obtain** *ls* **where**
    fs: *image-mset Mp fs* = {# *Mp g* #} + *ls* **by** *auto*
  from *f*[*unfolded fs split factorization-m-def*] **show** *g dvdm f*
    unfolding *dvdm-def*
    by (*intro exI*[*of - smult l* (*prod-mset ls*)], *auto simp del*: *Mp-smult*
        simp add: *Mp-smult*(*2*)[*of - Mp g* ∗ *prod-mset ls*, *symmetric*], *simp*)
qed

lemma *dvdm-degree*: *monic u* $\Longrightarrow$ *u dvdm f* $\Longrightarrow$ *Mp f* $\neq$ *0* $\Longrightarrow$ *degree u* $\leq$ *degree f*
  using *dvdm-imp-degree-le m1* **by** *blast*
end

context *poly-mod-prime*

begin
lemma *pl-dvdm-imp-p-dvdm*:
  assumes *l0*: $l \neq 0$
  and *pl-dvdm*: *poly-mod.dvdm* (*p*ˆ*l*) *a b*
  shows *a dvdm b*
proof −
  from *l0* **have** *l-gt-0*: $l > 0$ **by** *auto*
  with *m1* **interpret** *pl*: *poly-mod-2 p*ˆ*l* **by** (*unfold-locales*, *auto*)
  have *p-rw*: $p * p$ ˆ $(l - 1) = p$ ˆ *l* **by** (*rule power-minus-simp*[*symmetric*, *OF l-gt-0*])
  obtain *q r* **where** *b*: $b = q * a +$ *smult* (*p*ˆ*l*) *r* **using** *pl.dvdm-imp-div-mod*[*OF pl-dvdm*] **by** *auto*
  have *smult* (*p*ˆ*l*) *r* = *smult p* (*smult* (*p* ˆ $(l - 1)$) *r*) **unfolding** *smult-smult p-rw* **..**
  hence *b2*: $b = q * a +$ *smult p* (*smult* (*p* ˆ $(l - 1)$) *r*) **using** *b* **by** *auto*
  show *?thesis*
    by (*rule div-mod-imp-dvdm*, *rule exI*[*of - q*],
        rule exI[*of -* (*smult* (*p* ˆ $(l - 1)$) *r*)], *auto simp add*: *b2*)
qed

lemma *coprime-exp-mod*: *coprime lu p* ⟹ $n \neq 0$ ⟹ *lu mod p* ˆ $n \neq 0$
  using *prime* **by** *fastforce*

lemma *unique-factorization-m-factor-partition*: **assumes** *l0*: $l \neq 0$
  and *uf*: *poly-mod.unique-factorization-m* (*p*ˆ*l*) *f* (*lead-coeff f*, *mset gs*)
  and *f*: $f = f1 * f2$
  and *cop*: *coprime* (*lead-coeff f*) *p*
  and *sf*: *square-free-m f*
  and *part*: *partition* (λ*gi*. *gi dvdm f1*) *gs* = (*gs1*, *gs2*)
shows *poly-mod.unique-factorization-m* (*p*ˆ*l*) *f1* (*lead-coeff f1*, *mset gs1*)
  poly-mod.unique-factorization-m (*p*ˆ*l*) *f2* (*lead-coeff f2*, *mset gs2*)
proof −
  interpret *pl*: *poly-mod-2 p*ˆ*l* **by** (*standard*, *insert m1 l0*, *auto*)
  let *?I* = *image-mset pl.Mp*
  note *Mp-pow* [*simp*] = *Mp-Mp-pow-is-Mp*[*OF l0 m1*]
  have [*simp*]: *pl.Mp x dvdm u* = (*x dvdm u*) **for** *x u* **unfolding** *dvdm-def* **using** *Mp-pow*[*of x*]
    by (*metis poly-mod.mult-Mp*(*1*))
  have *gs-split*: *set gs* = *set gs1* ∪ *set gs2* **using** *part* **by** *auto*
  from *pl.unique-factorization-m-factor*[*OF prime uf*[*unfolded f*] *- - l0 refl*, *folded f*, *OF cop sf*]
  obtain *hs1 hs2* **where** *uf'*: *pl.unique-factorization-m f1* (*lead-coeff f1*, *hs1*)
      pl.unique-factorization-m f2 (*lead-coeff f2*, *hs2*)
    and *gs-hs*: *?I* (*mset gs*) = *hs1* + *hs2*
    unfolding *pl.Mf-def split* **by** *auto*

```
  have gs-gs: ?I (mset gs) = ?I (mset gs1) + ?I (mset gs2) using part
    by (auto, induct gs arbitrary: gs1 gs2, auto)
  with gs-hs have gs-hs12: ?I (mset gs1) + ?I (mset gs2) = hs1 + hs2 by auto
  note pl-dvdm-imp-p-dvdm = pl-dvdm-imp-p-dvdm[OF l0]
  note fact = pl.unique-factorization-m-imp-factorization[OF uf]
  have gs1: ?I (mset gs1) = {#x ∈# ?I (mset gs). x dvdm f1#}
    using part by (auto, induct gs arbitrary: gs1 gs2, auto)
   also have ... = {#x ∈# hs1. x dvdm f1#} + {#x ∈# hs2. x dvdm f1#}
unfolding gs-hs by simp
  also have {#x ∈# hs2. x dvdm f1#} = {#}
  proof (rule ccontr)
    assume ¬ ?thesis
    then obtain x where x: x ∈# hs2 and dvd: x dvdm f1 by fastforce
    from x gs-hs have x ∈# ?I (mset gs) by auto
    with fact[unfolded pl.factorization-m-def]
    have xx: pl.irreducible_d-m x monic x by auto
    from square-free-m-prod-imp-coprime-m[OF sf[unfolded f]]
    have cop-h-f: coprime-m f1 f2 by auto
   from pl.factorization-m-mem-dvdm[OF pl.unique-factorization-m-imp-factorization[OF
uf'(2)], of x] x
    have pl.dvdm x f2 by auto
    hence x dvdm f2 by (rule pl-dvdm-imp-p-dvdm)
    from cop-h-f[unfolded coprime-m-def, rule-format, OF dvd this]
    have x dvdm 1 by auto
    from dvdm-imp-degree-le[OF this xx(2) - m1] have degree x = 0 by auto
    with xx show False unfolding pl.irreducible_d-m-def by auto
  qed
  also have {#x ∈# hs1. x dvdm f1#} = hs1
  proof (rule ccontr)
    assume ¬ ?thesis
    from filter-mset-inequality[OF this]
    obtain x where x: x ∈# hs1 and dvd: ¬ x dvdm f1 by blast
   from pl.factorization-m-mem-dvdm[OF pl.unique-factorization-m-imp-factorization[OF
uf'(1)],
      of x] x dvd
    have pl.dvdm x f1 by auto
    from pl-dvdm-imp-p-dvdm[OF this] dvd show False by auto
  qed
  finally have gs-hs1: ?I (mset gs1) = hs1 by simp
  with gs-hs12 have ?I (mset gs2) = hs2 by auto
  with uf' gs-hs1 have pl.unique-factorization-m f1 (lead-coeff f1, ?I (mset gs1))
     pl.unique-factorization-m f2 (lead-coeff f2, ?I (mset gs2)) by auto
  thus pl.unique-factorization-m f1 (lead-coeff f1, mset gs1)
     pl.unique-factorization-m f2 (lead-coeff f2, mset gs2)
    unfolding pl.unique-factorization-m-def
    by (auto simp: pl.Mf-def image-mset.compositionality o-def)
qed
end
```

definition *find-indices* **where** *find-indices x xs* ≡ [*i* ← [*0..<length xs*]. *xs*!*i* = *x*]

lemma *find-indices-Nil* [*simp*]:
  find-indices x [] = []
  by (*simp add*: *find-indices-def*)

lemma *find-indices-Cons*:
  find-indices x (*y#ys*) = (*if x* = *y then Cons 0 else id*) (*map Suc* (*find-indices x ys*))
apply (*unfold find-indices-def length-Cons*, *subst upt-conv-Cons*, *simp*)
apply (*fold map-Suc-upt*, *auto simp*: *filter-map o-def*) **done**

lemma *find-indices-snoc* [*simp*]:
  find-indices x (*ys*@[*y*]) = *find-indices x ys* @ (*if x* = *y then* [*length ys*] *else* [])
  by (*unfold find-indices-def*, *auto intro*!: *filter-cong simp*: *nth-append*)

lemma *mem-set-find-indices* [*simp*]: $i \in$ *set* (*find-indices x xs*) $\longleftrightarrow i <$ *length xs* $\wedge$ *xs*!*i* = *x*
  by (*auto simp*: *find-indices-def*)

lemma *distinct-find-indices*: *distinct* (*find-indices x xs*)
  unfolding *find-indices-def* **by** *simp*

context *module* **begin**

definition *lincomb-list*
where *lincomb-list c vs* = *sumlist* (*map* ($\lambda i.\ c\ i \odot_M vs\ !\ i$) [*0..<length vs*])

lemma *lincomb-list-carrier*:
  assumes *set vs* ⊆ *carrier M* **and** *c* : {*0..<length vs*} → *carrier R*
  shows *lincomb-list c vs* ∈ *carrier M*
 by (*insert assms*, *unfold lincomb-list-def*, *intro sumlist-carrier*, *auto intro*!: *smult-closed*)

lemma *lincomb-list-Nil* [*simp*]: *lincomb-list c* [] = $\mathbf{0}_M$
  by (*simp add*: *lincomb-list-def*)

lemma *lincomb-list-Cons* [*simp*]:
  lincomb-list c (*v#vs*) = $c\ 0 \odot_M v \oplus_M$ *lincomb-list* (*c o Suc*) *vs*
 by (*unfold lincomb-list-def length-Cons*, *subst upt-conv-Cons*, *simp*, *fold map-Suc-upt*, *simp add*: *o-def*)

lemma *lincomb-list-eq-0*:
  assumes $\bigwedge i.\ i <$ *length vs* $\Longrightarrow c\ i \odot_M vs\ !\ i = \mathbf{0}_M$
  shows *lincomb-list c vs* = $\mathbf{0}_M$

proof (*insert assms*, *induct vs arbitrary*:*c*)
  case (*Cons v vs*)
  from *Cons.prems*[*of 0*] **have** [*simp*]: $c\ 0 \odot_M v = \mathbf{0}_M$ **by** *auto*
  from *Cons.prems*[*of Suc -*] *Cons.hyps* **have** *lincomb-list* $(c \circ Suc)\ vs = \mathbf{0}_M$ **by** *auto*
  then show *?case* **by** (*simp add*: *o-def*)
qed *simp*

definition *mk-coeff* **where** *mk-coeff vs c v* ≡ *R.sumlist* (*map c* (*find-indices v vs*))

lemma *mk-coeff-carrier*:
  assumes *c* : {*0..<length vs*} → *carrier R* **shows** *mk-coeff vs c w* ∈ *carrier R*
  by (*insert assms*, *auto simp*: *mk-coeff-def find-indices-def intro*!:*R.sumlist-carrier elim*!:*funcset-mem*)

lemma *mk-coeff-Cons*:
  assumes *c* : {*0..<length* (*v#vs*)} → *carrier R*
  shows *mk-coeff* (*v#vs*) *c* = (λ*w*. (*if w = v then c 0 else* **0**) ⊕ *mk-coeff vs* (*c o Suc*) *w*)
proof−
  from *assms* **have** *c o Suc* : {*0..<length vs*} → *carrier R* **by** *auto*
  from *mk-coeff-carrier*[*OF this*] *assms*
  show *?thesis* **by** (*auto simp add*: *mk-coeff-def find-indices-Cons*)
qed

lemma *mk-coeff-0*[*simp*]:
  assumes *v* ∉ *set vs*
  shows *mk-coeff vs c v* = **0**
proof −
  have (*find-indices v vs*) = [] **using** *assms* **unfolding** *find-indices-def*
    by (*simp add*: *in-set-conv-nth*)
  thus *?thesis* **unfolding** *mk-coeff-def* **by** *auto*
qed

lemma *lincomb-list-as-lincomb*:
  assumes *vs-M*: *set vs* ⊆ *carrier M* **and** *c*: *c* : {*0..<length vs*} → *carrier R*
  shows *lincomb-list c vs* = *lincomb* (*mk-coeff vs c*) (*set vs*)
proof (*insert assms*, *induct vs arbitrary*: *c*)
  case (*Cons v vs*)
  have *mk-coeff-Suc-closed*: *mk-coeff vs* (*c* ∘ *Suc*) *a* ∈ *carrier R* **for** *a*
    apply (*rule mk-coeff-carrier*)
    using *Cons.prems* **unfolding** *Pi-def* **by** *auto*
  have *x-in*: *x* ∈ *carrier M* **if** *x*: *x*∈ *set vs* **for** *x* **using** *Cons.prems x* **by** *auto*
  show *?case* **apply** (*unfold mk-coeff-Cons*[*OF Cons.prems*(*2*)] *lincomb-list-Cons*)
    apply (*subst Cons*) **using** *Cons* **apply** (*force*, *force*)
  proof (*cases v* ∈ *set vs*, *auto simp*:*insert-absorb*)
    case *False*
    let *?f* = (λ*va*. ((*if va = v then c 0 else* **0**) ⊕ *mk-coeff vs* (*c* ∘ *Suc*) *va*) $\odot_M$ *va*)

```
    have mk-0: mk-coeff vs (c ∘ Suc) v = 0 using False by auto
    have [simp]: (c 0 ⊕ 0) = c 0
      using Cons.prems(2) by force
    have finsum-rw: (⨁ Mva∈insert v (set vs). ?f va) = (?f v) ⊕M (⨁ Mva∈(set
vs). ?f va)
   proof (rule finsum-insert, auto simp add: False, rule smult-closed, rule R.a-closed)
      fix x
      show mk-coeff vs (c ∘ Suc) x ∈ carrier R
        using mk-coeff-Suc-closed by auto
      show c 0 ⊙M v ∈ carrier M
      proof (rule smult-closed)
        show c 0 ∈ carrier R
          using Cons.prems(2) by fastforce
        show v ∈ carrier M
          using Cons.prems(1) by auto
      qed
      show 0 ∈ carrier R
        by simp
      assume x: x ∈ set vs show x ∈ carrier M
        using Cons.prems(1) x by auto
    qed
    have finsum-rw2:
      (⨁ Mva∈(set vs). ?f va) = (⨁ Mva∈set vs. (mk-coeff vs (c ∘ Suc) va) ⊙M
va)
    proof (rule finsum-cong2, auto simp add: False)
      fix i assume i: i ∈ set vs
      have c ∘ Suc ∈ {0..<length vs} → carrier R using Cons.prems by auto
      then have [simp]: mk-coeff vs (c ∘ Suc) i ∈ carrier R
        using mk-coeff-Suc-closed by auto
     have 0 ⊕ mk-coeff vs (c ∘ Suc) i = mk-coeff vs (c ∘ Suc) i by (rule R.l-zero,
simp)
      then show (0 ⊕ mk-coeff vs (c ∘ Suc) i) ⊙M i = mk-coeff vs (c ∘ Suc) i
⊙M i
        by auto
      show (0 ⊕ mk-coeff vs (c ∘ Suc) i) ⊙M i ∈ carrier M
        using Cons.prems(1) i by auto
    qed
    show c 0 ⊙M v ⊕M lincomb (mk-coeff vs (c ∘ Suc)) (set vs) =
   lincomb (λa. (if a = v then c 0 else 0) ⊕ mk-coeff vs (c ∘ Suc) a) (insert v (set
vs))
      unfolding lincomb-def
      unfolding finsum-rw mk-0
      unfolding finsum-rw2 by auto
  next
    case True
    let ?f = λva. ((if va = v then c 0 else 0) ⊕ mk-coeff vs (c ∘ Suc) va) ⊙M va
    have rw: (c 0 ⊕ mk-coeff vs (c ∘ Suc) v) ⊙M v
      = (c 0 ⊙M v) ⊕M (mk-coeff vs (c ∘ Suc) v) ⊙M v
      using Cons.prems(1) Cons.prems(2) atLeast0-lessThan-Suc-eq-insert-0
```

using *mk-coeff-Suc-closed smult-l-distr* **by** *auto*
have *rw2*: $((\textit{mk-coeff vs } (c \circ Suc)\ v) \odot_M v)$
$\oplus_M (\bigoplus_M va{\in}(set\ vs - \{v\}).\ ?f\ va) = (\bigoplus_M v{\in}set\ vs.\ \textit{mk-coeff vs } (c \circ Suc)\ v \odot_M v)$
proof $-$
have $(\bigoplus_M va{\in}(set\ vs - \{v\}).\ ?f\ va) = (\bigoplus_M v{\in}set\ vs - \{v\}.\ \textit{mk-coeff vs } (c \circ Suc)\ v \odot_M v)$
by (*rule finsum-cong2*, *unfold Pi-def*, *auto simp add*: *mk-coeff-Suc-closed x-in*)
moreover have $(\bigoplus_M v{\in}set\ vs.\ \textit{mk-coeff vs } (c \circ Suc)\ v \odot_M v) = ((\textit{mk-coeff vs } (c \circ Suc)\ v) \odot_M v)$
$\oplus_M (\bigoplus_M v{\in}set\ vs - \{v\}.\ \textit{mk-coeff vs } (c \circ Suc)\ v \odot_M v)$
by (*rule M.add.finprod-split*, *auto simp add*: *mk-coeff-Suc-closed True x-in*)
ultimately show *?thesis* **by** *auto*
qed
have *lincomb* $(\lambda a.\ (\textit{if } a = v \textit{ then } c\ 0 \textit{ else } \mathbf{0}) \oplus \textit{mk-coeff vs } (c \circ Suc)\ a)\ (set\ vs)$
$= (\bigoplus_M va{\in}set\ vs.\ ?f\ va)$ **unfolding** *lincomb-def* **..**
also have ... $= ?f\ v \oplus_M (\bigoplus_M va{\in}(set\ vs - \{v\}).\ ?f\ va)$
proof (*rule M.add.finprod-split*)
have *c0-mkcoeff-in*: $c\ 0 \oplus \textit{mk-coeff vs } (c \circ Suc)\ v \in carrier\ R$
proof (*rule R.a-closed*)
show $c\ 0 \in carrier\ R$ **using** *Cons.prems* **by** *auto*
show $\textit{mk-coeff vs } (c \circ Suc)\ v \in carrier\ R$
using *mk-coeff-Suc-closed* **by** *auto*
qed
moreover have $(\mathbf{0} \oplus \textit{mk-coeff vs } (c \circ Suc)\ va) \odot_M va \in carrier\ M$
if *va*: $va \in carrier\ M$ **for** *va*
by (*rule smult-closed*[*OF - va*], *rule R.a-closed*, *auto simp add*: *mk-coeff-Suc-closed*)
ultimately show $?f$ ‘ $set\ vs \subseteq carrier\ M$ **using** *Cons.prems*(*1*) **by** *auto*
show *finite* (*set vs*) **by** *simp*
show $v \in set\ vs$ **using** *True* **by** *simp*
qed
also have ... $= (c\ 0 \oplus \textit{mk-coeff vs } (c \circ Suc)\ v) \odot_M v$
$\oplus_M (\bigoplus_M va{\in}(set\ vs - \{v\}).\ ?f\ va)$ **by** *auto*
also have ... $= ((c\ 0 \odot_M v) \oplus_M (\textit{mk-coeff vs } (c \circ Suc)\ v) \odot_M v)$
$\oplus_M (\bigoplus_M va{\in}(set\ vs - \{v\}).\ ?f\ va)$ **unfolding** *rw* **by** *simp*
also have ... $= (c\ 0 \odot_M v) \oplus_M (((\textit{mk-coeff vs } (c \circ Suc)\ v) \odot_M v)$
$\oplus_M (\bigoplus_M va{\in}(set\ vs - \{v\}).\ ?f\ va))$
proof (*rule M.a-assoc*)
show $c\ 0 \odot_M v \in carrier\ M$
using *Cons.prems*(*1*) *Cons.prems*(*2*) **by** *auto*
show $\textit{mk-coeff vs } (c \circ Suc)\ v \odot_M v \in carrier\ M$
using *Cons.prems*(*1*) *mk-coeff-Suc-closed* **by** *auto*
show $(\bigoplus_M va{\in}set\ vs - \{v\}.\ ((\textit{if } va = v \textit{ then } c\ 0 \textit{ else } \mathbf{0})$
$\oplus\ \textit{mk-coeff vs } (c \circ Suc)\ va) \odot_M va) \in carrier\ M$
by (*rule M.add.finprod-closed*) (*auto simp add*: *mk-coeff-Suc-closed x-in*)
qed
also have ... $= c\ 0 \odot_M v \oplus_M (\bigoplus_M v{\in}set\ vs.\ \textit{mk-coeff vs } (c \circ Suc)\ v \odot_M v)$

unfolding *rw2* **..**
also have ... = $c\ 0 \odot_M v \oplus_M$ *lincomb* (*mk-coeff vs* ($c \circ Suc$)) (*set vs*)
unfolding *lincomb-def* **..**
finally show $c\ 0 \odot_M v \oplus_M$ *lincomb* (*mk-coeff vs* ($c \circ Suc$)) (*set vs*)
= *lincomb* ($\lambda a.$ (*if* $a = v$ *then* $c\ 0$ *else* **0**) $\oplus$ *mk-coeff vs* ($c \circ Suc$) a) (*set vs*)
..
qed
qed *simp*

definition *span-list vs* $\equiv$ {*lincomb-list c vs* | *c. c* : {*0..<length vs*} $\rightarrow$ *carrier R*}

lemma *in-span-listI*:
assumes *c* : {*0..<length vs*} $\rightarrow$ *carrier R* **and** v = *lincomb-list c vs*
shows $v \in$ *span-list vs*
using *assms* **by** (*auto simp*: *span-list-def*)

lemma *in-span-listE*:
assumes $v \in$ *span-list vs*
and $\bigwedge c.$ *c* : {*0..<length vs*} $\rightarrow$ *carrier R* $\Longrightarrow$ v = *lincomb-list c vs* $\Longrightarrow$ *thesis*
shows *thesis*
using *assms* **by** (*auto simp*: *span-list-def*)

lemmas *lincomb-insert2* = *lincomb-insert*[*unfolded insert-union*[*symmetric*]]

lemma *lincomb-zero*:
assumes *U*: *U* $\subseteq$ *carrier M* **and** *a*: *a* : *U* $\rightarrow$ {*zero R*}
shows *lincomb a U* = *zero M*
using *U a*
proof (*induct U rule*: *infinite-finite-induct*)
case *empty* **show** *?case* **unfolding** *lincomb-def* **by** *auto* **next**
case (*insert u U*)
hence *a* $\in$ *insert u U* $\rightarrow$ *carrier R* **using** *zero-closed* **by** *force*
thus *?case* **using** *insert* **by** (*subst lincomb-insert2*; *auto*)
qed (*auto simp*: *lincomb-def*)

end

context *vec-module*
begin

lemma *lincomb-list-as-mat-mult*:
assumes $\forall w \in$ *set ws*. *dim-vec w* = *n*
shows *lincomb-list c ws* = *mat-of-cols n ws* $*_v$ *vec* (*length ws*) *c* (**is** *?l ws c* = *?r ws c*)
proof (*insert assms*, *induct ws arbitrary*: *c*)
case *Nil*

```
  then show ?case by (auto simp: mult-mat-vec-def scalar-prod-def)
next
  case (Cons w ws)
  { fix i assume i: i < n
    have ?l (w#ws) c = c 0 ·v w + mat-of-cols n ws *v vec (length ws) (c ∘ Suc)
      by (simp add: Cons o-def)
    also have ... $ i = ?r (w#ws) c $ i
      using Cons i index-smult-vec
        by (simp add: mat-of-cols-Cons-index-0 mat-of-cols-Cons-index-Suc o-def
vec-Suc mult-mat-vec-def row-def length-Cons)
    finally have ?l (w#ws) c $ i = ....
  }
  with Cons show ?case by (intro eq-vecI, auto)
qed

lemma lincomb-union2:
    assumes A: A ⊆ carrier-vec n
    and BA: B ⊆ A and fin-A: finite A
    and f: f ∈ A → UNIV shows lincomb f A = lincomb f (A−B) + lincomb f B
proof −
  have A − B ∪ B = A using BA by auto
  hence lincomb f A = lincomb f (A − B ∪ B) by simp
  also have ... = lincomb f (A−B) + lincomb f B
    by (rule lincomb-union, insert assms, auto intro: finite-subset)
  finally show ?thesis .
qed

lemma dim-sumlist:
  assumes ∀ x∈set xs. dim-vec x = n
  shows dim-vec (M.sumlist xs) = n using assms by (induct xs, auto)

lemma sumlist-nth:
  assumes ∀ x∈set xs. dim-vec x = n and i<n
  shows (M.sumlist xs) $ i= sum (λj. (xs ! j) $ i) {0..<length xs}
  using assms
proof (induct xs rule: rev-induct)
  case (snoc a xs)
  have [simp]: x ∈ carrier-vec n if x: x∈set xs for x
    using snoc.prems x unfolding carrier-vec-def by auto
  have [simp]: a ∈ carrier-vec n
    using snoc.prems unfolding carrier-vec-def by auto
  have hyp: M.sumlist xs $ i = (∑j = 0..<length xs. xs ! j $ i)
    by (rule snoc.hyps, auto simp add: snoc.prems)
  have M.sumlist (xs @ [a]) = M.sumlist xs + M.sumlist [a]
    by (rule M.sumlist-append, auto simp add: snoc.prems)
  also have ... = M.sumlist xs + a by auto
  also have ... $ i = (M.sumlist xs $ i) + (a $ i)
    by (rule index-add-vec(1), auto simp add: snoc.prems)
  also have ... = (∑j = 0..<length xs. xs ! j $ i) + (a $ i) unfolding hyp by
```

simp
  also have ... = ($\sum j = 0..<length\ (xs\ @\ [a]).\ (xs\ @\ [a])\ !\ j\ \$\ i$)
    by (*auto*, *rule sum.cong*, *auto simp add*: *nth-append*)
  finally show *?case* **.**
qed *auto*

lemma *lincomb-as-lincomb-list-distinct*:
  assumes *s*: *set ws* $\subseteq$ *carrier-vec n* **and** *d*: *distinct ws*
  shows *lincomb f* (*set ws*) = *lincomb-list* ($\lambda i.\ f\ (ws\ !\ i)$) *ws*
proof (*insert assms*, *induct ws*)
  case *Nil*
  then show *?case* **by** *auto*
next
  case (*Cons a ws*)
  have [*simp*]: $\bigwedge v.\ v \in set\ ws \Longrightarrow v \in carrier\text{-}vec\ n$ **using** *Cons.prems*(*1*) **by** *auto*
  then have *ws*: *set ws* $\subseteq$ *carrier-vec n* **by** *auto*
  have *hyp*: *lincomb f* (*set* (*ws*)) = *lincomb-list* ($\lambda i.\ f\ (ws\ !\ i)$) *ws*
  proof (*intro Cons.hyps ws*)
    show *distinct ws* **using** *Cons.prems*(*2*) **by** *auto*
  qed
  have ($map\ (\lambda i.\ f\ (ws\ !\ i) \cdot_v ws\ !\ i)\ [0..<length\ ws]) = (map\ (\lambda v.\ f\ v \cdot_v v)\ ws)$
    by (*simp add*: *nth-map-conv*)
  with *ws* **have** *sumlist-rw*: $sumlist\ (map\ (\lambda i.\ f\ (ws\ !\ i) \cdot_v ws\ !\ i)\ [0..<length\ ws])$
    $= sumlist\ (map\ (\lambda v.\ f\ v \cdot_v v)\ ws)$
    by (*subst* (*1 2*) *sumlist-as-summset*, *auto*)
  have $lincomb\ f\ (set\ (a\ \#\ ws)) = (\bigoplus_V v{\in}set\ (a\ \#\ ws).\ f\ v \cdot_v v)$ **unfolding** *lincomb-def* **..**
  also have ... = $(\bigoplus_V v{\in}\ insert\ a\ (set\ ws).\ f\ v \cdot_v v)$ **by** *simp*
  also have ... = $(f\ a \cdot_v a) + (\bigoplus_V v{\in}\ (set\ ws).\ f\ v \cdot_v v)$
    by (*rule finsum-insert*, *insert Cons.prems*, *auto*)
  also have ... = $f\ a \cdot_v a + lincomb\text{-}list\ (\lambda i.\ f\ (ws\ !\ i))\ ws$ **using** *hyp lincomb-def* **by** *auto*
  also have ... = $f\ a \cdot_v a + sumlist\ (map\ (\lambda v.\ f\ v \cdot_v v)\ ws)$
    unfolding *lincomb-list-def sumlist-rw* **by** *auto*
  also have ... = $sumlist\ (map\ (\lambda v.\ f\ v \cdot_v v)\ (a\ \#\ ws))$
  proof −
    let $?a = (map\ (\lambda v.\ f\ v \cdot_v v)\ [a])$
    have *a*: $a \in carrier\text{-}vec\ n$ **using** *Cons.prems*(*1*) **by** *auto*
    have $f\ a \cdot_v a = sumlist\ (map\ (\lambda v.\ f\ v \cdot_v v)\ [a])$ **using** *Cons.prems*(*1*) **by** *auto*
    hence $f\ a \cdot_v a + sumlist\ (map\ (\lambda v.\ f\ v \cdot_v v)\ ws)$
      $= sumlist\ ?a + sumlist\ (map\ (\lambda v.\ f\ v \cdot_v v)\ ws)$ **by** *simp*
    also have ... = $sumlist\ (?a\ @\ (map\ (\lambda v.\ f\ v \cdot_v v)\ ws))$
      by (*rule sumlist-append*[*symmetric*], *auto simp add*: *a*)
    finally show *?thesis* **by** *auto*
  qed
  also have ... = $sumlist\ (map\ (\lambda i.\ f\ ((a\ \#\ ws)\ !\ i) \cdot_v (a\ \#\ ws)\ !\ i)\ [0..<length\ (a\ \#\ ws)])$
  proof −

have *u*: (*map* (λ*i*. *f* ((*a* # *ws*) ! *i*) $\cdot_v$ (*a* # *ws*) ! *i*) [*0*..<*length* (*a* # *ws*)])
= (*map* (λ*v*. *f v* $\cdot_v$ *v*) (*a* # *ws*))
proof (*rule nth-map-conv*)
show *length* [*0*..<*length* (*a* # *ws*)] = *length* (*a* # *ws*) **by** *auto*
show ∀ *i*<*length* [*0*..<*length* (*a* # *ws*)]. *f* ((*a* # *ws*) ! ([*0*..<*length* (*a* # *ws*)] ! *i*)) $\cdot_v$ (*a* # *ws*) !
([*0*..<*length* (*a* # *ws*)] ! *i*) = *f* ((*a* # *ws*) ! *i*) $\cdot_v$ (*a* # *ws*) ! *i*
by (*metis* ‹*length* [*0*..<*length* (*a* # *ws*)] = *length* (*a* # *ws*)› *add.left-neutral nth-upt*)
qed
show *?thesis* **unfolding** *u* **..**
qed
also have ... = *lincomb-list* (λ*i*. *f* ((*a* # *ws*) ! *i*)) (*a* # *ws*)
unfolding *lincomb-list-def* **..**
finally show *?case* **.**
qed

end

locale *idom-vec* = *vec-module f-ty* **for** *f-ty* :: ′*a* :: *idom itself*
begin

lemma *lin-dep-cols-imp-det-0* ′:
fixes *ws*
defines *A* ≡ *mat-of-cols n ws*
assumes *dimv-ws*: ∀ *w*∈*set ws*. *dim-vec w* = *n*
assumes *A*: *A* ∈ *carrier-mat n n* **and** *ld-cols*: *lin-dep* (*set* (*cols A*))
shows *det A* = *0*
proof (*cases distinct ws*)
case *False*
obtain *i j* **where** *ij*: *i*≠*j* **and** *c*: *col A i* = *col A j* **and** *i*: *i*<*n* **and** *j*: *j*<*n*
using *False A* **unfolding** *A-def*
by (*metis dimv-ws distinct-conv-nth carrier-matD*(*2*)
col-mat-of-cols mat-of-cols-carrier(*3*) *nth-mem carrier-vecI*)
show *?thesis* **by** (*rule det-identical-columns*[*OF A ij i j c*])
next
case *True*
have *d1* [*simp*]: ⋀*x*. *x* ∈ *set ws* ⟹ *x* ∈ *carrier-vec n* **using** *dimv-ws* **by** *auto*
obtain *A*′ *f* ′ *v* **where** *f* ′*-in*: *f* ′ ∈ *A*′ → *UNIV*
and *lc-f* ′: *lincomb f* ′ *A*′ = 0_v *n* **and** *f* ′*-v*: *f* ′ *v* ≠ *0*
and *v-A*′: *v* ∈ *A*′ **and** *A*′*-in-rows*: *A*′ ⊆ *set* (*cols A*)
using *ld-cols* **unfolding** *lin-dep-def* **by** *auto*
define *f* **where** *f* ≡ λ*x*. *if x* ∉ *A*′ *then 0 else f* ′ *x*
have *f-in*: *f* ∈ (*set* (*cols A*)) → *UNIV* **using** *f* ′*-in* **by** *auto*
have *A*′*-in-carrier*: *A*′ ⊆ *carrier-vec n*
by (*metis* (*no-types*) *A*′*-in-rows A-def cols-dim carrier-matD*(*1*) *mat-of-cols-carrier*(*1*) *subset-trans*)
have *lc-f*: *lincomb f* (*set* (*cols A*)) = 0_v *n*
proof −

have *l1*: *lincomb f* (*set* (*cols A*) − *A′*) = 0_v *n*
by (*rule lincomb-zero*, *auto simp add*: *f-def*, *insert A cols-dim*, *blast*)
have *l2*: *lincomb f A′* = 0_v *n* **using** *lc-f′* **unfolding** *f-def* **using** *A′-in-carrier* **by** *auto*
have *lincomb f* (*set* (*cols A*)) = *lincomb f* (*set* (*cols A*) − *A′*) + *lincomb f A′*
proof (*rule lincomb-union2*)
show *set* (*cols A*) ⊆ *carrier-vec n*
using *A cols-dim* **by** *blast*
show *A′* ⊆ *set* (*cols A*)
using *A′-in-rows* **by** *blast*
show *finite* (*set* (*cols A*)) **by** *auto*
show *f* ∈ *set* (*cols A*) → *UNIV* **by** *auto*
qed
also have ... = 0_v *n* **using** *l1 l2* **by** *auto*
finally show *?thesis* **.**
qed
have *v-in*: *v* ∈ (*set* (*cols A*)) **using** *v-A′ A′-in-rows* **by** *auto*
have *fv*: *f v* ≠ *0* **using** *f′-v v-A′* **unfolding** *f-def* **by** *auto*
let *?c* = (λ*i*. *f* (*ws* ! *i*))
have *lincomb f* (*set ws*) = *lincomb-list ?c ws*
by (*rule lincomb-as-lincomb-list-distinct*[*OF - True*], *auto*)
have ∃ *v*. *v* ∈ *carrier-vec n* ∧ *v* ≠ 0_v *n* ∧ *A* $*_v$ *v* = 0_v *n*
proof (*rule exI*[*of - vec* (*length ws*) *?c*], *rule conjI*)
show *vec* (*length ws*) *?c* ∈ *carrier-vec n* **using** *A A-def* **by** *auto*
have *vec-not0*: *vec* (*length ws*) *?c* ≠ 0_v *n*
proof −
obtain *i* **where** *ws-i*: (*ws* ! *i*) = *v* **and** *i*: *i*<*length ws* **using** *v-in* **unfolding** *A-def*
by (*metis d1 cols-mat-of-cols in-set-conv-nth subset-eq*)
have *vec* (*length ws*) *?c* \$ *i* = *?c i* **by** (*rule index-vec*[*OF i*])
also have ... = *f v* **using** *ws-i* **by** *simp*
also have ... ≠ *0* **using** *fv* **by** *simp*
finally show *?thesis*
using *A A-def i* **by** *fastforce*
qed
have *A* $*_v$ *vec* (*length ws*) *?c* = *mat-of-cols n ws* $*_v$ *vec* (*length ws*) *?c* **unfolding** *A-def* **..**
also have ... = *lincomb-list ?c ws* **by** (*rule lincomb-list-as-mat-mult*[*symmetric*, *OF dimv-ws*])
also have ... = *lincomb f* (*set ws*)
by (*rule lincomb-as-lincomb-list-distinct*[*symmetric*, *OF - True*], *auto*)
also have ... = 0_v *n*
using *lc-f* **unfolding** *A-def* **using** *A* **by** (*simp add*: *subset-code*(*1*))
finally show *vec* (*length ws*) (λ*i*. *f* (*ws* ! *i*)) ≠ 0_v *n* ∧ *A* $*_v$ *vec* (*length ws*) (λ*i*. *f* (*ws* ! *i*)) = 0_v *n*
using *vec-not0* **by** *fast*
qed
thus *?thesis* **unfolding** *det-0-iff-vec-prod-zero*[*OF A*] **.**
qed

lemma *lin-dep-cols-imp-det-0*:
assumes *A*: $A \in$ *carrier-mat n n* **and** *ld*: *lin-dep* (*set* (*cols A*))
shows *det A* = *0*
proof −
have *col-rw*: (*cols* (*mat-of-cols n* (*cols A*))) = *cols A*
using *A* **by** *auto*
have *m*: *mat-of-cols n* (*cols A*) = *A* **using** *A* **by** *auto*
show *?thesis*
by (*rule A lin-dep-cols-imp-det-0'*[*of cols A*, *unfolded col-rw*, *unfolded m*, *OF* - *A ld*])
(*metis A cols-dim carrier-matD*(*1*) *subsetCE carrier-vecD*)
qed

corollary *lin-dep-rows-imp-det-0*:
assumes *A*: $A \in$ *carrier-mat n n* **and** *ld*: *lin-dep* (*set* (*rows A*))
shows *det A* = *0*
by (*subst det-transpose*[*OF A*, *symmetric*], *rule lin-dep-cols-imp-det-0*, *auto simp add*: *ld A*)

lemma *det-not-0-imp-lin-indpt-rows*:
assumes *A*: $A \in$ *carrier-mat n n* **and** *det*: *det A* $\neq$ *0*
shows *lin-indpt* (*set* (*rows A*))
using *lin-dep-rows-imp-det-0*[*OF A*] *det* **by** *auto*

lemma *upper-triangular-imp-lin-indpt-rows*:
assumes *A*: $A \in$ *carrier-mat n n*
and *tri*: *upper-triangular A*
and *diag*: *0* $\notin$ *set* (*diag-mat A*)
shows *lin-indpt* (*set* (*rows A*))
using *det-not-0-imp-lin-indpt-rows upper-triangular-imp-det-eq-0-iff assms*
by *auto*

lemma *lincomb-as-lincomb-list*:
fixes *ws f*
assumes *s*: *set ws* $\subseteq$ *carrier-vec n*
shows *lincomb f* (*set ws*) = *lincomb-list* (λi. *if* $\exists j<i$. *ws*!*i* = *ws*!*j then 0 else f* (*ws* ! *i*)) *ws*
using *assms*
proof (*induct ws rule*: *rev-induct*)
case (*snoc a ws*)
let *?f* = λi. *if* $\exists j<i$. *ws* ! *i* = *ws* ! *j then 0 else f* (*ws* ! *i*)
let *?g* = λi. (*if* $\exists j<i$. (*ws* @ [*a*]) ! *i* = (*ws* @ [*a*]) ! *j then 0 else f* ((*ws* @ [*a*]) ! *i*)) $\cdot_v$ (*ws* @ [*a*]) ! *i*
let *?g2*= (λi. (*if* $\exists j<i$. *ws* ! *i* = *ws* ! *j then 0 else f* (*ws* ! *i*)) $\cdot_v$ *ws* ! *i*)
have [*simp*]: $\bigwedge v$. $v \in$ *set ws* $\Longrightarrow$ $v \in$ *carrier-vec n* **using** *snoc.prems*(*1*) **by** *auto*
then have *ws*: *set ws* $\subseteq$ *carrier-vec n* **by** *auto*

have *hyp*: *lincomb f* (*set ws*) = *lincomb-list ?f ws*
by (*intro snoc.hyps ws*)
show *?case*
proof (*cases a∈set ws*)
case *True*
have *g-length*: *?g* (*length ws*) = 0_v *n* **using** *True*
by (*auto*, *metis in-set-conv-nth nth-append*)
have (*map ?g* [*0..<length* (*ws* @ [*a*])]) = (*map ?g* [*0..<length ws*]) @ [*?g* (*length ws*)]
by *auto*
also have ... = (*map ?g* [*0..<length ws*]) @ [0_v *n*] **using** *g-length* **by** *simp*
finally have *map-rw*: (*map ?g* [*0..<length* (*ws* @ [*a*])]) = (*map ?g* [*0..<length ws*]) @ [0_v *n*] **.**
have *M.sumlist* (*map ?g2* [*0..<length ws*]) = *M.sumlist* (*map ?g* [*0..<length ws*])
by (*rule arg-cong*[*of - - M.sumlist*], *rule nth-map-conv*, *auto simp add*: *nth-append*)
also have ... = *M.sumlist* (*map ?g* [*0..<length ws*]) + 0_v *n*
by (*metis M.r-zero calculation hyp lincomb-closed lincomb-list-def ws*)
also have ... = *M.sumlist* (*map ?g* [*0..<length ws*] @ [0_v *n*])
by (*rule M.sumlist-snoc*[*symmetric*], *auto simp add*: *nth-append*)
finally have *summlist-rw*: *M.sumlist* (*map ?g2* [*0..<length ws*])
= *M.sumlist* (*map ?g* [*0..<length ws*] @ [0_v *n*]) **.**
have *lincomb f* (*set* (*ws* @ [*a*])) = *lincomb f* (*set ws*) **using** *True* **unfolding** *lincomb-def*
by (*simp add*: *insert-absorb*)
thus *?thesis*
unfolding *hyp lincomb-list-def map-rw summlist-rw*
by *auto*
next
case *False*
have *g-length*: *?g* (*length ws*) = *f a* $\cdot_v$ *a* **using** *False* **by** (*auto simp add*: *nth-append*)
have (*map ?g* [*0..<length* (*ws* @ [*a*])]) = (*map ?g* [*0..<length ws*]) @ [*?g* (*length ws*)]
by *auto*
also have ... = (*map ?g* [*0..<length ws*]) @ [(*f a* $\cdot_v$ *a*)] **using** *g-length* **by** *simp*
finally have *map-rw*: (*map ?g* [*0..<length* (*ws* @ [*a*])]) = (*map ?g* [*0..<length ws*]) @ [(*f a* $\cdot_v$ *a*)] **.**
have *summlist-rw*: *M.sumlist* (*map ?g2* [*0..<length ws*]) = *M.sumlist* (*map ?g* [*0..<length ws*])
by (*rule arg-cong*[*of - - M.sumlist*], *rule nth-map-conv*, *auto simp add*: *nth-append*)
have *lincomb f* (*set* (*ws* @ [*a*])) = *lincomb f* (*set* (*a* # *ws*)) **by** *auto*
also have ... = ($\bigoplus_V$*v*∈*set* (*a* # *ws*). *f v* $\cdot_v$ *v*) **unfolding** *lincomb-def* **..**
also have ... = ($\bigoplus_V$*v*∈ *insert a* (*set ws*). *f v* $\cdot_v$ *v*) **by** *simp*
also have ... = (*f a* $\cdot_v$ *a*) + ($\bigoplus_V$*v*∈ (*set ws*). *f v* $\cdot_v$ *v*)
proof (*rule finsum-insert*)
show *finite* (*set ws*) **by** *auto*

```
      show a ∉ set ws using False by auto
      show (λv. f v ·v v) ∈ set ws → carrier-vec n
        using snoc.prems(1) by auto
      show f a ·v a ∈ carrier-vec n using snoc.prems by auto
    qed
    also have ... = (f a ·v a) + lincomb f (set ws) unfolding lincomb-def ..
    also have ... = (f a ·v a) + lincomb-list ?f ws using hyp by auto
    also have ... =  lincomb-list ?f ws  + (f a ·v a)
      using M.add.m-comm lincomb-list-carrier snoc.prems by auto
    also have ... = lincomb-list (λi. if ∃j<i. (ws @ [a]) ! i
      = (ws @ [a]) ! j then 0 else f ((ws @ [a]) ! i)) (ws @ [a])
   proof (unfold lincomb-list-def map-rw summlist-rw, rule M.sumlist-snoc[symmetric])
      show set (map ?g [0..<length ws]) ⊆ carrier-vec n using snoc.prems
        by (auto simp add: nth-append)
      show f a ·v a ∈ carrier-vec n
        using snoc.prems by auto
    qed
    finally show ?thesis .
  qed
qed auto

lemma span-list-as-span:
  assumes set vs ⊆ carrier-vec n
  shows span-list vs = span (set vs)
  using assms
proof (auto simp: span-list-def span-def)
  fix f show ∃ a A. lincomb-list f vs = lincomb a A ∧ finite A ∧ A ⊆ set vs
    using assms lincomb-list-as-lincomb by auto
next
  fix f::'a vec ⇒'a and A assume fA: finite A and A: A ⊆ set vs
  have [simp]: x ∈ carrier-vec n if x: x ∈ A for x using A x assms by auto
  have [simp]:  v ∈ carrier-vec n if v: v ∈ set vs for v using assms v by auto
  have set-vs-Un: ((set vs) − A) ∪ A = set vs using A by auto
  let ?f = (λx. if x∈(set vs) − A then 0 else f x)
  have f0: (⨁Vv∈(set vs) − A. ?f v ·v v) = 0v n by (rule M.finsum-all0, auto)

  have lincomb f A = lincomb ?f A
    by (auto simp add: lincomb-def intro!: finsum-cong2)
  also have ... = (⨁Vv∈(set vs) − A. ?f v ·v v) + (⨁Vv∈A. ?f v ·v v)
    unfolding f0 lincomb-def by auto
  also have ... = lincomb ?f (((set vs) − A) ∪ A)
    unfolding lincomb-def
    by (rule M.finsum-Un-disjoint[symmetric], auto simp add: fA)
  also have ... = lincomb ?f (set vs) using set-vs-Un by auto
  finally have lincomb f A = lincomb ?f (set vs) .
  with lincomb-as-lincomb-list[OF assms]
  show ∃ c. lincomb f A = lincomb-list c vs by auto
qed
```

lemma *in-spanI*[*intro*]:
  assumes $v = lincomb\ a\ A\ finite\ A\ A \subseteq W$
  shows $v \in span\ W$
unfolding *span-def* **using** *assms* **by** *auto*
lemma *in-spanE*:
  assumes $v \in span\ W$
  shows $\exists\ a\ A.\ v = lincomb\ a\ A \wedge finite\ A \wedge A \subseteq W$
using *assms* **unfolding** *span-def* **by** *auto*

declare *in-own-span*[*intro*]

lemma *smult-in-span*:
  assumes $W \subseteq carrier\text{-}vec\ n$ **and** *insp*: $x \in span\ W$
  shows $c \cdot_v x \in span\ W$
proof −
  from *in-spanE*[*OF insp*] **obtain** *a A* **where** *a*: $x = lincomb\ a\ A\ finite\ A\ A \subseteq W$ **by** *blast*
  have $c \cdot_v x = lincomb\ (\lambda\ x.\ c * a\ x)\ A$ **using** *a*(*1*) **unfolding** *lincomb-def a*
    apply(*subst finsum-smult*) **using** *assms a* **by** (*auto simp*:*smult-smult-assoc*)
  thus $c \cdot_v x \in span\ W$ **using** *a*(*2*,*3*) **by** *auto*
qed

lemma *span-subsetI*: **assumes** *ws*: $ws \subseteq carrier\text{-}vec\ n$
  $us \subseteq span\ ws$
shows $span\ us \subseteq span\ ws$
  by (*simp add*: *assms*(*1*) *span-is-submodule span-is-subset subsetI ws*)

end

context *vec-space* **begin**
sublocale *idom-vec***.**

lemma *sumlist-in-span*: **assumes** *W*: $W \subseteq carrier\text{-}vec\ n$
  shows $(\bigwedge x.\ x \in set\ xs \Longrightarrow x \in span\ W) \Longrightarrow sumlist\ xs \in span\ W$
proof (*induct xs*)
  case *Nil*
  thus *?case* **using** *W* **by** *force*
next
  case (*Cons x xs*)
  from *span-is-subset2*[*OF W*] *Cons*(*2*) **have** *xs*: $x \in carrier\text{-}vec\ n\ set\ xs \subseteq carrier\text{-}vec\ n$ **by** *auto*
  from *span-add1*[*OF W Cons*(*2*)[*of x*] *Cons*(*1*)[*OF Cons*(*2*)]]
  have $x + sumlist\ xs \in span\ W$ **by** *auto*
  also have $x + sumlist\ xs = sumlist\ ([x]\ @\ xs)$
    by (*subst sumlist-append*, *insert xs*, *auto*)
  finally show *?case* **by** *simp*
qed

lemma *span-span*[*simp*]:

assumes $W \subseteq$ *carrier-vec n*
shows *span* (*span W*) = *span W*
proof(*standard*,*standard*,*goal-cases*)
case (*1 x*) **with** *in-spanE* **obtain** *a A* **where** *a*: x = *lincomb a A finite A A* $\subseteq$ *span W* **by** *blast*
from *a*(*3*) *assms* **have** *AC*:$A \subseteq$ *carrier-vec n* **by** *auto*
show *?case* **unfolding** *a*(*1*)[*unfolded lincomb-def*]
proof(*insert a*(*3*),*atomize* (*full*),*rule finite-induct*[*OF a*(*2*)],*goal-cases*)
case *1*
then show *?case* **using** *span-zero* **by** *auto*
next
case (*2 x F*)
{ **assume** *F*:*insert x F* $\subseteq$ *span W*
hence $a\ x \cdot_v x \in$ *span W* **by** (*intro smult-in-span*[*OF assms*],*auto*)
hence $a\ x \cdot_v x + (\bigoplus_V v{\in}F.\ a\ v \cdot_v v) \in$ *span W*
using *span-add1 F 2 assms* **by** *auto*
hence $(\bigoplus_V v{\in}$*insert x F*$.\ a\ v \cdot_v v) \in$ *span W*
apply(*subst M.finsum-insert*[*OF 2*(*1*,*2*)]) **using** *F assms* **by** *auto*
}
then show *?case* **by** *auto*
qed
next
case *2*
show *?case* **using** *assms* **by**(*intro in-own-span*, *auto*)
qed

lemma *upper-triangular-imp-basis*:
assumes *A*: $A \in$ *carrier-mat n n*
and *tri*: *upper-triangular A*
and *diag*: $0 \notin$ *set* (*diag-mat A*)
shows *basis* (*set* (*rows A*))
using *upper-triangular-imp-distinct*[*OF assms*]
using *upper-triangular-imp-lin-indpt-rows*[*OF assms*] *A*
by (*auto intro*: *dim-li-is-basis simp*: *distinct-card dim-is-n set-rows-carrier*)
end

lemma (**in** *zero-hom*) *hom-upper-triangular*:
$A \in$ *carrier-mat n n* $\Longrightarrow$ *upper-triangular A* $\Longrightarrow$ *upper-triangular* (*map-mat hom A*)
by (*auto simp*: *upper-triangular-def*)

end

4 Norms

In this theory we provide the basic definitions and properties of several norms of vectors and polynomials.

theory *Norms*
  imports *Polynomial Adhoc-Overloading*
    Jordan-Normal-Form.Conjugate
    Algebraic-Numbers.Resultant
    Missing-Lemmas
begin

4.1 L-∞ Norms

consts *linf-norm* :: $'a \Rightarrow 'b$ ($\|(\text{-})\|_\infty$)

definition *linf-norm-vec* **where** *linf-norm-vec v* $\equiv$ *max-list* (*map abs* (*list-of-vec v*) @ [*0*])
adhoc-overloading *linf-norm linf-norm-vec*

definition *linf-norm-poly* **where** *linf-norm-poly f* $\equiv$ *max-list* (*map abs* (*coeffs f*) @ [*0*])
adhoc-overloading *linf-norm linf-norm-poly*

lemma *linf-norm-vec*: $\|vec\ n\ f\|_\infty = max\text{-}list\ (map\ (abs \circ f)\ [0..<n]\ @\ [0])$
  by (*simp add*: *linf-norm-vec-def*)

lemma *linf-norm-vec-vCons*[*simp*]: $\|vCons\ a\ v\|_\infty = max\ |a|\ \|v\|_\infty$
  by (*auto simp*: *linf-norm-vec-def max-list-Cons*)

lemma *linf-norm-vec-0* [*simp*]: $\|vec\ 0\ f\|_\infty = 0$ **by** (*simp add*: *linf-norm-vec-def*)

lemma *linf-norm-zero-vec* [*simp*]: $\|0_v\ n :: 'a :: ordered\text{-}ab\text{-}group\text{-}add\text{-}abs\ vec\|_\infty = 0$
  by (*induct n*, *simp add*: *zero-vec-def*, *auto simp*: *zero-vec-Suc*)

lemma *linf-norm-vec-ge-0* [*intro*!]:
  fixes *v* :: *'a* :: *ordered-ab-group-add-abs vec*
  shows $\|v\|_\infty \geq 0$
  by (*induct v*, *auto simp*: *max-def*)

lemma *linf-norm-vec-eq-0* [*simp*]:
  fixes *v* :: *'a* :: *ordered-ab-group-add-abs vec*
  assumes $v \in carrier\text{-}vec\ n$
  shows $\|v\|_\infty = 0 \longleftrightarrow v = 0_v\ n$
  by (*insert assms*, *induct rule*: *carrier-vec-induct*, *auto simp*: *zero-vec-Suc max-def*)

lemma *linf-norm-vec-greater-0* [*simp*]:
  fixes *v* :: *'a* :: *ordered-ab-group-add-abs vec*
  assumes $v \in carrier\text{-}vec\ n$

shows $\|v\|_\infty > 0 \longleftrightarrow v \neq 0_v\ n$
by (*insert assms*, *induct rule*: *carrier-vec-induct*, *auto simp*: *zero-vec-Suc max-def*)

lemma *linf-norm-poly-0* [*simp*]: $\|0{::}\text{-}\ poly\|_\infty = 0$
by (*simp add*: *linf-norm-poly-def*)

lemma *linf-norm-pCons* [*simp*]:
fixes $p :: {}'a :: ordered\text{-}ab\text{-}group\text{-}add\text{-}abs\ poly$
shows $\|pCons\ a\ p\|_\infty = max\ |a|\ \|p\|_\infty$
by (*cases* $p = 0$, *cases* $a = 0$, *auto simp*: *linf-norm-poly-def max-list-Cons*)

lemma *linf-norm-poly-ge-0* [*intro*!]:
fixes $f :: {}'a :: ordered\text{-}ab\text{-}group\text{-}add\text{-}abs\ poly$
shows $\|f\|_\infty \geq 0$
by (*induct f*, *auto simp*: *max-def*)

lemma *linf-norm-poly-eq-0* [*simp*]:
fixes $f :: {}'a :: ordered\text{-}ab\text{-}group\text{-}add\text{-}abs\ poly$
shows $\|f\|_\infty = 0 \longleftrightarrow f = 0$
by (*induct f*, *auto simp*: *max-def*)

lemma *linf-norm-poly-greater-0* [*simp*]:
fixes $f :: {}'a :: ordered\text{-}ab\text{-}group\text{-}add\text{-}abs\ poly$
shows $\|f\|_\infty > 0 \longleftrightarrow f \neq 0$
by (*induct f*, *auto simp*: *max-def*)

4.2 Square Norms

consts *sq-norm* :: $'a \Rightarrow {}'b$ ($\|(\text{-})\|^2$)

abbreviation *sq-norm-conjugate* $x \equiv x * conjugate\ x$
adhoc-overloading *sq-norm sq-norm-conjugate*

4.2.1 Square norms for vectors

We prefer sum_list over sum because it is not essentially dependent on commutativity, and easier for proving.

definition *sq-norm-vec* $v \equiv \sum x \leftarrow list\text{-}of\text{-}vec\ v.\ \|x\|^2$
adhoc-overloading *sq-norm sq-norm-vec*

lemma *sq-norm-vec-vCons*[*simp*]: $\|vCons\ a\ v\|^2 = \|a\|^2 + \|v\|^2$
by (*simp add*: *sq-norm-vec-def*)

lemma *sq-norm-vec-0*[*simp*]: $\|vec\ 0\ f\|^2 = 0$
by (*simp add*: *sq-norm-vec-def*)

lemma *sq-norm-vec-as-cscalar-prod*:
fixes $v :: {}'a :: conjugatable\text{-}ring\ vec$
shows $\|v\|^2 = v \cdot c\ v$

by (*induct v*, *simp-all add*: *sq-norm-vec-def*)

lemma *sq-norm-zero-vec*[*simp*]: $\|0_v\ n :: 'a :: conjugatable\text{-}ring\ vec\|^2 = 0$
by (*simp add*: *sq-norm-vec-as-cscalar-prod*)

lemmas *sq-norm-vec-ge-0* [*intro*!] = *conjugate-square-ge-0-vec*[*folded sq-norm-vec-as-cscalar-prod*]

lemmas *sq-norm-vec-eq-0* [*simp*] = *conjugate-square-eq-0-vec*[*folded sq-norm-vec-as-cscalar-prod*]

lemmas *sq-norm-vec-greater-0* [*simp*] = *conjugate-square-greater-0-vec*[*folded sq-norm-vec-as-cscalar-prod*]

4.2.2 Square norm for polynomials

definition *sq-norm-poly* **where** *sq-norm-poly p* $\equiv \sum a{\leftarrow}coeffs\ p.\ \|a\|^2$

adhoc-overloading *sq-norm sq-norm-poly*

lemma *sq-norm-poly-0* [*simp*]: $\|0{::}\text{-}poly\|^2 = 0$
by (*auto simp*: *sq-norm-poly-def*)

lemma *sq-norm-poly-pCons* [*simp*]:
fixes $a :: 'a :: conjugatable\text{-}ring$
shows $\|pCons\ a\ p\|^2 = \|a\|^2 + \|p\|^2$
by (*cases* $p = 0$; *cases* $a = 0$, *auto simp*: *sq-norm-poly-def*)

lemma *sq-norm-poly-ge-0* [*intro*!]:
fixes $p :: 'a :: conjugatable\text{-}ordered\text{-}ring\ poly$
shows $\|p\|^2 \geq 0$
by (*unfold sq-norm-poly-def*, *rule sum-list-nonneg*, *auto intro*!:*conjugate-square-positive*)

lemma *sq-norm-poly-eq-0* [*simp*]:
fixes $p :: 'a :: \{conjugatable\text{-}ordered\text{-}ring, ring\text{-}no\text{-}zero\text{-}divisors\}\ poly$
shows $\|p\|^2 = 0 \longleftrightarrow p = 0$
proof (*induct p*)
case *IH*: (*pCons a p*)
show *?case*
proof (*cases* $a = 0$)
case *True*
with *IH* **show** *?thesis* **by** *simp*
next
case *False*
then have $\|a\|^2 + \|p\|^2 > 0$ **by** (*intro add-pos-nonneg*, *auto*)
then show *?thesis* **by** *auto*
qed
qed *simp*

lemma *sq-norm-poly-pos* [*simp*]:
fixes $p :: 'a :: \{conjugatable\text{-}ordered\text{-}ring, ring\text{-}no\text{-}zero\text{-}divisors\}\ poly$
shows $\|p\|^2 > 0 \longleftrightarrow p \neq 0$

by (*auto simp*: *less-le*)

lemma *sq-norm-vec-of-poly* [*simp*]:
fixes p :: $'a$ :: *conjugatable-ring poly*
shows $\|$*vec-of-poly* $p\|^2 = \|p\|^2$
apply (*unfold sq-norm-poly-def sq-norm-vec-def*)
apply (*fold sum-mset-sum-list*)
apply *auto*.

lemma *sq-norm-poly-of-vec* [*simp*]:
fixes v :: $'a$ :: *conjugatable-ring vec*
shows $\|$*poly-of-vec* $v\|^2 = \|v\|^2$
apply (*unfold sq-norm-poly-def sq-norm-vec-def coeffs-poly-of-vec*)
apply (*fold rev-map*)
apply (*fold sum-mset-sum-list*)
apply (*unfold mset-rev*)
apply (*unfold sum-mset-sum-list*)
by (*auto intro*: *sum-list-map-dropWhile0*)

4.3 Relating Norms

A class where ordering around 0 is linear.

abbreviation (**in** *ordered-semiring*) *is-real* **where** *is-real* $a \equiv a < 0 \vee a = 0 \vee 0 < a$

class *semiring-real-line* = *ordered-semiring-strict* + *ordered-semiring-0* +
assumes *add-pos-neg-is-real*: $a > 0 \Longrightarrow b < 0 \Longrightarrow$ *is-real* $(a + b)$
and *mult-neg-neg*: $a < 0 \Longrightarrow b < 0 \Longrightarrow 0 < a * b$
and *pos-pos-linear*: $0 < a \Longrightarrow 0 < b \Longrightarrow a < b \vee a = b \vee b < a$
and *neg-neg-linear*: $a < 0 \Longrightarrow b < 0 \Longrightarrow a < b \vee a = b \vee b < a$
begin

lemma *add-neg-pos-is-real*: $a < 0 \Longrightarrow b > 0 \Longrightarrow$ *is-real* $(a + b)$
using *add-pos-neg-is-real*[*of b a*] **by** (*simp add*: *ac-simps*)

lemma *nonneg-linorder-cases* [*consumes 2*, *case-names less eq greater*]:
assumes $0 \le a$ **and** $0 \le b$
and $a < b \Longrightarrow$ *thesis* $a = b \Longrightarrow$ *thesis* $b < a \Longrightarrow$ *thesis*
shows *thesis*
using *assms pos-pos-linear* **by** (*auto simp*: *le-less*)

lemma *nonpos-linorder-cases* [*consumes 2*, *case-names less eq greater*]:
assumes $a \le 0$ $b \le 0$
and $a < b \Longrightarrow$ *thesis* $a = b \Longrightarrow$ *thesis* $b < a \Longrightarrow$ *thesis*
shows *thesis*
using *assms neg-neg-linear* **by** (*auto simp*: *le-less*)

lemma *real-linear*:
assumes *is-real* a **and** *is-real* b **shows** $a < b \vee a = b \vee b < a$

using *pos-pos-linear neg-neg-linear assms* **by** (*auto dest*: *less-trans*[*of* - *0*])

lemma *real-linorder-cases* [*consumes 2*, *case-names less eq greater*]:
assumes *real*: *is-real a is-real b*
and *cases*: $a < b \Longrightarrow$ *thesis* $a = b \Longrightarrow$ *thesis* $b < a \Longrightarrow$ *thesis*
shows *thesis*
using *real-linear*[*OF real*] *cases* **by** *auto*

lemma
assumes *a*: *is-real a* **and** *b*: *is-real b*
shows *real-add-le-cancel-left-pos*: $c + a \leq c + b \longleftrightarrow a \leq b$
and *real-add-less-cancel-left-pos*: $c + a < c + b \longleftrightarrow a < b$
and *real-add-le-cancel-right-pos*: $a + c \leq b + c \longleftrightarrow a \leq b$
and *real-add-less-cancel-right-pos*: $a + c < b + c \longleftrightarrow a < b$
using *add-strict-left-mono*[*of b a c*] *add-strict-left-mono*[*of a b c*]
using *add-strict-right-mono*[*of b a c*] *add-strict-right-mono*[*of a b c*]
by (*atomize*(*full*), *cases rule*: *real-linorder-cases*[*OF a b*], *auto*)

lemma
assumes *a*: *is-real a* **and** *b*: *is-real b* **and** *c*: $0 < c$
shows *real-mult-le-cancel-left-pos*: $c * a \leq c * b \longleftrightarrow a \leq b$
and *real-mult-less-cancel-left-pos*: $c * a < c * b \longleftrightarrow a < b$
and *real-mult-le-cancel-right-pos*: $a * c \leq b * c \longleftrightarrow a \leq b$
and *real-mult-less-cancel-right-pos*: $a * c < b * c \longleftrightarrow a < b$
using *mult-strict-left-mono*[*of b a c*] *mult-strict-left-mono*[*of a b c*] *c*
using *mult-strict-right-mono*[*of b a c*] *mult-strict-right-mono*[*of a b c*] *c*
by (*atomize*(*full*), *cases rule*: *real-linorder-cases*[*OF a b*], *auto*)

lemma
assumes *a*: *is-real a* **and** *b*: *is-real b*
shows *not-le-real*: $\neg\ a \geq b \longleftrightarrow a < b$
and *not-less-real*: $\neg\ a > b \longleftrightarrow a \leq b$
by (*atomize*(*full*), *cases rule*: *real-linorder-cases*[*OF a b*], *auto simp*: *less-imp-le*)

lemma *real-mult-eq-0-iff*:
assumes *a*: *is-real a* **and** *b*: *is-real b*
shows $a * b = 0 \longleftrightarrow a = 0 \lor b = 0$
proof−
{ assume *l*: $a * b = 0$ **and** $a \neq 0$ **and** $b \neq 0$
with *a b* **have** $a < 0 \lor 0 < a$ **and** $b < 0 \lor 0 < b$ **by** *auto*
then have *False* **using** *mult-pos-pos*[*of a b*] *mult-pos-neg*[*of a b*] *mult-neg-pos*[*of a b*] *mult-neg-neg*[*of a b*]
by (*auto simp*:*l*)
} **then show** *?thesis* **by** *auto*
qed

end

lemma *real-pos-mult-max*:

```
  fixes a b c :: 'a :: semiring-real-line
  assumes c: c > 0 and a: is-real a and b: is-real b
  shows c * max a b = max (c * a) (c * b)
  by (rule hom-max, simp add: real-mult-le-cancel-left-pos[OF a b c])

class ring-abs-real-line = ordered-ring-abs + semiring-real-line

class semiring-1-real-line = semiring-real-line + monoid-mult + zero-less-one
begin

subclass ordered-semiring-1 by (unfold-locales, auto)

lemma power-both-mono: 1 ≤ a ⟹ m ≤ n ⟹ a ≤ b ⟹ a ^ m ≤ b ^ n
  using power-mono[of a b n] power-increasing[of m n a]
  by (auto simp: order.trans[OF zero-le-one])

lemma power-pos:
  assumes a0: 0 < a shows 0 < a ^ n
  by (induct n, insert mult-strict-mono[OF a0] a0, auto)

lemma power-neg:
  assumes a0: a < 0 shows odd n ⟹ a ^ n < 0 and even n ⟹ a ^ n > 0
 by (atomize(full), induct n, insert a0, auto simp add: mult-pos-neg2 mult-neg-neg)

lemma power-ge-0-iff:
  assumes a: is-real a
  shows 0 ≤ a ^ n ⟷ 0 ≤ a ∨ even n
using a proof (elim disjE)
  assume a < 0
  with power-neg[OF this, of n] show ?thesis by(cases even n, auto)
next
  assume 0 < a
  with power-pos[OF this] show ?thesis by auto
next
  assume a = 0
  then show ?thesis by (auto simp:power-0-left)
qed

lemma nonneg-power-less:
  assumes 0 ≤ a and 0 ≤ b shows a^n < b^n ⟷ n > 0 ∧ a < b
proof (insert assms, induct n arbitrary: a b)
  case 0
  then show ?case by auto
next
  case (Suc n)
  note a = ‹0 ≤ a›
  note b = ‹0 ≤ b›
  show ?case
  proof (cases n > 0)
```

case *True*
from *a b* **show** *?thesis*
proof (*cases rule*: *nonneg-linorder-cases*)
case *less*
then show *?thesis* **by** (*auto simp*: *Suc.hyps*[*OF a b*] *True intro*!:*mult-strict-mono'* *a b zero-le-power*)
next
case *eq*
then show *?thesis* **by** *simp*
next
case *greater*
with *Suc.hyps*[*OF b a*] *True* **have** $b \hat{} n < a \hat{} n$ **by** *auto*
with *mult-strict-mono'*[*OF greater this*] *b greater*
show *?thesis* **by** *auto*
qed
qed *auto*
qed

lemma *power-strict-mono*:
shows $a < b \Longrightarrow 0 \le a \Longrightarrow 0 < n \Longrightarrow a \hat{} n < b \hat{} n$
by (*subst nonneg-power-less*, *auto*)

lemma *nonneg-power-le*:
assumes $0 \le a$ **and** $0 \le b$ **shows** $a\hat{}n \le b\hat{}n \longleftrightarrow n = 0 \vee a \le b$
using *assms* **proof** (*cases rule*: *nonneg-linorder-cases*)
case *less*
with *power-strict-mono*[*OF this*, *of n*] *assms* **show** *?thesis* **by** (*cases n*, *auto*)
next
case *eq*
then show *?thesis* **by** *auto*
next
case *greater*
with *power-strict-mono*[*OF this*, *of n*] *assms* **show** *?thesis* **by** (*cases n*, *auto*)
qed

end

subclass (**in** *linordered-idom*) *semiring-1-real-line*
apply *unfold-locales*
by (*auto simp*: *mult-strict-left-mono mult-strict-right-mono mult-neg-neg*)

class *ring-1-abs-real-line* = *ring-abs-real-line* + *semiring-1-real-line*
begin

subclass *ring-1* **..**

lemma *abs-cases*:
assumes $a = 0 \Longrightarrow thesis$ **and** $|a| > 0 \Longrightarrow thesis$ **shows** *thesis*
using *assms* **by** *auto*

lemma *abs-linorder-cases*[*case-names less eq greater*]:
  assumes $|a| < |b| \Longrightarrow$ *thesis* **and** $|a| = |b| \Longrightarrow$ *thesis* **and** $|b| < |a| \Longrightarrow$ *thesis*
  shows *thesis*
  apply (*cases rule*: *nonneg-linorder-cases*[*of* $|a|$ $|b|$])
  using *assms* **by** *auto*

lemma [*simp*]:
  shows *not-le-abs-abs*: $\neg\ |a| \geq |b| \longleftrightarrow |a| < |b|$
    and *not-less-abs-abs*: $\neg\ |a| > |b| \longleftrightarrow |a| \leq |b|$
  by (*atomize*(*full*), *cases a b rule*: *abs-linorder-cases*, *auto simp*: *less-imp-le*)

lemma *abs-power-less* [*simp*]: $|a|\hat{}n < |b|\hat{}n \longleftrightarrow n > 0 \wedge |a| < |b|$
  by (*subst nonneg-power-less*, *auto*)

lemma *abs-power-le* [*simp*]: $|a|\hat{}n \leq |b|\hat{}n \longleftrightarrow n = 0 \vee |a| \leq |b|$
  by (*subst nonneg-power-le*, *auto*)

lemma *abs-power-pos* [*simp*]: $|a|\hat{}n > 0 \longleftrightarrow a \neq 0 \vee n = 0$
  using *power-pos*[*of* $|a|$] **by** (*cases n*, *auto*)

lemma *abs-power-nonneg* [*intro*!]: $|a|\hat{}n \geq 0$ **by** *auto*

lemma *abs-power-eq-0* [*simp*]: $|a|\hat{}n = 0 \longleftrightarrow a = 0 \wedge n \neq 0$
  apply (*induct n*, *force*)
  apply (*unfold power-Suc*)
  apply (*subst real-mult-eq-0-iff*, *auto*)**.**

end

instance *nat* :: *semiring-1-real-line* **by** (*intro-classes*, *auto*)
instance *int* :: *ring-1-abs-real-line***..**

lemma *vec-index-vec-of-list* [*simp*]: *vec-of-list xs* \$ i = *xs* ! i
  by *transfer* (*auto simp*: *mk-vec-def undef-vec-def dest*: *empty-nth*)

lemma *vec-of-list-append*: *vec-of-list* (*xs* @ *ys*) = *vec-of-list xs* $@_v$ *vec-of-list ys*
  by (*auto simp*: *nth-append*)

lemma *linf-norm-vec-of-list*:
  $\|$*vec-of-list xs*$\|_\infty$ = *max-list* (*map abs xs* @ [*0*])
  by (*simp add*: *linf-norm-vec-def*)

lemma *linf-norm-vec-as-Greatest*:
  fixes *v* :: *'a* :: *ring-1-abs-real-line vec*
  shows $\|v\|_\infty$ = (*GREATEST a*. $a \in$ *abs* ‘ *set* (*list-of-vec v*) $\cup$ {*0*})
  unfolding *linf-norm-vec-of-list*[*of list-of-vec v*, *simplified*]
  by (*subst max-list-as-Greatest*, *auto*)

```
lemma vec-of-poly-pCons:
  assumes f ≠ 0
  shows vec-of-poly (pCons a f) = vec-of-poly f @v vec-of-list [a]
  using assms
  by (auto simp: vec-eq-iff Suc-diff-le)

lemma vec-of-poly-as-vec-of-list:
  assumes f ≠ 0
  shows vec-of-poly f = vec-of-list (rev (coeffs f))
proof (insert assms, induct f)
  case 0
  then show ?case by auto
next
  case (pCons a f)
  then show ?case
    by (cases f = 0, auto simp: vec-of-list-append vec-of-poly-pCons)
qed

lemma linf-norm-vec-of-poly [simp]:
  fixes f :: 'a :: ring-1-abs-real-line poly
  shows ‖vec-of-poly f‖∞ = ‖f‖∞
proof (cases f = 0)
  case False
  then show ?thesis
    apply (unfold vec-of-poly-as-vec-of-list linf-norm-vec-of-list linf-norm-poly-def)
    apply (subst (1 2) max-list-as-Greatest, auto).
qed simp

lemma linf-norm-poly-as-Greatest:
  fixes f :: 'a :: ring-1-abs-real-line poly
  shows ‖f‖∞ = (GREATEST a. a ∈ abs ` set (coeffs f) ∪ {0})
  using linf-norm-vec-as-Greatest[of vec-of-poly f]
  by simp

lemma vec-index-le-linf-norm:
  fixes v :: 'a :: ring-1-abs-real-line vec
  assumes i < dim-vec v
  shows |v$i| ≤ ‖v‖∞
apply (unfold linf-norm-vec-def, rule le-max-list) using assms
apply (auto simp: in-set-conv-nth intro!: imageI exI[of - i]).

lemma coeff-le-linf-norm:
  fixes f :: 'a :: ring-1-abs-real-line poly
  shows |coeff f i| ≤ ‖f‖∞
  using vec-index-le-linf-norm[of degree f − i vec-of-poly f]
  by (cases i ≤ degree f, auto simp: coeff-eq-0)

class conjugatable-ring-1-abs-real-line = conjugatable-ring + ring-1-abs-real-line +
power +
```

assumes *sq-norm-as-sq-abs* [*simp*]: $\|a\|^2 = |a|^2$
begin
subclass *conjugatable-ordered-ring* **by** (*unfold-locales*, *simp*)
end

instance *int* :: *conjugatable-ring-1-abs-real-line*
by (*intro-classes*, *simp add*: *numeral-2-eq-2*)

instance *rat* :: *conjugatable-ring-1-abs-real-line*
by (*intro-classes*, *simp add*: *numeral-2-eq-2*)

instance *real* :: *conjugatable-ring-1-abs-real-line*
by (*intro-classes*, *simp add*: *numeral-2-eq-2*)

instance *complex* :: *semiring-1-real-line*
apply *intro-classes*
by (*auto simp*: *complex-Re-Im-cancel-iff mult-le-cancel-left mult-le-cancel-right mult-neg-neg*)

Due to the assumption $?a \leq |?a|$ from Groups.thy, *complex* cannot be *ring-1-abs-real-line*!

instance *complex* :: *ordered-ab-group-add-abs* **oops**

lemma *sq-norm-as-sq-abs* [*simp*]: (*sq-norm* :: $'a$ :: *conjugatable-ring-1-abs-real-line* $\Rightarrow$ $'a$) = *power2* $\circ$ *abs*
by *auto*

lemma *sq-norm-vec-le-linf-norm*:
fixes v :: $'a$ :: {*conjugatable-ring-1-abs-real-line*} *vec*
assumes $v \in$ *carrier-vec* n
shows $\|v\|^2 \leq$ *of-nat* $n * \|v\|_\infty{}^2$
proof (*insert assms*, *induct rule*: *carrier-vec-induct*)
case (*Suc n a v*)
have [*dest*!]: $\neg\ |a| \leq \|v\|_\infty \Longrightarrow$ *of-nat* $n * \|v\|_\infty{}^2 \leq$ *of-nat* $n * |a|^2$
by (*rule real-linorder-cases*[*of* $|a|$ $\|v\|_\infty$], *insert Suc*, *auto simp*: *less-le intro*!: *power-mono mult-left-mono*)
from *Suc* **show** *?case*
by (*auto simp*: *ring-distribs max-def intro*!:*add-mono power-mono*)
qed *simp*

lemma *sq-norm-poly-le-linf-norm*:
fixes p :: $'a$ :: {*conjugatable-ring-1-abs-real-line*} *poly*
shows $\|p\|^2 \leq$ *of-nat* (*degree* p + *1*) $* \|p\|_\infty{}^2$
using *sq-norm-vec-le-linf-norm*[*of vec-of-poly p degree p* + *1*]
by (*auto simp*: *carrier-dim-vec*)

lemma *coeff-le-sq-norm*:
fixes f :: $'a$ :: {*conjugatable-ring-1-abs-real-line*} *poly*
shows $|$*coeff* f $i|^2 \leq \|f\|^2$

proof (*induct f arbitrary*: *i*)
  case (*pCons a f*)
  show *?case*
  proof (*cases i*)
    case (*Suc ii*)
    note *pCons*(*2*)[*of ii*]
    also have $\|f\|^2 \leq |a|^2 + \|f\|^2$ **by** *auto*
    finally show *?thesis* **unfolding** *Suc* **by** *auto*
  qed *auto*
qed *simp*

lemma *max-norm-witness*:
  fixes *f* :: *'a* :: *ordered-ring-abs poly*
  shows ∃ *i*. $\|f\|_\infty = |$*coeff f i*$|$
  by (*induct f*, *auto simp add*: *max-def intro*: *exI*[*of - Suc -*] *exI*[*of - 0*])

lemma *max-norm-le-sq-norm*:
  fixes *f* :: *'a* :: *conjugatable-ring-1-abs-real-line poly*
shows $\|f\|_\infty{}^2 \leq \|f\|^2$
proof −
  from *max-norm-witness*[*of f*] **obtain** *i* **where** *id*: $\|f\|_\infty = |$*coeff f i*$|$ **by** *auto*
  show *?thesis* **unfolding** *id* **using** *coeff-le-sq-norm*[*of f i*] **by** *auto*
qed

lemma (**in** *conjugatable-ring*) *conjugate-minus*: *conjugate* (*x* − *y*) = *conjugate x* − *conjugate y*
  by (*unfold diff-conv-add-uminus conjugate-dist-add conjugate-neg*, *rule*)

lemma *conjugate-1*[*simp*]: (*conjugate 1* :: *'a* :: {*conjugatable-ring*, *ring-1*}) = *1*
proof−
  have *conjugate 1* ∗ *1* = (*conjugate 1* :: *'a*) **by** *simp*
  also have *conjugate* . . . = *1* **by** *simp*
  finally show *?thesis* **by** (*unfold conjugate-dist-mul*, *simp*)
qed

lemma *conjugate-of-int* [*simp*]:
  (*conjugate* (*of-int x*) :: *'a* :: {*conjugatable-ring*,*ring-1*}) = *of-int x*
proof (*induct x*)
  case (*nonneg n*)
  then show *?case* **by** (*induct n*, *auto simp*: *conjugate-dist-add*)
next
  case (*neg n*)
  then show *?case* **apply** (*induct n*, *auto simp*: *conjugate-minus conjugate-neg*)
    by (*metis conjugate-1 conjugate-dist-add one-add-one*)
qed

lemma *sq-norm-of-int*: $\|$*map-vec of-int v* :: *'a* :: {*conjugatable-ring*,*ring-1*} *vec*$\|^2$

$= \textit{of-int}\ \|v\|^2$
unfolding *sq-norm-vec-as-cscalar-prod scalar-prod-def*
unfolding *hom-distribs*
by (*rule sum.cong*, *auto*)

end

5 Lattice

This theory implements the mathematical definition of lattice by means of locales and shows it forms a (HOL-Algebra) module.

theory *Vector-Lattice-Locale*
imports
HOL−Library.Multiset
Norms
Missing-Lemmas
begin

locale *vlattice* = *abelian-group G* **for** *G* (**structure**)
begin

fun *nat-mult* **where** *nat-mult 0 v* = **0** | *nat-mult* (*Suc n*) *v* = *v* ⊕ *nat-mult n v*

lemma *nat-mult-closed* [*simp*]: *v* ∈ *carrier G* ⟹ *nat-mult n v* ∈ *carrier G*
by (*induct n*, *auto*)

lemma *nat-mult-add-distrib1* [*simp*]:
assumes *v*: *v* ∈ *carrier G* **shows** *nat-mult* (*x*+*y*) *v* = *nat-mult x v* ⊕ *nat-mult y v*
by (*induct x*, *insert v*, *auto intro*!: *a-assoc*[*symmetric*])

lemma *nat-mult-add-distrib2* [*simp*]:
assumes *v* ∈ *carrier G* **and** *w* ∈ *carrier G*
shows *nat-mult x* (*v* ⊕ *w*) = *nat-mult x v* ⊕ *nat-mult x w*
proof(*induct x*)
case (*Suc x*)
have *nat-mult* (*Suc x*) (*v* ⊕ *w*) = *v* ⊕ *w* ⊕ *nat-mult x* (*v* ⊕ *w*) **by** *simp*
also have ... = *v* ⊕ (*w* ⊕ *nat-mult x v*) ⊕ *nat-mult x w*
using *assms a-assoc* **by** (*auto simp*: *Suc*)
also have *w* ⊕ *nat-mult x v* = *nat-mult x v* ⊕ *w* **using** *assms a-comm* **by** *auto*
finally show *?case* **using** *a-assoc assms* **by** *auto*
qed *simp*

definition *int-mult* (**infixl** · *70*)
where *x* · *v* ≡ *if x* ≥ *0 then nat-mult* (*Int.nat x*) *v else* ⊖ *nat-mult* (*Int.nat* (−*x*)) *v*

lemma *int-mult-closed* [*simp*]: $v \in carrier\ G \Longrightarrow x \cdot v \in carrier\ G$
  by (*unfold int-mult-def*, *auto*)

lemma [*simp*]: **assumes** $v \in carrier\ G$
 shows *zero-int-mult*: $0 \cdot v = \mathbf{0}$ **and** *one-int-mult*: $1 \cdot v = v$ **and** *uminus-int-mult*: $-x \cdot v = \ominus (x \cdot v)$
  using *assms* **by** (*simp-all add*: *int-mult-def*)

lemma *int-mult-add-1*:
  assumes v: $v \in carrier\ G$
  shows $(x + 1) \cdot v = v \oplus x \cdot v$ (**is** $?l = ?r$)
proof (*cases x* -1::*int rule*:*linorder-cases*)
  case *greater*
  then have $x \geq 0$ **by** *auto*
  then obtain n **where** x: $x = int\ n$ **using** *zero-le-imp-eq-int* **by** *auto*
  have $?l = int\text{-}mult\ (x + int\ 1)\ v$ **by** *simp*
  also have ... $= ?r$ **using** v **by** (*unfold x int-mult-def nat-int-add*, *auto*)
  finally show *?thesis*.
next
  case *equal*
  with v **show** *?thesis* **by** (*auto simp*: *a-inv-def*)
next
  case *less*
  then have $- x - 2 \geq 0$ **by** *auto*
  from *zero-le-imp-eq-int*[*OF this*] **obtain** n **where** $- x - 2 = int\ n$ **by** *auto*
  then have x: $x = - (int\ n + int\ 2)$ **by** *auto*
  have $?r = \ominus v \ominus int\ n \cdot v$
    using v
    by (*unfold x int-mult-def add.inverse-inverse nat-int-add*,
        simp add: *a-assoc*[*symmetric*] *minus-add r-neg minus-eq*)
  also have ... $= (- (int\ 1 + int\ n)) \cdot v$
    using v
    by (*unfold int-mult-def add.inverse-inverse nat-int-add*,
        simp add:*add.inv-mult-group a-comm minus-eq*)
  also have ... $= ?l$ **by** (*auto simp*: x)
  finally show *?thesis* **by** *simp*
qed

lemmas *int-mult-1-add* = *int-mult-add-1*[*folded add.commute*[*of 1*]]

lemma *int-mult-minus-1*:
  assumes v: $v \in carrier\ G$
  shows $(x - 1) \cdot v = \ominus v \oplus x \cdot v$ (**is** $?l = ?r$)
proof (*cases x 1*::*int rule*:*linorder-cases*)
  case *less*
  then have $-x \geq 0$ **by** *auto*
  from *zero-le-imp-eq-int*[*OF this*] **obtain** n **where** x: $-x = int\ n$ **by** *auto*
  have $?l = (-(int\ n + int\ 1)) \cdot v$ **by** (*simp add*: *x*[*symmetric*])
  also have ... $= \ominus v \oplus - int\ n \cdot v$

using v **by** (*unfold int-mult-def add.inverse-inverse nat-int-add*, *simp add*: *minus-add*)
also have ... $=$ *?r* **by** (*fold* x, *auto*)
finally show *?thesis*.
next
case *equal*
with v **show** *?thesis* **by** (*auto simp*: *l-neg*)
next
case *greater*
then have $x - 2 \geq 0$ **by** *auto*
from *zero-le-imp-eq-int*[*OF this*] **obtain** n **where** $x - 2 = int\ n$ **by** *auto*
then have x: $x = int\ n + int\ 2$ **by** *auto*
have $?r = \ominus\ v \oplus (v \oplus (v \oplus nat\text{-}mult\ n\ v))$
by (*unfold x int-mult-def add.inverse-inverse nat-int-add*, *simp*)
also have $\ldots = v \oplus int\ n \cdot v$ **using** v **by** (*simp add*: *a-assoc*[*symmetric*] *l-neg int-mult-def*)
also have ... $= (int\ 1 + int\ n) \cdot v$ **by** (*unfold int-mult-def minus-minus nat-int-add*, *simp*)
also have ... $=$ *?l* **by** (*auto simp*: x)
finally show *?thesis* **by** *simp*
qed

lemma *int-mult-add-distrib1*:
assumes v [*simp*]: $v \in carrier\ G$
shows $(x+y) \cdot v = x \cdot v \oplus y \cdot v$
proof (*induct* x)
case (*nonneg* n)
then show *?case* **using** v
by (*induct* n, *auto simp*: *ac-simps a-assoc*[*symmetric*] *int-mult-1-add*)
next
case (*neg* n)
show *?case*
proof(*induct* n)
case *0* **show** *?case* **using** v **by** (*auto simp add*: *int-mult-minus-1 minus-eq*)
case *IH*: (*Suc* n)
have $(-\ int\ (Suc\ (Suc\ n)) + y) = (-\ int\ (Suc\ n) + y - 1)$ **by** *simp*
also have $\ldots \cdot v = \ominus\ v \oplus (-\ int\ (Suc\ n) + y) \cdot v$ **unfolding** *int-mult-minus-1*[*OF* v] **by** *simp*
also have $\ldots = \ominus\ v \oplus (-\ int\ (Suc\ n) \cdot v \oplus y \cdot v)$ **by** (*unfold IH*, *simp*)
also have $\ldots = (\ominus\ v \oplus (-\ int\ (Suc\ n) \cdot v)) \oplus y \cdot v$ **by** (*auto simp*: *a-assoc*)
also have $\ldots = (-\ int\ (Suc\ (Suc\ n)) \cdot v) \oplus y \cdot v$ **by** (*subst int-mult-minus-1*[*symmetric*], *auto*)
finally show *?case*.
qed
qed

lemma *int-mult-minus-distrib1*:
assumes $v \in carrier\ G$
shows $(x - y) \cdot v = x \cdot v \ominus y \cdot v$

using *assms* **by** (*unfold diff-conv-add-uminus int-mult-add-distrib1*, *simp add*: *minus-eq*)

lemma *int-mult-mult*:
assumes v [*simp*]: $v \in$ *carrier G*
shows $x \cdot (y \cdot v) = x * y \cdot v$
proof (*cases x*)
case *x*: (*nonneg n*)
show *?thesis* **by** (*unfold x*, *induct n*, *auto simp*: *field-simps int-mult-add-distrib1*)
next
case *x*: (*neg n*)
show *?thesis*
proof (*unfold x*, *induct n*)
case *0*
then show *?case* **by** *simp*
next
case (*Suc n*)
have $-$ *int* (*Suc* (*Suc n*)) $\cdot$ ($y \cdot v$) = ($-$ *int* (*Suc n*) $-$ *1*) $\cdot$ ($y \cdot v$) **by** *simp*
also have ... = $\ominus$ ($y \cdot v$) $\oplus$ $-$ *int* (*Suc n*) $\cdot$ ($y \cdot v$) **by** (*rule int-mult-minus-1*, *simp*)
also have ... = ($-$ $y \cdot v$) $\oplus$ $-$ *int* (*Suc n*) $*$ $y \cdot v$ **unfolding** *Suc* **by** *simp*
also have ... = $-$ *int* (*Suc* (*Suc n*)) $*$ $y \cdot v$
by (*subst int-mult-add-distrib1*[*symmetric*], *auto simp*: *left-diff-distrib*)
finally show *?case* **by** (*simp add*: *field-simps*)
qed
qed

lemma *int-mult-add-distrib2*[*simp*]:
assumes $v \in$ *carrier G* **and** $w \in$ *carrier G*
shows $x \cdot (v \oplus w) = x \cdot v \oplus x \cdot w$ **using** *assms* **by** (*auto simp*: *int-mult-def minus-add*)

abbreviation *int-ring*
where *int-ring* $\equiv$ (| *carrier* = *UNIV*::*int set*, *monoid.mult* = *op* $*$, *one* = *1*, *zero* = *0*, *add* = *op* + |)

abbreviation *lattice-module*
where *lattice-module* $\equiv$
(| *carrier* = *carrier G*, *monoid.mult* = *op* $\otimes$, *one* = **1**, *zero* = **0**, *add* = *op* $\oplus$, *module.smult* = *int-mult* |)

sublocale *module*: *module int-ring lattice-module*
rewrites *carrier int-ring* = *UNIV*
and *monoid.mult int-ring* = *op* $*$
and *one int-ring* = *1*
and *zero int-ring* = *0*
and *add int-ring* = *op* +
and *carrier lattice-module* = *carrier G*
and *monoid.mult lattice-module* = *op* $\otimes$

and *one lattice-module* = **1**
and *zero lattice-module* = **0**
and *add lattice-module* = *op* ⊕
and *module.smult lattice-module* = *int-mult*
by (*unfold-locales*,
auto simp: *field-simps Units-def int-mult-mult l-neg r-neg int-mult-add-distrib1*
intro!: *a-assoc a-comm exI*[*of* - ⊖ -] *bexI*[*of* - ⊖ -])

end

end

6 Gram-Schmidt

theory *Gram-Schmidt-2*
imports *Jordan-Normal-Form.Gram-Schmidt*
Jordan-Normal-Form.Show-Matrix
Jordan-Normal-Form.Matrix-Impl
Norms
begin

no-notation *Gram-Schmidt.cscalar-prod* (**infix** ·*c 70*)

lemma *vec-conjugate-connect*[*simp*]: *Gram-Schmidt.vec-conjugate* = *conjugate*
by (*auto simp*: *vec-conjugate-def conjugate-vec-def*)

lemma *scalar-prod-ge-0*: (*x* :: ′*a* :: *linordered-idom vec*) · *x* ≥ *0*
unfolding *scalar-prod-def*
by (*rule sum-nonneg*, *auto*)

class *trivial-conjugatable-ordered-field* =
conjugatable-ordered-field + *linordered-idom* +
assumes *conjugate-id* [*simp*]: *conjugate x* = *x*

lemma *cscalar-prod-is-scalar-prod*[*simp*]: (*x* :: ′*a* :: *trivial-conjugatable-ordered-field vec*) ·*c y* = *x* · *y*
unfolding *conjugate-id*
by (*rule arg-cong*[*of* - - *scalar-prod x*], *auto*)

lemma *corthogonal-is-orthogonal*[*simp*]:
corthogonal (*xs* :: ′*a* :: *trivial-conjugatable-ordered-field vec list*) = *orthogonal xs*
unfolding *corthogonal-def orthogonal-def* **by** *simp*

instance *rat* :: *trivial-conjugatable-ordered-field*
by (*standard*, *auto*)

instance *real* :: *trivial-conjugatable-ordered-field*
by (*standard*, *auto*)

lemma *vec-right-zero*[*simp*]:
(v :: $'a$ :: *monoid-add vec*) $\in$ *carrier-vec* n $\Longrightarrow v + 0_v\ n = v$
by *auto*

context *vec-module* **begin**

lemma *sumlist-dim*: **assumes** $\bigwedge x.\ x \in$ *set xs* $\Longrightarrow x \in$ *carrier-vec* n
shows *dim-vec* (*sumlist xs*) $= n$
using *sumlist-carrier assms*
by *fastforce*

lemma *sumlist-vec-index*: **assumes** $\bigwedge x.\ x \in$ *set xs* $\Longrightarrow x \in$ *carrier-vec* n
and $i < n$
shows *sumlist xs* \$ i = *sum-list* (*map* (λ x. x \$ i) *xs*)
unfolding *M.sumlist-def* **using** *assms*(*1*) **proof**(*induct xs*)
case (*Cons a xs*)
hence *cond*:$\bigwedge x.\ x \in$ *set xs* $\Longrightarrow x \in$ *carrier-vec* n **by** *auto*
from *Cons*(*1*)[*OF cond*] **have** *IH*:*foldr op* + *xs* (0_v n) \$ i = ($\sum x \leftarrow xs.\ x$ \$ i)
by *auto*
have (a + *foldr op* + *xs* (0_v n)) \$ i = a \$ i + ($\sum x \leftarrow xs.\ x$ \$ i)
apply(*subst index-add-vec*) **unfolding** *IH*
using *sumlist-dim*[*OF cond*,*unfolded M.sumlist-def*] *assms* **by** *auto*
then show *?case* **by** *auto* **next**
case *Nil* **thus** *?case* **using** *assms* **by** *auto*
qed

lemma *scalar-prod-left-sum-distrib*:
assumes *vs*: $\bigwedge v.\ v \in$ *set vvs* $\Longrightarrow v \in$ *carrier-vec* n **and** w: $w \in$ *carrier-vec* n
shows *sumlist vvs* $\cdot$ w = *sum-list* (*map* (λ v. $v \cdot w$) *vvs*)
using *vs*
proof (*induct vvs*)
case (*Cons v vs*)
from *Cons* **have** v: $v \in$ *carrier-vec* n **and** *vs*: *sumlist vs* $\in$ *carrier-vec* n
by (*auto intro*!: *sumlist-carrier*)
have *sumlist* (v # *vs*) $\cdot$ w = *sumlist* ([v] @ *vs*) $\cdot$ w **by** *auto*
also have ... = (v + *sumlist vs*) $\cdot$ w
by (*subst sumlist-append*, *insert Cons v vs*, *auto*)
also have ... = $v \cdot w$ + (*sumlist vs* $\cdot$ w)
by (*rule add-scalar-prod-distrib*[*OF v vs w*])
finally show *?case* **using** *Cons* **by** *auto*
qed (*insert w*, *auto*)

definition *lattice-of* :: $'a$ *vec list* $\Rightarrow$ $'a$ *vec set* **where**
lattice-of fs = *range* (λ c. *sumlist* (*map* (λ i. *of-int* (c i) $\cdot_v$ *fs* ! i) [*0* ..< *length fs*]))

lemma *in-latticeE*: **assumes** $f \in$ *lattice-of fs* **obtains** c **where**
$f = \textit{sumlist}\ (\textit{map}\ (\lambda\ i.\ \textit{of-int}\ (c\ i) \cdot_v \textit{fs}\ !\ i)\ [0\ ..<\ \textit{length fs}])$
using *assms* **unfolding** *lattice-of-def* **by** *auto*

lemma *in-latticeI*: **assumes** $f = \textit{sumlist}\ (\textit{map}\ (\lambda\ i.\ \textit{of-int}\ (c\ i) \cdot_v \textit{fs}\ !\ i)\ [0\ ..<\ \textit{length fs}])$
shows $f \in$ *lattice-of fs*
using *assms* **unfolding** *lattice-of-def* **by** *auto*

lemma *basis-in-latticeI*: **assumes** *fs*: *set fs* $\subseteq$ *carrier-vec n*
and f: $f \in$ *set fs*
shows $f \in$ *lattice-of fs*
proof −
from f **obtain** i **where** f: $f = \textit{fs}\ !\ i$ **and** i: $i < \textit{length fs}$ **unfolding** *set-conv-nth* **by** *auto*
let $?c = \lambda\ j.\ \textit{if}\ j = i\ \textit{then}\ 1\ \textit{else}\ 0$
have *id*: $[0\ ..<\ \textit{length fs}] = [0\ ..<\ i]\ @\ [i]\ @\ [\textit{Suc}\ i\ ..<\ \textit{length fs}]$
by (*rule nth-equalityI*, *insert i*, *auto simp*: *nth-append*, *rename-tac k*, *case-tac* $k = i$, *auto*)
from *fs* **have** *fs*[*intro*!]: $\bigwedge\ i.\ i < \textit{length fs} \Longrightarrow \textit{fs}\ !\ i \in \textit{carrier-vec}\ n$ **unfolding** *set-conv-nth* **by** *auto*
have [*simp*]: $\bigwedge\ i.\ i < \textit{length fs} \Longrightarrow \textit{dim-vec}\ (\textit{fs}\ !\ i) = n$ **using** *fs* **by** *auto*
show *?thesis* **unfolding** f
apply (*rule in-latticeI*[*of* - *?c*], *unfold id map-append*, *insert i*)
apply (*subst sumlist-append*,*force*,*force*, *subst sumlist-append*, *force*, *force*)
by (*subst sumlist-neutral*, *force*, *subst sumlist-neutral*, *force*, *auto*)
qed

lemma *lattice-of-swap*: **assumes** *fs*: *set fs* $\subseteq$ *carrier-vec n*
and *ij*: $i < \textit{length fs}\ j < \textit{length fs}\ i \neq j$
and *gs*: $\textit{gs} = \textit{fs}[\ i := \textit{fs}\ !\ j,\ j := \textit{fs}\ !\ i]$
shows *lattice-of gs* = *lattice-of fs*
proof −
{
fix $i\ j$ **and** *fs* :: *'a vec list*
assume $*$: $i < j\ j < \textit{length fs}$ **and** *fs*: *set fs* $\subseteq$ *carrier-vec n*
let $?gs = \textit{fs}[\ i := \textit{fs}\ !\ j,\ j := \textit{fs}\ !\ i]$
let $?len = [0..<i]\ @\ [i]\ @\ [\textit{Suc}\ i..<j]\ @\ [j]\ @\ [\textit{Suc}\ j..<\textit{length fs}]$
have $[0\ ..<\ \textit{length fs}] = [0\ ..<\ j]\ @\ [j]\ @\ [\textit{Suc}\ j\ ..<\ \textit{length fs}]$ **using** $*$
by (*metis append-Cons append-self-conv2 less-Suc-eq-le less-imp-add-positive upt-add-eq-append*
upt-conv-Cons zero-less-Suc)
also have $[0\ ..<\ j] = [0\ ..<\ i]\ @\ [i]\ @\ [\textit{Suc}\ i\ ..<\ j]$ **using** $*$
by (*metis append-Cons append-self-conv2 less-Suc-eq-le less-imp-add-positive upt-add-eq-append*
upt-conv-Cons zero-less-Suc)
finally have *len*: $[0..<\textit{length fs}] = ?len$ **by** *simp*
from *fs* **have** *fs*: $\bigwedge\ i.\ i < \textit{length fs} \Longrightarrow \textit{fs}\ !\ i \in \textit{carrier-vec}\ n$ **unfolding**

set-conv-nth **by** *auto*
{
fix *f*
assume $f \in$ *lattice-of fs*
from *in-latticeE*[*OF this*, *unfolded len*] **obtain** *c* **where**
f: f = *sumlist* (*map* (λi. *of-int* (*c i*) $\cdot_v$ *fs* ! *i*) *?len*) **by** *auto*
define *sc* **where** *sc* = (λ *xs*. *sumlist* (*map* (λi. *of-int* (*c i*) $\cdot_v$ *fs* ! *i*) *xs*))
define *d* **where** *d* = (λ *k*. *if k* = *i then c j else if k* = *j then c i else c k*)
define *sd* **where** *sd* = (λ *xs*. *sumlist* (*map* (λi. *of-int* (*d i*) $\cdot_v$ *?gs* ! *i*) *xs*))
have *isc*: *set is* $\subseteq$ {*0* ..< *length fs*} $\Longrightarrow$ *sc is* $\in$ *carrier-vec n* **for** *is*
unfolding *sc-def* **by** (*intro sumlist-carrier*, *auto simp*: *fs*)
let *?a* = *sc* [*0*..<*i*] **let** *?b* = *sc* [*i*] **let** *?c* = *sc* [*Suc i* ..< *j*] **let** *?d* = *sc* [*j*]
let *?e* = *sc* [*Suc j* ..< *length fs*]
let *?A* = *sd* [*0*..<*i*] **let** *?B* = *sd* [*i*] **let** *?C* = *sd* [*Suc i* ..< *j*] **let** *?D* = *sd* [*j*]
let *?E* = *sd* [*Suc j* ..< *length fs*]
let *?CC* = *carrier-vec n*
have *ae*: *?a* $\in$ *?CC* *?b* $\in$ *?CC* *?c* $\in$ *?CC* *?d* $\in$ *?CC* *?e* $\in$ *?CC*
using * **by** (*auto intro*: *isc*)
have *sc-sd*: {*i*,*j*} $\cap$ *set is* $\subseteq$ {} $\Longrightarrow$ *sc is* = *sd is* **for** *is*
unfolding *sc-def sd-def* **by** (*rule arg-cong*[*of - - sumlist*], *rule map-cong*, *auto simp*: *d-def*)
have *f* = *?a* + (*?b* + (*?c* + (*?d* + *?e*)))
unfolding *f map-append sc-def* **using** *fs* *
by ((*subst sumlist-append*, *force*, *force*)+, *simp*)
also have ... = *?a* + (*?d* + (*?c* + (*?b* + *?e*))) **using** * **by** *auto*
also have ... = *?A* + (*?d* + (*?C* + (*?b* + *?E*)))
using * **by** (*auto simp*: *sc-sd*)
also have *?b* = *?D* **unfolding** *sd-def sc-def d-def* **using** * **by** (*auto simp*: *d-def*)
also have *?d* = *?B* **unfolding** *sd-def sc-def* **using** * **by** (*auto simp*: *d-def*)

finally have *f* = *?A* + (*?B* + (*?C* + (*?D* + *?E*))) .
also have ... = *sumlist* (*map* (λi. *of-int* (*d i*) $\cdot_v$ *?gs* ! *i*) *?len*)
unfolding *f map-append sd-def* **using** *fs* *
by ((*subst sumlist-append*, *force*, *force*)+, *simp*)
also have ... = *sumlist* (*map* (λi. *of-int* (*d i*) $\cdot_v$ *?gs* ! *i*) [*0* ..< *length ?gs*])
unfolding *len*[*symmetric*] **by** *simp*
finally have *f* = *sumlist* (*map* (λi. *of-int* (*d i*) $\cdot_v$ *?gs* ! *i*) [*0* ..< *length ?gs*]) .
from *in-latticeI*[*OF this*] **have** $f \in$ *lattice-of ?gs* .
}
hence *lattice-of fs* $\subseteq$ *lattice-of ?gs* **by** *blast*
} **note** *main* = *this*
{
fix *i j* **and** *fs* :: *'a vec list*
assume *: *i* < *length fs* *j* < *length fs* *i* $\neq$ *j* **and** *fs*: *set fs* $\subseteq$ *carrier-vec n*
let *?gs* = *fs*[*i* := *fs* ! *j*, *j* := *fs* ! *i*]
have *lattice-of fs* $\subseteq$ *lattice-of ?gs*

```
    proof (cases i < j)
      case True
      from main[OF this *(2) fs] show ?thesis .
    next
      case False
      with *(3) have j < i by auto
      from main[OF this *(1) fs]
      have lattice-of fs ⊆ lattice-of (fs[j := fs ! i, i := fs ! j]) .
      also have fs[j := fs ! i, i := fs ! j] = ?gs using *
        by (metis list-update-swap)
      finally show ?thesis .
    qed
  } note sub = this
  from sub[OF ij fs]
  have one: lattice-of fs ⊆ lattice-of gs unfolding gs .
  have lattice-of gs ⊆ lattice-of (gs[i := gs ! j, j := gs ! i])
    by (rule sub, insert ij fs, auto simp: gs)
  also have gs[i := gs ! j, j := gs ! i] = fs unfolding gs using ij
    by (intro nth-equalityI, force, intro allI,
      rename-tac k, case-tac k = i, force, case-tac k = j, auto)
  finally show ?thesis using one by auto
qed

lemma lattice-of-add: assumes fs: set fs ⊆ carrier-vec n
  and ij: i < length fs j < length fs i ≠ j
  and gs: gs = fs[ i := fs ! i + of-int l ·v fs ! j]
shows lattice-of gs = lattice-of fs
proof −
  {
    fix i j l and fs :: 'a vec list
    assume *: i < j j < length fs and fs: set fs ⊆ carrier-vec n
    note * = ij(1) *
    let ?gs = fs[ i := fs ! i + of-int l ·v fs ! j]
    let ?len = [0..<i] @ [i] @ [Suc i..<j] @ [j] @ [Suc j..<length fs]
    have [0 ..< length fs] = [0 ..< j] @ [j] @ [Suc j ..< length fs] using *
      by (metis append-Cons append-self-conv2 less-Suc-eq-le less-imp-add-positive
upt-add-eq-append
          upt-conv-Cons zero-less-Suc)
    also have [0 ..< j] = [0 ..< i] @ [i] @ [Suc i ..< j] using *
      by (metis append-Cons append-self-conv2 less-Suc-eq-le less-imp-add-positive
upt-add-eq-append
          upt-conv-Cons zero-less-Suc)
    finally have len: [0..<length fs] = ?len by simp
     from fs have fs: ⋀ i. i < length fs ⟹ fs ! i ∈ carrier-vec n unfolding
set-conv-nth by auto
    from fs have fsd: ⋀ i. i < length fs ⟹ dim-vec (fs ! i) = n by auto
    from fsd[of i] fsd[of j] * have fsd: dim-vec (fs ! i) = n dim-vec (fs ! j) = n
by auto
    {
```

fix f
assume $f \in$ *lattice-of fs*
from *in-latticeE*[*OF this*, *unfolded len*] **obtain** c **where**
f: $f = $ *sumlist* (*map* ($\lambda i.$ *of-int* (c i) $\cdot_v$ *fs* ! i) *?len*) **by** *auto*
define *sc* **where** *sc* = (λ *xs*. *sumlist* (*map* ($\lambda i.$ *of-int* (c i) $\cdot_v$ *fs* ! i) *xs*))
define d **where** $d = (\lambda\ k.\ \textit{if}\ k = j\ \textit{then}\ c\ j - c\ i * l\ \textit{else}\ c\ k)$
define *sd* **where** *sd* = (λ *xs*. *sumlist* (*map* ($\lambda i.$ *of-int* (d i) $\cdot_v$ *?gs* ! i) *xs*))
have *isc*: *set is* $\subseteq$ {*0* ..< *length fs*} $\Longrightarrow$ *sc is* $\in$ *carrier-vec n* **for** *is*
unfolding *sc-def* **by** (*intro sumlist-carrier*, *auto simp*: *fs*)
have *isd*: *set is* $\subseteq$ {*0* ..< *length fs*} $\Longrightarrow$ *sd is* $\in$ *carrier-vec n* **for** *is*
unfolding *sd-def* **using** $*$ **by** (*intro sumlist-carrier*, *auto*, *rename-tac k*, *case-tac* $k = i$, *auto simp*: *fs*)
let *?a* = *sc* [*0*..<i] **let** *?b* = *sc* [i] **let** *?c* = *sc* [*Suc i* ..< j] **let** *?d* = *sc* [j]
let *?e* = *sc* [*Suc j* ..< *length fs*]
let *?A* = *sd* [*0*..<i] **let** *?B* = *sd* [i] **let** *?C* = *sd* [*Suc i* ..< j] **let** *?D* = *sd* [j]
let *?E* = *sd* [*Suc j* ..< *length fs*]
let *?CC* = *carrier-vec n*
have *ae*: *?a* $\in$ *?CC* *?b* $\in$ *?CC* *?c* $\in$ *?CC* *?d* $\in$ *?CC* *?e* $\in$ *?CC*
using $*$ **by** (*auto intro*: *isc*)
have *AE*: *?A* $\in$ *?CC* *?B* $\in$ *?CC* *?C* $\in$ *?CC* *?D* $\in$ *?CC* *?E* $\in$ *?CC*
using $*$ **by** (*auto intro*: *isd*)
have *sc-sd*: {i,j} $\cap$ *set is* $\subseteq$ {} $\Longrightarrow$ *sc is* = *sd is* **for** *is*
unfolding *sc-def sd-def* **by** (*rule arg-cong*[*of - - sumlist*], *rule map-cong*, *auto simp*: *d-def*,
rename-tac k, *case-tac* $i = k$, *auto*)
have f = *?a* + (*?b* + (*?c* + (*?d* + *?e*)))
unfolding f *map-append sc-def* **using** *fs* $*$
by ((*subst sumlist-append*, *force*, *force*)+, *simp*)
also have ... = *?a* + ((*?b* + *?d*) + (*?c* + *?e*)) **using** *ae* **by** *auto*
also have ... = *?A* + ((*?b* + *?d*) + (*?C* + *?E*))
using $*$ **by** (*auto simp*: *sc-sd*)
also have *?b* + *?d* = *?B* + *?D* **unfolding** *sd-def sc-def d-def sumlist-def*
by (*rule eq-vecI*, *insert* $*$ *fsd*, *auto simp*: *algebra-simps*)
finally have f = *?A* + (*?B* + (*?C* + (*?D* + *?E*))) **using** *AE* **by** *auto*
also have ... = *sumlist* (*map* ($\lambda i.$ *of-int* (d i) $\cdot_v$ *?gs* ! i) *?len*)
unfolding f *map-append sd-def* **using** *fs* $*$
by ((*subst sumlist-append*, *force*, *force*)+, *simp*)
also have ... = *sumlist* (*map* ($\lambda i.$ *of-int* (d i) $\cdot_v$ *?gs* ! i) [*0* ..< *length ?gs*])
unfolding *len*[*symmetric*] **by** *simp*
finally have f = *sumlist* (*map* ($\lambda i.$ *of-int* (d i) $\cdot_v$ *?gs* ! i) [*0* ..< *length ?gs*]) **.**
from *in-latticeI*[*OF this*] **have** $f \in$ *lattice-of ?gs* **.**
}
hence *lattice-of fs* $\subseteq$ *lattice-of ?gs* **by** *blast*
} **note** *main* = *this*
{
fix i j **and** *fs* :: *'a vec list*
assume $*$: $i < j$ $j <$ *length fs* **and** *fs*: *set fs* $\subseteq$ *carrier-vec n*

let *?gs* = *fs*[i := *fs* ! i + *of-int* l $\cdot_v$ *fs* ! j]
define *gs* **where** *gs* = *?gs*
from *main*[*OF* ∗ *fs*, *of* l, *folded gs-def*]
have *one*: *lattice-of fs* ⊆ *lattice-of gs* **.**
have ∗: i < j j < *length gs set gs* ⊆ *carrier-vec* n **using** ∗ *fs* **unfolding** *gs-def set-conv-nth*
by (*auto*, *rename-tac* k, *case-tac* k = i, (*force intro*!: *add-carrier-vec*)+)
from *fs* **have** *fs*: ⋀ i. i < *length fs* ⟹ *fs* ! i ∈ *carrier-vec* n **unfolding** *set-conv-nth* **by** *auto*
from *fs* **have** *fsd*: ⋀ i. i < *length fs* ⟹ *dim-vec* (*fs* ! i) = n **by** *auto*
from *fsd*[*of* i] *fsd*[*of* j] ∗ **have** *fsd*: *dim-vec* (*fs* ! i) = n *dim-vec* (*fs* ! j) = n **by** (*auto simp*: *gs-def*)
from *main*[*OF* ∗, *of* −l]
have *lattice-of gs* ⊆ *lattice-of* (*gs*[i := *gs* ! i + *of-int* (− l) $\cdot_v$ *gs* ! j]) **.**
also have *gs*[i := *gs* ! i + *of-int* (− l) $\cdot_v$ *gs* ! j] = *fs* **unfolding** *gs-def*
by (*rule nth-equalityI*, *auto*, *insert* ∗ *fsd*, *rename-tac* k, *case-tac* k = i, *auto*)
ultimately have *lattice-of fs* = *lattice-of ?gs* **using** *one* **unfolding** *gs-def* **by** *auto*
} **note** *main* = *this*
show *?thesis*
proof (*cases* i < j)
case *True*
from *main*[*OF this ij*(*2*) *fs*] **show** *?thesis* **unfolding** *gs* **by** *simp*
next
case *False*
with *ij* **have** *ji*: j < i **by** *auto*
define *hs* **where** *hs* = *fs*[i := *fs* ! j, j := *fs* ! i]
define *ks* **where** *ks* = *hs*[j := *hs* ! j + *of-int* l $\cdot_v$ *hs* ! i]
from *ij fs* **have** *ij'*: i < *length hs set hs* ⊆ *carrier-vec* n **unfolding** *hs-def* **by** *auto*
hence *ij''*: *set ks* ⊆ *carrier-vec* n i < *length ks* j < *length ks* i ≠ j
using *ji* **unfolding** *ks-def set-conv-nth* **by** (*auto*, *rename-tac* k, *case-tac* k = i,
force, *case-tac* k = j, (*force intro*!: *add-carrier-vec*)+)
from *lattice-of-swap*[*OF fs ij refl*]
have *lattice-of fs* = *lattice-of hs* **unfolding** *hs-def* **by** *auto*
also have ... = *lattice-of ks*
using *main*[*OF ji ij'*] **unfolding** *ks-def* **.**
also have ... = *lattice-of* (*ks*[i := *ks* ! j, j := *ks* ! i])
by (*rule sym*, *rule lattice-of-swap*[*OF ij'' refl*])
also have *ks*[i := *ks* ! j, j := *ks* ! i] = *gs* **unfolding** *gs ks-def hs-def*
by (*rule nth-equalityI*, *insert ij*, *auto*,
rename-tac k, *case-tac* k = i, *force*, *case-tac* k = j, *auto*)
finally show *?thesis* **by** *simp*
qed
qed

definition *orthogonal-complement* W = {x. x ∈ *carrier-vec* n ∧ (∀ y ∈ W. x • y = *0*)}

lemma *orthogonal-complement-subset*:
  assumes $A \subseteq B$
  shows *orthogonal-complement* $B \subseteq$ *orthogonal-complement* A
unfolding *orthogonal-complement-def* **using** *assms* **by** *auto*

end

context *vec-space*
begin
sublocale *vec-module* - *n* **.**

lemma *in-orthogonal-complement-span*[*simp*]:
  assumes [*intro*]:$S \subseteq$ *carrier-vec n*
  shows *orthogonal-complement* (*span S*) = *orthogonal-complement S*
proof
  show *orthogonal-complement* (*span S*) $\subseteq$ *orthogonal-complement S*
    by(*fact orthogonal-complement-subset*[*OF in-own-span*[*OF assms*]])
  {**fix** *x* :: *'a vec*
    fix *a* **fix** *A* :: *'a vec set*
    assume *x* [*intro*]:$x \in$ *carrier-vec n* **and** *f*: *finite A* **and** *S*:$A \subseteq S$
    assume *i0*:$\forall y \in S.\ x \cdot y = 0$
    have $x \cdot$ *lincomb a A* = *0*
        unfolding *comm-scalar-prod*[*OF x lincomb-closed*[*OF subset-trans*[*OF S assms*]]]
    proof(*insert S*,*atomize*(*full*),*rule finite-induct*[*OF f*],*goal-cases*)
      case *1* **thus** *?case* **using** *assms x* **by** *force*
    next
      case (*2 f F*)
      { **assume** *i*:*insert f F* $\subseteq S$
        hence *F*:$F \subseteq S$ **and** *f*: $f \in S$ **by** *auto*
        from *F f assms*
        have [*intro*]:$F \subseteq$ *carrier-vec n*
          and *fc*[*intro*]:$f \in$ *carrier-vec n*
          and [*intro*]:$x \in F \Longrightarrow x \in$ *carrier-vec n* **for** *x* **by** *auto*
        have *laf*:*lincomb a F* $\cdot$ *x* = *0* **using** *F 2* **by** *auto*
        have [*simp*]:($\sum u \in F.\ (a\ u \cdot_v u) \cdot x$) = *0*
        by(*insert laf*[*unfolded lincomb-def*],*atomize*(*full*),*subst finsum-scalar-prod-sum*) *auto*
        from *f i0* **have** [*simp*]:$f \cdot x = 0$ **by** (*subst comm-scalar-prod*) *auto*
        from *lincomb-closed*[*OF subset-trans*[*OF i assms*]]
        have *lincomb a* (*insert f F*) $\cdot$ *x* = *0* **unfolding** *lincomb-def*
          apply(*subst finsum-scalar-prod-sum*,*force*,*force*)
          using *2*(*1*,*2*) *smult-scalar-prod-distrib*[*OF fc x*] **by** *auto*
      } **thus** *?case* **by** *auto*
      qed
  }
  thus *orthogonal-complement S* $\subseteq$ *orthogonal-complement* (*span S*)

unfolding *orthogonal-complement-def span-def* **by** *auto*
qed

lemma *lincomb-list-add-vec-2*: **assumes** *us*: $set\ us \subseteq carrier\text{-}vec\ n$
and *x*: $x = lincomb\text{-}list\ lc\ (us\ [i := us\ !\ i + c \cdot_v us\ !\ j])$
and *i*: $j < length\ us\ i < length\ us\ i \neq j$
shows $x = lincomb\text{-}list\ (lc\ (j := lc\ j + lc\ i * c))\ us$ (**is** - = *?x*)
proof −
let $?xx = lc\ j + lc\ i * c$
let $?i = us\ !\ i$
let $?j = us\ !\ j$
let $?v = ?i + c \cdot_v ?j$
let $?ws = us\ [i := us\ !\ i + c \cdot_v us\ !\ j]$
from *us* **have** *usk*: $k < length\ us \Longrightarrow us\ !\ k \in carrier\text{-}vec\ n$ **for** *k* **by** *auto*
from *usk i* **have** *ij*: $?i \in carrier\text{-}vec\ n\ ?j \in carrier\text{-}vec\ n$ **by** *auto*
hence *v*: $c \cdot_v ?j \in carrier\text{-}vec\ n\ ?v \in carrier\text{-}vec\ n$ **by** *auto*
with *us* **have** *ws*: $set\ ?ws \subseteq carrier\text{-}vec\ n$ **unfolding** *set-conv-nth* **using** *i*
by (*auto*, *rename-tac k*, *case-tac* $k = i$, *auto*)
from *us* **have** *us'*: $\forall w \in set\ us.\ dim\text{-}vec\ w = n$ **by** *auto*
from *ws* **have** *ws'*: $\forall w \in set\ ?ws.\ dim\text{-}vec\ w = n$ **by** *auto*
have *mset*: $mset\text{-}set\ \{0..<length\ us\} = \{\#i\#\} + \{\#j\#\} + (mset\text{-}set\ (\{0..<length\ us\} - \{i,j\}))$
by (*rule multiset-eqI*, *insert i*, *auto*, *rename-tac x*, *case-tac* $x \in \{0\ ..<\ length\ us\}$, *auto*)
define *M2* **where** $M2 = M.summset$
$\{\#lc\ ia \cdot_v ?ws\ !\ ia.\ ia \in\#\ mset\text{-}set\ (\{0..<length\ us\} - \{i, j\})\#\}$
define *M1* **where** $M1 = M.summset\ \{\#(if\ i = j\ then\ ?xx\ else\ lc\ i) \cdot_v us\ !\ i.\ i \in\#\ mset\text{-}set\ (\{0..<length\ us\} - \{i, j\})\#\}$
have *M1*: $M1 \in carrier\text{-}vec\ n$ **unfolding** *M1-def* **using** *usk* **by** *fastforce*
have *M2*: $M1 = M2$ **unfolding** *M2-def M1-def*
by (*rule arg-cong*[*of* - - *M.summset*], *rule multiset.map-cong0*, *insert i usk*, *auto*)
have *x1*: $x = lc\ j \cdot_v ?j + (lc\ i \cdot_v ?i + lc\ i \cdot_v (c \cdot_v ?j) + M1)$
unfolding *x lincomb-list-def M2 M2-def*
apply (*subst sumlist-as-summset*, (*insert us ws i v ij*, *auto simp*: *set-conv-nth*)[*1*], *insert i ij v us ws usk*,
simp add: *mset smult-add-distrib-vec*[*OF ij*(*1*) *v*(*1*)])
by (*subst M.summset-add-mset*, *auto*)+
have *x2*: $?x = ?xx \cdot_v ?j + (lc\ i \cdot_v ?i + M1)$
unfolding *x lincomb-list-def M1-def*
apply (*subst sumlist-as-summset*, (*insert us ws i v ij*, *auto simp*: *set-conv-nth*)[*1*], *insert i ij v us ws usk*,
simp add: *mset smult-add-distrib-vec*[*OF ij*(*1*) *v*(*1*)])
by (*subst M.summset-add-mset*, *auto*)+
show *?thesis* **unfolding** *x1 x2* **using** *M1 ij*
by (*intro eq-vecI*, *auto simp*: *field-simps*)
qed

lemma *lincomb-list-add-vec-1*: **assumes** *us*: $set\ us \subseteq carrier\text{-}vec\ n$

and *x*: x = *lincomb-list lc us*
and *i*: $j <$ *length us* $i <$ *length us* $i \neq j$
shows x = *lincomb-list* (*lc* (j := *lc* j − *lc* $i * c$)) (*us* [i := *us* ! i + $c \cdot_v$ *us* ! j])
(**is** - = *?x*)
proof −
let *?i* = *us* ! i
let *?j* = *us* ! j
let *?v* = *?i* + $c \cdot_v$ *?j*
let *?ws* = *us* [i := *us* ! i + $c \cdot_v$ *us* ! j]
from *us* **have** *usk*: $k <$ *length us* $\Longrightarrow$ *us* ! $k \in$ *carrier-vec n* **for** k **by** *auto*
from *usk i* **have** *ij*: *?i* $\in$ *carrier-vec n* *?j* $\in$ *carrier-vec n* **by** *auto*
hence *v*: $c \cdot_v$ *?j* $\in$ *carrier-vec n* *?v* $\in$ *carrier-vec n* **by** *auto*
with *us* **have** *ws*: *set ?ws* $\subseteq$ *carrier-vec n* **unfolding** *set-conv-nth* **using** *i*
by (*auto*, *rename-tac k*, *case-tac k* = *i*, *auto*)
from *us* **have** *us'*: $\forall$ *w*$\in$*set us*. *dim-vec w* = *n* **by** *auto*
from *ws* **have** *ws'*: $\forall$ *w*$\in$*set ?ws*. *dim-vec w* = *n* **by** *auto*
have *mset*: *mset-set* {*0*..<*length us*} = {#*i*#} + {#*j*#} + (*mset-set* ({*0*..<*length us*} − {*i*,*j*}))
by (*rule multiset-eqI*, *insert i*, *auto*, *rename-tac x*, *case-tac x* $\in$ {*0* ..< *length us*}, *auto*)
define *M2* **where** *M2* = *M.summset*
{#(*if ia* = *j then lc j* − *lc i* $*$ *c else lc ia*) $\cdot_v$ *?ws* ! *ia*
. *ia* $\in$# *mset-set* ({*0*..<*length us*} − {*i*, *j*})#}
define *M1* **where** *M1* = *M.summset* {#*lc i* $\cdot_v$ *us* ! *i*. *i* $\in$# *mset-set* ({*0*..<*length us*} − {*i*, *j*})#}
have *M1*: *M1* $\in$ *carrier-vec n* **unfolding** *M1-def* **using** *usk* **by** *fastforce*
have *M2*: *M1* = *M2* **unfolding** *M2-def M1-def*
by (*rule arg-cong*[*of - - M.summset*], *rule multiset.map-cong0*, *insert i usk*, *auto*)
have *x1*: x = *lc j* $\cdot_v$ *?j* + (*lc i* $\cdot_v$ *?i* + *M1*)
unfolding *x lincomb-list-def M1-def*
apply (*subst sumlist-as-summset*, (*insert us ws i v ij*, *auto simp*: *set-conv-nth*)[*1*], *insert i ij v us ws usk*,
simp add: *mset smult-add-distrib-vec*[*OF ij*(*1*) *v*(*1*)])
by (*subst M.summset-add-mset*, *auto*)+
have *x2*: *?x* = (*lc j* − *lc i* $*$ *c*) $\cdot_v$ *?j* + (*lc i* $\cdot_v$ *?i* + *lc i* $\cdot_v$ (*c* $\cdot_v$ *?j*) + *M1*)
unfolding *x lincomb-list-def M2 M2-def*
apply (*subst sumlist-as-summset*, (*insert us ws i v ij*, *auto simp*: *set-conv-nth*)[*1*], *insert i ij v us ws usk*,
simp add: *mset smult-add-distrib-vec*[*OF ij*(*1*) *v*(*1*)])
by (*subst M.summset-add-mset*, *auto*)+
show *?thesis* **unfolding** *x1 x2* **using** *M1 ij*
by (*intro eq-vecI*, *auto simp*: *field-simps*)
qed

lemma *add-vec-span*: **assumes** *us*: *set us* $\subseteq$ *carrier-vec n*
and *i*: $j <$ *length us* $i <$ *length us* $i \neq j$
shows *span* (*set us*) = *span* (*set* (*us* [i := *us* ! i + $c \cdot_v$ *us* ! j])) (**is** - = *span* (*set* *?ws*))

```
proof -
  let ?i = us ! i
  let ?j = us ! j
  let ?v = ?i + c ·v ?j
  from us i have ij: ?i ∈ carrier-vec n ?j ∈ carrier-vec n by auto
  hence v: ?v ∈ carrier-vec n by auto
  with us have ws: set ?ws ⊆ carrier-vec n unfolding set-conv-nth using i
    by (auto, rename-tac k, case-tac k = i, auto)
  have span (set us) = span-list us unfolding span-list-as-span[OF us] ..
  also have ... = span-list ?ws
  proof -
    {
      fix x
      assume x ∈ span-list us
      then obtain lc where x = lincomb-list lc us by (metis in-span-listE)
      from lincomb-list-add-vec-1[OF us this i, of c]
      have x ∈ span-list ?ws unfolding span-list-def by auto
    }
    moreover
    {
      fix x
      assume x ∈ span-list ?ws
      then obtain lc where x = lincomb-list lc ?ws by (metis in-span-listE)
      from lincomb-list-add-vec-2[OF us this i]
      have x ∈ span-list us unfolding span-list-def by auto
    }
    ultimately show ?thesis by blast
  qed
  also have ... = span (set ?ws) unfolding span-list-as-span[OF ws] ..
  finally show ?thesis .
qed

lemma prod-in-span[intro!]:
  assumes b ∈ carrier-vec n S ⊆ carrier-vec n a = 0 ∨ b ∈ span S
  shows a ·v b ∈ span S
proof(cases a = 0)
  case True
  then show ?thesis by (auto simp:lmult-0[OF assms(1)] span-zero)
next
  case False with assms have b ∈ span S by auto
  from this[THEN in-spanE]
  obtain aa A where a[intro!]: b = lincomb aa A finite A A ⊆ S by auto
  hence [intro!]:(λv. aa v ·v v) ∈ A → carrier-vec n using assms by auto
  show ?thesis proof
    show a ·v b = lincomb (λ v. a * aa v) A using a(1) unfolding lincomb-def
smult-smult-assoc[symmetric]
      by(subst finsum-smult[symmetric]) force+
  qed auto
qed
```

lemma *det-nonzero-congruence*:
  assumes *eq*:$A * M = B * M$ **and** *det*:*det* $(M::'a\ mat) \neq 0$
  and *M*: $M \in$ *carrier-mat n n* **and** *carr*:$A \in$ *carrier-mat n n* $B \in$ *carrier-mat n n*
  shows $A = B$
proof −
  have 1_m $n \in$ *carrier-mat n n* **by** *auto*
  from *det-non-zero-imp-unit*[*OF M det*] *gauss-jordan-check-invertable*[*OF M this*]
  have *gj-fst*:(*fst* (*gauss-jordan M* (1_m *n*)) = 1_m *n*) **by** *metis*
  define *Mi* **where** *Mi* = *snd* (*gauss-jordan M* (1_m *n*))
  with *gj-fst* **have** *gj*:*gauss-jordan M* (1_m *n*) = (1_m *n*, *Mi*)
    unfolding *fst-def snd-def* **by** (*auto split*:*prod.split*)
  from *gauss-jordan-compute-inverse*(*1*,*3*)[*OF M gj*]
  have *Mi*: $Mi \in$ *carrier-mat n n* **and** *is1*:$M * Mi = 1_m$ *n* **by** *metis*+
  from *arg-cong*[*OF eq*, *of* λ *M*. $M * Mi$]
  show $A = B$ **unfolding** *carr*[*THEN assoc-mult-mat*[*OF - M Mi*]] *is1 carr*[*THEN right-mult-one-mat*]**.**
qed

end

context *cof-vec-space*
begin

definition *lin-indpt-list* :: *'a vec list* $\Rightarrow$ *bool* **where**
  lin-indpt-list fs = (*set fs* $\subseteq$ *carrier-vec n* $\wedge$ *distinct fs* $\wedge$ *lin-indpt* (*set fs*))

definition *basis-list* :: *'a vec list* $\Rightarrow$ *bool* **where**
  basis-list fs = (*set fs* $\subseteq$ *carrier-vec n* $\wedge$ *length fs* = *n* $\wedge$ *carrier-vec n* $\subseteq$ *span* (*set fs*))

lemma *upper-triangular-imp-lin-indpt-list*:
  assumes *A*: $A \in$ *carrier-mat n n*
    and *tri*: *upper-triangular A*
    and *diag*: $0 \notin$ *set* (*diag-mat A*)
  shows *lin-indpt-list* (*rows A*)
  using *upper-triangular-imp-distinct*[*OF assms*]
  using *upper-triangular-imp-lin-indpt-rows*[*OF assms*] *A*
  unfolding *lin-indpt-list-def* **by** (*auto simp*: *rows-def*)

lemma *basis-list-basis*: **assumes** *basis-list fs*
  shows *distinct fs lin-indpt* (*set fs*) *basis* (*set fs*)
proof −
  from *assms*[*unfolded basis-list-def*]
  have *len*: *length fs* = *n* **and** *C*: *set fs* $\subseteq$ *carrier-vec n*
    and *span*: *carrier-vec n* $\subseteq$ *span* (*set fs*) **by** *auto*
  show *b*: *basis* (*set fs*)
  proof (*rule dim-gen-is-basis*[*OF finite-set C*])

show *card* (*set fs*) $\leq$ *dim* **unfolding** *dim-is-n* **unfolding** *len*[*symmetric*] **by** (*rule card-length*)
show *span* (*set fs*) = *carrier-vec n* **using** *span C* **by** *auto*
qed
thus *lin-indpt* (*set fs*) **unfolding** *basis-def* **by** *auto*
show *distinct fs*
proof (*rule ccontr*)
assume ¬ *distinct fs*
hence *card* (*set fs*) < *length fs* **using** *antisym-conv1 card-distinct card-length* **by** *auto*
also have ... = *dim* **unfolding** *len dim-is-n* **..**
finally have *card* (*set fs*) < *dim* **by** *auto*
also have ... $\leq$ *card* (*set fs*) **using** *span finite-set*[*of fs*]
using *b basis-def gen-ge-dim* **by** *auto*
finally show *False* **by** *simp*
qed
qed

lemma *basis-list-imp-lin-indpt-list*: **assumes** *basis-list fs* **shows** *lin-indpt-list fs*
using *basis-list-basis*[*OF assms*] *assms* **unfolding** *lin-indpt-list-def basis-list-def* **by** *auto*

lemma *mat-of-rows-mult-as-finsum*:
assumes $v \in$ *carrier-vec* (*length lst*) $\bigwedge$ *i*. *i* < *length lst* $\Longrightarrow$ *lst* ! *i* $\in$ *carrier-vec n*
defines *f l* $\equiv$ *sum* (λ *i*. *if l* = *lst* ! *i then v* \$ *i else 0*) {*0..*<*length lst*}
shows *mat-of-cols-mult-as-finsum*:*mat-of-cols n lst* $*_v$ *v* = *lincomb f* (*set lst*)
proof −
from *assms* **have** ∀ *i* < *length lst*. *lst* ! *i* $\in$ *carrier-vec n* **by** *blast*
note *an* = *all-nth-imp-all-set*[*OF this*] **hence** *slc*:*set lst* $\subseteq$ *carrier-vec n* **by** *auto*
hence *dn* [*simp*]:$\bigwedge$ *x*. *x* $\in$ *set lst* $\Longrightarrow$ *dim-vec x* = *n* **by** *auto*
have *dl* [*simp*]:*dim-vec* (*lincomb f* (*set lst*)) = *n* **using** *an* **by** (*intro lincomb-dim*,*auto*)
show *?thesis* **proof**
show *dim-vec* (*mat-of-cols n lst* $*_v$ *v*) = *dim-vec* (*lincomb f* (*set lst*)) **using** *assms*(*1*,*2*) **by** *auto*
fix *i* **assume** *i*:*i* < *dim-vec* (*lincomb f* (*set lst*)) **hence** *i'*:*i* < *n* **by** *auto*
with *an* **have** *fcarr*:(λ*v*. *f v* $\cdot_v$ *v*) $\in$ *set lst* $\rightarrow$ *carrier-vec n* **by** *auto*
from *i'* **have** (*mat-of-cols n lst* $*_v$ *v*) \$ *i* = *row* (*mat-of-cols n lst*) *i* $\cdot$ *v* **by** *auto*
also have ... = ($\sum$ *ia* = *0..*<*dim-vec v*. *lst* ! *ia* \$ *i* $*$ *v* \$ *ia*)
unfolding *mat-of-cols-def row-def scalar-prod-def*
apply(*rule sum.cong*[*OF refl*]) **using** *i an assms*(*1*) **by** *auto*
also have ... = ($\sum$ *ia* = *0..*<*length lst*. *lst* ! *ia* \$ *i* $*$ *v* \$ *ia*) **using** *assms*(*1*) **by** *auto*
also have ... = ($\sum$ *x*$\in$*set lst*. *f x* $*$ *x* \$ *i*)
unfolding *f-def sum-distrib-right* **apply** (*subst sum.commute*)
apply(*rule sum.cong*[*OF refl*])
unfolding *if-distrib if-distrib-ap mult-zero-left sum.delta*[*OF finite-set*] **by** *auto*

also have ... $= (\sum x{\in}set\ lst.\ (f\ x \cdot_v x)\ \$\ i)$
apply(*rule sum.cong*[*OF refl*],*subst index-smult-vec*) **using** *i slc* **by** *auto*
also have ... $= (\bigoplus_V v{\in}set\ lst.\ f\ v \cdot_v v)\ \$\ i$
unfolding *finsum-index*[*OF i′ fcarr slc*] **by** *auto*
finally show (*mat-of-cols n lst* $*_v$ *v*) \$ *i* = *lincomb f* (*set lst*) \$ *i*
by (*auto simp*:*lincomb-def*)
qed
qed

lemma *basis-det-nonzero*:
assumes *db*:*basis* (*set G*) **and** *len*:*length G = n*
shows *det* (*mat-of-rows n G*) $\neq$ *0*
proof −
have *M-car1*:*mat-of-rows n G* $\in$ *carrier-mat n n* **using** *assms* **by** *auto*
hence *M-car*:$(mat\text{-}of\text{-}rows\ n\ G)^T \in carrier\text{-}mat\ n\ n$ **by** *auto*
have *li*:*lin-indpt* (*set G*)
and *inc-2*:*set G* $\subseteq$ *carrier-vec n*
and *issp*:*carrier-vec n = span* (*set G*)
and *RG-in-carr*:$\bigwedge i.\ i < length\ G \Longrightarrow G\ !\ i \in carrier\text{-}vec\ n$
using *assms*[*unfolded basis-def*] **by** *auto*
hence *basis-list G* **unfolding** *basis-list-def* **using** *len* **by** *auto*
from *basis-list-basis*[*OF this*] **have** *di*:*distinct G* **by** *auto*
have *det* $((mat\text{-}of\text{-}rows\ n\ G)^T) \neq 0$ **unfolding** *det-0-iff-vec-prod-zero*[*OF M-car*]

proof
assume $\exists v.\ v \in carrier\text{-}vec\ n \wedge v \neq 0_v\ n \wedge (mat\text{-}of\text{-}rows\ n\ G)^T *_v v = 0_v\ n$
then obtain *v* **where** *v*:$v \in span\ (set\ G)$
$v \neq 0_v\ n\ (mat\text{-}of\text{-}rows\ n\ G)^T *_v v = 0_v\ n$
unfolding *issp* **by** *blast*
from *finite-in-span*[*OF finite-set inc-2 v*(*1*)] **obtain** *a*
where *aA*: *v = lincomb a* (*set G*) **by** *blast*
from *v*(*1*,*2*)[*folded issp*] **obtain** *i* **where** *i*:*v* \$ *i* $\neq$ *0 i* $<$ *n* **by** *fastforce*
hence *inG*:*G* ! *i* $\in$ *set G* **using** *len* **by** *auto*
have *di2*: *distinct* [*0..*$<$*length G*] **by** *auto*
define *f* **where** $f = (\lambda l.\ \sum i \in set\ [0..{<}length\ G].\ if\ l = G\ !\ i\ then\ v\ \$\ i\ else\ 0)$
hence *f′*:$f\ (G\ !\ i) = (\sum ia{\leftarrow}[0..{<}n].\ if\ G\ !\ ia = G\ !\ i\ then\ v\ \$\ ia\ else\ 0)$
unfolding *f-def sum.distinct-set-conv-list*[*OF di2*] **unfolding** *len* **by** *metis*
from *v* **have** *mat-of-cols n G* $*_v$ *v* = 0_v *n*
unfolding *transpose-mat-of-rows* **by** *auto*
with *mat-of-cols-mult-as-finsum*[*OF v*(*1*)[*folded issp len*] *RG-in-carr*]
have *f*:*lincomb f* (*set G*) = 0_v *n* **unfolding** *len f-def* **by** *auto*
note [*simp*] = *list-trisect*[*OF i*(*2*)[*folded len*],*unfolded len*]
note *x* = *i*(*2*)[*folded len*]
have [*simp*]:$(\sum x{\leftarrow}[0..{<}i].\ if\ G\ !\ x = G\ !\ i\ then\ v\ \$\ x\ else\ 0) = 0$
by (*rule sum-list-0*,*auto simp*: *nth-eq-iff-index-eq*[*OF di less-trans*[*OF - x*] *x*])
have [*simp*]:$(\sum x{\leftarrow}[Suc\ i..{<}n].\ if\ G\ !\ x = G\ !\ i\ then\ v\ \$\ x\ else\ 0) = 0$
apply (*rule sum-list-0*) **using** *nth-eq-iff-index-eq*[*OF di - x*] *len* **by** *auto*
from *i*(*1*) **have** *f* (*G* ! *i*) $\neq$ *0* **unfolding** *f′* **by** *auto*

```
  from lin-dep-crit[OF finite-set subset-refl TrueI inG this f]
    have lin-dep (set G).
    thus False using li by auto
  qed
  thus det0:det (mat-of-rows n G) ≠ 0 by (unfold det-transpose[OF M-car1])
qed

lemma lin-indpt-list-add-vec: assumes
      i: j < length us i < length us i ≠ j
   and indep: lin-indpt-list  us
shows lin-indpt-list (us [i := us ! i + c ·v us ! j]) (is lin-indpt-list ?V)
proof −
  from indep[unfolded lin-indpt-list-def] have us: set us ⊆ carrier-vec n
    and dist: distinct us and indep: lin-indpt (set us) by auto
  let ?E = set us − {us ! i}
  let ?us = insert (us ! i) ?E
  let ?v = us ! i + c ·v us ! j
  from us i have usi: us ! i ∈ carrier-vec n us ! i ∉ ?E us ! i ∈ set us
    and usj: us ! j ∈ carrier-vec n by auto
  from usi usj have v: ?v ∈ carrier-vec n by auto
  have fin: finite ?E by auto
  have id: set us = insert (us ! i) (set us − {us ! i}) using i(2) by auto
  from dist i have diff': us ! i ≠ us ! j unfolding distinct-conv-nth by auto
  from subset-li-is-li[OF indep] have indepE: lin-indpt ?E by auto
  have Vid: set ?V = insert ?v ?E using set-update-distinct[OF dist i(2)] by auto
  have E: ?E ⊆ carrier-vec n using us by auto
  have V: set ?V ⊆ carrier-vec n using us v unfolding Vid by auto
  from dist i have diff: us ! i ≠ us ! j unfolding distinct-conv-nth by auto
  have vspan: ?v ∉ span ?E
  proof
    assume mem: ?v ∈ span ?E
    from diff i have us ! j ∈ ?E by auto
    hence us ! j ∈ span ?E using E by (metis span-mem)
    hence − c ·v us ! j ∈ span ?E using smult-in-span[OF E] by auto
    from span-add1[OF E mem this] have ?v + (− c ·v us ! j) ∈ span ?E .
    also have ?v + (− c ·v us ! j) = us ! i using usi usj by auto
    finally have mem: us ! i ∈ span ?E .
     from in-spanE[OF this] obtain a A where lc: us ! i = lincomb a A and A:
finite A
      A ⊆ set us − {us ! i}
      by auto
    let ?a = a (us ! i := −1) let ?A = insert (us ! i) A
    from A have fin: finite ?A by auto
    have lc: lincomb ?a A = us ! i unfolding lc
      by (rule lincomb-cong, insert A us lc, auto)
    have lincomb ?a ?A = 0v n
      by (subst lincomb-insert2[OF A(1)], insert A us lc usi diff, auto)
    from not-lindepD[OF indep - - - this] A usi
    show False by auto
```

```
  qed
  hence vmem: ?v ∉ ?E using span-mem[OF E, of ?v] by auto
  from lin-dep-iff-in-span[OF E indepE v this] vspan
  have indep1: lin-indpt (set ?V) unfolding Vid by auto
  from vmem dist have distinct ?V by (metis distinct-list-update)
  with indep1 V show ?thesis unfolding lin-indpt-list-def by auto
qed

lemma scalar-prod-lincomb-orthogonal: assumes ortho: orthogonal gs and gs: set
gs ⊆ carrier-vec n
  shows k ≤ length gs ⟹ sumlist (map (λ i. g i ·v gs ! i) [0 ..< k]) · sumlist
(map (λ i. g i ·v gs ! i) [0 ..< k])
  = sum-list (map (λ i. g i * g i * (gs ! i · gs ! i)) [0 ..< k])
proof (induct k)
  case (Suc k)
  note ortho = orthogonalD[OF ortho]
  let ?m = length gs
  from gs Suc(2) have gsi[simp]: ⋀ i. i ≤ k ⟹ gs ! i ∈ carrier-vec n by auto
  from Suc have kn: k ≤ ?m and k: k < ?m by auto
  let ?v1 = sumlist (map (λi. g i ·v gs ! i) [0..<k])
  let ?v2 = (g k ·v gs ! k)
  from Suc have id: [0 ..< Suc k] = [0 ..< k] @ [k] by simp
  have id: sumlist (map (λi. g i ·v gs ! i) [0..<Suc k]) = ?v1 + ?v2
    unfolding id map-append
    by (subst sumlist-append, insert Suc(2), auto)
  have v1: ?v1 ∈ carrier-vec n by (rule sumlist-carrier, insert Suc(2), auto)
  have v2: ?v2 ∈ carrier-vec n by (insert Suc(2), auto)
  have gsk: gs ! k ∈ carrier-vec n by simp
  have v12: ?v1 + ?v2 ∈ carrier-vec n using v1 v2 by auto
  have 0: i < k ⟹ (g i ·v gs ! i) · (g k ·v gs ! k) = 0 for i
    by (subst scalar-prod-smult-distrib[OF - gsk], (insert k, auto)[1],
    subst smult-scalar-prod-distrib[OF - gsk], (insert k, auto)[1], insert ortho[of i k]
k, auto)
  have 0: ?v1 · ?v2 = 0
     by (subst scalar-prod-left-sum-distrib[OF - v2], (insert Suc(2), auto)[1], rule
sum-list-neutral,
        insert 0, auto)
  show ?case unfolding id
    unfolding scalar-prod-add-distrib[OF v12 v1 v2]
      add-scalar-prod-distrib[OF v1 v2 v1]
      add-scalar-prod-distrib[OF v1 v2 v2]
      scalar-prod-smult-distrib[OF v2 gsk]
      smult-scalar-prod-distrib[OF gsk gsk]
    unfolding Suc(1)[OF kn]
    by (simp add: 0 comm-scalar-prod[OF v2 v1])
qed auto
end
```

```
locale gram-schmidt = cof-vec-space n f-ty
  for n :: nat and f-ty :: 'a :: trivial-conjugatable-ordered-field itself
begin

definition Gramian-matrix where
  Gramian-matrix G k = (let M = mat k n (λ (i,j). (G ! i) $ j) in M * M^T)

lemma Gramian-matrix-alt-def: k ≤ length G ⟹
  Gramian-matrix G k = (let M = mat-of-rows n (take k G) in M * M^T)
  unfolding Gramian-matrix-def Let-def
 by (rule arg-cong[of - - λ x. x * x^T], unfold mat-of-rows-def, intro eq-matI, auto)

definition Gramian-determinant where
  Gramian-determinant G k = det (Gramian-matrix G k)

lemma orthogonal-imp-lin-indpt-list:
  assumes ortho: orthogonal gs and gs: set gs ⊆ carrier-vec n
  shows lin-indpt-list gs
proof –
  from corthogonal-distinct[of gs] ortho have dist: distinct gs by simp
  show ?thesis unfolding lin-indpt-list-def
  proof (intro conjI gs dist finite-lin-indpt2 finite-set)
    fix lc
    assume 0: lincomb lc (set gs) = 0_v n (is ?lc = -)
    have lc: ?lc ∈ carrier-vec n by (rule lincomb-closed[OF gs])
    let ?m = length gs
    from 0 have 0 = ?lc · ?lc by simp
    also have ?lc = lincomb-list (λi. lc (gs ! i)) gs
      unfolding lincomb-as-lincomb-list-distinct[OF gs dist] ..
    also have ... = sumlist (map (λi. lc (gs ! i) ·_v gs ! i) [0..< ?m])
      unfolding lincomb-list-def by auto
    also have ... · ... = (∑ i←[0..<?m]. (lc (gs ! i) * lc (gs ! i)) * sq-norm (gs
! i)) (is - = sum-list ?sum)
      unfolding scalar-prod-lincomb-orthogonal[OF ortho gs le-refl]
      by (auto simp: sq-norm-vec-as-cscalar-prod power2-eq-square)
    finally have sum-0: sum-list ?sum = 0 ..
    have nonneg: ⋀ x. x ∈ set ?sum ⟹ x ≥ 0
      using zero-le-square[of lc (gs ! i) for i] sq-norm-vec-ge-0[of gs ! i for i] by
auto
    {
      fix x
      assume x: x ∈ set gs
      then obtain i where i: i < ?m and x: x = gs ! i unfolding set-conv-nth
        by auto
      hence lc x * lc x * sq-norm x ∈ set ?sum by auto
      with sum-list-nonneg-eq-0-iff[of ?sum, OF nonneg] sum-0
      have lc x = 0 ∨ sq-norm x = 0 by auto
      with orthogonalD[OF ortho, OF i i, folded x]
      have lc x = 0 by (auto simp: sq-norm-vec-as-cscalar-prod)
```

}
thus $\forall v{\in}set\ gs.\ lc\ v = 0$ **by** *auto*
qed
qed

lemma *projection-alt-def*:
assumes *carr*:$(W::'a\ vec\ set) \subseteq carrier\text{-}vec\ n\ x \in carrier\text{-}vec\ n$
and *alt1*:$y1 \in W\ x - y1 \in orthogonal\text{-}complement\ W$
and *alt2*:$y2 \in W\ x - y2 \in orthogonal\text{-}complement\ W$
shows $y1 = y2$
proof −
have *carr*:$y1 \in carrier\text{-}vec\ n\ y2 \in carrier\text{-}vec\ n\ x \in carrier\text{-}vec\ n - y1 \in carrier\text{-}vec\ n$
$0_v\ n \in carrier\text{-}vec\ n$
using *alt1 alt2 carr* **by** *auto*
hence $y1 - y2 \in carrier\text{-}vec\ n$ **by** *auto*
note *carr = this carr*
from *alt1* **have** $ya{\in}W \Longrightarrow (x - y1) \cdot ya = 0$ **for** *ya*
unfolding *orthogonal-complement-def* **by** *blast*
hence $(x - y1) \cdot y2 = 0\ (x - y1) \cdot y1 = 0$ **using** *alt2 alt1* **by** *auto*
hence *eq1*:$y1 \cdot y2 = x \cdot y2\ y1 \cdot y1 = x \cdot y1$ **using** *carr minus-scalar-prod-distrib* **by** *force+*
from *this(1)* **have** *eq2*:$y2 \cdot y1 = x \cdot y2$ **using** *carr comm-scalar-prod* **by** *force*
from *alt2* **have** $ya{\in}W \Longrightarrow (x - y2) \cdot ya = 0$ **for** *ya*
unfolding *orthogonal-complement-def* **by** *blast*
hence $(x - y2) \cdot y1 = 0\ (x - y2) \cdot y2 = 0$ **using** *alt2 alt1* **by** *auto*
hence *eq3*:$y2 \cdot y2 = x \cdot y2\ y2 \cdot y1 = x \cdot y1$ **using** *carr minus-scalar-prod-distrib* **by** *force+*
with *eq2* **have** *eq4*:$x \cdot y1 = x \cdot y2$ **by** *auto*
have $\|(y1 - y2)\|^2 = 0$ **unfolding** *sq-norm-vec-as-cscalar-prod cscalar-prod-is-scalar-prod* **using** *carr*
apply(*subst minus-scalar-prod-distrib*) **apply** *force+*
apply(*subst (0 0) scalar-prod-minus-distrib*) **apply** *force+*
unfolding *eq1 eq2 eq3 eq4* **by** *auto*
with *sq-norm-vec-eq-0*[*of* $(y1 - y2)$] *carr* **have** $y1 - y2 = 0_v\ n$ **by** *fastforce*
hence $y1 - y2 + y2 = y2$ **using** *carr* **by** *fastforce*
also have $y1 - y2 + y2 = y1$ **using** *carr* **by** *auto*
finally show $y1 = y2$ **.**
qed

definition *weakly-reduced* :: $'a \Rightarrow nat \Rightarrow 'a\ vec\ list \Rightarrow bool$

where $weakly\text{-}reduced\ \alpha\ k\ gs = (\forall\ i.\ Suc\ i < k \longrightarrow$
$sq\text{-}norm\ (gs\ !\ i) \le \alpha * sq\text{-}norm\ (gs\ !\ (Suc\ i)))$

definition *strictly-reduced* :: $nat \Rightarrow 'a \Rightarrow 'a\ vec\ list \Rightarrow (nat \Rightarrow nat \Rightarrow 'a) \Rightarrow bool$

where $strictly\text{-}reduced\ N\ \alpha\ gs\ mu = (weakly\text{-}reduced\ \alpha\ N\ gs\ \wedge$
$(\forall\ i\ j.\ i < N \longrightarrow j < i \longrightarrow abs\ (mu\ i\ j) \le 1/2))$

definition
is-projection w S v = (*w* ∈ *carrier-vec n* ∧ *v* − *w* ∈ *span S* ∧ (∀ *u*. *u* ∈ *S* ⟶ *w* · *u* = *0*))

definition *projection* **where**
projection S fi ≡ (*SOME v*. *is-projection v S fi*)

context
fixes *fs* :: *'a vec list*
begin

fun *gso* **and** μ **where**
gso i = *fs* ! *i* + *sumlist* (*map* (λ *j*. − μ *i j* $\cdot_v$ *gso j*) [*0* ..< *i*])
| μ *i j* = (*if j* < *i then* (*fs* ! *i* · *gso j*)/ *sq-norm* (*gso j*) *else if i* = *j then 1 else 0*)

declare *gso.simps*[*simp del*]
declare μ*.simps*[*simp del*]

fun *adjuster-wit* :: *'a list* ⇒ *'a vec* ⇒ *'a vec list* ⇒ *'a list* × *'a vec*
where *adjuster-wit wits w* [] = (*wits*, 0_v *n*)
| *adjuster-wit wits w* (*u*#*us*) = (*let a* = (*w* · *u*)/ *sq-norm u in*
case adjuster-wit (*a* # *wits*) *w us of* (*wit*, *v*)
⇒ (*wit*, −*a* $\cdot_v$ *u* + *v*))

fun *sub2-wit* **where**
sub2-wit us [] = ([], [])
| *sub2-wit us* (*w* # *ws*) =
(*case adjuster-wit* (*1* # *replicate* (*n* − *length us* − *1*) *0*) *w us of* (*wit*,*aw*) ⇒ *let u* = *aw* + *w in*
case sub2-wit (*u* # *us*) *ws of* (*wits*, *vvs*) ⇒ (*wit* # *wits*, *u* # *vvs*))

definition *main* :: *'a vec list* ⇒ *'a list list* × *'a vec list* **where**
main us = *sub2-wit* [] *us*

lemma *gso-carrier'*[*intro*]:
assumes ⋀ *i*. *i* ≤ *j* ⟹ *fs* ! *i* ∈ *carrier-vec n*
shows *gso j* ∈ *carrier-vec n*
using *assms* **proof**(*induct j rule*:*nat-less-induct*[*rule-format*])
case (*1 j*)
then show *?case* **unfolding** *gso.simps*[*of j*] **by** (*auto intro*!:*sumlist-carrier add-carrier-vec*)
qed

lemma *adjuster-wit*: **assumes** *res*: *adjuster-wit wits w us* = (*wits'*,*a*)
and *w*: *w* ∈ *carrier-vec n*
and *us*: ⋀ *i*. *i* ≤ *j* ⟹ *fs* ! *i* ∈ *carrier-vec n*
and *us-gs*: *us* = *map gso* (*rev* [*0* ..< *j*])
and *wits*: *wits* = *map* (μ *i*) [*j* ..< *n*]
and *j*: *j* ≤ *n j* ≤ *i*

and *wi*: *w* = *fs* ! *i*
shows *adjuster n w us* = *a* ∧ *a* ∈ *carrier-vec n* ∧ *wits'* = *map* (μ *i*) [*0* ..< *n*] ∧
(*a* = *sumlist* (*map* (λ*j*. − μ *i j* $\cdot_v$ *gso j*) [*0*..<*j*]))
using *res us us-gs wits j*
proof (*induct us arbitrary*: *wits wits' a j*)
case (*Cons u us wits wits' a j*)
note *us-gs* = *Cons*(*4*)
note *wits* = *Cons*(*5*)
note *jn* = *Cons*(*6*−*7*)
from *us-gs* **obtain** *jj* **where** *j*: *j* = *Suc jj* **by** (*cases j*, *auto*)
from *jn j* **have** *jj*: *jj* ≤ *n jj* < *n jj* ≤ *i jj* < *i* **by** *auto*
have *zj*: [*0* ..< *j*] = [*0* ..< *jj*] @ [*jj*] **unfolding** *j* **by** *simp*
have *jjn*: [*jj* ..< *n*] = *jj* # [*j* ..< *n*] **using** *jj*(*2*) **unfolding** *j* **by** (*rule upt-conv-Cons*)
from *us-gs*[*unfolded zj*] **have** *ugs*: *u* = *gso jj* **and** *us*: *us* = *map gso* (*rev* [*0*..<*jj*])
by *auto*
let *?w* = *w* • *u* / (*u* • *u*)
have *muij*: *?w* = μ *i jj* **unfolding** μ.*simps*[*of i jj*] *ugs wi sq-norm-vec-as-cscalar-prod*
using *jj* **by** *auto*
have *wwits*: *?w* # *wits* = *map* (μ *i*) [*jj*..<*n*] **unfolding** *jjn wits muij* **by** *simp*
obtain *wwits b* **where** *rec*: *adjuster-wit* (*?w* # *wits*) *w us* = (*wwits*,*b*) **by** *force*
from *Cons*(*1*)[*OF this Cons*(*3*) *us wwits jj*(*1*,*3*),*unfolded j*] **have** *IH*:
adjuster n w us = *b wwits* = *map* (μ *i*) [*0*..<*n*]
b = *sumlist* (*map* (λ*j*. − μ *i j* $\cdot_v$ *gso j*) [*0*..<*jj*])
and *b*: *b* ∈ *carrier-vec n* **by** *auto*
from *Cons*(*2*)[*simplified*, *unfolded Let-def rec split sq-norm-vec-as-cscalar-prod*
cscalar-prod-is-scalar-prod]
have *id*: *wits'* = *wwits* **and** *a*: *a* = − *?w* $\cdot_v$ *u* + *b* **by** *auto*
have *1*: *adjuster n w* (*u* # *us*) = *a* **unfolding** *a IH*(*1*)[*symmetric*] **by** *auto*
from *id IH*(*2*) **have** *wits'*: *wits'* = *map* (μ *i*) [*0*..<*n*] **by** *simp*
have *carr*:*set* (*map* (λ*j*. − μ *i j* $\cdot_v$ *gso j*) [*0*..<*j*]) ⊆ *carrier-vec n*
set (*map* (λ*j*. − μ *i j* $\cdot_v$ *gso j*) [*0*..<*jj*]) ⊆ *carrier-vec n* **and** *u*:*u* ∈
carrier-vec n
using *Cons j* **by** (*auto intro*!:*gso-carrier'*)
from *u b a* **have** *ac*: *a* ∈ *carrier-vec n dim-vec* (−*?w* $\cdot_v$ *u*) = *n dim-vec b* = *n*
dim-vec u = *n* **by** *auto*
show *?case*
apply (*intro conjI*[*OF 1*] *ac exI conjI wits'*)
unfolding *carr a IH zj muij ugs*[*symmetric*] *map-append*
apply (*subst sumlist-append*)
using *Cons.prems j* **apply** *force*
using *b u ugs IH*(*3*) **by** *auto*
qed *auto*

lemma *sub2-wit*:
assumes *set us* ⊆ *carrier-vec n set ws* ⊆ *carrier-vec n length us* + *length ws* =
m
and *ws* = *map* (λ *i*. *fs* ! *i*) [*i* ..< *m*]
and *us* = *map gso* (*rev* [*0* ..< *i*])
and *us*: ⋀ *j*. *j* < *m* ⟹ *fs* ! *j* ∈ *carrier-vec n*

and *mn*: $m \leq n$
shows *snd* (*sub2-wit us ws*) = *vvs* $\Longrightarrow$ *gram-schmidt-sub2 n us ws* = *vvs*
$\wedge$ *vvs* = *map gso* [*i* ..< *m*]
using *assms*(*1*−*6*)
proof (*induct ws arbitrary*: *us vvs i*)
case (*Cons w ws us vs*)
note *us* = *Cons*(*3*) **note** *wws* = *Cons*(*4*)
note *wsf'* = *Cons*(*6*)
note *us-gs* = *Cons*(*7*)
from *wsf'* **have** $i < m$ $i \leq m$ **by** (*cases* $i < m$, *auto*)+
hence *i-m*: [*i* ..< *m*] = *i* # [*Suc i* ..< *m*] **by** (*metis upt-conv-Cons*)
from ‹$i < m$› *mn* **have** $i < n$ $i \leq n$ $i \leq m$ **by** *auto*
hence *i-n*: [*i* ..< *n*] = *i* # [*Suc i* ..< *n*] **by** (*metis upt-conv-Cons*)
from *wsf' i-m* **have** *wsf*: *ws* = *map* (λ *i*. *fs* ! *i*) [*Suc i* ..< *m*]
and *fiw*: *fs* ! *i* = *w* **by** *auto*
from *wws* **have** *w*: $w \in$ *carrier-vec n* **and** *ws*: *set ws* $\subseteq$ *carrier-vec n* **by** *auto*
let *?list* = *1* # *replicate* (*n* − *Suc* (*length us*)) *0*
have *map* (μ *i*) [*Suc i* ..< *n*] = *map* (λ *i*. *0*) [*Suc i* ..< *n*]
by (*rule map-cong*[*OF refl*], *unfold* μ*.simps*[*of i*], *auto*)
moreover have μ *i i* = *1* **unfolding** μ*.simps* **by** *simp*
ultimately have *map* (μ *i*) [*i* ..< *n*] = *1* # *map* (λ *i*. *0*) [*Suc i* ..< *n*] **unfolding** *i-n* **by** *auto*
also have ... = *?list* **using** ‹$i < n$› **unfolding** *map-replicate-const* **by** (*auto simp*: *us-gs*)
finally have *list*: *?list* = *map* (μ *i*) [*i* ..< *n*] **by** *auto*
let *?a* = *adjuster-wit ?list w us*
obtain *wit a* **where** *a*: *?a* = (*wit*,*a*) **by** *force*
obtain *vv* **where** *gs*: *snd* (*sub2-wit* ((*a* + *w*) # *us*) *ws*) = *vv* **by** *force*
from *adjuster-wit*[*OF a w Cons*(*8*) *us-gs list* ‹$i \leq n$› - *fiw*[*symmetric*]] *us wws* ‹$i < m$›
have *awus*: *set* ((*a* + *w*) # *us*) $\subseteq$ *carrier-vec n*
and *aa*: *adjuster n w us* = *a* $a \in$ *carrier-vec n*
and *aaa*: *a* = *sumlist* (*map* (λ*j*. − μ *i j* $\cdot_v$ *gso j*) [*0*..<*i*])
and *wit*: *wit* = *map* (μ *i*) [*0*..<*n*]
by *auto*
have *aw-gs*: *a* + *w* = *gso i* **unfolding** *gso.simps*[*of i*] *fiw aaa*[*symmetric*] **using** *aa*(*2*) *w* **by** *auto*
with *us-gs* **have** *us-gs'*: (*a* + *w*) # *us* = *map gso* (*rev* [*0*..<*Suc i*]) **by** *auto*
from *Cons*(*1*)[*OF gs awus ws - wsf us-gs' Cons*(*8*)] *Cons*(*5*)
have *IH*: *gram-schmidt-sub2 n* ((*a* + *w*) # *us*) *ws* = *vv*
and *vv*: *vv* = *map gso* [*Suc i*..<*m*] **by** *auto*
from *gs a aa IH Cons*(*5*)
have *gs-vs*: *gram-schmidt-sub2 n us* (*w* # *ws*) = *vs* **and** *vs*: *vs* = (*a* + *w*) # *vv* **using** *Cons*(*2*)
by (*auto simp add*: *Let-def snd-def split*:*prod.splits*)
from *vs vv aw-gs* **have** *vs*: *vs* = *map gso* [*i* ..< *m*] **unfolding** *i-m* **by** *auto*
with *gs-vs* **show** *?case* **by** *auto*
qed *auto*

lemma *inv-in-span*:
  assumes *incarr*[*intro*]:$U \subseteq$ *carrier-vec n* **and** *insp*:$a \in$ *span U*
  shows $- a \in$ *span U*
proof $-$
  from *insp*[*THEN in-spanE*] **obtain** *aa A* **where** *a*:$a =$ *lincomb aa A finite A A*
$\subseteq U$ **by** *auto*
  with *assms* **have** [*intro*!]:($\lambda v.$ *aa v* $\cdot_v$ *v*) $\in A \rightarrow$ *carrier-vec n* **by** *auto*
  from *a*(*1*) **have** *e1*:$- a =$ *lincomb* (λ *x*. $-$ *1* $*$ *aa x*) *A* **unfolding** *smult-smult-assoc*[*symmetric*]
lincomb-def
    by(*subst finsum-smult*[*symmetric*]) *force*+
  show *?thesis* **using** *e1 a span-def* **by** *blast*
qed

lemma *non-span-det-zero*:
  assumes *len*: *length G* $= n$
  and *nonb*:$\neg$ (*carrier-vec n* $\subseteq$ *span* (*set G*))
  and *carr*:*set G* $\subseteq$ *carrier-vec n*
  shows *det* (*mat-of-rows n G*) $=$ *0* **unfolding** *det-0-iff-vec-prod-zero*
proof $-$
  let *?A* $=$ (*mat-of-rows n G*)^T **let** *?B* $=$ 1_m *n*
  from *carr* **have** *carr-mat*:*?A* $\in$ *carrier-mat n n ?B* $\in$ *carrier-mat n n mat-of-rows*
n G $\in$ *carrier-mat n n*
    using *len mat-of-rows-carrier*(*1*) **by** *auto*
  from *carr* **have** *g-len*:$\bigwedge$ *i*. *i* $<$ *length G* $\Longrightarrow$ *G* ! *i* $\in$ *carrier-vec n* **by** *auto*
  from *nonb* **obtain** *v* **where** *v*:$v \in$ *carrier-vec n v* $\notin$ *span* (*set G*) **by** *fast*
  hence $v \neq 0_v$ *n* **using** *span-zero* **by** *auto*
  obtain *B C* **where** *gj*:*gauss-jordan ?A ?B* $=$ (*B*,*C*) **by** *force*
  note *gj* $=$ *carr-mat*(*1*,*2*) *gj*
  hence *B*:*B* $=$ *fst* (*gauss-jordan ?A ?B*) **by** *auto*
  from *gauss-jordan*[*OF gj*] **have** *BC*:*B* $\in$ *carrier-mat n n* **by** *auto*
  from *gauss-jordan-transform*[*OF gj*] **obtain** *P* **where**
   P:*P*$\in$*Units* (*ring-mat TYPE*($'a$) *n ?B*) *B* $=$ *P* $*$ *?A* **by** *fast*
  hence *PC*:*P* $\in$ *carrier-mat n n* **unfolding** *Units-def* **by** (*simp add*: *ring-mat-simps*)
  from *mat-inverse*[*OF PC*] *P* **obtain** *PI* **where** *mat-inverse P* $=$ *Some PI* **by**
fast
  from *mat-inverse*(*2*)[*OF PC this*]
  have *PI*:*P* $*$ *PI* $=$ 1_m *n PI* $*$ *P* $=$ 1_m *n PI* $\in$ *carrier-mat n n* **by** *auto*
  have *B* $\neq$ 1_m *n* **proof**
    assume *B* $=$ *?B*
    hence *?A* $*$ *P* $=$ *?B* **unfolding** *P*
      using *PC P*(*2*) *carr-mat*(*1*) *mat-mult-left-right-inverse* **by** *blast*
    hence *?A* $*$ *P* $*_v$ *v* $=$ *v* **using** *v* **by** *auto*
    hence *?A* $*_v$ (*P* $*_v$ *v*) $=$ *v* **unfolding** *assoc-mult-mat-vec*[*OF carr-mat*(*1*) *PC*
v(*1*)].
    hence *v-eq*:*mat-of-cols n G* $*_v$ (*P* $*_v$ *v*) $=$ *v*
      unfolding *transpose-mat-of-rows* **by** *auto*
    have *pvc*:*P* $*_v$ *v* $\in$ *carrier-vec* (*length G*) **using** *PC v len* **by** *auto*
    from *mat-of-cols-mult-as-finsum*[*OF pvc g-len*,*unfolded v-eq*] **obtain** *a* **where**
      v $=$ *lincomb a* (*set G*) **by** *auto*

hence *v* ∈ *span* (*set G*) **by** (*intro in-spanI*[*OF - finite-set subset-refl*])
thus *False* **using** *v* **by** *auto*
qed
with *det-non-zero-imp-unit*[*OF carr-mat*(*1*)] **show** *?thesis*
unfolding *gauss-jordan-check-invertable*[*OF carr-mat*(*1*,*2*)] *B det-transpose*[*OF carr-mat*(*3*)]
by *metis*
qed

lemma *span-basis-det-zero-iff*:
assumes *length G* = *n set G* ⊆ *carrier-vec n*
shows *carrier-vec n* ⊆ *span* (*set G*) ⟷ *det* (*mat-of-rows n G*) ≠ *0* (**is** *?q1*)
carrier-vec n ⊆ *span* (*set G*) ⟷ *basis* (*set G*) (**is** *?q2*)
det (*mat-of-rows n G*) ≠ *0* ⟷ *basis* (*set G*) (**is** *?q3*)
proof −
have *dc*:*det* (*mat-of-rows n G*) ≠ *0* ⟹ *carrier-vec n* ⊆ *span* (*set G*)
using *assms non-span-det-zero* **by** *auto*
have *cb*:*carrier-vec n* ⊆ *span* (*set G*) ⟹ *basis* (*set G*) **using** *assms basis-list-basis*

by (*auto simp*: *basis-list-def*)
have *bd*:*basis* (*set G*) ⟹ *det* (*mat-of-rows n G*) ≠ *0* **using** *assms basis-det-nonzero* **by** *auto*
show *?q1 ?q2 ?q3* **using** *dc cb bd* **by** *metis+*
qed

lemma *partial-connect*: **fixes** *vs*
assumes *length fs* = *m k* ≤ *m m* ≤ *n set us* ⊆ *carrier-vec n snd* (*main us*) = *vs*
us = *take k fs set fs* ⊆ *carrier-vec n*
shows *gram-schmidt n us* = *vs*
vs = *map gso* [*0*..<*k*]
proof −
have [*simp*]: *map* (*op* ! *fs*) [*0*..<*k*] = *take k fs* **using** *assms*(*1*,*2*) **by** (*intro nth-equalityI*, *auto*)
have *carr*: *j* < *m* ⟹ *fs* ! *j* ∈ *carrier-vec n* **for** *j* **using** *assms* **by** *auto*
from *sub2-wit*[*OF - assms*(*4*) - - - *carr - assms*(*5*)[*unfolded main-def*], *of k 0*] *assms*
show *gram-schmidt n us* = *vs vs* = *map gso* [*0*..<*k*] **unfolding** *gram-schmidt-code* **by** *auto*
qed

lemma *adjuster-wit-small*:
(*adjuster-wit v a xs*) = (*x1*,*x2*)
⟷ (*fst* (*adjuster-wit v a xs*) = *x1* ∧ *x2* = *adjuster n a xs*)
proof(*induct xs arbitrary*: *a v x1 x2*)
case (*Cons a xs*)
then show *?case*
by (*auto simp*:*Let-def sq-norm-vec-as-cscalar-prod split*:*prod.splits*)
qed *auto*

lemma *rev-unsimp*: *rev xs* @ (*r* # *rs*) = *rev* (*r*#*xs*) @ *rs* **by**(*induct xs*,*auto*)

lemma *sub2*: *rev xs* @ *snd* (*sub2-wit xs us*) = *rev* (*gram-schmidt-sub n xs us*)
proof −
  have *sub2-wit xs us* = (*x1*, *x2*) ⟹ *rev xs* @ *x2* = *rev* (*gram-schmidt-sub n xs us*)
    for *x1 x2 xs us*
    apply(*induct us arbitrary*: *xs x1 x2*)
   by (*auto simp*:*Let-def rev-unsimp adjuster-wit-small split*:*prod.splits simp del*:*rev.simps*)
  thus *?thesis*
    apply (*cases us*)
   by (*auto simp*:*Let-def rev-unsimp adjuster-wit-small split*:*prod.splits simp del*:*rev.simps*)
qed

lemma *gso-connect*: *snd* (*main us*) = *gram-schmidt n us* **unfolding** *main-def gram-schmidt-def*
  using *sub2*[*of Nil us*] **by** *auto*

lemma *lin-indpt-list-nonzero*:
  assumes *lin-indpt-list G*
  shows 0_v *n* ∉ *set G*
proof−
  from *assms*[*unfolded lin-indpt-list-def*] **have** *lin-indpt* (*set G*) **by** *auto*
  from *vs-zero-lin-dep*[*OF - this*] *assms*[*unfolded lin-indpt-list-def*] **show** *zero*: 0_v *n* ∉ *set G* **by** *auto*
qed

context
  fixes *m* :: *nat*
begin
definition *M* **where** *M* ≡ *mat m m* (λ (*i*,*j*). μ *i j*)

context
  fixes *vs*
  assumes *indep*: *lin-indpt-list fs*
   and *len-fs*: *length fs* = *m*
   and *snd-main*: *snd* (*main fs*) = *vs*
begin

lemma *fs-carrier*[*simp*]: *set fs* ⊆ *carrier-vec n*
  and *dist*: *distinct fs*
  and *lin-indpt*: *lin-indpt* (*set fs*)
  using *indep*[*unfolded lin-indpt-list-def*] **by** *auto*

lemmas *assm* = *len-fs fs-carrier snd-main*

lemma *f-carrier*[*simp*]: *i* < *m* ⟹ *fs* ! *i* ∈ *carrier-vec n*

using *fs-carrier len-fs* **unfolding** *set-conv-nth* **by** *force*

lemma *gso-carrier*[*simp*]: $i < m \Longrightarrow gso\ i \in carrier\text{-}vec\ n$
using *gso-carrier′ f-carrier* **by** *auto*

lemma *gso-dim*[*simp*]: $i < m \Longrightarrow dim\text{-}vec\ (gso\ i) = n$ **by** *auto*
lemma *f-dim*[*simp*]: $i < m \Longrightarrow dim\text{-}vec\ (fs\ !\ i) = n$ **by** *auto*

lemma *mn*: $m \leq n$
proof –
have *n*: $n = dim$ **by** (*simp add*: *dim-is-n*)
have *m*: $m = card\ (set\ fs)$ **unfolding** *len-fs*[*symmetric*]
using *distinct-card*[*OF dist*] **by** *simp*
from *m n* **have** *mn*: $m \leq n \longleftrightarrow card\ (set\ fs) \leq dim$ **by** *simp*
show *?thesis* **unfolding** *mn*
by (*rule li-le-dim*[*OF - fs-carrier lin-indpt*], *simp*)
qed

lemma *main-connect*:
gram-schmidt n fs = vs
vs = map gso [0..<m]
proof –
have *gram-schmidt-sub2 n [] fs = vs* $\wedge$ *vs = map gso [0..<m]*
by (*rule sub2-wit*[*OF - assm(2) - - - - mn*], *insert snd-main len-fs*,
auto simp: *main-def intro*!: *nth-equalityI*)
thus *gram-schmidt n fs = vs vs = map gso [0..<m]* **by** (*auto simp*: *gram-schmidt-code*)
qed

lemma *reduced-vs-E*: *weakly-reduced* α *k vs* $\Longrightarrow k \leq m \Longrightarrow Suc\ i < k \Longrightarrow$
$sq\text{-}norm\ (gso\ i) \leq \alpha * sq\text{-}norm\ (gso\ (Suc\ i))$
unfolding *weakly-reduced-def main-connect*(*2*) **by** *auto*

abbreviation (*input*) *FF* **where** $FF \equiv$ *mat-of-rows n fs*
abbreviation (*input*) *Fs* **where** $Fs \equiv$ *mat-of-rows n vs*

lemma *FF-dim*[*simp*]: *dim-row FF = m dim-col FF = n FF* $\in$ *carrier-mat m n*
unfolding *mat-of-rows-def* **by** (*auto simp*: *assm len-fs*)

lemma *Fs-dim*[*simp*]: *dim-row Fs = m dim-col Fs = n Fs* $\in$ *carrier-mat m n*
unfolding *mat-of-rows-def* **by** (*auto simp*: *assm main-connect*)

lemma *M-dim*[*simp*]: *dim-row M = m dim-col M = m M* $\in$ *carrier-mat m m*
unfolding *M-def* **by** *auto*

lemma *FF-index*[*simp*]: $i < m \Longrightarrow j < n \Longrightarrow FF\ \$\$\ (i,j) = fs\ !\ i\ \$\ j$
unfolding *mat-of-rows-def* **using** *assm* **by** *auto*

lemma *M-index*[*simp*]:$i < m \Longrightarrow j < m \Longrightarrow M\ \$\$\ (i,j) = \mu\ i\ j$
unfolding *M-def* **by** *auto*

lemma *matrix-equality*: *FF* = *M* ∗ *Fs*
proof −
let *?P* = *M* ∗ *Fs*
have *dim*: *dim-row FF* = *m dim-col FF* = *n dim-row ?P* = *m dim-col ?P* = *n* *dim-row M* = *m dim-col M* = *m*
dim-row Fs = *m dim-col Fs* = *n*
by (*auto simp*: *assm mat-of-rows-def mat-of-rows-list-def main-connect len-fs*)
show *?thesis*
proof (*rule eq-matI*; *unfold dim*)
fix *i j*
assume *i*: *i* < *m* **and** *j*: *j* < *n*
from *i* **have** *split*: [*0* ..< *m*] = [*0* ..< *i*] @ [*i*] @ [*Suc i* ..< *m*]
by (*metis append-Cons append-self-conv2 less-Suc-eq-le less-imp-add-positive upt-add-eq-append upt-rec zero-less-Suc*)
let *?prod* = λ *k*. μ *i k* ∗ *gso k* $ *j*
have *dim2*: *dim-vec* (*col Fs j*) = *m* **using** *j dim* **by** *auto*
define *idx* **where** *idx* = [*0*..<*i*]
have *idx*: *set idx* ⊆ {*0* ..< *i*} **unfolding** *idx-def* **using** *i* **by** *auto*
let *?vec* = *sumlist* (*map* (λ*j*. − μ *i j* $\cdot_v$ *gso j*) *idx*)
have *vec*: *?vec* ∈ *carrier-vec n* **by** (*rule sumlist-carrier*, *insert idx gso-carrier i*, *auto*)
hence *dimv*: *dim-vec ?vec* = *n* **by** *auto*
have *?P* $$ (*i*,*j*) = *row M i* · *col Fs j* **using** *dim i j* **by** *auto*
also have . . . = (∑ *k* = *0*..<*m*. *row M i* $ *k* ∗ *col Fs j* $ *k*)
unfolding *scalar-prod-def dim2* **by** *auto*
also have . . . = (∑ *k* = *0*..<*m*. *?prod k*)
by (*rule sum.cong*[*OF refl*], *insert i j dim*, *auto simp*: *mat-of-rows-list-def mat-of-rows-def main-connect*(*2*))
also have . . . = *sum-list* (*map ?prod* [*0* ..< *m*])
by (*subst sum-list-distinct-conv-sum-set*, *auto*)
also have . . . = *sum-list* (*map ?prod idx*) + *?prod i* + *sum-list* (*map ?prod* [*Suc i* ..< *m*])
unfolding *split idx-def* **by** *auto*
also have *?prod i* = *gso i* $ *j* **unfolding** *μ.simps* **by** *simp*
also have . . . = *fs* ! *i* $ *j* + *sum-list* (*map* (λ*k*. − μ *i k* ∗ *gso k* $ *j*) *idx*) **unfolding** *gso.simps*[*of i*] *idx-def*[*symmetric*]
by (*subst index-add-vec*, *unfold dimv*, *rule j*, *subst sumlist-vec-index*[*OF - j*], *insert idx gso-carrier i j*,
auto simp: *o-def intro*!: *sum-list-cong*)
also have *sum-list* (*map* (λ*k*. − μ *i k* ∗ *gso k* $ *j*) *idx*) = − *sum-list* (*map* (λ*k*. μ *i k* ∗ *gso k* $ *j*) *idx*)
by (*induct idx*, *auto*)
also have *sum-list* (*map ?prod* [*Suc i* ..< *m*]) = *0*
by (*rule sum-list-neutral*, *auto simp*: *μ.simps*)
finally have *?P* $$ (*i*,*j*) = *fs* ! *i* $ *j* **by** *simp*
with *FF-index*[*OF i j*]
show *FF* $$ (*i*,*j*) = *?P* $$ (*i*,*j*) **by** *simp*

qed *auto*
qed

lemma *fi-is-sum-of-mu-gso*: **assumes** *i*: $i < m$
shows *fs* ! i = *sumlist* (*map* (λ *j*. μ *i j* $\cdot_v$ *gso j*) [*0* ..< *Suc i*])
proof −
let *?l* = *sumlist* (*map* (λ *j*. μ *i j* $\cdot_v$ *gso j*) [*0* ..< *Suc i*])
have *?l* $\in$ *carrier-vec n* **by** (*rule sumlist-carrier*, *insert gso-carrier i*, *auto*)
hence *dim*: *dim-vec ?l* = *n* **by** (*rule carrier-vecD*)
show *?thesis*
proof (*rule eq-vecI*, *unfold dim f-dim*[*OF i*])
fix *j*
assume *j*: $j < n$
from *i* **have** *split*: [*0* ..< *m*] = [*0* ..< *Suc i*] @ [*Suc i* ..< *m*]
by (*metis Suc-lessI append.assoc append-same-eq less-imp-add-positive order-refl upt-add-eq-append zero-le*)
let *?prod* = λ *k*. μ *i k* $*$ *gso k* \$ *j*
have *fs* ! *i* \$ *j* = *FF* \$\$ (*i*,*j*) **using** *i j* **by** *simp*
also have … = (*M* $*$ *Fs*) \$\$ (*i*,*j*) **using** *matrix-equality* **by** *simp*
also have … = *row M i* $\cdot$ *col Fs j* **using** *i j* **by** *auto*
also have … = ($\sum$ *k* = *0*..<*m*. *row M i* \$ *k* $*$ *col Fs j* \$ *k*)
unfolding *scalar-prod-def* **by** (*auto simp*: *main-connect*(*2*))
also have … = ($\sum$ *k* = *0*..<*m*. *?prod k*)
by (*rule sum.cong*[*OF refl*], *insert i j dim*, *auto simp*: *mat-of-rows-list-def mat-of-rows-def main-connect*(*2*))
also have … = *sum-list* (*map ?prod* [*0* ..< *m*])
by (*subst sum-list-distinct-conv-sum-set*, *auto*)
also have … = *sum-list* (*map ?prod* [*0* ..< *Suc i*]) + *sum-list* (*map ?prod* [*Suc i* ..< *m*])
unfolding *split* **by** *auto*
also have *sum-list* (*map ?prod* [*Suc i* ..< *m*]) = *0*
by (*rule sum-list-neutral*, *auto simp*: μ.*simps*)
also have *sum-list* (*map ?prod* [*0* ..< *Suc i*]) = *?l* \$ *j*
by (*subst sumlist-vec-index*[*OF - j*], (*insert i*, *auto simp*: *intro*!: *gso-carrier*)[*1*],

rule arg-cong[*of - - sum-list*], *insert i j*, *auto*)
finally show *fs* ! *i* \$ *j* = *?l* \$ *j* **by** *simp*
qed *simp*
qed

lemma *vs*:
shows *set vs* $\subseteq$ *carrier-vec n distinct vs corthogonal* (*rev vs*)
span (*set fs*) = *span* (*set vs*) *length vs* = *length fs*
proof −
from *main-connect*(*1*)[*unfolded gram-schmidt-def*] **have** *e*:*gram-schmidt-sub n* [] *fs* = *rev vs* **by** *auto*
have *set* [] $\subseteq$ *carrier-vec n distinct* ([] @ *fs*) *lin-indpt* (*set* ([] @ *fs*))
corthogonal [] **using** *assm lin-indpt dist* **by** *auto*
from *cof-vec-space.gram-schmidt-sub-result*[*OF e assm*(*2*) *this*]

```
  show set vs ⊆ carrier-vec n distinct vs  corthogonal (rev vs)
    span (set fs) = span (set vs) length vs = length fs
    by auto
qed

lemma gso-inj[intro]:
  shows i < m ⟹ inj-on gso {0..<i}
proof
  fix x y assume assms:i < m x ∈ {0..<i} y ∈ {0..<i} gso x = gso y
  have distinct vs x < length vs y < length vs using vs assms assm by auto
  from nth-eq-iff-index-eq[OF this] assms main-connect show x = y by auto
qed

lemmas gram-schmidt = cof-vec-space.gram-schmidt-result[OF fs-carrier dist lin-indpt
main-connect(1)[symmetric]]

lemma partial-span: assumes i: i ≤ m shows span (gso ‘ {0 ..< i}) = span (set
(take i fs))
proof −
  let ?f = λ i. fs ! i
  let ?us = take i fs
  have len: length ?us = i using len-fs i by auto
  from fs-carrier len-fs i have us: set ?us ⊆ carrier-vec n
    by (meson set-take-subset subset-trans)
  obtain vi where main: snd (main ?us) = vi by force
  from dist have dist: distinct ?us by auto
  from lin-indpt have indpt: lin-indpt (set ?us)
    using supset-ld-is-ld[of set ?us, of set (?us @ drop i fs)]
    by (auto simp: set-take-subset)
  from partial-connect[OF len-fs i mn us main refl fs-carrier]
  have gso: vi = gram-schmidt n ?us and vi: vi = map gso [0 ..< i] by auto
  from cof-vec-space.gram-schmidt-result(1)[OF us dist indpt gso, unfolded vi]
  show ?thesis by auto
qed

lemma partial-span': assumes i: i ≤ m shows span (gso ‘ {0 ..< i}) = span ((λ
j. fs ! j) ‘ {0 ..< i})
  unfolding partial-span[OF i]
  by (rule arg-cong[of - - span], subst nth-image, insert i len-fs, auto)


lemma det: assumes m: m = n shows det FF = det Fs
  unfolding matrix-equality
  apply (subst det-mult[OF M-dim(3)], (insert Fs-dim(3) m, auto)[1])
  apply (subst det-lower-triangular[OF - M-dim(3)])
 by (subst M-index, (auto simp: μ.simps)[3], unfold prod-list-diag-prod, auto simp:
μ.simps)
```

lemma *orthogonal*: $i < m \Longrightarrow j < m \Longrightarrow i \neq j \Longrightarrow$ *gso i* $\cdot$ *gso j* $= 0$
  using *gram-schmidt*(*2*)[*unfolded main-connect corthogonal-def*] **by** *auto*

lemma *same-base*: *span* (*set fs*) = *span* (*gso* ‘ {*0*..<*m*})
  using *gram-schmidt*(*1*)[*unfolded - main-connect*] **by** *auto*

lemma *sq-norm-gso-le-f*: **assumes** *i*: $i < m$
  shows *sq-norm* (*gso i*) $\leq$ *sq-norm* (*fs* ! *i*)
proof −
  have *id*: [*0* ..< *Suc i*] = [*0* ..< *i*] @ [*i*] **by** *simp*
  let *?sum* = *sumlist* (*map* ($\lambda j.\ \mu\ i\ j \cdot_v$ *gso j*) [*0*..<*i*])
  have *sum*: *?sum* $\in$ *carrier-vec n* **and** *gsoi*: *gso i* $\in$ *carrier-vec n* **using** *i*
    by (*auto intro*!: *sumlist-carrier gso-carrier*)
  from *fi-is-sum-of-mu-gso*[*OF i*, *unfolded id*]
  have *sq-norm* (*fs* ! *i*) = *sq-norm* (*sumlist* (*map* ($\lambda j.\ \mu\ i\ j \cdot_v$ *gso j*) [*0*..<*i*] @ [*gso i*])) **by** (*simp add*: μ*.simps*)
  also have ... = *sq-norm* (*?sum* + *gso i*)
    by (*subst sumlist-append*, *insert gso-carrier i*, *auto*)
  also have ... = (*?sum* + *gso i*) $\cdot$ (*?sum* + *gso i*) **by** (*simp add*: *sq-norm-vec-as-cscalar-prod*)
  also have ... = *?sum* $\cdot$ (*?sum* + *gso i*) + *gso i* $\cdot$ (*?sum* + *gso i*)
    by (*rule add-scalar-prod-distrib*[*OF sum gsoi*], *insert sum gsoi*, *auto*)
  also have ... = (*?sum* $\cdot$ *?sum* + *?sum* $\cdot$ *gso i*) + (*gso i* $\cdot$ *?sum* + *gso i* $\cdot$ *gso i*)
    by (*subst* (*1 2*) *scalar-prod-add-distrib*[*of - n*], *insert sum gsoi*, *auto*)
  also have *?sum* $\cdot$ *?sum* = *sq-norm ?sum* **by** (*simp add*: *sq-norm-vec-as-cscalar-prod*)
  also have *gso i* $\cdot$ *gso i* = *sq-norm* (*gso i*) **by** (*simp add*: *sq-norm-vec-as-cscalar-prod*)
  also have *gso i* $\cdot$ *?sum* = *?sum* $\cdot$ *gso i* **using** *gsoi sum* **by** (*simp add*: *comm-scalar-prod*)
  finally have *sq-norm* (*fs* ! *i*) = *sq-norm ?sum* + *2* $*$ (*?sum* $\cdot$ *gso i*) + *sq-norm* (*gso i*) **by** *simp*
  also have ... $\geq$ *2* $*$ (*?sum* $\cdot$ *gso i*) + *sq-norm* (*gso i*) **using** *sq-norm-vec-ge-0*[*of ?sum*] **by** *simp*
  also have *?sum* $\cdot$ *gso i* = ($\sum$ *v*$\leftarrow$*map* ($\lambda j.\ \mu\ i\ j \cdot_v$ *gso j*) [*0*..<*i*]. *v* $\cdot$ *gso i*)
    by (*subst scalar-prod-left-sum-distrib*[*OF - gsoi*], *insert i gso-carrier*, *auto*)
  also have ... = *0*
  proof (*rule sum-list-neutral*, *goal-cases*)
    case (*1 x*)
    then obtain *j* **where** *j*: $j < i$ **and** *x*: *x* = ($\mu\ i\ j \cdot_v$ *gso j*) $\cdot$ *gso i* **by** *auto*
    from *j i* **have** *gsoj*: *gso j* $\in$ *carrier-vec n* **by** *auto*
    have *x* = $\mu\ i\ j$ $*$ (*gso j* $\cdot$ *gso i*) **using** *gsoi gsoj* **unfolding** *x* **by** *simp*
    also have *gso j* $\cdot$ *gso i* = *0*
      by (*rule orthogonal*, *insert j i*, *auto*)
    finally show *x* = *0* **by** *simp*
  qed
  finally show *?thesis* **by** *simp*
qed

```
lemma projection-exist:
  assumes i < m
  shows fs ! i − gso i ∈ span (gso ‘ {0..<i})
proof
  let ?A = gso ‘ {0..<i}
  show finA:finite ?A by auto
  have carA[intro!]:?A ⊆ carrier-vec n using gso-dim assms by auto
  let ?a v = ∑n←[0..<i]. if v = gso n then μ i n else 0
  have d:(sumlist (map (λj. − μ i j ·v gso j) [0..<i])) ∈ carrier-vec n
    using gso.simps[of i] gso-dim[OF assms] unfolding carrier-vec-def by auto
  note [intro] = f-carrier[OF assms] gso-carrier[OF assms] d
  have [intro!]:(λv. ?a v ·v v) ∈ gso ‘ {0..<i} → carrier-vec n
      using gso-carrier assms by auto
  {fix ia assume ia[intro]:ia < n
    have (∑x∈gso ‘ {0..<i}. (?a x ·v x) $ ia) =
      − (∑x←map (λj. − μ i j ·v gso j) [0..<i]. x $ ia)
   unfolding map-map comm-monoid-add-class.sum.reindex[OF gso-inj[OF assms]]
   unfolding atLeastLessThan-upt sum-set-upt-conv-sum-list-nat uminus-sum-list-map
o-def
    proof(rule sum-list-cong[OF refl],goal-cases)
      case (1 x) hence x:x < m x < i using assms by auto
      hence d:insert x (set [0..<i]) = {0..<i}
            count (mset [0..<i]) x = 1 by auto
      hence inj-on gso (insert x (set [0..<i])) using gso-inj[OF assms] by auto
      from inj-on-filter-key-eq[OF this,folded replicate-count-mset-eq-filter-eq]
     have [n←[0..<i] . gso x = gso n] = [x] using x assms d replicate.simps(2)[of
0] by auto
      hence (∑n←[0..<i]. if gso x = gso n then μ i n else 0) = μ i x
        unfolding sum-list-map-filter'[symmetric] by auto
      with ia gso-dim x show ?case apply(subst index-smult-vec) by force+
    qed
    hence (⨁Vv∈gso ‘ {0..<i}. ?a v ·v v) $ ia =
          (− local.sumlist (map (λj. − μ i j ·v gso j) [0..<i])) $ ia
      using d assms
      apply (subst (0 0) finsum-index index-uminus-vec) apply force+
      apply (subst sumlist-vec-index) by force+
  }
  hence id: (⨁Vv∈?A. ?a v ·v v) = − sumlist (map (λj. − μ i j ·v gso j) [0..<i])
   using d lincomb-dim[OF finA carA,unfolded lincomb-def] by(intro eq-vecI,auto)
  show fs ! i − gso i = lincomb ?a ?A unfolding lincomb-def gso.simps[of i] id
    by (rule eq-vecI, auto)
qed auto

lemma orthocompl-span:
  assumes ⋀ x. x ∈ S ⟹ v · x = 0 S ⊆ carrier-vec n and [intro]: v ∈ carrier-vec
n
  and y ∈ span S
  shows v · y = 0
proof −
```

{**fix** *a A*
assume $y = lincomb\ a\ A\ finite\ A\ A \subseteq S$
note *assms = assms this*
hence [*intro*!]:$lincomb\ a\ A \in carrier\text{-}vec\ n\ (\lambda v.\ a\ v \cdot_v v) \in A \to carrier\text{-}vec\ n$ **by** *auto*
have $\forall x{\in}A.\ (a\ x \cdot_v x) \cdot v = 0$ **proof fix** *x* **assume** $x \in A$ **note** *assms = assms this*
hence $x{:}x \in S$ **by** *auto*
with *assms* **have** [*intro*]:$x \in carrier\text{-}vec\ n$ **by** *auto*
from *assms*(*1*)[*OF x*] **have** $x \cdot v = 0$ **by**(*subst comm-scalar-prod*) *force+*
thus $(a\ x \cdot_v x) \cdot v = 0$
apply(*subst smult-scalar-prod-distrib*) **by** *force+*
qed
hence $v \cdot lincomb\ a\ A = 0$ **apply**(*subst comm-scalar-prod*) **apply** *force+* **unfolding** *lincomb-def*
apply(*subst finsum-scalar-prod-sum*) **by** *force+*
}
thus *?thesis* **using** ‹$y \in span\ S$› **unfolding** *span-def* **by** *auto*
qed

lemma *projection-unique*:
assumes $i < m$
$v \in carrier\text{-}vec\ n$
$\bigwedge x.\ x \in gso\ `\ \{0..<i\} \Longrightarrow v \cdot x = 0$
$fs\ !\ i - v \in span\ (gso\ `\ \{0..<i\})$
shows $v = gso\ i$
proof −
from *assms* **have** *carr-span*:$span\ (gso\ `\ \{0..<i\}) \subseteq carrier\text{-}vec\ n$ **by**(*intro span-is-subset2*) *auto*
from *assms* **have** *carr*: $gso\ `\ \{0..<i\} \subseteq carrier\text{-}vec\ n$ **by** *auto*
from *assms* **have** *eq*:$fs\ !\ i - (fs\ !\ i - v) = v$ **for** *v* **by** *auto*
from *orthocompl-span*[*OF assms*(*3*) *carr assms*(*2*)]
have $y \in span\ (gso\ `\ \{0..<i\}) \Longrightarrow v \cdot y = 0$ **for** *y* **by** *auto*
hence *oc1*:$fs\ !\ i - (fs\ !\ i - v) \in orthogonal\text{-}complement\ (span\ (gso\ `\ \{0..<i\}))$
unfolding *eq orthogonal-complement-def* **using** *assms* **by** *auto*
have $x \in gso\ `\ \{0..<i\} \Longrightarrow gso\ i \cdot x = 0$ **for** *x* **using** *assms orthogonal* **by** *auto*
hence $y \in span\ (gso\ `\ \{0..<i\}) \Longrightarrow gso\ i \cdot y = 0$ **for** *y*
by (*rule orthocompl-span*[*OF - carr gso-carrier*[*OF assms*(*1*)],*rule-format*])
hence *oc2*:$fs\ !\ i - (fs\ !\ i - gso\ i) \in orthogonal\text{-}complement\ (span\ (gso\ `\ \{0..<i\}))$
unfolding *eq orthogonal-complement-def* **using** *assms* **by** *auto*
note *pe= projection-exist*[*OF assms*(*1*)]
note *prerec = carr-span f-carrier*[*OF assms*(*1*)] *assms*(*4*) *oc1 projection-exist*[*OF assms*(*1*)] *oc2*
have *gsoi*: $gso\ i \in carrier\text{-}vec\ n\ fs\ !\ i \in carrier\text{-}vec\ n$ **by** (*rule gso-carrier*[*OF* ‹$i < m$›], *rule f-carrier*[*OF* ‹$i < m$›])
note *main = arg-cong*[*OF projection-alt-def*[*OF carr-span f-carrier*[*OF assms*(*1*)] *assms*(*4*) *oc1 pe oc2*],
of $\lambda\ v.\ - v\ \$\ j + fs\ !\ i\ \$\ j$ **for** *j*]

show $v = gso\ i$
proof (*intro eq-vecI*)
fix j
show $j < dim\text{-}vec\ (gso\ i) \Longrightarrow v\ \$\ j = gso\ i\ \$\ j$
using *assms*(*2*) *gsoi main*[*of j*] **by** *auto*
qed (*insert assms*(*2*) *gsoi*, *auto*)
qed

lemma *gso-projection*:
assumes $i < m$
shows $gso\ i = projection\ (gso\ \text{`}\ \{0..<i\})\ (fs\ !\ i)$
unfolding *projection-def is-projection-def*
proof (*rule some-equality*[*symmetric*,*OF - projection-unique*[*OF assms*]])
have *orthogonal*:$\bigwedge xa.\ xa < i \Longrightarrow gso\ i \cdot gso\ xa = 0$ **by** (*rule orthogonal*,*insert assms*, *auto*)
show $gso\ i \in carrier\text{-}vec\ n \land$
$fs\ !\ i - gso\ i \in span\ (gso\ \text{`}\ \{0..<i\}) \land$
$(\forall x.\ x \in gso\ \text{`}\ \{0..<i\} \longrightarrow gso\ i \cdot x = 0)$
using *gso-carrier*[*OF assms*] *projection-exist*[*OF assms*] *orthogonal* **by** *auto*
qed *auto*

lemma *gso-projection-span*:
assumes $i < m$
shows $gso\ i = projection\ (span\ (gso\ \text{`}\ \{0..<i\}))\ (fs\ !\ i)$
and $is\text{-}projection\ (gso\ i)\ (span\ (gso\ \text{`}\ \{0..<i\}))\ (fs\ !\ i)$
unfolding *projection-def is-projection-def*
proof (*rule some-equality*[*symmetric*,*OF - projection-unique*[*OF assms*]])
let $?P\ v = v \in carrier\text{-}vec\ n \land fs\ !\ i - v \in span\ (span\ (gso\ \text{`}\ \{0..<i\}))$
$\land (\forall x.\ x \in span\ (gso\ \text{`}\ \{0..<i\}) \longrightarrow v \cdot x = 0)$
have *carr*:$gso\ \text{`}\ \{0..<i\} \subseteq carrier\text{-}vec\ n$ **using** *assms* **by** *auto*
have $*$: $\bigwedge xa.\ xa < i \Longrightarrow gso\ i \cdot gso\ xa = 0$ **by** (*rule orthogonal*,*insert assms*, *auto*)
have *orthogonal*:$\bigwedge x.\ x \in span\ (gso\ \text{`}\ \{0..<i\}) \Longrightarrow gso\ i \cdot x = 0$
apply(*rule orthocompl-span*) **using** *assms* $*$ **by** *auto*
show $?P\ (gso\ i)\ ?P\ (gso\ i)$ **unfolding** *span-span*[*OF carr*]
using *gso-carrier*[*OF assms*] *projection-exist*[*OF assms*] *orthogonal* **by** *auto*
fix v **assume** p:$?P\ v$
then show $v \in carrier\text{-}vec\ n$ **by** *auto*
from p **show** $fs\ !\ i - v \in span\ (gso\ \text{`}\ \{0..<i\})$ **unfolding** *span-span*[*OF carr*] **by** *auto*
fix xa **assume** $xa \in gso\ \text{`}\ \{0..<i\}$
hence $xa \in span\ (gso\ \text{`}\ \{0..<i\})$ **using** *in-own-span*[*OF carr*] **by** *auto*
thus $v \cdot xa = 0$ **using** p **by** *auto*
qed

lemma *is-projection-eq*:
assumes *ispr*:$is\text{-}projection\ a\ S\ v\ is\text{-}projection\ b\ S\ v$
and *carr*: $S \subseteq carrier\text{-}vec\ n\ v \in carrier\text{-}vec\ n$
shows $a = b$

proof −
  from *carr* **have** *c2*:*span S* ⊆ *carrier-vec n v* ∈ *carrier-vec n* **by** *auto*
  have *a*:*v* − (*v* − *a*) = *a* **using** *carr ispr* **by** *auto*
  have *b*:*v* − (*v* − *b*) = *b* **using** *carr ispr* **by** *auto*
  have (*v* − *a*) = (*v* − *b*)
    apply(*rule projection-alt-def*[*OF c2*])
    using *ispr a b* **unfolding** *in-orthogonal-complement-span*[*OF carr*(*1*)]
    unfolding *orthogonal-complement-def is-projection-def* **by** *auto*
  hence *v* − (*v* − *a*) = *v* − (*v* − *b*) **by** *metis*
  thus *?thesis* **unfolding** *a b*.
qed

lemma *scalar-prod-lincomb-gso*: **assumes** *k*: $k \leq m$
  shows *sumlist* (*map* (λ *i*. *g i* $\cdot_v$ *gso i*) [*0* ..< *k*]) • *sumlist* (*map* (λ *i*. *g i* $\cdot_v$ *gso i*) [*0* ..< *k*])
    = *sum-list* (*map* (λ *i*. *g i* ∗ *g i* ∗ (*gso i* • *gso i*)) [*0* ..< *k*])
proof −
  have *id1*: *map* (λ*i*. *g i* $\cdot_v$ *map gso* [*0*..<*m*] ! *i*) [*0*..<*k*] = *map* (λ*i*. *g i* $\cdot_v$ *gso i*) [*0*..<*k*] **using** *k*
    by *auto*
  have *id2*: (∑ *i*←[*0*..<*k*]. *g i* ∗ *g i* ∗ (*map gso* [*0*..<*m*] ! *i* • *map gso* [*0*..<*m*] ! *i*))
    = (∑ *i*←[*0*..<*k*]. *g i* ∗ *g i* ∗ (*gso i* • *gso i*)) **using** *k*
    by (*intro sum-list-cong*, *auto*)
  from *k len-fs gram-schmidt* **have** *orthogonal vs set vs* ⊆ *carrier-vec n k* ≤ *length vs* **by** *auto*
  from *scalar-prod-lincomb-orthogonal*[*OF this*, *of g*, *unfolded main-connect*(*2*) *id1 id2*]
  show *?thesis* .
qed

lemma *gso-times-self-is-norm*:
  assumes *j* < *m* **shows** *fs* ! *j* • *gso j* = *sq-norm-vec* (*gso j*) (**is** *?lhs* = *?rhs*)
proof −
  have *?lhs* = *fs* ! *j* ·*c gso j* + *0* **by** *auto*
  also have *0* = *M*.*sumlist* (*map* (λ*ja*. − μ *j ja* $\cdot_v$ *gso ja*) [*0*..<*j*]) ·*c gso j* **using** *assms orthogonal*
    apply(*subst scalar-prod-left-sum-distrib*,*force*,*force*)
    by(*intro sum-list-0*[*symmetric*],*auto*)
  finally show *?thesis* **unfolding** *sq-norm-vec-as-cscalar-prod vec-conjugate-rat* **using** *assms*
    apply(*subst* (*2*) *gso*.*simps*)
    apply(*subst add-scalar-prod-distrib*[*OF f-carrier M*.*sumlist-carrier*])
    by *auto*
qed

lemma *gram-schmidt-short-vector*: **assumes** *in-L*: *h* ∈ *lattice-of fs* − {0_v *n*}

shows $\exists\ i < m.\ \|h\|^2 \geq \|gso\ i\|^2$
proof $-$
from *in-L* **have** *non-0*: $h \neq 0_v\ n$ **by** *auto*
from *in-L*[*unfolded lattice-of-def*] **obtain** *lam* **where**
h: $h = \ sumlist\ (map\ (\lambda\ i.\ of\text{-}int\ (lam\ i) \cdot_v fs\ !\ i)\ [0\ ..<\ length\ fs])$
by *auto*
have *in-L*: $h = sumlist\ (map\ (\lambda\ i.\ of\text{-}int\ (lam\ i) \cdot_v fs\ !\ i)\ [0\ ..<\ m])$ **unfolding** *assm length-map h*
by (*rule arg-cong*[*of - - sumlist*], *rule map-cong*, *auto simp*: *len-fs*)
let $?n = [0\ ..<\ m]$
let $?f = (\lambda\ i.\ of\text{-}int\ (lam\ i) \cdot_v fs\ !\ i)$
let $?vs = map\ ?f\ ?n$
let $?P = \lambda\ k.\ k < m \wedge lam\ k \neq 0$
define *k* **where** $k = (GREATEST\ kk.\ ?P\ kk)$
{
assume $*$: $\forall\ i < m.\ lam\ i = 0$
have *vs*: $?vs = map\ (\lambda\ i.\ 0_v\ n)\ ?n$
by (*rule map-cong*, *insert f-dim* $*$, *auto*)
have $h = 0_v\ n$ **unfolding** *in-L vs*
by (*rule sumlist-neutral*, *auto*)
with *non-0* **have** *False* **by** *auto*
}
then obtain *kk* **where** *?P kk* **by** *auto*
from *GreatestI-nat*[*of ?P*, *OF this*, *of m*] **have** *Pk*: *?P k* **unfolding** *k-def* **by** *auto*
hence *kn*: $k < m$ **by** *auto*
let $?gso = (\lambda i\ j.\ \mu\ i\ j \cdot_v gso\ j)$
have *k*: $k < i \Longrightarrow i < m \Longrightarrow lam\ i = 0$ **for** *i*
using *Greatest-le-nat*[*of ?P i m*, *folded k-def*] **by** *auto*
define *l* **where** $l = lam\ k$
from *Pk* **have** *l*: $l \neq 0$ **unfolding** *l-def* **by** *auto*
define *idx* **where** $idx = [0\ ..<\ k]$
have *idx*: $\bigwedge\ i.\ i \in set\ idx \Longrightarrow i < k\ \bigwedge\ i.\ i \in set\ idx \Longrightarrow i < m$ **unfolding** *idx-def* **using** *kn* **by** *auto*
from *Pk* **have** *split*: $[0\ ..<\ m] = idx\ @\ [k]\ @\ [Suc\ k\ ..<\ m]$ **unfolding** *idx-def*
by (*metis append-Cons append-self-conv2 less-Suc-eq-le less-imp-add-positive upt-add-eq-append*
upt-rec zero-less-Suc)
define *gg* **where** $gg = sumlist$
$(map\ (\lambda i.\ of\text{-}int\ (lam\ i) \cdot_v fs\ !\ i)\ idx) + of\text{-}int\ l \cdot_v sumlist\ (map\ (\lambda j.\ \mu\ k\ j \cdot_v gso\ j)\ idx)$
have $h = sumlist\ ?vs$ **unfolding** *in-L* **..**
also have $\ldots = sumlist\ ((map\ ?f\ idx\ @\ [?f\ k])\ @\ map\ ?f\ [Suc\ k\ ..<\ m])$ **unfolding** *split* **by** *auto*
also have $\ldots = sumlist\ (map\ ?f\ idx\ @\ [?f\ k]) + sumlist\ (map\ ?f\ [Suc\ k\ ..<\ m])$

by (*rule sumlist-append*, *auto intro*!: *f-carrier*, *insert Pk idx*, *auto*)
also have $sumlist\ (map\ ?f\ [Suc\ k\ ..<\ m]) = 0_v\ n$ **by** (*rule sumlist-neutral*, *auto simp*: *k*)

```
  also have sumlist (map ?f idx @ [?f k]) = sumlist (map ?f idx) + ?f k
    by (subst sumlist-append, auto intro!: f-carrier, insert Pk idx, auto)
  also have fs ! k = sumlist (map (?gso k) [0..<Suc k]) using fi-is-sum-of-mu-gso[OF
kn] by simp
  also have ... = sumlist (map (?gso k) idx @ [gso k]) by (simp add: μ.simps[of
k k] idx-def)
  also have ... = sumlist (map (?gso k) idx) + gso k
    by (subst sumlist-append, auto intro!: f-carrier, insert Pk idx, auto)
  also have of-int (lam k) ·v ... = of-int (lam k) ·v (sumlist (map (?gso k) idx))
    + of-int (lam k) ·v gso k
    unfolding idx-def
    by (rule smult-add-distrib-vec[OF sumlist-carrier], auto intro!: gso-carrier, in-
sert kn, auto)
  finally have h = sumlist (map ?f idx) +
       (of-int (lam k) ·v sumlist (map (?gso k) idx) + of-int (lam k) ·v gso k) +
0v n  by simp
  also have ... = gg + of-int l ·v gso k unfolding gg-def l-def
    by (rule eq-vecI, insert idx kn, auto simp: sumlist-vec-index,
      subst index-add-vec, auto simp: sumlist-dim kn, subst sumlist-dim, auto)
  finally have hgg: h = gg + of-int l ·v gso k .
  let ?k = [0 ..< k]
  define R where R = {gg. ∃ nu. gg = sumlist (map (λ i. nu i ·v gso i) idx)}
  {
    fix nu
    have dim-vec (sumlist (map (λ i. nu i ·v gso i) idx)) = n
      by (rule sumlist-dim, insert kn, auto simp: idx-def)
  } note dim-nu[simp] = this
  define kk where kk = ?k
  {
    fix v
    assume v ∈ R
   then obtain nu where v: v = sumlist (map (λ i. nu i ·v gso i) idx) unfolding
R-def by auto
    have dim-vec v = n unfolding gg-def v by simp
  } note dim-R = this
  {
    fix v1 v2
    assume v1 ∈ R v2 ∈ R
    then obtain nu1 nu2 where v1: v1 = sumlist (map (λ i. nu1 i ·v gso i) idx)
and
      v2: v2 = sumlist (map (λ i. nu2 i ·v gso i) idx)
      unfolding R-def by auto
    have v1 + v2 ∈ R unfolding R-def
      by (standard, rule exI[of - λ i. nu1 i + nu2 i], unfold v1 v2, rule eq-vecI,
        (subst sumlist-vec-index, insert idx, auto intro!: gso-carrier simp: o-def)+,
        unfold sum-list-addf[symmetric], induct idx, auto simp: algebra-simps)
  } note add-R = this
  have gg ∈ R unfolding gg-def
  proof (rule add-R)
```

show *of-int l* $\cdot_v$ *sumlist* (*map* ($\lambda j.$ μ *k j* $\cdot_v$ *gso j*) *idx*) $\in$ *R*
unfolding *R-def*
by (*standard*, *rule exI*[*of* - $\lambda i.$ *of-int l* $*$ μ *k i*], *rule eq-vecI*,
(*subst sumlist-vec-index*, *insert idx*, *auto intro*!: *gso-carrier simp*: *o-def*)+,
induct idx, *auto simp*: *algebra-simps*)
show *sumlist* (*map ?f idx*) $\in$ *R* **using** *idx*
proof (*induct idx*)
case *Nil*
show *?case* **by** (*simp add*: *R-def*, *intro exI*[*of* - λ *i*. *0*], *rule eq-vecI*,
(*subst sumlist-vec-index*, *insert idx*, *auto intro*!: *gso-carrier simp*: *o-def*)+,
induct idx, *auto*)
next
case (*Cons i idxs*)
have *sumlist* (*map ?f* (*i* # *idxs*)) = *sumlist* ([*?f i*] @ *map ?f idxs*) **by** *simp*
also have ... = *?f i* + *sumlist* (*map ?f idxs*)
by (*subst sumlist-append*, *insert Cons*(*3*), *auto intro*!: *f-carrier*)
finally have *id*: *sumlist* (*map ?f* (*i* # *idxs*)) = *?f i* + *sumlist* (*map ?f idxs*)
.
show *?case* **unfolding** *id*
proof (*rule add-R*[*OF* - *Cons*(*1*)[*OF Cons*(*2*−*3*)]])
from *Cons*(*2*−*3*) **have** *i*: *i* < *m i* < *k* **by** *auto*
hence *idx-split*: *idx* = [*0* ..< *Suc i*] @ [*Suc i* ..< *k*] **unfolding** *idx-def*
by (*metis Suc-lessI append-Nil2 less-imp-add-positive upt-add-eq-append upt-rec zero-le*)
{
fix *j*
assume *j*: *j* < *n*
define *idxs* **where** *idxs* = [*0* ..< *Suc i*]
let *?f* = λ *x*. ((*if x* < *Suc i then of-int* (*lam i*) $*$ μ *i x else 0*) $\cdot_v$ *gso x*) $ *j*
have ($\sum$*x*←*idx*. *?f x*) = ($\sum$*x*←[*0* ..< *Suc i*]. *?f x*) + ($\sum$*x*← [*Suc i* ..< *k*]. *?f x*)
unfolding *idx-split* **by** *auto*
also have ($\sum$*x*← [*Suc i* ..< *k*]. *?f x*) = *0* **by** (*rule sum-list-neutral*, *insert j kn*, *auto*)
also have ($\sum$*x*←[*0* ..< *Suc i*]. *?f x*) = ($\sum$*x*←*idxs*. *of-int* (*lam i*) $*$ (μ *i x* $\cdot_v$ *gso x*) $ *j*)
unfolding *idxs-def* **by** (*rule arg-cong*[*of* - - *sum-list*], *rule map-cong*[*OF refl*],
subst index-smult-vec, *insert j i kn*, *auto*)
also have ... = *of-int* (*lam i*) $*$ (($\sum$*x*←[*0*..<*Suc i*]. (μ *i x* $\cdot_v$ *gso x*) $ *j*))
unfolding *idxs-def*[*symmetric*] **by** (*induct idxs*, *auto simp*: *algebra-simps*)
finally have ($\sum$*x*←*idx*. *?f x*) = *of-int* (*lam i*) $*$ (($\sum$*x*←[*0*..<*Suc i*]. (μ *i x* $\cdot_v$ *gso x*) $ *j*))
by *simp*
} **note** *main* = *this*
show *?f i* $\in$ *R* **unfolding** *fi-is-sum-of-mu-gso*[*OF i*(*1*)] *R-def*
apply (*standard*, *rule exI*[*of* - λ *j*. *if j* < *Suc i then of-int* (*lam i*) $*$ μ *i j else 0*], *rule eq-vecI*)

apply (*subst sumlist-vec-index*, *insert idx i*, *auto intro*!: *gso-carrier sumlist-dim simp*: *o-def*)
apply (*subst index-smult-vec*, *subst sumlist-dim*, *auto*)
apply (*subst sumlist-vec-index*, *auto*, *insert idx i main*, *auto simp*: *o-def*)
done
qed *auto*
qed
qed
then obtain *nu* **where** *gg*: $gg = sumlist\ (map\ (\lambda\ i.\ nu\ i \cdot_v gso\ i)\ idx)$ **unfolding** *R-def* **by** *auto*
let $?ff = sumlist\ (map\ (\lambda i.\ nu\ i \cdot_v gso\ i)\ idx) + of\text{-}int\ l \cdot_v gso\ k$
define *hh* **where** $hh = (\lambda\ i.\ (if\ i < k\ then\ nu\ i\ else\ of\text{-}int\ l))$
let $?hh = sumlist\ (map\ (\lambda\ i.\ hh\ i \cdot_v gso\ i)\ [0\ ..< Suc\ k])$
have *ffhh*: $?hh = sumlist\ (map\ (\lambda\ i.\ hh\ i \cdot_v gso\ i)\ [0\ ..< k]\ @\ [hh\ k \cdot_v gso\ k])$
by *simp*
also have $\ldots = sumlist\ (map\ (\lambda\ i.\ hh\ i \cdot_v gso\ i)\ [0\ ..< k]) + sumlist\ [hh\ k \cdot_v gso\ k]$
by (*rule sumlist-append*, *insert kn*, *auto*)
also have $sumlist\ [hh\ k \cdot_v gso\ k] = hh\ k \cdot_v gso\ k$ **using** *kn* **by** *auto*
also have $\ldots = of\text{-}int\ l \cdot_v gso\ k$ **unfolding** *hh-def* **by** *auto*
also have $map\ (\lambda\ i.\ hh\ i \cdot_v gso\ i)\ [0\ ..< k] = map\ (\lambda\ i.\ nu\ i \cdot_v gso\ i)\ [0\ ..< k]$
by (*rule map-cong*, *auto simp*: *hh-def*)
finally have *ffhh*: $?ff = ?hh$ **by** (*simp add*: *idx-def*)
from *hgg*[*unfolded gg*]
have *h*: $h = ?ff$ **by** *auto*
have $gso\ k \bullet gso\ k \le 1 * (gso\ k \bullet gso\ k)$ **by** *simp*
also have $\ldots \le of\text{-}int\ (l * l) * (gso\ k \bullet gso\ k)$
proof (*rule mult-right-mono*)
from *l* **have** $l * l \ge 1$ **by** (*meson eq-iff int-one-le-iff-zero-less mult-le-0-iff not-le*)
thus $1 \le (of\text{-}int\ (l * l) :: 'a)$ **by** *presburger*
show $0 \le gso\ k \bullet gso\ k$ **by** (*rule scalar-prod-ge-0*)
qed
also have $\ldots = 0 + of\text{-}int\ (l * l) * (gso\ k \bullet gso\ k)$ **by** *simp*
also have $\ldots \le sum\text{-}list\ (map\ (\lambda\ i.\ (nu\ i * nu\ i) * (gso\ i \bullet gso\ i))\ idx) + of\text{-}int\ (l * l) * (gso\ k \bullet gso\ k)$
by (*rule add-right-mono*, *rule sum-list-nonneg*, *auto*, *rule mult-nonneg-nonneg*, *auto simp*: *scalar-prod-ge-0*)
also have $map\ (\lambda\ i.\ (nu\ i * nu\ i) * (gso\ i \bullet gso\ i))\ idx = map\ (\lambda\ i.\ hh\ i * hh\ i * (gso\ i \bullet gso\ i))\ [0..<k]$
unfolding *idx-def* **by** (*rule map-cong*, *auto simp*: *hh-def*)
also have $of\text{-}int\ (l * l) = hh\ k * hh\ k$ **unfolding** *hh-def* **by** *auto*
also have $(\sum i{\leftarrow}[0..<k].\ hh\ i * hh\ i * (gso\ i \bullet gso\ i)) + hh\ k * hh\ k * (gso\ k \bullet gso\ k)$
$= (\sum i{\leftarrow}[0..<Suc\ k].\ hh\ i * hh\ i * (gso\ i \bullet gso\ i))$ **by** *simp*
also have $\ldots = ?hh \bullet ?hh$ **by** (*rule sym*, *rule scalar-prod-lincomb-gso*, *insert kn*, *auto*)
also have $\ldots = ?ff \bullet ?ff$ **by** (*simp add*: *ffhh*)
also have $\ldots = h \bullet h$ **unfolding** *h* **..**

finally show *?thesis* **using** *kn* **unfolding** *sq-norm-vec-as-cscalar-prod* **by** *auto*
qed

lemma *fs0-gso0*: $0 < m \Longrightarrow fs\ !\ 0 = gso\ 0$
unfolding *gso.simps*[*of 0*] **using** *f-dim*[*of 0*] *arg-cong*[*OF assm*(*3*), *of hd*] *len-fs*

by (*cases fs*, *auto simp add*: *upt-rec*)

lemma *weakly-reduced-imp-short-vector*: **assumes** *weakly-reduced* α *m vs*
and *in-L*: $h \in$ *lattice-of fs* $- \{0_v\ n\}$ **and** α*-pos*:$\alpha \geq 1$
shows $fs \neq [] \wedge$ *sq-norm* (*fs* ! *0*) $\leq \alpha\hat{}(m-1) *$ *sq-norm h*
proof $-$
from *gram-schmidt-short-vector*[*OF in-L*] **obtain** *i* **where**
i: $i < m$ **and** *le*: *sq-norm* (*gso i*) $\leq$ *sq-norm h* **by** *auto*
have *small*: *sq-norm* (*fs* ! *0*) $\leq \alpha\hat{}i *$ *sq-norm* (*gso i*) **using** *i*
proof (*induct i*)
case *0*
show *?case* **unfolding** *fs0-gso0*[*OF 0*] **by** *auto*
next
case (*Suc i*)
hence *sq-norm* (*fs* ! *0*) $\leq \alpha\hat{}i *$ *sq-norm* (*gso i*) **by** *auto*
also have $\ldots \leq \alpha\hat{}i * (\alpha *$ (*sq-norm* (*gso* (*Suc i*))))
using *reduced-vs-E*[*OF assms*(*1*) *le-refl Suc*(*2*)] α*-pos* **by** *auto*
finally show *?case* **unfolding** *class-semiring.nat-pow-Suc*[*of* α *i*] **by** *auto*
qed
also have $\ldots \leq \alpha\hat{}(m-1) *$ *sq-norm h*
by (*rule mult-mono*[*OF power-increasing le*], *insert i* α*-pos*, *auto*)
finally show *?thesis* **using** *i assm* **by** (*cases fs*, *auto*)
qed

lemma *sq-norm-pos*: **assumes** *j*: $j < m$
shows *sq-norm* (*vs* ! *j*) > 0
proof $-$
have *len-vs*: *length vs* $= m$ **using** *main-connect*(*2*) **by** *simp*
have *corthogonal vs* **by** (*rule gram-schmidt*)
from *corthogonalD*[*OF this*, *unfolded len-vs*, *OF j j*]
have *sq-norm* (*vs* ! *j*) $\neq 0$ **by** (*simp add*: *sq-norm-vec-as-cscalar-prod*)
moreover have *sq-norm* (*vs* ! *j*) ≥ 0 **by** *auto*
ultimately show $0 <$ *sq-norm* (*vs* ! *j*) **by** *auto*
qed

lemma *Gramian-determinant*: **assumes** *k*: $k \leq m$
shows *Gramian-determinant fs k* $= (\prod\ j<k.$ *sq-norm* (*vs* ! *j*))
Gramian-determinant fs k > 0
proof $-$
define *Gk* **where** $Gk = mat\ k\ n\ (\lambda\ (i,j).\ fs\ !\ i\ \$\ j)$
have *Gk*: $Gk \in$ *carrier-mat k n* **unfolding** *Gk-def* **by** *auto*
define *Mk* **where** $Mk = mat\ k\ k\ (\lambda\ (i,j).\ \mu\ i\ j)$

```
  have Mk-μ: i < k ⟹ j < k ⟹ Mk $$ (i,j) = μ i j for i j
    unfolding Mk-def using k by auto
  have Mk: Mk ∈ carrier-mat k k and [simp]: dim-row Mk = k dim-col Mk = k
unfolding Mk-def by auto
  have det Mk = prod-list (diag-mat Mk)
    by (rule det-lower-triangular[OF - Mk], auto simp: Mk-μ μ.simps)
  also have ... = 1
    by (rule prod-list-neutral, auto simp: diag-mat-def Mk-μ μ.simps)
  finally have detMk: det Mk = 1 .
  define Gsk where Gsk = mat k n (λ (i,j). vs ! i $ j)
  have Gsk: Gsk ∈ carrier-mat k n unfolding Gsk-def by auto
  have Gsk': Gsk^T ∈ carrier-mat n k using Gsk by auto
  have len-vs: length vs = m using main-connect(2) by simp
  let ?Rn = carrier-vec n
  have id: Gk = Mk * Gsk
  proof (rule eq-matI)
    from Gk Mk Gsk
    have dim: dim-row Gk = k dim-row (Mk * Gsk) = k dim-col Gk = n dim-col
(Mk * Gsk) = n by auto
     from dim show dim-row Gk = dim-row (Mk * Gsk) dim-col Gk = dim-col
(Mk * Gsk) by auto
    fix i j
    assume i < dim-row (Mk * Gsk) j < dim-col (Mk * Gsk)
    hence ij: i < k j < n and i: i < m using dim k by auto
    have Gi: fs ! i ∈ ?Rn using i by simp
    have Gk $$ (i, j) = fs ! i $ j unfolding Gk-def using ij k Gi by auto
    also have ... = FF $$ (i,j) using ij i by simp
    also have FF = M * Fs by (rule matrix-equality)
    also have ... $$ (i,j) = row M i · col Fs j
      by (rule index-mult-mat(1), insert i ij, auto simp: mat-of-rows-list-def)
    also have row M i = vec m (λ j. if j < k then Mk $$ (i,j) else 0)
      (is - = vec m ?Mk)
      unfolding Mk-def using ij i
      by (auto simp: mat-of-rows-list-def μ.simps)
   also have col Fs j = vec m (λ i'. if i' < k then Gsk $$ (i',j) else (Fs $$ (i',j)))

      (is - = vec m ?Gsk)
      unfolding Gsk-def using ij i len-vs by (auto simp: mat-of-rows-def)
    also have vec m ?Mk · vec m ?Gsk = (∑ i ∈ {0 ..< m}. ?Mk i * ?Gsk i)
      unfolding scalar-prod-def by auto
    also have ... = (∑ i ∈ {0 ..< k} ∪ {k ..< m}. ?Mk i * ?Gsk i)
      by (rule sum.cong, insert k, auto)
     also have ... = (∑ i ∈ {0 ..< k}. ?Mk i * ?Gsk i) + (∑ i ∈ {k ..< m}.
?Mk i * ?Gsk i)
      by (rule sum.union-disjoint, auto)
    also have (∑ i ∈ {k ..< m}. ?Mk i * ?Gsk i) = 0
      by (rule sum.neutral, auto)
    also have (∑ i ∈ {0 ..< k}. ?Mk i * ?Gsk i) = (∑ i' ∈ {0 ..< k}. Mk $$
(i,i') * Gsk $$ (i',j))
```

by (*rule sum.cong*, *auto*)
also have ... = *row Mk i* · *col Gsk j* **unfolding** *scalar-prod-def* **using** *ij*
by (*auto simp*: *Gsk-def Mk-def*)
also have ... = (*Mk* ∗ *Gsk*) $$ (*i*, *j*) **using** *ij Mk Gsk* **by** *simp*
finally show *Gk* $$ (*i*, *j*) = (*Mk* ∗ *Gsk*) $$ (*i*, *j*) **by** *simp*
qed
have *cong*: ⋀ *a b c d*. *a* = *b* ⟹ *c* = *d* ⟹ *a* ∗ *c* = *b* ∗ *d* **by** *auto*
have *Gramian-determinant fs k* = *det* (*Gk* ∗ Gk^T)
unfolding *Gramian-determinant-def Gramian-matrix-def Let-def*
by (*rule arg-cong*[*of - - det*], *rule cong*, *insert k*, *auto simp*: *Gk-def assm*(*3*))
also have Gk^T = Gsk^T ∗ Mk^T (**is** - = *?TGsk* ∗ *?TMk*) **unfolding** *id*
by (*rule transpose-mult*[*OF Mk Gsk*])
also have *Gk* = *Mk* ∗ *Gsk* **by** *fact*
also have ... ∗ (*?TGsk* ∗ *?TMk*) = *Mk* ∗ (*Gsk* ∗ (*?TGsk* ∗ *?TMk*))
by (*rule assoc-mult-mat*[*OF Mk Gsk*, *of - k*], *insert Gsk Mk*, *auto*)
also have *det* ... = *det Mk* ∗ *det* (*Gsk* ∗ (*?TGsk* ∗ *?TMk*))
by (*rule det-mult*[*OF Mk*], *insert Gsk Mk*, *auto*)
also have ... = *det* (*Gsk* ∗ (*?TGsk* ∗ *?TMk*)) **using** *detMk* **by** *simp*
also have *Gsk* ∗ (*?TGsk* ∗ *?TMk*) = (*Gsk* ∗ *?TGsk*) ∗ *?TMk*
by (*rule assoc-mult-mat*[*symmetric*, *OF Gsk*], *insert Gsk Mk*, *auto*)
also have *det* ... = *det* (*Gsk* ∗ *?TGsk*) ∗ *det ?TMk*
by (*rule det-mult*, *insert Gsk Mk*, *auto*)
also have ... = *det* (*Gsk* ∗ *?TGsk*) **using** *detMk det-transpose*[*OF Mk*] **by** *simp*
also have *Gsk* ∗ *?TGsk* = *mat k k* (λ (*i*,*j*). *if i* = *j then sq-norm* (*vs* ! *j*) *else* *0*) (**is** - = *?M*)
proof (*rule eq-matI*)
show *dim-row* (*Gsk* ∗ *?TGsk*) = *dim-row ?M* **unfolding** *Gsk-def* **by** *auto*
show *dim-col* (*Gsk* ∗ *?TGsk*) = *dim-col ?M* **unfolding** *Gsk-def* **by** *auto*
fix *i j*
assume *i* < *dim-row ?M j* < *dim-col ?M*
hence *ij*: *i* < *k j* < *k* **and** *ijn*: *i* < *m j* < *m* **using** *k* **by** *auto*
{
fix *i*
assume *i* < *k*
hence *i* < *m* **using** *k* **by** *auto*
hence *Gs*: *vs* ! *i* ∈ *?Rn* **using** *len-vs vs*(*1*) **by** *auto*
have *row Gsk i* = *vs* ! *i* **unfolding** *row-def Gsk-def*
by (*rule eq-vecI*, *insert Gs* ‹*i* < *k*›, *auto*)
} **note** *row* = *this*
have (*Gsk* ∗ *?TGsk*) $$ (*i*,*j*) = *row Gsk i* · *row Gsk j* **using** *ij Gsk* **by** *auto*
also have ... = *vs* ! *i* ·c *vs* ! *j* **using** *row ij* **by** *simp*
also have ... = (*if i* = *j then sq-norm* (*vs* ! *j*) *else 0*)
proof (*cases i* = *j*)
assume *i* = *j*
thus *?thesis* **by** (*simp add*: *sq-norm-vec-as-cscalar-prod*)
next
assume *i* ≠ *j*
have *corthogonal vs* **by** (*rule gram-schmidt*)
from ‹*i* ≠ *j*› *corthogonalD*[*OF this*, *unfolded len-vs*, *OF ijn*]

show *?thesis* **by** *auto*
qed
also have ... = *?M* $$ (i,j) **using** *ij* **by** *simp*
finally show (*Gsk* ∗ *?TGsk*) $$ (i,j) = *?M* $$ (i,j) **.**
qed
also have *det ?M* = *prod-list* (*diag-mat ?M*)
by (*rule det-upper-triangular*, *auto*)
also have *diag-mat ?M* = *map* (λ *j*. *sq-norm* (*vs* ! *j*)) [*0* ..< *k*] **unfolding** *diag-mat-def* **by** *auto*
also have *prod-list* ... = ($\prod$ *j* < *k*. *sq-norm* (*vs* ! *j*))
by (*subst prod.distinct-set-conv-list*[*symmetric*], *force*, *rule prod.cong*, *auto*)
finally show *Gramian-determinant fs k* = ($\prod j<k.\ \|vs\ !\ j\|^2$) **.**
also have ... > *0*
by (*rule prod-pos*, *intro ballI sq-norm-pos*, *insert k*, *auto*)
finally show *0* < *Gramian-determinant fs k* **by** *auto*
qed
end
end
end
end

lemma *prod-list-le-mono*: **fixes** *us* :: *'a* :: {*linordered-nonzero-semiring*,*ordered-ring*} *list*
assumes *length us* = *length vs*
and ⋀ *i*. *i* < *length vs* ⟹ *0* ≤ *us* ! *i* ∧ *us* ! *i* ≤ *vs* ! *i*
shows *0* ≤ *prod-list us* ∧ *prod-list us* ≤ *prod-list vs*
using *assms*
proof (*induction us vs rule*: *list-induct2*)
case (*Cons u us v vs*)
have *0* ≤ *prod-list us* ∧ *prod-list us* ≤ *prod-list vs*
by (*rule Cons.IH*, *insert Cons.prems*[*of Suc i* **for** *i*], *auto*)
moreover have *0* ≤ *u* ∧ *u* ≤ *v* **using** *Cons.prems*[*of 0*] **by** *auto*
ultimately show *?case* **by** (*auto intro*: *mult-mono*)
qed *simp*

lemma *lattice-of-of-int*: **assumes** *G*: *set F* ⊆ *carrier-vec n*
and *f* ∈ *vec-module.lattice-of n F*
shows *map-vec rat-of-int f* ∈ *vec-module.lattice-of n* (*map* (*map-vec of-int*) *F*)
(**is** *?f* ∈ *vec-module.lattice-of* - *?F*)
proof −
let *?sl* = *abelian-monoid.sumlist* (*module-vec TYPE*(*'a*::*semiring-1*) *n*)
note *d* = *vec-module.lattice-of-def*
note *dim* = *vec-module.sumlist-dim*
note *sumlist-vec-index* = *vec-module.sumlist-vec-index*
from *G* **have** *Gi*: ⋀ *i*. *i* < *length F* ⟹ *F* ! *i* ∈ *carrier-vec n* **by** *auto*
from *Gi* **have** *Gid*: ⋀ *i*. *i* < *length F* ⟹ *dim-vec* (*F* ! *i*) = *n* **by** *auto*
from *assms*(*2*)[*unfolded d*]
obtain *c* **where**
ffc: *f* = *?sl* (*map* (λ*i*. *of-int* (*c i*) $\cdot_v$ *F* ! *i*) [*0*..<*length F*]) (**is** - = *?g*) **by** *auto*

```
  have ?f = ?sl (map (λi. of-int (c i) ·v ?F ! i) [0..<length ?F]) (is - = ?gg)
  proof −
    have d1[simp]: dim-vec ?g = n by (subst dim, auto simp: Gi)
    have d2[simp]: dim-vec ?gg = n unfolding length-map by (subst vec-module.sumlist-dim,
auto simp: Gi G)
    show ?thesis
      unfolding ffc length-map
      apply (rule eq-vecI)
       apply (insert d1 d2, auto)[2]
      apply (subst (1 2) sumlist-vec-index, auto simp: o-def Gi G)
      apply (unfold of-int-hom.hom-sum-list)
      apply (intro arg-cong[of - - sum-list] map-cong)
        by (auto simp: G Gi, (subst index-smult-vec, simp add: Gid)+,
         subst index-map-vec, auto simp: Gid)
  qed
  thus ?f ∈ vec-module.lattice-of n ?F unfolding d by auto
qed


lemma Hadamard's-inequality:
  fixes A::real mat
  assumes A: A ∈ carrier-mat n n
  shows  abs (det A) ≤ sqrt (prod-list (map sq-norm (rows A)))
proof −
  interpret gso: gram-schmidt n TYPE(real) .
  let ?us = map (row A) [0 ..< n]
  have len: length ?us = n by simp
  have us: set ?us ⊆ carrier-vec n using A by auto
  obtain vs where main: snd (gso.main ?us) = vs by force
  show ?thesis
  proof (cases carrier-vec n ⊆ gso.span (set ?us))
    case True
    with us len have basis: gso.basis-list ?us unfolding gso.basis-list-def by auto
    note conn = gso.basis-list-imp-lin-indpt-list[OF basis] len main
    note gram = gso.gram-schmidt[OF conn]
    note main = gso.main-connect[OF conn]
    from main have len-vs: length vs = n by simp
     have last: 0 ≤ prod-list (map sq-norm vs) ∧ prod-list (map sq-norm vs) ≤
prod-list (map sq-norm ?us)
    proof (rule prod-list-le-mono, force simp: main(2), unfold length-map length-upt)
      fix i
      assume i < n − 0
      hence i: i < n by simp
      have vsi: map sq-norm vs ! i = sq-norm (vs ! i) using main(2) i by simp
      have usi: map sq-norm ?us ! i = sq-norm (row A i) using i by simp
      have zero: 0 ≤ sq-norm (vs ! i) by auto
     have le: sq-norm (vs ! i) ≤ sq-norm (row A i) using gso.sq-norm-gso-le-f[OF
```

conn i]
unfolding *main*(*2*) **using** *i* **by** *simp*
show $0 \leq$ *map sq-norm vs* ! *i* $\wedge$ *map sq-norm vs* ! *i* $\leq$ *map sq-norm ?us* ! *i*
unfolding *vsi usi* **using** *zero le* **by** *auto*
qed
have *Fs*: *gso.Fs ?us* $\in$ *carrier-mat n n* **by** *auto*
have *A-Fs*: *A* = *gso.Fs ?us*
by (*rule eq-matI*, *subst gso.FF-index*[*OF conn*], *insert A*, *auto*)
hence *abs* (*det A*) = *abs* (*det* (*gso.Fs ?us*)) **by** *simp*

also have . . . = *abs* (*sqrt* (*det* (*gso.Fs ?us*) ∗ *det* (*gso.Fs ?us*))) **by** *simp*
also have *det* (*gso.Fs ?us*) ∗ *det* (*gso.Fs ?us*) = *det* (*gso.Fs ?us*) ∗ *det* (*gso.Fs ?us*)^T
unfolding *det-transpose*[*OF Fs*] **..**
also have . . . = *det* (*gso.Fs ?us* ∗ (*gso.Fs ?us*)^T)
by (*subst det-mult*[*OF Fs*], *insert Fs*, *auto*)
also have . . . = *gso.Gramian-determinant ?us n*
unfolding *gso.Gramian-matrix-def gso.Gramian-determinant-def Let-def A-Fs*[*symmetric*]
by (*rule arg-cong*[*of - - det*], *rule arg-cong2*[*of - - - - op* ∗], *insert A*, *auto*)
also have . . . = ($\prod$ *j* $\in$ *set* [*0* ..< *n*]. $\|$*vs* ! *j*$\|^2$) **unfolding** *gso.Gramian-determinant*[*OF conn le-refl*]
by (*rule prod.cong*, *auto*)
also have . . . = *prod-list* (*map* (λ *i*. *sq-norm* (*vs* ! *i*)) [*0* ..< *n*])
by (*subst prod.distinct-set-conv-list*, *auto*)
also have *map* (λ *i*. *sq-norm* (*vs* ! *i*)) [*0* ..< *n*] = *map sq-norm vs*
using *len-vs* **by** (*intro nth-equalityI*, *auto*)
also have *abs* (*sqrt* (*prod-list* . . .)) $\leq$ *sqrt* (*prod-list* (*map sq-norm ?us*))
using *last* **by** *simp*
also have *?us* = *rows A* **unfolding** *rows-def* **using** *A* **by** *simp*
finally show *?thesis* **.**
next
case *False*
from *mat-of-rows-rows*[*unfolded rows-def*,*of A*] *A gram-schmidt.non-span-det-zero*[*OF len False us*]
have *zero*: *det A* = *0* **by** *auto*
have *ge*: *prod-list* (*map sq-norm* (*rows A*)) $\geq$ *0*
by (*rule prod-list-nonneg*, *auto simp*: *sq-norm-vec-ge-0*)
show *?thesis* **unfolding** *zero* **using** *ge* **by** *simp*
qed
qed

definition *gram-schmidt-wit* = *gram-schmidt.main*
lemmas *gram-schmidt-wit* = *gram-schmidt.weakly-reduced-imp-short-vector*[*folded gram-schmidt-wit-def*]
declare *gram-schmidt.adjuster-wit.simps*[*code*]
declare *gram-schmidt.sub2-wit.simps*[*code*]
declare *gram-schmidt.main-def*[*code*]

definition *gram-schmidt-int* :: *nat* ⇒ *int vec list* ⇒ *rat list list* × *rat vec list*
where
gram-schmidt-int n us = *gram-schmidt-wit n* (*map* (*map-vec of-int*) *us*)

lemma *snd-gram-schmidt-int* : *snd* (*gram-schmidt-int n us*) = *gram-schmidt n* (*map* (*map-vec of-int*) *us*)
unfolding *gram-schmidt-int-def gram-schmidt-wit-def gram-schmidt.gso-connect*
by *metis*

fun *adjuster-triv* :: *nat* ⇒ *'a* :: *trivial-conjugatable-ordered-field vec* ⇒ (*'a vec* × *'a*) *list* ⇒ *'a vec*
where *adjuster-triv n w* [] = 0_v *n*
| *adjuster-triv n w* ((*u*,*nu*)#*us*) = −(*w* · *u*)/ *nu* $\cdot_v$ *u* + *adjuster-triv n w us*

fun *gram-schmidt-sub-triv*
where *gram-schmidt-sub-triv n us* [] = *us*
| *gram-schmidt-sub-triv n us* (*w* # *ws*) = (*let u* = *adjuster-triv n w us* + *w in*
gram-schmidt-sub-triv n ((*u*, *sq-norm u*) # *us*) *ws*)

definition *gram-schmidt-triv* :: *nat* ⇒ *'a* :: *trivial-conjugatable-ordered-field vec list* ⇒ (*'a vec* × *'a*) *list*
where *gram-schmidt-triv n ws* = *rev* (*gram-schmidt-sub-triv n* [] *ws*)

lemma *adjuster-triv*: *adjuster-triv n w* (*map* (λ *x*. (*x*,*sq-norm x*)) *us*) = *adjuster n w us*
by (*induct us*, *auto simp*: *sq-norm-vec-as-cscalar-prod*)

lemma *gram-schmidt-sub-triv*: *gram-schmidt-sub-triv n* ((*map* (λ *x*. (*x*,*sq-norm x*)) *us*)) *ws* =
map (λ *x*. (*x*, *sq-norm x*)) (*gram-schmidt-sub n us ws*)
by (*rule sym*, *induct ws arbitrary*: *us*, *auto simp*: *adjuster-triv o-def Let-def*)

lemma *gram-schmidt-triv*[*simp*]: *gram-schmidt-triv n ws* = *map* (λ *x*. (*x*,*sq-norm x*)) (*gram-schmidt n ws*)
unfolding *gram-schmidt-def gram-schmidt-triv-def rev-map*[*symmetric*]
by (*auto simp*: *gram-schmidt-sub-triv*[*symmetric*])

end

7 The LLL algorithm

This theory provides an implementation and a soundness proof of the LLL algorithm to compute a "short" vector in a lattice.

theory *LLL*
imports
Gram-Schmidt-2

Missing-Lemmas
Determinant
SN-Order-Carrier
List-Representation
begin

7.1 Implementation of the LLL algorithm

definition *floor-ceil* **where** *floor-ceil x = floor (x + 1/2)*

definition *scalar-prod-int-rat :: int vec ⇒ rat vec ⇒ rat* (**infix** *·i 70*) **where**
x ·i y = (y · map-vec rat-of-int x)

type-synonym *f-repr = int vec list-repr*

type-synonym *g-repr = (rat vec × rat) list-repr*

definition *g-i :: g-repr ⇒ rat vec* **where** *g-i Gr = (fst (get-nth-i Gr))*
definition *sqnorm-g-i :: g-repr ⇒ rat* **where** *sqnorm-g-i Gr = (snd (get-nth-i Gr))*
definition *g-im1 :: g-repr ⇒ rat vec* **where** *g-im1 Gr = (fst (get-nth-im1 Gr))*
definition *sqnorm-g-im1 :: g-repr ⇒ rat* **where** *sqnorm-g-im1 Gr = (snd (get-nth-im1 Gr))*
definition *μ-i-im1 :: f-repr ⇒ g-repr ⇒ rat* **where**
μ-i-im1 Fr Gr = (get-nth-i Fr ·i g-im1 Gr) / sqnorm-g-im1 Gr

definition *μ-ij :: int vec ⇒ rat vec × rat ⇒ rat* **where**
μ-ij fi gj-norm = (case gj-norm of (gj,norm-gj) ⇒ (fi ·i gj) / norm-gj)

type-synonym *state = nat × f-repr × g-repr*

fun *basis-reduction-add-row-main :: state ⇒ int vec ⇒ rat ⇒ state × int* **where**
basis-reduction-add-row-main (i,F,G) fj mu = (let
c = floor-ceil mu
in if c = 0 then
((i,F,G), c)
else
let
fi = get-nth-i F − (c $\cdot_v$ fj);
F′ = update-i F fi
in ((i,F′,G), c))

definition *basis-reduction-add-row-i-im1 :: state ⇒ state × rat* **where**

basis-reduction-add-row-i-im1 state = (*case state of* (-,*F*,*G*) ⇒ *let*
mu = *μ-i-im1 F G*;
fj = *get-nth-im1 F*
in map-prod id (λ *c*. *mu* − *rat-of-int c*) (*basis-reduction-add-row-main state fj mu*))

definition *increase-i* :: *state* ⇒ *state* **where**
increase-i state = (*case state of* (*i*, *F*, *G*) ⇒ (*Suc i*, *inc-i F*, *inc-i G*))

fun *basis-reduction-swap* :: *state* ⇒ *rat* ⇒ *state* **where**
basis-reduction-swap (*i*,*F*,*G*) *mu* = (*let*
gi = *g-i G*;
gim1 = *g-im1 G*;
fi = *get-nth-i F*;
fim1 = *get-nth-im1 F*;
new-gim1 = *gi* + *mu* $\cdot_v$ *gim1*;
norm-gim1 = *sq-norm new-gim1*;
new-gi = *gim1* − (*fim1* •*i new-gim1* / *norm-gim1*) $\cdot_v$ *new-gim1*;
norm-gi = *sq-norm new-gi*;
G' = *dec-i* (*update-im1* (*update-i G* (*new-gi*,*norm-gi*)) (*new-gim1*,*norm-gim1*));
F' = *dec-i* (*update-im1* (*update-i F fim1*) *fi*)
in (*i* − *1*, *F'*, *G'*))

definition *basis-reduction-step* :: *rat* ⇒ *state* ⇒ *state* **where**
basis-reduction-step α *state* = (*if fst state* = *0 then increase-i state*
else case basis-reduction-add-row-i-im1 state of
(*state'*, *mu*) ⇒
case state' of (*i*, *F*, *G*) ⇒
if sqnorm-g-im1 G > α ∗ *sqnorm-g-i G*
then basis-reduction-swap state' mu
else increase-i state'
)

partial-function (*tailrec*) *basis-reduction-main* :: *rat* ⇒ *nat* ⇒ *state* ⇒ *state* **where**
[*code*]: *basis-reduction-main* α *m state* = (*case state of* (*i*,*F*,*G*) ⇒
if i < *m*
then basis-reduction-main α *m* (*basis-reduction-step* α *state*)
else state)

definition *basis-reduction-part-1* :: *nat* ⇒ *rat* ⇒ *int vec list* ⇒ *state* **where**
basis-reduction-part-1 n α *F* = (*let m* = *length F*;
G = *gram-schmidt-triv n* (*map* (*map-vec of-int*) *F*);
Fr = ([], *F*);
Gr = ([], *G*)
in basis-reduction-main α *m* (*0*, *Fr*, *Gr*))

definition *weakly-reduce-basis* :: *nat* ⇒ *rat* ⇒ *int vec list* ⇒ *int vec list* × *rat vec list* **where**

weakly-reduce-basis n α F = (λ *state*. ((*of-list-repr o fst o snd*) *state*,
(*map fst o of-list-repr o snd o snd*) *state*))
(*basis-reduction-part-1 n α F*)

definition *short-vector* :: *rat* ⇒ *int vec list* ⇒ *int vec* **where**
short-vector α F = (*hd o fst*) (*weakly-reduce-basis* (*dim-vec* (*hd F*)) *α F*)

fun *basis-reduction-add-row-i-all-main* :: *state* ⇒ *int vec list* ⇒ (*rat vec* × *rat*) *list* ⇒ *state* **where**
basis-reduction-add-row-i-all-main state (*Cons fj fjs*) (*Cons gj gjs*) = (*case state of* (*i*,*F*,*G*) ⇒
let fi = *get-nth-i F in*
basis-reduction-add-row-i-all-main (*fst* (*basis-reduction-add-row-main state fj* (*μ-ij fi gj*))) *fjs gjs*)
| *basis-reduction-add-row-i-all-main state* - - = *state*

definition *basis-reduction-add-row-i-all* :: *state* ⇒ *state* **where**
basis-reduction-add-row-i-all state = (*case state of* (*i*,*F*,*G*) ⇒
let fjs = *fst F*;
gjs = *fst G*
in basis-reduction-add-row-i-all-main state fjs gjs)

fun *basis-reduction-part-2-main* :: *state* ⇒ *state* **where**
basis-reduction-part-2-main (*i*,*F*,*G*) = (*if i* = *0 then* (*i*,*F*,*G*) *else*
case basis-reduction-add-row-i-all (*i* − *1*, *dec-i F*, *dec-i G*) *of*
(-, *F′*, *G′*) ⇒ *basis-reduction-part-2-main* (*i* − *1*, *F′*, *G′*))

definition *basis-reduction-part-2* :: *nat* ⇒ *rat* ⇒ *int vec list* ⇒ *state* **where**
basis-reduction-part-2 n α F = *basis-reduction-part-2-main*
(*basis-reduction-part-1 n α F*)

definition *strictly-reduce-basis* :: *nat* ⇒ *rat* ⇒ *int vec list* ⇒ *int vec list* × *rat vec list* **where**
strictly-reduce-basis n α F = (λ *state*. ((*of-list-repr o fst o snd*) *state*,
(*map fst o of-list-repr o snd o snd*) *state*))
(*basis-reduction-part-2 n α F*)

7.2 LLL algorithm is sound

lemma *floor-ceil*: |*x* − *rat-of-int* (*floor-ceil x*)| ≤ *inverse 2*
unfolding *floor-ceil-def* **by** (*metis* (*no-types*, *hide-lams*) *abs-divide abs-neg-one round-def*
div-by-1 div-minus-right inverse-eq-divide minus-diff-eq of-int-round-abs-le)

lemma *μ-i-im1-code*[*code-unfold*]:
μ-i-im1 F G = *μ-ij* (*get-nth-i F*) (*get-nth-im1 G*)
unfolding *μ-i-im1-def g-im1-def sqnorm-g-im1-def μ-ij-def*
by (*auto split*: *prod.splits*)

lemma *scalar-prod-int-rat*[*simp*]: *dim-vec x* = *dim-vec y* $\Longrightarrow$ *x* ·*i* *y* = *map-vec of-int x* · *y*
unfolding *scalar-prod-int-rat-def* **by** (*intro comm-scalar-prod*[*of - dim-vec x*], *auto intro*: *carrier-vecI*)

definition *int-times-rat* :: *int* ⇒ *rat* ⇒ *rat* **where** *int-times-rat i x* = *of-int i* ∗ *x*

lemma *scalar-prod-int-rat-code*[*code*]: *v* ·*i* *w* = ($\sum$ *i* = *0..*<*dim-vec v*. *int-times-rat* (*v* \$ *i*) (*w* \$ *i*))
unfolding *scalar-prod-int-rat-def Let-def scalar-prod-def int-times-rat-def*
by (*rule sum.cong*, *auto*)

lemma *int-times-rat-code*[*code abstract*]: *quotient-of* (*int-times-rat i x*) =
(*case quotient-of x of* (*n*,*d*) ⇒ *Rat.normalize* (*i* ∗ *n*, *d*))
unfolding *int-times-rat-def rat-times-code* **by** *auto*

locale *LLL* =
fixes *n* :: *nat* **and** *m* :: *nat*
begin

sublocale *vec-module TYPE*(*int*) *n*.

sublocale *gs*: *gram-schmidt n TYPE*(*rat*) **.**

abbreviation *RAT* **where** *RAT* ≡ *map* (*map-vec rat-of-int*)
abbreviation *SRAT* **where** *SRAT xs* ≡ *set* (*RAT xs*)
abbreviation *Rn* **where** *Rn* ≡ *carrier-vec n* :: *rat vec set*

definition *GSO* **where** *GSO F* = *gs.gso* (*RAT F*)

definition *g-repr* :: *nat* ⇒ *g-repr* ⇒ *int vec list* ⇒ *bool* **where**
g-repr i G F = (*i* ≤ *m* ∧ *list-repr i G* (*map* (λ *x*. (*x*,*sq-norm x*)) (*map* (*GSO F*) [*0..*<*m*])))

abbreviation (*input*) *f-repr* ≡ *list-repr*

lemma *g-i-GSO*: *g-repr i G F* $\Longrightarrow$ *i* < *m* $\Longrightarrow$ *g-i G* = *GSO F i*
unfolding *g-repr-def g-i-def* **by** (*cases F*, *auto simp*: *get-nth-i*)

lemma *sqnorm-g-i-GSO*: *g-repr i G F* $\Longrightarrow$ *i* < *m* $\Longrightarrow$ *sqnorm-g-i G* = *sq-norm* (*GSO F i*)
unfolding *g-repr-def sqnorm-g-i-def* **by** (*cases G*, *auto simp*: *get-nth-i*)

lemma *g-im1-GSO*: *g-repr i G F* $\Longrightarrow$ *i* ≠ *0* $\Longrightarrow$ *g-im1 G* = *GSO F* (*i* − *1*)
unfolding *g-repr-def g-im1-def* **by** (*cases G*, *cases i*, *auto simp*: *get-nth-im1*)

lemma *sqnorm-g-im1-GSO*: *g-repr i G F* $\Longrightarrow$ *i* ≠ *0* $\Longrightarrow$ *sqnorm-g-im1 G* =

```
sq-norm (GSO F (i − 1))
 unfolding g-repr-def sqnorm-g-im1-def by (cases G, cases i, auto simp: get-nth-im1)

lemma inc-i-gso: assumes i < m g-repr i G F
shows g-repr (Suc i) (inc-i G) F
  using assms unfolding g-repr-def by (auto simp: inc-i)

lemma μ-i-im1: assumes gr: f-repr i Fr F and gso:g-repr i G F
  and n:m = length F and i:i ≠ 0 i < m
  and dim: F ! i ∈ carrier-vec n gs.gso (RAT F) (i − 1) ∈ carrier-vec n
  shows μ-i-im1 Fr G = gs.μ (RAT F) i (i − 1)
  unfolding g-repr-def μ-i-im1-def gs.μ.simps
           get-nth-i[OF gr,unfolded length-map,OF i(2)[unfolded n]]
           of-list-repr[OF gr] g-im1-GSO[OF gso i(1)]
           sqnorm-g-im1-GSO[OF gso i(1)] GSO-def
  using i n dim by auto

context fixes L :: int vec set and α :: rat
begin

definition LLL-invariant :: state ⇒ int vec list ⇒ rat vec list ⇒ bool where
  LLL-invariant state F G = (case state of (i,Fr,Gr) ⇒
  snd (gram-schmidt-int n F) = G ∧
  gs.lin-indpt-list (RAT F) ∧
  lattice-of F = L ∧
  gs.weakly-reduced α i G ∧
  i ≤ m ∧
  length F = m ∧
  f-repr i Fr F ∧
  g-repr i Gr F
  )

lemma LLL-invD: assumes LLL-invariant (i,Fr,Gr) F G
  shows F = of-list-repr Fr
  snd (gram-schmidt-int n F) = G
  set F ⊆ carrier-vec n
  length F = m
  lattice-of F = L
  gs.weakly-reduced α i G
  i ≤ m
  f-repr i Fr F
  g-repr i Gr F
  gs.lin-indpt-list (RAT F)
 using assms gs.lin-indpt-list-def of-list-repr[of i Fr F] unfolding LLL-invariant-def
split by auto

lemma LLL-invI: assumes
  f-repr i Fr F
  g-repr i Gr F
```

snd (*gram-schmidt-int n F*) = *G*
lattice-of F = *L*
gs.weakly-reduced α *i G*
$i \leq m$
length F = *m*
gs.lin-indpt-list (*RAT F*)
shows *LLL-invariant* (*i*,*Fr*,*Gr*) *F G*
unfolding *LLL-invariant-def Let-def split* **using** *assms of-list-repr*[*OF assms*(*1*)]
by *auto*

lemma *gram-schmidt-int-connect*: **fixes** *F* :: *int vec list*
assumes *gs.lin-indpt-list* (*RAT F*) *snd* (*gram-schmidt-int n F*) = *G length F* = *m*
shows *G* = *map* (*gs.gso* (*RAT F*)) [*0*..<*m*]
proof −
from *assms* **have** *gsw*: *snd* (*gram-schmidt-wit n* (*RAT F*)) = *G*
by (*auto simp*: *gram-schmidt-int-def*)
from *gram-schmidt.main-connect*[*OF assms*(*1*) - *gsw*[*unfolded gram-schmidt-wit-def*], *of m*] *assms*(*3*)
show *G* = *map* (*gs.gso* (*RAT F*)) [*0*..<*m*] **by** *auto*
qed

lemma *LLL-connect*: **fixes** *F* :: *int vec list*
assumes *inv*: *LLL-invariant* (*i*,*Fr*,*Gr*) *F G*
shows *G* = *map* (*gs.gso* (*RAT F*)) [*0*..<*m*]
using *gram-schmidt-int-connect*[*of F G*] *LLL-invD*[*OF inv*] **by** *auto*

lemma *gs-gs-identical*: **assumes** $\bigwedge$ *i*. $i \leq x \Longrightarrow$ *f1* ! *i* = *f2* ! *i*
shows *gs.gso f1 x* = *gs.gso f2 x*
using *assms*
proof(*induct x rule*:*nat-less-induct*[*rule-format*])
case (*1 x*)
hence *fg*:*op* + (*f1* ! *x*) = *op* + (*f2* ! *x*) **by** *auto*
show *?case*
apply(*subst* (*1 2*) *gs.gso.simps*) **unfolding** *gs.μ.simps*
apply(*rule cong*[*OF fg cong*[*OF refl*[*of gs.sumlist*]]])
using *1* **by** *auto*
qed

lemma *gs-μ-identical*: **assumes** $\bigwedge$ *k*. $j < i \Longrightarrow k \leq j \Longrightarrow$ *f1* ! *k* = *f2* ! *k*
and $j < i \Longrightarrow$ *f1* ! *i* = *f2* ! *i*
shows *gs.μ f1 i j* = *gs.μ f2 i j*
proof −
from *gs-gs-identical*[*of j f1 f2*] *assms* **have** *id*: $j < i \Longrightarrow$ *gs.gso f1 j* = *gs.gso f2 j* **by** *auto*
show *?thesis* **unfolding** *gs.μ.simps* **using** *assms id* **by** *auto*
qed

lemma *g-i*: **assumes** *inv*: *LLL-invariant* (*i*,*Fr*,*Gr*) *F G* **and** *i*: *i* < *m*
shows *g-i Gr* = *G* ! *i*
sqnorm-g-i Gr = *sq-norm* (*G* ! *i*)
proof −
note *conn* = *LLL-connect*[*OF inv*]
note *inv* = *LLL-invD*[*OF inv*]
note *conn* = *conn*[*folded inv*(*1*)]
from *inv i* **have** *Gr*: *g-repr i Gr F*
and *len*: *length F* = *m* **by** *auto*
from *g-i-GSO*[*OF Gr i*] *sqnorm-g-i-GSO*[*OF Gr i*]
have *g-i Gr* = *GSO F i* ∧ *sqnorm-g-i Gr* = *sq-norm* (*GSO F i*) **by** *simp*
also have *GSO F i* = *G* ! *i* **unfolding** *GSO-def conn* **using** *i len* **by** *simp*
finally show *g-i Gr* = *G* ! *i*
sqnorm-g-i Gr = *sq-norm* (*G* ! *i*) **by** *auto*
qed

lemma *g-im1*: **assumes** *inv*: *LLL-invariant* (*i*,*Fr*,*Gr*) *F G* **and** *i*: *i* < *m i* ≠ *0*
shows *g-im1 Gr* = *G* ! (*i* − *1*)
sqnorm-g-im1 Gr = *sq-norm* (*G* ! (*i* − *1*))
proof −
note *conn* = *LLL-connect*[*OF inv*]
note *inv* = *LLL-invD*[*OF inv*]
note *conn* = *conn*[*folded inv*(*1*)]
from *inv i* **have** *Gr*: *g-repr i Gr F*
and *len*: *length F* = *m* **by** *auto*
from *g-im1-GSO*[*OF Gr i*(*2*)] *sqnorm-g-im1-GSO*[*OF Gr i*(*2*)]
have *g-im1 Gr* = *GSO F* (*i* − *1*) ∧ *sqnorm-g-im1 Gr* = *sq-norm* (*GSO F* (*i* − *1*)) **by** *simp*
also have *GSO F* (*i* − *1*) = *G* ! (*i* − *1*) **unfolding** *GSO-def conn* **using** *i len* **by** *simp*
finally show *g-im1 Gr* = *G* ! (*i* − *1*)
sqnorm-g-im1 Gr = *sq-norm* (*G* ! (*i* − *1*)) **by** *auto*
qed

definition *reduction* **where** *reduction* = (*4*+α)/(*4*∗α)

definition *dk* :: *nat* ⇒ *int vec list* ⇒ *int* **where** *dk k fs* = (*gs.Gramian-determinant fs k*)

definition *D* :: *int vec list* ⇒ *nat* **where** *D fs* = *nat* (∏ *i* < *m*. *dk i fs*)

definition *logD* :: *int vec list* ⇒ *nat*
where *logD fs* = (*if* α = *4*/*3* *then* (*D fs*) *else nat* (*floor* (*log* (*1* / *of-rat reduction*) (*D fs*))))

definition *LLL-measure* :: *state* ⇒ *nat* **where**
LLL-measure state = (*case state of* (*i*,*fs*,*gs*) ⇒ *2* ∗ *logD* (*of-list-repr fs*) + *m* − *i*)

lemma *Gramian-determinant*: **assumes** *LLL-invariant* (*i*,*Fr*,*Gr*) *F G*
and *k*: $k \leq m$
shows *of-int* (*gs.Gramian-determinant F k*) = ($\prod$ *j*<*k*. *sq-norm* (*G* ! *j*))
gs.Gramian-determinant F k > *0*
proof −
let *?f* = (λ*i*. *of-int-hom.vec-hom* (*F* ! *i*))
note *LLL* = *LLL-connect*[*OF assms*(*1*)]
note *LLLD* = *LLL-invD*[*OF assms*(*1*)]
let *?F* = *map of-int-hom.vec-hom F*
from *LLL* **have** *lenGs*: *length G* = *m* **by** *auto*
from *LLLD*(*2*−)[*unfolded gram-schmidt-int-def gram-schmidt-wit-def*]
have *main*: *snd* (*gs.main ?F*) = *G* **and** *len*: *length ?F* = *m* **and** *F*: *set ?F* ⊆ *Rn*
and *indep*: *gs.lin-indpt-list ?F* **by** (*auto intro*: *nth-equalityI*)
note *conn* = *indep len main*
have *Fi*: $\bigwedge$ *i*. *i* < *m* $\Longrightarrow$ *F* ! *i* ∈ *carrier-vec n* **using** *len F* **unfolding** *set-conv-nth* **by** *auto*
have *det*: *gs.Gramian-determinant ?F k* = ($\prod$*j*<*k*. ‖*G* ! *j*‖^2) (*0* :: *rat*) < *gs.Gramian-determinant ?F k*
using *gs.Gramian-determinant*[*OF conn k*] **by** *auto*
have *hom*: *gs.Gramian-determinant ?F k* = *of-int* (*gs.Gramian-determinant F k*)
unfolding *gs.Gramian-determinant-def of-int-hom.hom-det*[*symmetric*]
proof (*rule arg-cong*[*of - - det*])
have *cong*: $\bigwedge$ *a b c d*. *a* = *b* $\Longrightarrow$ *c* = *d* $\Longrightarrow$ *a* ∗ *c* = *b* ∗ *d* **by** *auto*
show *gs.Gramian-matrix ?F k* = *map-mat of-int* (*gs.Gramian-matrix F k*)
unfolding *gs.Gramian-matrix-def Let-def*
proof (*subst of-int-hom.mat-hom-mult*[*of - k n - k*], (*auto*)[*2*], *rule cong*)
show *id*: *mat k n* (λ (*i*,*j*). *?F* ! *i* $ *j*) = *map-mat of-int* (*mat k n* (λ (*i*, *j*). *F* ! *i* $ *j*)) (**is** *?L* = *map-mat - ?R*)
proof (*rule eq-matI*, *goal-cases*)
case (*1 i j*)
hence *ij*: *i* < *k j* < *n i* < *length F dim-vec* (*F* ! *i*) = *n* **using** *len k Fi*[*of i*] **by** *auto*
show *?case* **using** *ij* **by** *simp*
qed *auto*
show *?L*^{T} = *map-mat of-int ?R*^{T} **unfolding** *id* **by** (*rule eq-matI*, *auto*)
qed
qed
show *of-int* (*gs.Gramian-determinant F k*) = ($\prod$ *j*<*k*. *sq-norm* (*G* ! *j*))
gs.Gramian-determinant F k > (*0* :: *int*) **using** *det*[*unfolded hom*] **by** *auto*
qed

lemma *LLL-dk-pos* [*intro*]: **assumes** *inv*: *LLL-invariant state F G*
and *k*: $k \leq m$
shows *dk k F* > *0*
proof −
obtain *i Gr gso* **where** *trip*: *state* = (*i*, *Gr*, *gso*) **by** (*cases state*, *auto*)

note *inv* = *inv*[*unfolded trip*]
from *Gramian-determinant*[*OF inv k*] **show** *?thesis* **unfolding** *trip dk-def* **by** *auto*
qed

lemma *LLL-D-pos*: **assumes** *inv*: *LLL-invariant state F G*
shows *D F* > *0*
proof −
have ($\prod$ *j* < *m*. *dk j F*) > *0*
by (*rule prod-pos*, *insert LLL-dk-pos*[*OF inv*], *auto*)
thus *?thesis* **unfolding** *D-def* **by** *auto*
qed

lemma *increase-i*: **assumes** *LLL*: *LLL-invariant* (*i*,*Fr*,*Gr*) *F G*
and *i*: *i* < *m*
and *red-i*: *i* ≠ *0* ⟹ *sq-norm* (*G* ! (*i* − *1*)) ≤ α ∗ *sq-norm* (*G* ! *i*)
shows *LLL-invariant* (*increase-i* (*i*,*Fr*,*Gr*)) *F G*
proof −
note *inv* = *LLL-invD*[*OF LLL*]
from *inv* **have** *Gr*: *g-repr i Gr F* **and** *Fr*: *f-repr i Fr F*
and *red*: *gs.weakly-reduced* α *i G* **by** *auto*
from *inv i inc-i-gso*[*OF i Gr*] *inc-i*[*OF Fr*]
have *Gr′*: *g-repr* (*Suc i*) (*inc-i Gr*) *F* **and** *Fr′*: *f-repr* (*Suc i*) (*inc-i Fr*) *F*
by *auto*
from *red red-i* **have** *red*: *gs.weakly-reduced* α (*Suc i*) *G*
unfolding *gs.weakly-reduced-def*
by (*intro allI impI*, *rename-tac ii*, *case-tac Suc ii* = *i*, *auto*)
show *?thesis* **unfolding** *increase-i-def split*
by (*rule LLL-invI*[*OF Fr′ Gr′*], *insert inv red i*, *auto*)
qed

lemma *basis-reduction-add-row-main*: **assumes** *Linv*: *LLL-invariant* (*i*,*Fr*,*Gr*) *F G*
and *i*: *i* < *m* **and** *j*: *j* < *i*
and *res*: *basis-reduction-add-row-main* (*i*,*Fr*,*Gr*) *fj mu* = ((*i′*,*Fr′*,*Gr′*), *c*)
and *fj*: *fj* = *F* ! *j*
and *mu*: *mu* = *gs*.μ (*RAT F*) *i j*
shows ∃ *v*. *LLL-invariant* (*i′*,*Fr′*,*Gr′*) (*F*[*i* := *v*]) *G* ∧ *i′* = *i* ∧ *Gr′* = *Gr* ∧ *abs* (*mu* − *of-int c*) ≤ *inverse 2*
∧ *mu* − *of-int c* = *gs*.μ (*RAT* (*F*[*i* := *v*])) *i j*
∧ (∀ *i′ j′*. *i′* < *m* ⟶ *j′* < *m* ⟶ (*i′* ≠ *i* ∨ *j′* > *j*)
⟶ *gs*.μ (*RAT* (*F*[*i* := *v*])) *i′ j′* = *gs*.μ (*RAT F*) *i′ j′*)
proof −
define *M* **where** *M* = *map* (λ*i*. *map* (*gs*.μ (*RAT F*) *i*) [*0*..<*m*]) [*0*..<*m*]
note *inv* = *LLL-invD*[*OF Linv*]
note *Gr* = *inv*(*1*)
have *ji*: *j* ≤ *i j* < *m* **and** *jstrict*: *j* < *i*
and *add*: *set F* ⊆ *carrier-vec n i* < *length F j* < *length F i* ≠ *j*
and *len*: *length F* = *m* **and** *red*: *gs.weakly-reduced* α *i G*

and *gs*:*snd* (*gram-schmidt-int n F*) = *G*
and *Fr*: *f-repr i Fr F*
and *Gr*: *g-repr i Gr F*
and *indep*: *gs.lin-indpt-list* (*RAT F*)
using *inv j i* **by** *auto*
let *?R* = *rat-of-int*
let *?RV* = *map-vec ?R*
from *add*[*unfolded set-conv-nth*]
have *Fij*: $F\ !\ i \in$ *carrier-vec n* $F\ !\ j \in$ *carrier-vec n* **by** *auto*
let $?x = F\ !\ i - c \cdot_v F\ !\ j$
define *F1* **where** $F1 = F[i := ?x]$
let *?g* = *gs.gso* (*RAT F*)
from *add*[*unfolded set-conv-nth*]
have *Fi*:$\bigwedge$ *i*. *i* < *length* (*RAT F*) $\Longrightarrow$ (*RAT F*) ! *i* $\in$ *carrier-vec n* **by** *auto*
with *len j i*
have *gs-carr*: *?g j* $\in$ *carrier-vec n*
?g i $\in$ *carrier-vec n*
$\bigwedge$ *i*. *i* < *j* $\Longrightarrow$ *?g i* $\in$ *carrier-vec n*
$\bigwedge$ *j*. *j* < *i* $\Longrightarrow$ *?g j* $\in$ *carrier-vec n*
by (*intro gs.gso-carrier′*,*force*)+
have *RAT-F1*: *RAT F1* = (*RAT F*)[*i* := (*RAT F*) ! *i* − *?R c* $\cdot_v$ (*RAT F*) ! *j*]
unfolding *F1-def*
proof (*rule nth-equalityI*[*rule-format*], *goal-cases*)
case (*2 k*)
show *?case*
proof (*cases k* = *i*)
case *False*
thus *?thesis* **using** *2* **by** *auto*
next
case *True*
hence *?thesis* = (*?RV* (*F* ! *i* − *c* $\cdot_v$ *F* ! *j*) =
?RV (*F* ! *i*) − *?R c* $\cdot_v$ *?RV* (*F* ! *j*))
using *2 add* **by** *auto*
also have . . . **by** (*rule eq-vecI*, *insert Fij*, *auto*)
finally show *?thesis* **by** *simp*
qed
qed *auto*
hence *RAT-F1-i*:*RAT F1* ! *i* = (*RAT F*) ! *i* − *?R c* $\cdot_v$ (*RAT F*) ! *j* (**is** - = - − *?mui*)
using *i len* **by** *auto*
have *uminus*: *F* ! *i* − *c* $\cdot_v$ *F* ! *j* = *F* ! *i* + −*c* $\cdot_v$ *F* ! *j*
by (*subst minus-add-uminus-vec*, *insert Fij*, *auto*)
obtain *G1* **where** *gs′*: *snd* (*gram-schmidt-int n F1*) = *G1* **by** *force*
have *F1-F*: *lattice-of F1* = *lattice-of F* **unfolding** *F1-def uminus*
by (*rule lattice-of-add*[*OF add*, *of* - − *c*], *auto*)
from *len* **have** *len′*:*length* (*RAT F*) = *m* **by** *auto*
from *add* **have** *add′*:*set* (*map ?RV F*) ⊆ *carrier-vec n* **by** *auto*
from *add len*
have *k* < *length F* $\Longrightarrow$ ¬ *k* ≠ *i* $\Longrightarrow$ *F1* ! *k* $\in$ *carrier-vec n* **for** *k*

unfolding *F1-def*
by (*metis* (*no-types*, *lifting*) *nth-list-update nth-mem subset-eq carrier-dim-vec index-minus-vec*(*2*)
index-smult-vec(*2*))
hence $k < \mathit{length}\ F \Longrightarrow F1\ !\ k \in \mathit{carrier\text{-}vec}\ n$ **for** k
unfolding *F1-def* **using** *add len* **by** (*cases* $k \neq i$,*auto*)
with *len* **have** *F1*: *set F1* $\subseteq$ *carrier-vec n length F1* = *m* **unfolding** *F1-def* **by** (*auto simp*: *set-conv-nth*)
hence *F1'*: *length* (*RAT F1*) = *m SRAT F1* $\subseteq$ *Rn* **by** *auto*
from *indep* **have** *dist*: *distinct* (*RAT F*) **by** (*auto simp*: *gs.lin-indpt-list-def*)
have *Fij'*: (*RAT F*) ! *i* $\in$ *Rn* (*RAT F*) ! *j* $\in$ *Rn* **using** *add'*[*unfolded set-conv-nth*] *i* ‹*j* < *m*› *len* **by** *auto*
have *uminus'*: (*RAT F*) ! *i* − *rat-of-int c* $\cdot_v$ (*RAT F*) ! *j* = (*RAT F*) ! *i* + − *rat-of-int c* $\cdot_v$ (*RAT F*) ! *j*
by (*subst minus-add-uminus-vec*[**where** *n* = *n*], *insert Fij'*, *auto*)
have *span-F-F1*: *gs.span* (*SRAT F*) = *gs.span* (*SRAT F1*) **unfolding** *RAT-F1 uminus'*
by (*rule gs.add-vec-span*, *insert len add*, *auto*)
have ∗∗: *of-int-hom.vec-hom* (*F* ! *i*) + − *rat-of-int c* $\cdot_v$ (*RAT F*) ! *j*
= *of-int-hom.vec-hom* (*F* ! *i* − *c* $\cdot_v$ *F* ! *j*)
by (*rule eq-vecI*, *insert Fij len i j*, *auto*)
from *i j len* **have** *j* < *length* (*RAT F*) *i* < *length* (*RAT F*) *i* $\neq$ *j* **by** *auto*
from *gs.lin-indpt-list-add-vec*[*OF this indep*, *of* − *of-int c*]
have *gs.lin-indpt-list* ((*RAT F*) [*i* := (*RAT F*) ! *i* + − *of-int c* $\cdot_v$ (*RAT F*) ! *j*]) (**is** *gs.lin-indpt-list ?F1*) **.**
also have *?F1* = *RAT F1* **unfolding** *F1-def* **using** *i len Fij'* ∗∗
by (*auto simp*: *map-update*)
finally have *indep-F1*: *gs.lin-indpt-list* (*RAT F1*) **.**
note *conn1* = *indep len' gs*[*unfolded gram-schmidt-int-def gram-schmidt-wit-def*]
note *conn2* = *indep-F1 F1'*(*1*) *gs'*[*unfolded gram-schmidt-int-def gram-schmidt-wit-def*]
from *gs.main-connect*[*OF conn1*] *gs.main-connect*[*OF conn2*]
have *G-def*: *G* = *map ?g* [*0* ..< *m*] *G* = *gram-schmidt n* (*RAT F*)
and *G1-def*: *G1* = *map* (*gs.gso* (*RAT F1*)) [*0*..< *m*] *G1* = *gram-schmidt n* (*RAT F1*)
by (*auto simp*:*o-def*)
from *gs.gram-schmidt*[*OF conn1*] *gs.gram-schmidt*[*OF conn2*] *span-F-F1 len*
have *span-G-G1*: *gs.span* (*set G*) = *gs.span* (*set G1*)
and *lenG*: *length G* = *m*
and *Gi*: $i < \mathit{length}\ G \Longrightarrow G\ !\ i \in Rn$
and *G1i*: $i < \mathit{length}\ G1 \Longrightarrow G1\ !\ i \in Rn$ **for** *i*
by *auto*
have *eq*: $x \neq i \Longrightarrow$ *RAT F1* ! *x* = (*RAT F*) ! *x* **for** *x* **unfolding** *RAT-F1* **by** *auto*
hence *eq-part*: $x < i \Longrightarrow$ *gs.gso* (*RAT F1*) *x* = *gs.gso* (*RAT F*) *x* **for** *x*
by (*intro gs-gs-identical*, *insert len*, *auto*)
have *G*: $i < m \Longrightarrow$ (*RAT F*) ! *i* $\in$ *Rn*
$i < m \Longrightarrow F\ !\ i \in \mathit{carrier\text{-}vec}\ n$ **for** *i* **by**(*insert add len'*, *auto*)
note *carr1*[*intro*] = *this*[*OF i*] *this*[*OF ji*(*2*)]
from *Gi*[*unfolded gs.main-connect*[*OF conn1*]] *G1i*[*unfolded gs.main-connect*[*OF

conn2]]
have $x < m \Longrightarrow ?g\ x \in Rn$
$x < m \Longrightarrow gs.gso\ (RAT\ F1)\ x \in Rn$
$x < m \Longrightarrow dim\text{-}vec\ (gs.gso\ (RAT\ F1)\ x) = n$
$x < m \Longrightarrow dim\text{-}vec\ (gs.gso\ (RAT\ F)\ x) = n$
for x **by** (*auto simp*:*o-def*)
hence *carr2*[*intro*!]:$?g\ i \in Rn$ $gs.gso\ (RAT\ F1)\ i \in Rn$
$?g$ ‘ $\{0..<i\} \subseteq Rn$
$?g$ ‘ $\{0..<Suc\ i\} \subseteq Rn$ **using** i **by** *auto*
have *F1-RV*: $?RV\ (F1\ !\ i) = RAT\ F1\ !\ i$ **using** *i F1* **by** *auto*
have *F-RV*: $?RV\ (F\ !\ i) = (RAT\ F)\ !\ i$ **using** *i len* **by** *auto*
have $x < i \Longrightarrow gs.gso\ (RAT\ F1)\ x = gs.gso\ (RAT\ F)\ x$ **for** x **using** *eq*
by (*rule gs-gs-identical*, *auto*)
hence *span-G1-G*: *gs.span* (*gs.gso* (*RAT F1*) ‘ $\{0..<i\}$)
$=$ *gs.span* (*gs.gso* (*RAT F*) ‘ $\{0..<i\}$) (**is** $?ls = ?rs$)
apply(*intro cong*[*OF refl*[*of gs.span*]],*rule image-cong*[*OF refl*]) **using** *eq* **by** *auto*
have $(RAT\ F1)\ !\ i - gs.gso\ (RAT\ F1)\ i = ((RAT\ F)\ !\ i - gs.gso\ (RAT\ F1)\ i) - ?mui$
unfolding *RAT-F1-i* **using** *carr1 carr2*
by (*intro eq-vecI*, *auto*)
hence *in1*:$((RAT\ F)\ !\ i - gs.gso\ (RAT\ F1)\ i) - ?mui \in ?rs$
using *gram-schmidt.projection-exist*[*OF conn2 i*]
unfolding *span-G1-G* **by** *auto*
from ‹$j < i$› **have** *Gj-mem*: $(RAT\ F)\ !\ j \in (\lambda\ x.\ ((RAT\ F)\ !\ x))$ ‘ $\{0\ ..<\ i\}$ **by** *auto*
have *id1*: *set* (*take i* (*map of-int-hom.vec-hom F*)) $= (\lambda x.$ *of-int-hom.vec-hom* $(F\ !\ x))$ ‘ $\{0..<i\}$
using ‹$i \le m$› *len*
by (*subst nth-image*[*symmetric*], *force*+)
have $(RAT\ F)\ !\ j \in ?rs \longleftrightarrow (RAT\ F)\ !\ j \in gs.span\ ((\lambda x.\ ?RV\ (F\ !\ x))$ ‘ $\{0..<i\})$
unfolding *gs.partial-span*[*OF conn1* ‹$i \le m$›] *id1* **..**
also have $(\lambda x.\ ?RV\ (F\ !\ x))$ ‘ $\{0..<i\} = (\lambda x.\ ((RAT\ F)\ !\ x))$ ‘ $\{0..<i\}$ **using** ‹$i < m$› *len* **by** *force*
also have $(RAT\ F)\ !\ j \in gs.span\ \ldots$
by (*rule gs.span-mem*[*OF - Gj-mem*], *insert* ‹$i < m$› *G*, *auto*)
finally have $(RAT\ F)\ !\ j \in ?rs$ **.**
hence *in2*:$?mui \in ?rs$
apply(*intro gs.prod-in-span*) **by** *force*+
have *ineq*:$((RAT\ F)\ !\ i - gs.gso\ (RAT\ F1)\ i) + ?mui - ?mui = ((RAT\ F)\ !\ i - gs.gso\ (RAT\ F1)\ i)$
using *carr1 carr2* **by** (*intro eq-vecI*, *auto*)
have *cong'*: $A = B \Longrightarrow A \in C \Longrightarrow B \in C$ **for** $A\ B$ :: *'a vec* **and** C **by** *auto*
have $*$: *gs.gso* (*RAT F*) ‘ $\{0..<i\} \subseteq Rn$ **by** *auto*
have *in-span*: $(RAT\ F)\ !\ i - gs.gso\ (RAT\ F1)\ i \in ?rs$
by (*rule cong'*[*OF eq-vecI gs.span-add1*[*OF * in1 in2*,*unfolded ineq*]], *insert carr1 carr2*, *auto*)
{

```
    fix x assume x:x < i hence x < m i ≠ x using i by auto
    from gram-schmidt.orthogonal[OF conn2,OF i this] this
    have gs.gso (RAT F1) i • gs.gso (RAT F1) x = 0 by auto
  }
  hence G1-G: gs.gso (RAT F1) i = gs.gso (RAT F) i
   apply(intro gram-schmidt.projection-unique[OF conn1 i gs.gso-carrier[OF conn2
i]])
    using in-span by (auto simp: eq-part[symmetric])
  have eq-fs:x < m ⟹ gs.gso (RAT F1) x = gs.gso (RAT F) x
    for x proof(induct x rule:nat-less-induct[rule-format])
    case (1 x)
    hence ind: m < x ⟹ gs.gso (RAT F1) m = gs.gso (RAT F) m
      for m by auto
    { assume x > i
      hence ?case apply(subst (1 2) gs.gso.simps) unfolding gs.μ.simps
              apply(rule cong[OF - cong[OF refl[of gs.sumlist]]])
              using ind eq by auto
    } note eq-rest = this
     show ?case by (rule linorder-class.linorder-cases[of x i],insert G1-G eq-part
eq-rest,auto)
  qed
  with G-def G1-def cof-vec-space.gram-schmidt-result
  have Hs:G1 = G by (auto simp:o-def)
  hence red: gs.weakly-reduced α i G1 using red by auto
  let ?Mi = M ! i ! j
  let ?x′ = get-nth-i Fr − floor-ceil ?Mi ·v F ! j
  define Fr1 where Fr1 = update-i Fr ?x′
  have Hr: update-i Fr (?x′) = Fr1 unfolding Fr1-def by simp
  have Gjn: dim-vec (F ! j) = n using Fij(2) carrier-vecD by blast
  define E where E = addrow-mat m (− ?R c) i j
  define M′ where M′ = gs.M (RAT F) m
  define N′ where N′ = gs.M (RAT F1) m
  have E: E ∈ carrier-mat m m unfolding E-def by simp
 have M: M′ ∈ carrier-mat m m unfolding gram-schmidt.M-def M′-def by auto
  have N: N′ ∈ carrier-mat m m unfolding gram-schmidt.M-def N′-def by auto
  let ?GsM = gs.Fs G
  have Gs: ?GsM ∈ carrier-mat m n using G-def by auto
  hence GsT: ?GsM^T ∈ carrier-mat n m by auto
 have Gnn: gs.Fs (RAT F) ∈ carrier-mat m n unfolding mat-of-rows-def using
len by auto
  have gs.Fs (RAT F1) = addrow (− ?R c) i j (gs.Fs (RAT F))
   unfolding RAT-F1 by (rule eq-matI, insert Gjn ji(2), auto simp: len mat-of-rows-def)
  also have ... = E ∗ gs.Fs (RAT F) unfolding E-def
    by (rule addrow-mat, insert j i, auto simp: mat-of-rows-def len)
  finally have HEG: gs.Fs (RAT F1) = E ∗ gs.Fs (RAT F) .
  have (E ∗ M′) ∗ gs.Fs G = E ∗ (M′ ∗ gs.Fs G)
    by (rule assoc-mult-mat[OF E M Gs])
  also have M′ ∗ ?GsM = gs.Fs (RAT F) using gs.matrix-equality[OF conn1]
M′-def by simp
```

also have $E * \ldots = gs.Fs\ (RAT\ F1)$ **unfolding** *HEG* **..**
also have $\ldots = N' * gs.Fs\ G1$ **using** *gs.matrix-equality*[*OF conn2*] *N'-def* **by** *simp*
also have $gs.Fs\ G1 = ?GsM$ **unfolding** *Hs* **..**
finally have $(E * M') * ?GsM = N' * ?GsM$ **.**
from *arg-cong*[*OF this*, *of* $\lambda\ x.\ x * ?GsM^T$] *E M N*
have *EMN*: $(E * M') * (?GsM * ?GsM^T) = N' * (?GsM * ?GsM^T)$
by (*subst* (*1 2*) *assoc-mult-mat*[*OF - Gs GsT*, *of - m*, *symmetric*], *auto*)
have *det* $(?GsM * ?GsM^T) = gs.Gramian\text{-}determinant\ G\ m$
unfolding *gs.Gramian-determinant-def*
by (*subst gs.Gramian-matrix-alt-def*, *auto simp*: *Let-def G-def*)
also have $\ldots > 0$
by (*rule gs.Gramian-determinant*(*2*)[*OF gs.orthogonal-imp-lin-indpt-list* ‹*length* $G = m$›],
insert gs.gram-schmidt(*2*−)[*OF conn1*], *auto*)
finally have *det* $(?GsM * ?GsM^T) \neq 0$ **by** *simp*
from *vec-space.det-nonzero-congruence*[*OF EMN this - - N*] *Gs E M*
have *EMN*: $E * M' = N'$ **by** *auto*

have *Mij*: $mu = M\ !\ i\ !\ j$ **unfolding** *M-def mu* **using** ‹$i < m$› *ji*(*2*) **by** *auto*
from *res*[*unfolded Mij*] **have** *c*: $c = floor\text{-}ceil\ (M\ !\ i\ !\ j)$
by (*auto simp*: *Let-def split*: *if-splits*)
have *x*: $?x = ?x'$ **by** (*subst get-nth-i*[*OF Fr*], *insert add*, *auto simp*: *c Mij j*)
{
assume *c0*: $c = 0$
have $Fr1 = update\text{-}i\ Fr\ (F\ !\ i)$ **unfolding** *Fr1-def x*[*symmetric*] *c0*
using ‹$F\ !\ i \in carrier\text{-}vec\ n$› ‹$F\ !\ j \in carrier\text{-}vec\ n$› **by** *auto*
from *update-i*[*OF Fr*, *of* (*F* ! *i*), *folded this*] *i len*
have *list-repr i Fr1* $(F[i := (F\ !\ i)])$ **by** *auto*
also have $F[i := F\ !\ i] = F$
by (*rule nth-equalityI*, *force*, *intro allI*, *rename-tac j*, *insert i len*, *case-tac j* $= i$, *auto*)
finally have *f-repr i Fr1 F* **.**
with *Fr* **have** $Fr1 = Fr$ **unfolding** *list-repr-def* **by** (*cases Fr*, *cases Fr1*, *auto*)
} **note** $c0 = this$
from *res*[*unfolded basis-reduction-add-row-main.simps Let-def fj Mij Hr*]
have *res*: $i' = i\ Fr' = Fr1\ Gr' = Gr$ **using** *Mij c0 i len*
by (*auto simp*: *j split*: *if-splits*)
{
from *Gr*[*unfolded g-repr-def*] *i*
have *list-repr i Gr* $(map\ (\lambda x.\ (x, \|x\|^2))\ (map\ (GSO\ F)\ [0..<m]))$
(**is** *list-repr - -* (*map ?f* (*map* (*GSO F*) *?is*)))
by *auto*
also have *map* (*GSO F*) *?is* = *map* (*GSO F1*) *?is* (**is** $?l = ?r$)
proof (*rule nth-equalityI*, *force*, *unfold length-map*, *intro allI impI*, *goal-cases*)
case (*1 ii*)
hence *ii*: $ii < m$ **using** *i* **by** *auto*
from *1* **have** *id*: $?is\ !\ ii = ii$
by (*metis add.commute add.right-neutral diff-zero length-upt nth-upt*)

have *GSO F ii = GSO F1 ii* **using** *arg-cong[OF Hs[unfolded G1-def G-def], of λ xs. xs ! ii] ii*
unfolding *GSO-def* **by** *auto*
thus *?case* **unfolding** *nth-map[OF 1] id* **.**
qed
finally have *list-repr i Gr (map ?f (map (GSO F1) [0..<m]))* **.**
hence *gso''*: *g-repr i Gr F1* **unfolding** *g-repr-def* **using** *i* **by** *simp*
have *repr'*: *f-repr i Fr1 F1* **unfolding** *F1-def Fr1-def x map-update*
by *(rule update-i[OF Fr], insert add, auto)*
have *LLL-invariant (i, Fr1, Gr) F1 G* **unfolding** *Hs[symmetric]*
apply *(rule LLL-invI[OF repr' gso'' G1-def(2)[folded snd-gram-schmidt-int,symmetric] - red inv(7)])*
by *(insert F1 F1-F inv(5) indep-F1, auto)*
} **note** *inv-gso = this*
{
fix *ia* **assume** *ia ≤ j* **hence** *ia < i* **using** *ji j* **by** *auto*
hence *(RAT F1) ! ia = (RAT F) ! ia*
using *F1-def i len* **by** *auto*
}
hence *fs-eq*:*gs.gso (RAT F1) j = gs.gso (RAT F) j*
by *(intro gs-gs-identical, auto)*
have *dima*:*dim-vec a = dim-vec b ⟹ ?RV (a + b) = ?RV a + ?RV b* **for** *a b* **by** *auto*
from *gs.gso-times-self-is-norm[OF conn1 ji(2)]*
have *gs-norm*:*(RAT F) ! j • gs.gso (RAT F) j =* $\|$*gs.gso (RAT F) j*$\|^2$ **by** *auto*
have *fc*:*floor-ceil (0::rat) = 0* **unfolding** *floor-ceil-def* **by** *linarith*
{
assume *sq-norm-vec (gs.gso (RAT F) j) = 0*
hence *gs.gso (RAT F) j =* 0_v *n* **using** *gs-carr(1) sq-norm-vec-eq-0 len* **by** *force*
hence *c = 0* **unfolding** *c M-def gs.μ.simps* **using** *j i fc* **by** *auto*
} **note** *zero = this*
from ‹*j < i*› **have** *if-True*: *(if j < i then t else e) = t* **for** *t e* **by** *simp*
have *id1*: *?RV (F ! j) = (RAT F) ! j* **using** *ji len* **by** *auto*
have *id*: *(RAT F1) ! i = (RAT F) ! i − ?R c* $\cdot_v$ *?RV (F ! j)* **unfolding** *F1-def* **using** *i len Fij* **by** *auto*
have *mudiff*: *mu = (RAT F) ! i • gs.gso (RAT F) j /* $\|$*gs.gso (RAT F) j*$\|^2$
unfolding *mu gs.μ.simps if-True id* **using** *i ji(2)* **by** *auto*
have *mudiff*:*mu − of-int c = gs.μ (RAT F1) i j*
unfolding *mudiff* **unfolding** *gs.μ.simps fs-eq if-True id*
apply *(subst minus-scalar-prod-distrib, (insert Fij gs-carr, auto)[3])*
apply *(subst scalar-prod-smult-left, (insert Fij gs-carr, auto)[1])*
apply *(unfold id1 gs-norm)*
using *zero divide-diff-eq-iff* **by** *fastforce*
{
fix *i' j'*
assume *ij*: *i' < m j' < m* **and** *choice*: *i' ≠ i ∨ j < j'*
have *gs.μ (RAT F1) i' j'*
= N' $$ (i',j') **using** *ij F1* **unfolding** *N'-def gs.M-def* **by** *auto*

also have ... = *addrow* (− *?R c*) *i j M′* $$ (*i′*,*j′*) **unfolding** *EMN*[*symmetric*] *E-def*
by (*subst addrow-mat*[*OF M*], *insert ji*, *auto*)
also have ... = (*if i* = *i′ then* − *?R c* ∗ *M′* $$ (*j*, *j′*) + *M′* $$ (*i′*, *j′*) *else* *M′* $$ (*i′*, *j′*))
by (*rule index-mat-addrow*, *insert ij M*, *auto*)
also have ... = *M′* $$ (*i′*, *j′*)
proof (*cases i* = *i′*)
case *True*
with *choice* **have** *jj*: *j* < *j′* **by** *auto*
have *M′* $$ (*j*, *j′*) = *gs.μ* (*RAT F*) *j j′*
using *ij ji len* **unfolding** *M′-def gs.M-def* **by** *auto*
also have ... = *0* **unfolding** *gs.μ.simps* **using** *jj* **by** *auto*
finally show *?thesis* **using** *True* **by** *auto*
qed *auto*
also have ... = *gs.μ* (*RAT F*) *i′ j′*
using *ij len* **unfolding** *M′-def gs.M-def* **by** *auto*
also note *calculation*
} **note** *mu-change* = *this*
have *abs* (*mu* − *of-int c*) ≤ *inverse 2* **unfolding** *res j Mij c*
by (*rule floor-ceil*)
thus *?thesis* **using** *mu-change inv-gso mudiff* **unfolding** *res j F1-def* **by** *auto*
qed

lemma *basis-reduction-add-row-i-im1*: **assumes** *Linv*: *LLL-invariant* (*i*,*Fr*,*Gr*) *F G*
and *i*: *i* < *m* **and** *i0*: *i* ≠ *0*
and *res*: *basis-reduction-add-row-i-im1* (*i*,*Fr*,*Gr*) = ((*i′*,*Fr′*,*Gr′*), *mu*)
shows ∃ *F′*. *LLL-invariant* (*i′*,*Fr′*,*Gr′*) *F′ G* ∧ *i′* = *i* ∧ *Gr′* = *Gr* ∧ *abs mu* ≤ *inverse 2* ∧
mu = *gs.μ* (*RAT F′*) *i* (*i*−*1*)
proof −
note *inv* = *LLL-invD*[*OF Linv*]
from *i* **have** *im1*: *i* − *1* < *m* **by** *auto*
from *i i0* **have** *im1′*: *i* − *1* < *i* **by** *auto*
from *inv* **have** *G*: *length F* = *m* **by** *auto*
have *fst*: *get-nth-im1 Fr* = *F* ! (*i* − *1*) **unfolding** *get-nth-im1*[*OF inv*(*8*) *i0*] **using** *im1 G*
by *auto*
have *Gi*: *F* ! *i* ∈ *carrier-vec n* **using** *i inv*(*3*,*4*) **by** *auto*
have *gs-gs*: *gs.gso* (*RAT F*) (*i* − *1*) ∈ *Rn*
by (*rule gs.gso-carrier′*, *insert i inv*(*3*,*4*), *auto*)
note *res* = *res*[*unfolded basis-reduction-add-row-i-im1-def Let-def split fst*]
from *res* **obtain** *c* **where**
res′: *basis-reduction-add-row-main* (*i*, *Fr*, *Gr*) (*F* ! (*i* − *1*)) (*μ-i-im1 Fr Gr*) = ((*i′*,*Fr′*,*Gr′*),*c*)
(**is** *?res* = -) **by** (*cases ?res*, *auto*)
from *res*[*unfolded res′*] **have** *mu*: *mu* = *μ-i-im1 Fr Gr* − *of-int c* **by** *auto*
have *id*: *μ-i-im1 Fr Gr* = *gs.μ* (*RAT F*) *i* (*i* − *1*)

    unfolding *μ-i-im1*[*OF inv*(*8*,*9*) *G*[*symmetric*] *i0 i Gi gs-gs*]
    by (*rule gs-μ-identical*, *insert G i*, *auto*)
  from *basis-reduction-add-row-main*[*OF Linv i im1′ res′ refl id*]
  obtain *G′* **where** *i′*: *i′* = *i Gr′* = *Gr LLL-invariant* (*i*, *Fr′*, *Gr*) *G′ G*
      |*μ-i-im1 Fr Gr* − *of-int c*| ≤ *inverse 2*
      μ-i-im1 Fr Gr − *of-int c* = *gs.μ* (*RAT G′*) *i* (*i* − *1*) **by** *auto*
  thus *?thesis* **unfolding** *mu* **by** *auto*
qed

lemma *basis-reduction-add-row-i-all*: **fixes** *Gr* **assumes** *Linv*: *LLL-invariant* (*i*,*Fr*,*Gr*) *F G*
  and *i*: *i* < *m*
  and *res*: *basis-reduction-add-row-i-all* (*i*,*Fr*,*Gr*) = (*i′*,*Fr′*,*Gr′*)
shows ∃ *F′*. *LLL-invariant* (*i*,*Fr′*,*Gr*) *F′ G* ∧ *i′* = *i* ∧ *Gr′* = *Gr* ∧
  (∀ *j* < *i*. *abs* (*gs.μ* (*RAT F′*) *i j*) ≤ *1*/*2*) ∧
  (∀ *i′ j′*. *i′* < *m* ⟶ *j′* < *m* ⟶ *i′* ≠ *i* ⟶
    gs.μ (*RAT F′*) *i′ j′* = *gs.μ* (*RAT F*) *i′ j′*)
proof −
  note *inv* = *LLL-invD*[*OF Linv*]
  let *?xs* = *map* (*gs.gso* (*RAT F*)) [*0*..<*i*]
  from *inv*(*8*)[*unfolded list-repr-def*] *inv*(*4*) *i*
  have *id*: *fst Fr* = *rev* (*take i F*)
    by (*auto simp*: *rev-map*[*symmetric*] *take-map o-def*)
  from *inv*(*9*) **have** *id′*: *fst Gr* = *rev* (*map* (λ*x*. (*x*, $\|x\|^2$)) *?xs*)
    unfolding *g-repr-def list-repr-def GSO-def* **by** (*auto simp*: *take-map*)
  note *res* = *res*[*unfolded basis-reduction-add-row-i-all-def split Let-def*]
  define *ii* **where** *ii* = *i*
  hence *id″*: [*0*..< *i*] = [*0* ..< *ii*] **by** *auto*
  have *id′*: *fst Gr* = *rev* (*map* (λ*x*. (*x*, $\|x\|^2$)) (*map* (*gs.gso* (*RAT F*)) [*0* ..< *ii*]))

    using *i* **unfolding** *id′ id″* **by** *auto*
  have *id*: *fst Fr* = *rev* (*take ii F*) **unfolding** *id ii-def* **by** *auto*
  let *?small* = λ *F j*. *abs* (*gs.μ* (*RAT F*) *i j*) ≤ *1*/*2*
  have *small*: ∀ *j*. *ii* ≤ *j* ⟶ *j* < *i* ⟶ *?small F j* **unfolding** *ii-def* **by** *auto*
  from *res*[*unfolded id id′*] **have**
    basis-reduction-add-row-i-all-main (*i*, *Fr*, *Gr*) (*rev* (*take ii F*))
     (*rev* (*map* (λ*x*. (*x*, $\|x\|^2$)) (*map* (*gs.gso* (*RAT F*)) [*0*..<*ii*]))) =
    (*i′*, *Fr′*, *Gr′*) *ii* ≤ *i* **unfolding** *ii-def* **by** *auto*
  thus *?thesis* **using** *Linv small*
  proof (*induct ii arbitrary*: *Fr F*)
    case (*Suc ii Fr F*)
    note *inv* = *LLL-invD*[*OF Suc*(*4*)]
    let *?fs* = *gs.gso* (*RAT F*) *ii*
    let *?fsn* = (*?fs*, $\|?fs\|^2$)
    let *?main* = *basis-reduction-add-row-main* (*i*, *Fr*, *Gr*) (*F* ! *ii*)
        (*μ-ij* (*get-nth-i Fr*) *?fsn*)
    obtain *i″ Fr″ Gr″ c* **where** *main*: *?main* = ((*i″*, *Fr″*, *Gr″*),*c*)
      by (*cases ?main*, *auto*)
    from *Suc*(*3*) **have** *ii*: *ii* < *i* (*ii* < *i*) = *True* **by** *auto*

have *Gi*: $F ! i \in$ *carrier-vec n* **using** *inv*(*3*,*4*) *i* **by** *auto*
have *gs-gs*: *gram-schmidt.gso n* (*RAT F*) *ii* $\in$ *Rn*
by (*rule gram-schmidt.gso-carrier'*, *insert inv*(*3*,*4*) *i ii*, *auto*)
from *get-nth-i*[*OF inv*(*8*)] *inv*(*4*) *i Gi gs-gs*
have *pair*: μ*-ij* (*get-nth-i Fr*) *?fsn* =
gs.μ (*RAT F*) *i ii*
unfolding μ*-ij-def split gs*.μ*.simps ii if-True id*
by *auto*
from *basis-reduction-add-row-main*[*OF Suc*(*4*) *i ii*(*1*) *main refl pair*]
obtain *v* **where**
Linv: *LLL-invariant* (*i*, *Fr''*, *Gr*) (*F*[*i* := *v*]) *G*
and *id*: *i''* = *i Gr''* = *Gr*
and *small*: *?small* (*F*[*i* := *v*]) *ii*
and *id'*: $\bigwedge$ *i' j'*. *i'* < *m* $\Longrightarrow$ *j'* < *m* $\Longrightarrow$ (*i'* $\neq$ *i* $\vee$ *ii* < *j'*) $\longrightarrow$
gs.μ (*RAT* (*F*[*i* := *v*])) *i' j'* =
gs.μ (*RAT F*) *i' j'* **by** *auto*
let *?G* = *F*[*i* := *v*]
from *inv Suc*(*3*) **have** *lt*: *ii* < *length F* **by** *auto*
have (*rev* (*map* (λx. (*x*, $\|x\|^2$)) (*map* (*gs.gso* (*RAT F*)) [*0*..<*Suc ii*])))
= (*?fsn* # *rev* (*map* (λx. (*x*, $\|x\|^2$)) (*map* (*gs.gso* (*RAT F*)) [*0*..<*ii*])))
by *auto*
from *id Suc*(*2*)[*unfolded rev-take-Suc*[*OF lt*] *this basis-reduction-add-row-i-all-main.simps split Let-def main*]
have *basis-reduction-add-row-i-all-main* ((*i*, *Fr''*, *Gr*)) (*rev* (*take ii F*))
(*rev* (*map* (λx. (*x*, $\|x\|^2$)) (*map* (*gs.gso* (*RAT F*)) [*0*..<*ii*]))) =
(*i'*, *Fr'*, *Gr'*) **by** *auto*
also have *rev* (*take ii F*) = *rev* (*take ii ?G*) **using** *ii* **by** *auto*
also have *map* (*gs.gso* (*RAT F*)) [*0*..<*ii*] = *map* (*gs.gso* (*RAT ?G*)) [*0*..<*ii*]
by (*rule nth-equalityI*, *auto intro*!: *gs-gs-identical*, *insert inv*(*4*) *i ii*, *auto*)
finally have *res*: *basis-reduction-add-row-i-all-main* ((*i*, *Fr''*, *Gr*)) (*rev* (*take ii ?G*))
(*rev* (*map* (λx. (*x*, $\|x\|^2$)) (*map* (*gs.gso* (*RAT ?G*)) [*0*..<*ii*]))) =
(*i'*, *Fr'*, *Gr'*) **by** *auto*
from *ii* **have** *ii-le*: *ii* $\leq$ *i* **by** *auto*
have *small*: $\forall j \geq ii$. *j* < *i* $\longrightarrow$ *?small ?G j*
proof (*intro allI impI*)
fix *j*
assume $*$: *ii* $\leq$ *j j* < *i*
show *?small ?G j*
proof (*cases j* = *ii*)
case *True*
with *small* **show** *?thesis* **by** *auto*
next
case *False*
with $*$ *Suc*(*5*)[*rule-format*, *of j*]
have *small*: *?small F j* **by** *auto*
with *id'*[*rule-format*, *OF i*, *of j*] *i* $*$ *False*
show *?thesis* **by** *auto*
qed

qed
from *Suc*(*1*)[*OF res ii-le Linv small*] **obtain** G' **where**
Linv: *LLL-invariant* (*i*, *Fr'*, *Gr*) G' *G*
and i': $i' = i$
and *gso'*: $Gr' = Gr$
and *small*: ($\forall j<i$. *?small* G' *j*)
and *id*: $\bigwedge i'\ j'.\ i' < m \Longrightarrow j' < m \Longrightarrow i' \neq i \Longrightarrow$
gs.μ (*RAT* G') i' j' =
gs.μ (*RAT* *?G*) i' j' **by** *blast*
show *?case*
proof (*intro exI conjI*, *rule Linv*, *rule i'*, *rule gso'*, *rule small*, *intro allI impI*, *goal-cases*)
case (*1* i' j')
show *?case* **unfolding** *id*[*OF 1*]
by (*rule id'*[*rule-format*], *insert 1 i'*, *auto*)
qed
qed *auto*
qed

lemma *basis-reduction-part-2-main*: **fixes** *Gr* **assumes** *Linv*: *LLL-invariant state F G*
and *n*: *fst state* = *m*
and *res*: *basis-reduction-part-2-main state* = (*i*,*Fr*,*Gr*)
shows $\exists$ F'. *LLL-invariant* (*i*,*Fr*,*Gr*) F' *G* $\wedge$ *i* = *0* $\wedge$
gs.strictly-reduced m α *G* (*gs*.μ (*RAT* F'))
proof −
from *n* **obtain** *i1 Fr1 Gr1* **where** *state*: *state* = (*i1*, *Fr1*, *Gr1*)
and *i1*: *i1* = *m i1* $\leq$ *m*
by (*cases state*, *auto*)
note *Linv* = *Linv*[*unfolded state*]
from *LLL-invD*[*OF Linv*] *i1*
have *weak*: *gs.weakly-reduced* α *m G* **by** *auto*
let *?small* = λ *F i*. $\forall$ *j*. *j* < *i* $\longrightarrow$ *abs* (*gs*.μ (*RAT F*) *i j*) $\leq$ *1*/*2*
have *small*: $\forall$ *i*. *i1* $\leq$ *i* $\longrightarrow$ *i* < *m* $\longrightarrow$ *?small F i* **unfolding** *i1* **by** *auto*
from *res*[*unfolded state*] *small Linv* ‹*i1* $\leq$ *m*›
show *?thesis*
proof (*induct i1 arbitrary*: *F Fr1 Gr1*)
case (*0 F Fr Gr*)
thus *?thesis* **using** *weak* **by** (*auto intro*!: *exI*[*of - F*] *simp*: *gs.strictly-reduced-def o-def*)
next
case (*Suc i1 F Fr1 Gr1*)
from *Suc*(*4*) **have** *1*: *list-repr* (*Suc i1*) *Fr1 F*
unfolding *LLL-invariant-def* **by** *auto*
hence *2*: *list-repr i1* (*dec-i Fr1*) *F* **by** (*simp add*: *dec-i-Suc*)
from *1 2* **have** ∗: *of-list-repr* (*dec-i Fr1*) = *of-list-repr Fr1*
using *of-list-repr* **by** *blast*
have *Linv*: *LLL-invariant* (*i1*, *dec-i Fr1*, *dec-i Gr1*) *F G*
using *Suc*(*4*) **unfolding** *LLL-invariant-def split*

```
      by (auto simp: g-repr-def dec-i-Suc gs.weakly-reduced-def ∗)
    obtain i2 Fr2 Gr2 where
      call: basis-reduction-add-row-i-all (i1, dec-i Fr1, dec-i Gr1) = (i2, Fr2, Gr2)

      (is ?call = -) by (cases ?call, auto)
    from Suc(3−) have i1: i1 < m i1 ≤ m by auto
    from basis-reduction-add-row-i-all[OF Linv i1(1) call]
    obtain F′ where
      Linv: LLL-invariant (i1, Fr2, dec-i Gr1) F′ G
      and i2: i2 = i1
      and gso2: Gr2 = dec-i Gr1
      and small: ?small F′ i1
      and id: ⋀ i′ j′. i′ < m =⇒ j′ < m =⇒ i′ ≠ i1 =⇒
            gs.μ (RAT F′) i′ j′ =
            gs.μ (RAT F) i′ j′ by auto
    from Suc(2)[unfolded basis-reduction-part-2-main.simps[of Suc i1]] call i2 gso2
     have res: basis-reduction-part-2-main (i1, Fr2, dec-i Gr1) = (i, Fr, Gr) by
auto
    show ?case
    proof (rule Suc(1)[OF res - Linv i1(2)], intro allI impI, goal-cases)
      case (1 i j)
     thus ?case using small[rule-format, of j] Suc(3)[rule-format, of i j] id[of i j]
        i1 by (cases i = i1, auto simp: o-def)
    qed
  qed
qed

context
  assumes α: α ≥ 4/3
begin
lemma α0: α > 0 α ≠ 0
  using α by auto

lemma reduction: 0 < reduction reduction ≤ 1
  α > 4/3 =⇒ reduction < 1
  α = 4/3 =⇒ reduction = 1
  using α unfolding reduction-def by auto


lemma dk-swap-unchanged: assumes len: length F1 = m
  and i0: i ≠ 0 and i: i < m and ki: k ≠ i and km: k ≤ m
  and swap: F2 = F1[i := F1 ! (i − 1), i − 1 := F1 ! i]
shows dk k F1 = dk k F2
proof −
  let ?F1-M = mat k n (λ(i, y). F1 ! i $ y)
  let ?F2-M = mat k n (λ(i, y). F2 ! i $ y)
  have ∃ P. P ∈ carrier-mat k k ∧ det P ∈ {−1, 1} ∧ ?F2-M = P ∗ ?F1-M
  proof cases
    assume ki: k < i
```

hence *H*: *?F2-M* = *?F1-M* **unfolding** *swap*
by (*intro eq-matI*, *auto*)
let *?P* = 1_m *k*
have *?P* ∈ *carrier-mat k k det ?P* ∈ {−*1*, *1*} *?F2-M* = *?P* ∗ *?F1-M* **unfolding** *H* **by** *auto*
thus *?thesis* **by** *blast*
next
assume ¬ *k* < *i*
with *ki* **have** *ki*: *k* > *i* **by** *auto*
let *?P* = *swaprows-mat k i* (*i* − *1*)
from *i0 ki* **have** *neq*: *i* ≠ *i* − *1* **and** *kmi*: *i* − *1* < *k* **by** *auto*
have ∗: *?P* ∈ *carrier-mat k k det ?P* ∈ {−*1*, *1*} **using** *det-swaprows-mat*[*OF ki kmi neq*] *ki* **by** *auto*
from *i len* **have** *iH*: *i* < *length F1 i* − *1* < *length F1* **by** *auto*
have *?P* ∗ *?F1-M* = *swaprows i* (*i* − *1*) *?F1-M*
by (*subst swaprows-mat*[*OF - ki kmi*], *auto*)
also have . . . = *?F2-M* **unfolding** *swap*
by (*intro eq-matI*, *rename-tac ii jj*,
case-tac ii = *i*, (*insert iH*, *simp add*: *nth-list-update*)[*1*],
case-tac ii = *i* − *1*, *insert iH neq ki*, *auto simp*: *nth-list-update*)
finally show *?thesis* **using** ∗ **by** *metis*
qed
then obtain *P* **where** *P*: *P* ∈ *carrier-mat k k* **and** *detP*: *det P* ∈ {−*1*, *1*} **and** *H'*: *?F2-M* = *P* ∗ *?F1-M* **by** *auto*
have *dk k F2* = *det* (*gs.Gramian-matrix F2 k*)
unfolding *dk-def gs.Gramian-determinant-def* **by** *simp*
also have . . . = *det* (*?F2-M* ∗ *?F2-M*^{T}) **unfolding** *gs.Gramian-matrix-def Let-def* **by** *simp*
also have *?F2-M* ∗ *?F2-M*^{T} = *?F2-M* ∗ (*?F1-M*^{T} ∗ *P*^{T}) **unfolding** *H'*
by (*subst transpose-mult*[*OF P*], *auto*)
also have . . . = *P* ∗ (*?F1-M* ∗ (*?F1-M*^{T} ∗ *P*^{T})) **unfolding** *H'*
by (*subst assoc-mult-mat*[*OF P*], *auto*)
also have *det* . . . = *det P* ∗ *det* (*?F1-M* ∗ (*?F1-M*^{T} ∗ *P*^{T}))
by (*rule det-mult*[*OF P*], *insert P*, *auto*)
also have *?F1-M* ∗ (*?F1-M*^{T} ∗ *P*^{T}) = (*?F1-M* ∗ *?F1-M*^{T}) ∗ *P*^{T}
by (*subst assoc-mult-mat*, *insert P*, *auto*)
also have *det* . . . = *det* (*?F1-M* ∗ *?F1-M*^{T}) ∗ *det P*
by (*subst det-mult*, *insert P*, *auto simp*: *det-transpose*)
also have *det* (*?F1-M* ∗ *?F1-M*^{T}) = *det* (*gs.Gramian-matrix F1 k*) **unfolding** *gs.Gramian-matrix-def Let-def* **by** *simp*
also have . . . = *dk k F1*
unfolding *dk-def gs.Gramian-determinant-def* **by** *simp*
finally have *dk k F2* = (*det P* ∗ *det P*) ∗ *dk k F1* **by** *simp*
also have *det P* ∗ *det P* = *1* **using** *detP* **by** *auto*
finally show *dk k F1* = *dk k F2* **by** *simp*
qed

lemma *basis-reduction-step*: **assumes** *inv*: *LLL-invariant* (*i*,*Fr*,*Gr*) *F G*

and *i*: $i < m$
and *res*: *basis-reduction-step* α (*i*,*Fr*,*Gr*) = *state*
shows ($\exists$ *F' G'*. *LLL-invariant state F' G'*)
and *LLL-measure state* < *LLL-measure* (*i*,*Fr*,*Gr*)
proof (*atomize(full)*, *cases i = 0*)
case *i0*: *False*
note *res* = *res*[*unfolded basis-reduction-step-def split*]
obtain *i1 Fr1 Gr1 mu* **where**
il: *basis-reduction-add-row-i-im1* (*i*,*Fr*,*Gr*) = ((*i1*, *Fr1*, *Gr1*), *mu*) (**is** *?b* = -)
by (*cases ?b*, *auto*)
from *basis-reduction-add-row-i-im1*[*OF inv i i0 il*] **obtain** *F1*
where *Linv'*: *LLL-invariant* (*i*, *Fr1*, *Gr1*) *F1 G* **and** *ii*: *i1* = *i*
and *m12*: |*mu*| $\leq$ *inverse 2*
and *mu*: *mu* = *gs*.μ (*RAT F1*) *i* (*i* − *1*) **by** *auto*
note *dk* = *dk-def*
note *Gd* = *Gramian-determinant*(*1*)
note *Gd12* = *Gd*[*OF inv*] *Gd*[*OF Linv'*]
have *dk-eq*: *k* $\leq$ *m* $\Longrightarrow$ *dk k F* = *dk k F1* **for** *k*
unfolding *dk* **using** *Gd12*[*of k*] **by** *auto*
have *D-eq*: *D F* = *D F1* **unfolding** *D-def*
by (*rule arg-cong*[*of - - nat*], *rule prod.cong*, *insert dk-eq*, *auto*)
hence *logD-eq*: *logD F* = *logD F1* **unfolding** *logD-def* **by** *simp*
note *inv* = *LLL-invD*[*OF inv*]
note *inv'* = *LLL-invD*[*OF Linv'*]
from *inv* **have** *repr*: *f-repr i Fr F* **by** *auto*
note *res* = *res*[*unfolded basis-reduction-step-def this split il id Let-def*]
let *?x* = *G* ! (*i* − *1*) **let** *?y* = *G* ! *i*
let *?x'* = *sqnorm-g-im1 Gr1* **let** *?y'* = *sqnorm-g-i Gr1*
let *?cond* = α ∗ *sq-norm ?y* < *sq-norm ?x*
let *?cond'* = α ∗ *?y'* < *?x'*
from *inv'* **have** *red*: *gs.weakly-reduced* α *i G*
and *repr*: *f-repr i Fr1 F1* **and** *gS*: *snd* (*gram-schmidt-int n F1*) = *G*
and *len*: *length F1* = *m* **and** *HC*: *set F1* $\subseteq$ *carrier-vec n*
and *gso*: *g-repr i Gr1 F1* **and** *L*: *lattice-of F1* = *L*
using *i* **by** *auto*
from *g-i*[*OF Linv' i*] **have** *y*: *?y'* = *sq-norm ?y* **by** *auto*
from *g-im1*[*OF Linv' i i0*] **have** *x*: *?x'* = *sq-norm ?x* **by** *auto*
hence *cond*: *?cond'* = *?cond* **using** *y* **by** *auto*
from *i0* **have** (*i* = *0*) = *False* **by** *auto*
note *res* = *res*[*unfolded cond fst-conv this if-False*]
show ($\exists$ *H*. *Ex* (*LLL-invariant state H*)) $\wedge$ *LLL-measure state* < *LLL-measure* (*i*, *Fr*, *Gr*)
proof (*cases ?cond*)
case *False*
from *len inc-i*[*OF repr*] *repr i* **have** *repr'*: *f-repr* (*Suc i*) (*inc-i Fr1*) *F1* **by** *auto*
from *of-list-repr*[*OF repr'*] **have** *Hr'*: *of-list-repr* (*inc-i Fr1*) = *F1* **by** *auto*
from *False* **have** *le*: *sq-norm* (*G* ! (*i* − *1*)) $\leq$ α ∗ *sq-norm* (*G* ! *i*) **by** *force*
have *red*: *gs.weakly-reduced* α (*Suc i*) *G*

unfolding *gs.weakly-reduced-def*
proof (*intro allI impI*)
fix *k*
assume *ki*: *Suc k* < *Suc i*
show *sq-norm* (*G* ! *k*) ≤ *α* ∗ *sq-norm* (*G* ! *Suc k*)
proof (*cases Suc k* < *i*)
case *True*
from *red*[*unfolded gs.weakly-reduced-def*, *rule-format*, *OF True*] **show** *?thesis*
.
next
case *False*
with *i0 ki* **have** *id*: *k* = *i* − *1 Suc k* = *i* **by** *auto*
with *le* **show** *?thesis* **by** *auto*
qed
qed
from *res False ii*
have *state*: *state* = *increase-i* (*i*, *Fr1*, *Gr1*) **by** *auto*
have *invS*: *LLL-invariant state F1 G* **unfolding** *state*
by (*rule increase-i*[*OF Linv′ i*], *insert False*, *auto*)
obtain *Fr′ Gr′* **where** *state*: *state* = (*Suc i*, *Fr′*, *Gr′*) **using** *state*
by (*cases state*, *auto simp*: *increase-i-def*)
have *LLL-measure state* < *LLL-measure* (*i*, *Fr*, *Gr*) **unfolding** *LLL-measure-def*
logD-eq split state
LLL-invD(*1*)[*OF invS*[*unfolded state*], *symmetric*] *inv*(*1*)[*symmetric*]
using *i* **by** *simp*
thus *?thesis* **using** *invS* **by** *blast*
next
case *True*
from *i0 inv′ True i* **have** *swap*: *set F1* ⊆ *carrier-vec n i* < *length F1 i* − *1* <
length F1 i ≠ *i* − *1*
unfolding *LLL-invariant-def Let-def* **by** *auto*
define *F2* **where** *F2* = *F1*[*i* := *F1* ! (*i* − *1*), *i* − *1* := *F1* ! *i*]
define *Fr2* **where** *Fr2* = *dec-i* (*update-im1* (*update-i Fr1* (*get-nth-im1 Fr1*))
(*get-nth-i Fr1*))
from *dec-i*[*OF update-im1*[*OF update-i*[*OF repr*]], *of get-nth-im1 Fr1 get-nth-i*
Fr1, *folded Fr2-def*] *swap*(*2*) *i0*
have *list-repr* (*i* − *1*) *Fr2* (*F1*[*i* := *get-nth-im1 Fr1*, *i* − *1* := *get-nth-i Fr1*])
by *auto*
hence *repr′*: *f-repr* (*i* − *1*) *Fr2 F2* **unfolding** *F2-def*
using *get-nth-im1*[*OF repr*] *get-nth-i*[*OF repr*] *i0 swap*(*2*) **by** (*auto simp*:
map-update)
note *Hr′* = *of-list-repr*[*OF repr′*]
obtain *G2* **where** *gH′*: *snd* (*gram-schmidt-int n F2*) = *G2* **by** *force*
let *?gso′* = *let gi* = *g-i Gr1*;
gim1 = *g-im1 Gr1*;
fim1 = *get-nth-im1 Fr1*;
new-gim1 = *gi* + *mu* $\cdot_v$ *gim1*;
norm-gim1 = *sq-norm new-gim1*;
new-gi = *gim1* − (*fim1* •*i new-gim1* / *norm-gim1*) $\cdot_v$ *new-gim1*;

norm-gi = *sq-norm new-gi*
in dec-i (*update-im1* (*update-i Gr1* (*new-gi*,*norm-gi*)) (*new-gim1*,*norm-gim1*))

define *Gr2* **where** *gso'*: *Gr2* = *?gso'*
have *span'*: *gs.span* (*SRAT F1*) = *gs.span* (*SRAT F2*)
by (*rule arg-cong*[*of - - gs.span*], *unfold F2-def*, *insert swap*, *auto*)
from *res gH' Fr2-def Hr' gso' True ii*
have *state*: *state* = (*i* − *1*, *Fr2*, *Gr2*) **by** (*auto simp*: *Let-def*)
have *lF2*: *lattice-of F2* = *lattice-of F1* **unfolding** *F2-def*
by (*rule lattice-of-swap*[*OF swap refl*])
have *len'*: *length F2* = *m* **using** *inv'* **unfolding** *F2-def* **by** *auto*
have *F2*: *set F2* $\subseteq$ *carrier-vec n* **using** *swap* **unfolding** *F2-def set-conv-nth*
by (*auto*, *rename-tac k*, *case-tac k* = *i*, *force*, *case-tac k* = *i* − *1*, *auto*)
let *?rv* = *map-vec rat-of-int*
from *inv'*(*10*) **have** *indepH*: *gs.lin-indpt-list* (*RAT F1*) **.**
from *i i0 len* **have** *i* < *length* (*RAT F1*) *i* − *1* < *length* (*RAT F1*) **by** *auto*
with *distinct-swap*[*OF this*] *len* **have** *distinct* (*RAT F2*) = *distinct* (*RAT F1*) **unfolding** *F2-def*
by (*auto simp*: *map-update*)
with *len' F2 span' indepH* **have** *indepH'*: *gs.lin-indpt-list* (*RAT F2*) **unfolding** *F2-def* **using** *i i0*
by (*auto simp*: *gs.lin-indpt-list-def*)
note *conn1* = *indepH inv'*(*2*) *len*
note *conn2* = *indepH' gH' len'*
from *gram-schmidt-int-connect*[*OF conn1*]
have *Gs-fs*: *G* = *map* (*gs.gso* (*RAT F1*)) [*0*..<*m*] **.**
from *gram-schmidt-int-connect*[*OF conn2*]
have *G2-F2*: *G2* = *map* (*gs.gso* (*RAT F2*)) [*0*..<*m*] **.**
have *F2-F1*: *k* < *i* − *1* $\Longrightarrow$ *F2* ! *k* = *F1* ! *k* **for** *k* **unfolding** *F2-def* **by** *auto*
{
fix *k*
assume *ki*: *k* < *i* − *1*
with *i* **have** *kn*: *k* < *m* **by** *simp*
have *G2* ! *k* = *gs.gso* (*RAT F2*) *k* **unfolding** *G2-F2* **using** *kn* **by** *auto*
also have ... = *gs.gso* (*RAT F1*) *k*
by (*rule gs-gs-identical*, *insert ki kn len*, *auto simp*: *F2-def*)
also have ... = *G* ! *k* **unfolding** *Gs-fs* **using** *kn* **by** *auto*
finally have *G2* ! *k* = *G* ! *k* **.**
} **note** *G2-G* = *this*
have *take-eq*: *take* (*Suc i* − *1* − *1*) *F2* = *take* (*Suc i* − *1* − *1*) *F1*
by (*intro nth-equalityI*, *insert len len' i swap*(*2*−), *auto intro*!: *F2-F1*)
from *inv'* **have** *gs.weakly-reduced* α *i G* **by** *auto*
hence *gs.weakly-reduced* α (*i* − *1*) *G* **unfolding** *gs.weakly-reduced-def* **by** *auto*
hence *red*: *gs.weakly-reduced* α (*i* − *1*) *G2*
unfolding *gs.weakly-reduced-def* **using** *G2-G* **by** *auto*
from *inv'* **have** *L*: *lattice-of F2* = *L* **unfolding** *lF2* **by** *auto*
have *ii*: *g-repr* (*i* − *1*) = *g-repr* ((*Suc i* − *1*) − *1*) **using** *True* **by** *auto*
have *i1n*: *i* − *1* < *m* **using** *i* **by** *auto*
let *?R* = *rat-of-int*

let *?RV* = *map-vec ?R*
let *?f1* = λ *i*. *RAT F1* ! *i*
let *?f2* = λ *i*. *RAT F2* ! *i*
have *heq*:*F1* ! (*i* − *1*) = *F2* ! *i take* (*i*−*1*) *F1* = *take* (*i*−*1*) *F2*
?f2 (*i* − *1*) = *?f1 i ?f2 i* = *?f1* (*i* − *1*)
unfolding *F2-def* **using** *i len i0* **by** *auto*
let *?g2* = *gs.gso* (*RAT F2*)
let *?g1* = *gs.gso* (*RAT F1*)
let *?mu1* = *gs*.μ (*RAT F1*)
let *?mu2* = *gs*.μ (*RAT F2*)
from *gH′*[*unfolded gram-schmidt-int-def gram-schmidt-wit-def*] *indepH′ len′*
have *connH′*:
gs.lin-indpt-list (*RAT F2*) *length* (*RAT F2*) = *m snd* (*gs.main* (*RAT F2*)) = *G2*
by (*auto intro*: *nth-equalityI*)
from *gS*[*unfolded gram-schmidt-int-def gram-schmidt-wit-def*] *indepH len*
have *connH*:
gs.lin-indpt-list (*RAT F1*) *length* (*RAT F1*) = *m snd* (*gs.main* (*RAT F1*)) = *G*
by (*auto intro*: *nth-equalityI*)
have *gs*: $\bigwedge$ *j*. *j* < *m* $\Longrightarrow$ *?g1 j* ∈ *Rn* **using** *gs.gso-carrier*[*OF connH*] **.**
have *g*: $\bigwedge$ *j*. *j* < *m* $\Longrightarrow$ *?f1 j* ∈ *Rn* **using** *gs.f-carrier*[*OF connH*] **.**
let *?fs1* = *?f1* ‘ {*0*..< (*i* − *1*)}
have *G*: *?fs1* ⊆ *Rn* **using** *g i* **by** *auto*
let *?gs1* = *?g1* ‘ {*0*..< (*i* − *1*)}
have *G′*: *?gs1* ⊆ *Rn* **using** *gs i* **by** *auto*
let *?S* = *gs.span ?fs1*
let *?S′* = *gs.span ?gs1*
have *S′S*: *?S′* = *?S*
by (*rule gs.partial-span′*[*OF connH*], *insert i*, *auto*)
have *gs.is-projection* (*?g2* (*i* − *1*)) (*gs.span* (*?g2* ‘ {*0*..< (*i* − *1*)})) (*?f2* (*i* − *1*))
by (*rule gs.gso-projection-span*(*2*)[*OF connH′* ‹*i* − *1* < *m*›])
also have *?f2* (*i* − *1*) = *?f1 i* **unfolding** *F2-def* **using** *len i* **by** *auto*
also have *gs.span* (*?g2* ‘ {*0* ..< (*i* − *1*)}) = *gs.span* (*?f2* ‘ {*0* ..< (*i* − *1*)})
by (*rule gs.partial-span′*[*OF connH′*], *insert i*, *auto*)
also have *?f2* ‘ {*0* ..< (*i* − *1*)} = *?fs1*
by (*rule image-cong*[*OF refl*], *insert len i*, *auto simp*: *F2-def*)
finally have *claim1*: *gs.is-projection* (*?g2* (*i* − *1*)) *?S* (*?f1 i*) **.**
have *?f1 i* = *gs.sumlist* (*map* (λ*j*. *?mu1 i j* $\cdot_v$ *?g1 j*) [*0* ..< *i*] @ [*?g1 i*])
unfolding *gs.fi-is-sum-of-mu-gso*[*OF connH* ‹*i* < *m*›] **by** (*simp add*: *gs*.μ.*simps*)
also have . . . = *gs.sumlist* (*map* (λ*j*. *?mu1 i j* $\cdot_v$ *?g1 j*) [*0* ..< *i*]) + *?g1 i*
by (*subst gs.sumlist-append*, *insert i gs*, *auto*)
finally have *claim2*: *?f1 i* = *gs.sumlist* (*map* (λ*j*. *?mu1 i j* $\cdot_v$ *?g1 j*) [*0* ..< *i*]) + *?g1 i* (**is** - = *?sum* + -) **.**
have *sum*: *?sum* ∈ *Rn* **by** (*rule gs.sumlist-carrier*, *insert gs i*, *auto*)
from *gs.span-closed*[*OF G*] **have** *S*: *?S* ⊆ *Rn* **by** *auto*
from *gs i* **have** *gs′*: $\bigwedge$ *j*. *j* < *i* − *1* $\Longrightarrow$ *?g1 j* ∈ *Rn* **and** *gsi*: *?g1* (*i* − *1*) ∈ *Rn* **by** *auto*

have $[0 ..< i] = [0 ..< Suc\ (i - 1)]$ **using** *i0* **by** *simp*
also have $\ldots = [0 ..< i - 1]$ @ $[i - 1]$ **by** *simp*
finally have *list*: $[0 ..< i] = [0 ..< i - 1]$ @ $[i - 1]$ **.**
have *g2-im1*: $?g2\ (i - 1) = ?g1\ i + ?mu1\ i\ (i - 1) \cdot_v ?g1\ (i - 1)$ (**is** - = - + *?mu-f1*)
proof (*rule gs.is-projection-eq*[*OF connH claim1 - S g*[*OF i*]])
show *gs.is-projection* (*?g1 i* + *?mu-f1*) *?S* (*?f1 i*) **unfolding** *gs.is-projection-def*
proof (*intro conjI allI impI*)
let $?sum' = gs.sumlist\ (map\ (\lambda j.\ ?mu1\ i\ j \cdot_v ?g1\ j)\ [0 ..< i - 1])$
have *sum′*: $?sum' \in Rn$ **by** (*rule gs.sumlist-carrier*, *insert gs i*, *auto*)
show *inRn*: $(?g1\ i + ?mu\text{-}f1) \in Rn$ **using** *gs*[*OF i*] *gsi i* **by** *auto*
have *carr*: $?sum \in Rn$ $?g1\ i \in Rn$ $?mu\text{-}f1 \in Rn$ $?sum' \in Rn$ **using** *sum′ sum gs*[*OF i*] *gsi i* **by** *auto*
have $?f1\ i - (?g1\ i + ?mu\text{-}f1) = (?sum + ?g1\ i) - (?g1\ i + ?mu\text{-}f1)$
unfolding *claim2* **by** *simp*
also have $\ldots = ?sum - ?mu\text{-}f1$ **using** *carr* **by** *auto*
also have $?sum = gs.sumlist\ (map\ (\lambda j.\ ?mu1\ i\ j \cdot_v ?g1\ j)\ [0 ..< i - 1]$ @ $[?mu\text{-}f1])$
unfolding *list* **by** *simp*
also have $\ldots = ?sum' + ?mu\text{-}f1$
by (*subst gs.sumlist-append*, *insert gs′ gsi*, *auto*)
also have $\ldots - ?mu\text{-}f1 = ?sum'$ **using** *sum′ gsi* **by** *auto*
finally have *id*: $?f1\ i - (?g1\ i + ?mu\text{-}f1) = ?sum'$ **.**
show $?f1\ i - (?g1\ i + ?mu\text{-}f1) \in gs.span\ ?S$ **unfolding** *id gs.span-span*[*OF G*]
proof (*rule gs.sumlist-in-span*[*OF G*])
fix v
assume $v \in set\ (map\ (\lambda j.\ ?mu1\ i\ j \cdot_v ?g1\ j)\ [0 ..< i - 1])$
then obtain j **where** j: $j < i - 1$ **and** v: $v = ?mu1\ i\ j \cdot_v ?g1\ j$ **by** *auto*
show $v \in ?S$ **unfolding** v
by (*rule gs.smult-in-span*[*OF G*], *unfold S′S*[*symmetric*], *rule gs.span-mem*, *insert gs i j*, *auto*)
qed
fix x
assume $x \in ?S$
hence x: $x \in ?S'$ **using** *S′S* **by** *simp*
show $(?g1\ i + ?mu\text{-}f1) \cdot x = 0$
proof (*rule gs.orthocompl-span*[*OF connH - G′ inRn x*])
fix x
assume $x \in ?gs1$
then obtain j **where** j: $j < i - 1$ **and** *x-id*: $x = ?g1\ j$ **by** *auto*
from *j i x-id gs*[*of j*] **have** x: $x \in Rn$ **by** *auto*
{
fix k
assume k: $k > j$ $k < m$
have $?g1\ k \cdot x = 0$ **unfolding** *x-id*
by (*rule gs.orthogonal*[*OF connH*], *insert k*, *auto*)
}
from *this*[*of i*] *this*[*of i* − *1*] *j i*

have *main*: *?g1 i* · *x* = *0* *?g1* (*i* − *1*) · *x* = *0* **by** *auto*
have (*?g1 i* + *?mu-f1*) · *x* = *?g1 i* · *x* + *?mu-f1* · *x*
by (*rule add-scalar-prod-distrib*[*OF gs*[*OF i*] - *x*], *insert gsi*, *auto*)
also have ... = *0* **using** *main*
by (*subst smult-scalar-prod-distrib*[*OF gsi x*], *auto*)
finally show (*?g1 i* + *?mu-f1*) · *x* = *0* **.**
qed
qed
qed
{
fix *k*
assume *kn*: *k* < *m*
and *ki*: *k* ≠ *i k* ≠ *i* − *1*
have *?g2 k* = *gs.projection* (*gs.span* (*?g2* ' {*0*..<*k*})) (*?f2 k*)
by (*rule gs.gso-projection-span*[*OF connH' kn*])
also have *gs.span* (*?g2* ' {*0*..<*k*}) = *gs.span* (*?f2* ' {*0*..<*k*})
by (*rule gs.partial-span'*[*OF connH'*], *insert kn*, *auto*)
also have *?f2* ' {*0*..<*k*} = *?f1* ' {*0*..<*k*}
proof(*cases k*≤*i*)
case *True* **hence** *k* < *i* − *1* **using** *ki* **by** *auto*
then show *?thesis* **apply**(*intro image-cong*) **unfolding** *F2-def* **using** *len i* **by** *auto*
next
case *False*
have *?f2* ' {*0*..<*k*} = *Fun.swap i* (*i* − *1*) *?f1* ' {*0*..<*k*}
unfolding *Fun.swap-def F2-def o-def* **using** *len i*
by (*intro image-cong*, *insert len kn*, *force*+)
also have ... = *?f1* ' {*0*..<*k*}
apply(*rule swap-image-eq*) **using** *False* **by** *auto*
finally show *?thesis***.**
qed
also have *gs.span* ... = *gs.span* (*?g1* ' {*0*..<*k*})
by (*rule sym*, *rule gs.partial-span'*[*OF connH*], *insert kn*, *auto*)
also have *?f2 k* = *?f1 k* **using** *ki kn len* **unfolding** *F2-def* **by** *auto*
also have *gs.projection* (*gs.span* (*?g1* ' {*0*..<*k*})) ... = *?g1 k*
by (*subst gs.gso-projection-span*[*OF connH kn*], *auto*)
finally have *?g2 k* = *?g1 k* **.**
} **note** *g2-g1-identical* = *this*
{
fix *jj*
assume *jj*: *jj* < *i* − *1*
hence *id1*: *jj* < *i* − *1* ⟷ *True jj* < *i* ⟷ *True* **by** *auto*
have *id2*: *?g2 jj* = *?g1 jj* **by** (*subst g2-g1-identical*, *insert jj i*, *auto*)
have *?mu2 i jj* = *?mu1* (*i* − *1*) *jj*
unfolding *gs.μ.simps id1 id2 if-True* **using** *len i i0* **by** (*auto simp*: *F2-def*)
} **note** *mu'-mu-i* = *this*
let *?g2-im1* = *?g2* (*i* − *1*)
have *g2-im1-Rn*: *?g2-im1* ∈ *Rn* **using** *i* **by** (*auto intro*!: *gs.gso-carrier*[*OF connH'*])

```
{
  let ?mu2-f2 = λ j. − ?mu2 i j ·v ?g2 j
  let ?sum = gs.sumlist (map (λj. − ?mu1 (i − 1) j ·v ?g1 j) [0 ..< i − 1])
  have mhs: ?mu2-f2 (i − 1) ∈ Rn using i by (auto intro!: gs.gso-carrier[OF connH'])
  have sum': ?sum ∈ Rn by (rule gs.sumlist-carrier, insert gs i, auto)
  have gim1: ?f1 (i − 1) ∈ Rn using g i by auto
  have ?g2 i = ?f2 i + gs.sumlist (map ?mu2-f2 [0 ..< i−1] @ [?mu2-f2 (i−1)])
    unfolding gs.gso.simps[of - i] list by simp
  also have ?f2 i = ?f1 (i − 1) unfolding F2-def using len i i0 by auto
  also have map ?mu2-f2 [0 ..< i−1] = map (λj. − ?mu1 (i − 1) j ·v ?g1 j) [0 ..< i − 1]
      by (rule map-cong[OF refl], subst g2-g1-identical, insert i, auto simp: mu'-mu-i)
  also have gs.sumlist (... @ [?mu2-f2 (i − 1)]) = ?sum + ?mu2-f2 (i − 1)
    by (subst gs.sumlist-append, insert gs i mhs, auto)
  also have ?f1 (i − 1) + ... = (?f1 (i − 1) + ?sum) + ?mu2-f2 (i − 1)
    using gim1 sum' mhs by auto
  also have ?f1 (i − 1) + ?sum = ?g1 (i − 1) unfolding gs.gso.simps[of - i − 1] by simp
  also have ?mu2-f2 (i − 1) = − (?f2 i • ?g2-im1 / sq-norm ?g2-im1) ·v ?g2-im1 unfolding gs.μ.simps using i0 by simp
  also have ... = − ((?f2 i • ?g2-im1 / sq-norm ?g2-im1) ·v ?g2-im1) by auto
  also have ?g1 (i − 1) + ... = ?g1 (i − 1) − ((?f2 i • ?g2-im1 / sq-norm ?g2-im1) ·v ?g2-im1)
    by (rule sym, rule minus-add-uminus-vec[of - n], insert gsi g2-im1-Rn, auto)
  also have ?f2 i = ?f1 (i − 1) by fact
  finally have ?g2 i = ?g1 (i − 1) − (?f1 (i − 1) • ?g2 (i − 1) / sq-norm (?g2 (i − 1))) ·v ?g2 (i − 1) .
} note g2-i = this
{
  from i1n have i1n': i − 1 ≤ m by simp
  have upd-im1: list-repr i ba xs ⟹ ys = (xs [i − 1 := x]) ⟹ list-repr i (update-im1 ba x) ys
    for ba xs x ys using update-im1[of i ba xs] i0 by force
  from gso[unfolded g-repr-def]
  have gsoH: list-repr i Gr1 (map (λx. (x, ‖x‖²)) (map (GSO F1) [0..<m])) by auto
  let ?f-im1 = get-nth-im1 Fr1
  let ?g2'-im1 = g-i Gr1 + mu ·v g-im1 Gr1
  let ?norm-im1 = sq-norm ?g2'-im1
  let ?g2'-i = g-im1 Gr1 − (?f-im1 •i ?g2'-im1 / ?norm-im1) ·v ?g2'-im1
  define g2'-i where g2'-i = ?g2'-i
  define g2'-im1 where g2'-im1 = ?g2'-im1
  have ?g2 (i − 1) = ?g1 i + ?mu1 i (i − 1) ·v ?g1 (i − 1) by fact
  also have ?g1 i = g-i Gr1 unfolding g-i[OF Linv' i] Gs-fs o-def using i by simp
  also have ?g1 (i − 1) = g-im1 Gr1 unfolding g-im1[OF Linv' i i0] Gs-fs
```

o-def **using** *i* **by** *simp*
finally have *g2im1*: *?g2 (i − 1) = g2'-im1*
unfolding *mu g2'-im1-def* **by** *blast*
have *?f-im1 ∈ carrier-vec n* **using** *inv'(3−4) ‹i − 1 < m›* **unfolding** *get-nth-im1[OF inv'(8) i0]*
by *auto*
hence *dim*: *dim-vec ?f-im1 = n dim-vec ?g2'-im1 = n* **unfolding** *g2'-im1-def[symmetric] g2im1[symmetric]*
using *‹?g2-im1 ∈ Rn›* **by** *auto*
have *?g2 i = ?g1 (i − 1) − (?f1 (i − 1) · ?g2 (i − 1) / sq-norm (?g2 (i − 1))) $\cdot_v$?g2 (i − 1)*
by *(rule g2-i)*
also have *?g2 (i − 1) = g2'-im1* **by** *(simp add: g2im1[symmetric])*
also have *?g1 (i − 1) = g-im1 Gr1* **by** *fact*
also have *?f1 (i − 1) = map-vec of-int ?f-im1*
unfolding *get-nth-im1[OF repr i0] o-def* **using** *len i* **by** *simp*
finally have *g2i*: *?g2 i = g2'-i* **using** *dim* **unfolding** *g2'-i-def g2'-im1-def* **by** *simp*
have *map (λx. (x, ‖x‖²)) (map (GSO F1) [0..<m])*
[i := (g2'-i, sq-norm g2'-i), i − 1 := (g2'-im1, sq-norm g2'-im1)]
= map (λx. (x, ‖x‖²)) ((map (GSO F1) [0..<m])[i := g2'-i, i − 1 := g2'-im1])
unfolding *map-update* **by** *auto*
also have *GSOH*: *map (GSO F1) [0..<m] = map ?g1 [0..<m]*
by *(rule map-cong[OF refl], auto simp: GSO-def len intro!: gs-gs-identical)*
also have *id*: *... [i := g2'-i, i − 1 := g2'-im1] = map ?g2 [0..<m]* (**is** *?Gs = ?Gs2*)
proof −
{
fix *k*
assume *k*: *k < m*
consider *(ki) k = i | (im1) k = i − 1 | (other) k ∉ {i−1, i}* **by** *auto*
hence *?Gs ! k = ?g2 k*
proof *(cases)*
case *other*
hence *?Gs ! k = ?g1 k* **using** *k* **by** *simp*
also have *... = ?g2 k* **using** *g2-g1-identical[OF k] other* **by** *auto*
finally show *?thesis* **.**
next
case *ki*
have *?g2 i = g2'-i* **unfolding** *g2i* **..**
also have *... = ?Gs ! k* **using** *i len i0 ki* **by** *simp*
finally show *?thesis* **unfolding** *ki* **by** *simp*
next
case *im1*
hence *?Gs ! k = g2'-im1* **using** *i len* **by** *simp*
also have *... = ?g2 (i − 1)* **unfolding** *g2im1* **..**
finally show *?thesis* **unfolding** *im1* **by** *simp*
qed

also have *?g2 k = ?Gs2 ! k* **using** *k* **by** *simp*
finally have *?Gs ! k = ?Gs2 ! k* **by** *simp*
} **note** *main = this*
show *?thesis*
by (*rule nth-equalityI*, *force*, *insert main*, *auto*)
qed
also have ... = *map* (*GSO F2*) [*0..<m*]
by (*rule map-cong*[*OF refl*], *auto simp*: *GSO-def len′ intro*!: *gs-gs-identical*)
finally
have *map* (λ*x*. (*x*, $\|x\|^2$)) (*map* (*GSO F2*) [*0..<m*])
= *map* (λ*x*. (*x*, $\|x\|^2$)) (*map* (*GSO F1*) [*0..<m*])[*i* := (*g2′-i*, $\|g2'\text{-}i\|^2$), *i* − *1* := (*g2′-im1*, $\|g2'\text{-}im1\|^2$)]
unfolding *GSOH o-def* **by** *auto*
hence *g-repr* (*i* − *1*) *Gr2 F2* **unfolding** *gso′ Let-def* **unfolding** *g-repr-def Let-def g2′-i-def*[*symmetric*]
unfolding *g2′-im1-def*[*symmetric*] **apply** (*intro conjI i1n′*)
apply(*rule dec-i*[*OF - i0*]) **by**(*auto simp*: *i intro*!:*upd-im1 update-i*[*OF gsoH*])
} **note** *g-repr = this*

have *newInv*: *LLL-invariant* (*i* − *1*, *Fr2*, *Gr2*) *F2 G2*
by (*rule LLL-invI*[*OF repr′ g-repr gH′ L red*], *insert connH′ len′ span′ i m12 F2*, *auto*)

{
have *sq-norm* (*?g2* (*i* − *1*)) = *sq-norm* (*?g1 i* + *?mu-f1*) **unfolding** *g2-im1* **by** *simp*
also have ... = (*?g1 i* + *?mu-f1*) • (*?g1 i* + *?mu-f1*)
by (*simp add*: *sq-norm-vec-as-cscalar-prod*)
also have ... = (*?g1 i* + *?mu-f1*) • *?g1 i* + (*?g1 i* + *?mu-f1*) • *?mu-f1*
by (*rule scalar-prod-add-distrib*, *insert gs i*, *auto*)
also have (*?g1 i* + *?mu-f1*) • *?g1 i* = *?g1 i* • *?g1 i* + *?mu-f1* • *?g1 i*
by (*rule add-scalar-prod-distrib*, *insert gs i*, *auto*)
also have (*?g1 i* + *?mu-f1*) • *?mu-f1* = *?g1 i* • *?mu-f1* + *?mu-f1* • *?mu-f1*
by (*rule add-scalar-prod-distrib*, *insert gs i*, *auto*)
also have *?mu-f1* • *?g1 i* = *?g1 i* • *?mu-f1*
by (*rule comm-scalar-prod*, *insert gs i*, *auto*)
also have *?g1 i* • *?g1 i* = *sq-norm* (*?g1 i*)
by (*simp add*: *sq-norm-vec-as-cscalar-prod*)
also have *?g1 i* • *?mu-f1* = *?mu1 i* (*i* − *1*) ∗ (*?g1 i* • *?g1* (*i* − *1*))
by (*rule scalar-prod-smult-right*, *insert gs*[*OF i*] *gs*[*OF* ‹*i* − *1* < *m*›], *auto*)
also have *?g1 i* • *?g1* (*i* − *1*) = *0*
using *corthogonalD*[*OF gs.gram-schmidt*(*2*)[*OF connH*], *of i i* − *1*] *i len i0* **unfolding** *Gs-fs*
by (*auto simp*: *o-def*)
also have *?mu-f1* • *?mu-f1* = *?mu1 i* (*i* − *1*) ∗ (*?mu-f1* • *?g1* (*i* − *1*))
by (*rule scalar-prod-smult-right*, *insert gs*[*OF i*] *gs*[*OF* ‹*i* − *1* < *m*›], *auto*)
also have *?mu-f1* • *?g1* (*i* − *1*) = *?mu1 i* (*i* − *1*) ∗ (*?g1* (*i* − *1*) • *?g1* (*i* − *1*))

by (*rule scalar-prod-smult-left*, *insert gs*[*OF i*] *gs*[*OF* ‹*i* − *1* < *m*›], *auto*)
also have *?g1* (*i* − *1*) · *?g1* (*i* − *1*) = *sq-norm* (*?g1* (*i* − *1*))
by (*simp add*: *sq-norm-vec-as-cscalar-prod*)
finally have *sq-norm* (*?g2* (*i* − *1*)) =
sq-norm (*?g1 i*) + (*?mu1 i* (*i* − *1*) ∗ *?mu1 i* (*i* − *1*)) ∗ *sq-norm* (*?g1* (*i* − *1*))
by (*simp add*: *ac-simps o-def*)
also have ... < *1*/α ∗ (*sq-norm* (*?g1* (*i* − *1*))) + (*1*/*2* ∗ *1*/*2*) ∗ (*sq-norm* (*?g1* (*i* − *1*)))
proof (*rule add-less-le-mono*[*OF* - *mult-mono*])
from *True*[*unfolded mult.commute*[*of* α] *Gs-fs*,
THEN linordered-field-class.mult-imp-less-div-pos[*OF* α*0*(*1*)]]
show *sq-norm* (*?g1 i*) < *1*/α ∗ (*sq-norm* (*?g1* (*i* − *1*)))
unfolding *Gs-fs o-def* **using** *len i* **by** *auto*
from *m12* **have** *abs*: *abs* (*?mu1 i* (*i* − *1*)) ≤ *1*/*2* **unfolding** *mu* **by** *auto*
have *?mu1 i* (*i* − *1*) ∗ *?mu1 i* (*i* − *1*) ≤ *abs* (*?mu1 i* (*i* − *1*)) ∗ *abs* (*?mu1 i* (*i* − *1*)) **by** *auto*
also have ... ≤ *1*/*2* ∗ *1*/*2* **using** *mult-mono*[*OF abs abs*] **by** *auto*
finally show *?mu1 i* (*i* − *1*) ∗ *?mu1 i* (*i* − *1*) ≤ *1*/*2* ∗ *1*/*2* **by** *auto*
qed *auto*
also have ... = *reduction* ∗ *sq-norm* (*?g1* (*i* − *1*)) **unfolding** *reduction-def*

using α*0* **by** (*simp add*: *ring-distribs add-divide-distrib*)
finally have *sq-norm* (*?g2* (*i* − *1*)) < *reduction* ∗ *sq-norm* (*?g1* (*i* − *1*)) **.**
} **note** *g-reduction* = *this*
have *norm-pos*: *j* < *m* ⟹ *sq-norm* (*?g2 j*) > *0* **for** *j*
using *gs.sq-norm-pos*[*OF connH'*,*of j*] **unfolding** *G2-F2 o-def* **by** *simp*
{
fix *k*
assume *k*: *k* = *i*
hence *kn*: *k* ≤ *m* **using** *i* **by** *auto*
from *Gd*[*OF newInv*, *folded dk-def*, *folded state*, *OF kn*] *k*
have *?R* (*dk k F2*) = (∏*j*<*i*. *sq-norm* (*G2* ! *j*)) **by** *auto*
also have ... = *prod* (λ *j*. *sq-norm* (*?g2 j*)) ({*0* ..< *i*−*1*} ∪ {*i* − *1*})
by (*rule sym*, *rule prod.cong*, (*insert i0*, *auto*)[*1*], *insert G2-F2 i*, *auto simp*: *o-def*)
also have ... = *sq-norm* (*?g2* (*i* − *1*)) ∗ *prod* (λ *j*. *sq-norm* (*?g2 j*)) ({*0* ..< *i*−*1*})
by *simp*
also have ... < (*reduction* ∗ *sq-norm* (*?g1* (*i* − *1*))) ∗ *prod* (λ *j*. *sq-norm* (*?g2 j*)) ({*0* ..< *i*−*1*})
by (*rule mult-strict-right-mono*[*OF g-reduction prod-pos*], *insert norm-pos i*, *auto*)
also have *prod* (λ *j*. *sq-norm* (*?g2 j*)) ({*0* ..< *i*−*1*}) = *prod* (λ *j*. *sq-norm* (*?g1 j*)) ({*0* ..< *i*−*1*})
by (*rule prod.cong*[*OF refl*], *subst g2-g1-identical*, *insert i*, *auto*)
also have (*reduction* ∗ *sq-norm* (*?g1* (*i* − *1*))) ∗ *prod* (λ *j*. *sq-norm* (*?g1 j*)) ({*0* ..< *i*−*1*})
= *reduction* ∗ *prod* (λ *j*. *sq-norm* (*?g1 j*)) ({*0* ..< *i*−*1*} ∪ {*i* − *1*}) **by** *simp*

also have *prod* (λ *j*. *sq-norm* (*?g1 j*)) ($\{0 ..< i-1\} \cup \{i - 1\}$) = ($\prod j<i$. *sq-norm* (*?g1 j*))
by (*rule prod.cong*, *insert i0*, *auto*)
also have ... = *?R* (*dk k F1*) **unfolding** *dk-def Gd*[*OF Linv′ kn*] **unfolding** *k*
by (*rule prod.cong*[*OF refl*], *insert i*, *auto simp*: *Gs-fs o-def*)
also have ... = *?R* (*dk k F*) **unfolding** *dk-eq*[*OF kn*] **by** *simp*
finally have *dk k F2* < *real-of-rat reduction* $*$ *dk k F*
using *of-rat-less of-rat-mult of-rat-of-int-eq* **by** *metis*
} **note** *dk-i* = *this*[*OF refl*]
{
fix *k*
assume *kn*: $k \leq m$ **and** *ki*: $k \neq i$
from *dk-swap-unchanged*[*OF len i0 i ki kn F2-def*] *dk-eq*[*OF kn*]
have *dk k F* = *dk k F2* **by** *simp*
} **note** *dk* = *this*
have *pos*: $k < m \Longrightarrow 0 <$ *dk k F2* $k < m \Longrightarrow 0 \leq$ *dk k F2* **for** *k*
using *LLL-dk-pos*[*OF newInv*, *folded state*, *of k*] **by** *auto*
have *prodpos*:0< ($\prod i<m$. *dk i F2*) **apply** (*rule prod-pos*)
using *LLL-dk-pos*[*OF newInv*, *folded state*] **by** *auto*
have *prod-pos′*:0 < ($\prod x \in \{0..<m\} - \{i\}$. *real-of-int* (*dk x F2*)) **apply** (*rule prod-pos*)
using *LLL-dk-pos*[*OF newInv*, *folded state*] *pos* **by** *auto*
have *prod-nonneg*:0 $\leq$ ($\prod x \in \{0..<m\} - \{i\}$. *real-of-int* (*dk x F2*)) **apply** (*rule prod-nonneg*)
using *LLL-dk-pos*[*OF newInv*, *folded state*] *pos* **by** *auto*
have *prodpos2*:0<($\prod ia<m$. *dk ia F*) **apply** (*rule prod-pos*)
using *LLL-dk-pos*[*OF assms*(*1*)] **by** *auto*
have *D F2* = *real-of-int* ($\prod i<m$. *dk i F2*) **unfolding** *D-def* **using** *prodpos* **by** *simp*
also have ($\prod i<m$. *dk i F2*) = ($\prod j \in \{0 ..< m\} - \{i\} \cup \{i\}$. *dk j F2*)
by (*rule prod.cong*, *insert i*, *auto*)
also have *real-of-int* ... = *real-of-int* ($\prod j \in \{0 ..< m\} - \{i\}$. *dk j F2*) $*$ *real-of-int* (*dk i F2*)
by (*subst prod.union-disjoint*, *auto*)
also have ... < ($\prod j \in \{0 ..< m\} - \{i\}$. *dk j F2*) $*$ (*of-rat reduction* $*$ *dk i F*)
by(*rule mult-strict-left-mono*[*OF dk-i*],*insert prod-pos′*,*auto*)
also have ($\prod j \in \{0 ..< m\} - \{i\}$. *dk j F2*) = ($\prod j \in \{0 ..< m\} - \{i\}$. *dk j F*)
by (*rule prod.cong*, *insert dk*, *auto*)
also have ... $*$ (*of-rat reduction* $*$ *dk i F*)
= *of-rat reduction* $*$ ($\prod j \in \{0 ..< m\} - \{i\} \cup \{i\}$. *dk j F*)
by (*subst prod.union-disjoint*, *auto*)
also have ($\prod j \in \{0 ..< m\} - \{i\} \cup \{i\}$. *dk j F*) = ($\prod j<m$. *dk j F*)
by (*subst prod.cong*, *insert i*, *auto*)
finally have *D*: *D F2* < *real-of-rat reduction* $*$ *D F*
unfolding *D-def* **using** *prodpos2* **by** *auto*
have *logD*: *logD F2* < *logD F*

proof (*cases* $\alpha = 4/3$)
case *True*
show *?thesis* **using** *D* **unfolding** *reduction*(*4*)[*OF True*] *logD-def* **unfolding** *True* **by** *simp*
next
case *False*
hence *False'*: $\alpha = 4/3 \longleftrightarrow$ *False* **by** *simp*
from *False* α **have** $\alpha > 4/3$ **by** *simp*
with *reduction* **have** *reduction1*: *reduction* < *1* **by** *simp*
let *?new* = *real* (*D F2*)
let *?old* = *real* (*D F*)
let *?log* = *log* (*1*/*of-rat reduction*)
note *pos* = *LLL-D-pos*[*OF newInv*[*folded state*]] *LLL-D-pos*[*OF assms*(*1*)]
from *reduction* **have** *real-of-rat reduction* > *0* **by** *auto*
hence *gediv*:*1*/*real-of-rat reduction* > *0* **by** *auto*
have (*1*/*of-rat reduction*) ∗ *?new* ≤ ((*1*/*of-rat reduction*) ∗ *of-rat reduction*) ∗ *?old*
unfolding *mult.assoc real-mult-le-cancel-iff2*[*OF gediv*] **using** *D* **by** *simp*
also have (*1*/*of-rat reduction*) ∗ *of-rat reduction* = *1* **using** *reduction* **by** *auto*
finally have (*1*/*of-rat reduction*) ∗ *?new* ≤ *?old* **by** *auto*
hence *?log* ((*1*/*of-rat reduction*) ∗ *?new*) ≤ *?log ?old*
by (*subst log-le-cancel-iff*, *auto simp*: *pos reduction1 reduction*)
hence *floor* (*?log* ((*1*/*of-rat reduction*) ∗ *?new*)) ≤ *floor* (*?log ?old*)
by (*rule floor-mono*)
hence *nat* (*floor* (*?log* ((*1*/*of-rat reduction*) ∗ *?new*))) ≤ *nat* (*floor* (*?log ?old*)) **by** *simp*
also have . . . = *logD F* **unfolding** *logD-def False'* **by** *simp*
also have *?log* ((*1*/*of-rat reduction*) ∗ *?new*) = *1* + *?log ?new*
by (*subst log-mult*, *insert reduction reduction1*, *auto simp*: *pos*)
also have *floor* (*1* + *?log ?new*) = *1* + *floor* (*?log ?new*) **by** *simp*
also have *nat* (*1* + *floor* (*?log ?new*)) = *1* + *nat* (*floor* (*?log ?new*))
by (*subst nat-add-distrib*, *insert pos reduction reduction1*, *auto*)
also have *nat* (*floor* (*?log ?new*)) = *logD F2* **unfolding** *logD-def False'* **by** *simp*
finally show *logD F2* < *logD F* **by** *simp*
qed
hence *LLL-measure state* < *LLL-measure* (*i*, *Fr*, *Gr*) **unfolding** *LLL-measure-def state split*
inv(*1*)[*symmetric*] *of-list-repr*[*OF repr'*]
using *i logD* **by** *simp*
thus *?thesis* **using** *newInv* **unfolding** *state* **by** *auto*
qed
next
case *i0*: *True*
from *res i0* **have** *state*: *state* = *increase-i* (*i*, *Fr*, *Gr*) **unfolding** *basis-reduction-step-def* **by** *auto*
with *increase-i*[*OF inv i*] *i0*
have *inv'*: *LLL-invariant state F G* **by** *auto*

from *LLL-invD*[*OF inv*] **have** *Gr*: *of-list-repr Fr* = *F* **by** *simp*
from *LLL-invD*[*OF inv*′[*unfolded increase-i-def state split*]]
have *Gr*′: *of-list-repr* (*inc-i Fr*) = *F* **by** *simp*
have *id*: *of-list-repr* (*inc-i Fr*) = *of-list-repr Fr* **by** (*simp add*: *Gr Gr*′)
have *dec*: *LLL-measure state* < *LLL-measure* (*i*, *Fr*, *Gr*) **using** *i* **unfolding** *state i0*
unfolding *LLL-measure-def* **by** (*simp add*: *increase-i-def id*)
show ($\exists$ *H*. *Ex* (*LLL-invariant state H*)) $\wedge$ *LLL-measure state* < *LLL-measure* (*i*, *Fr*, *Gr*)
by (*intro conjI exI dec*, *rule inv*′)
qed

lemma *D-approx*: **assumes** *LLL-invariant* (*i*, *Fr*, *Gr*) *F G*
shows $D\ F \le nat\ (\prod i{<}m.\ (\prod j{<}i.\ \|F\ !\ j\|^2))$
proof −
note *inv* = *LLL-invD*[*OF assms*]
note *conn* = *LLL-connect*[*OF assms*]
note *main* = *inv*(*2*)[*unfolded gram-schmidt-int-def gram-schmidt-wit-def*]
have *rat-of-int* ($\prod$ *i*<*m*. *dk i F*) = ($\prod$ *i*<*m*. *rat-of-int* (*dk i F*)) **by** *simp*
also have ... = $(\prod i{<}m.\ (\prod j{<}i.\ \|G\ !\ j\|^2))$ **unfolding** *dk-def*
by (*rule prod.cong*, *auto simp*: *Gramian-determinant* [*OF assms*])
also have ... = $(\prod i{<}m.\ (\prod j{<}i.\ \|gs.gso\ (RAT\ F)\ j\|^2))$
by (*intro prod.cong arg-cong*[*of - - sq-norm-vec*], *insert conn*, *auto*)
also have ... $\le (\prod i{<}m.\ (\prod j{<}i.\ of\text{-}int\ \|F\ !\ j\|^2))$
by (*intro prod-mono ballI conjI prod-nonneg*, *insert gs.sq-norm-gso-le-f*[*OF inv*(*10*) - *main*]
inv(*4*), *auto simp*: *sq-norm-of-int*)
also have ... = $of\text{-}int\ (\prod i{<}m.\ (\prod j{<}i.\ \|F\ !\ j\|^2))$
by *simp*
finally show $D\ F \le nat\ (\prod i{<}m.\ (\prod j{<}i.\ \|F\ !\ j\|^2))$ **unfolding** *D-def of-int-le-iff*
by (*simp add*: *prod-nonneg sq-norm-vec-ge-0*)
qed

lemma *LLL-measure-approx*: **assumes** *inv*: *LLL-invariant* (*i*, *Fr*, *Gr*) *F G*
and $\alpha > 4/3$
shows *LLL-measure* (*i*, *Fr*, *Gr*) ≤ *m* + *2* ∗ *m* ∗
$(\sum i{<}m.\ log\ ((4 * of\text{-}rat\ \alpha)\ /\ (4 + of\text{-}rat\ \alpha))\ (of\text{-}int\ \|F\ !\ i\|^2))$
proof −
have *id*: *1* / *real-of-rat reduction* = (*4* ∗ *of-rat* α) / (*4* + *of-rat* α)
unfolding *reduction-def of-rat-divide of-rat-add of-rat-mult* **by** *simp*
define *b* **where** *b* = (*1* / *real-of-rat reduction*)
have *b1*: *b* > *1* **using** *reduction*(*3*)[*OF assms*(*2*)] *reduction*(*1*) **unfolding** *b-def* **by** *auto*
from *LLL-D-pos*[*OF inv*] **have** *D1*: *real* (*D F*) ≥ *1* **by** *auto*
note *invD* = *LLL-invD*[*OF inv*]
from *invD*
have *F*: *set F* ⊆ *carrier-vec n* **and** *len*: *length F* = *m* **by** *auto*
from *gs.lin-indpt-list-nonzero*[*OF invD*(*10*)]
have 0_v *n* ∉ *set* (*RAT F*) **by** *auto*

hence $0_v\ n \notin set\ F$ **using** F **by** *force*
hence 0: $\bigwedge i.\ i < m \Longrightarrow sq\text{-}norm\ (F\ !\ i) \neq 0$ **using** F *sq-norm-vec-eq-0*[*of* F ! i n **for** i]
unfolding *set-conv-nth len* **by** *force*
have 1: $i < m \Longrightarrow sq\text{-}norm\ (F\ !\ i) \geq 1$ **for** i **using** 0[*of* i] *sq-norm-vec-ge-0*[*of* F ! i] **by** *simp*
from *D-approx*[*OF inv*]
have $D\ F \leq nat\ (\prod i{<}m.\ \prod j{<}i.\ \|F\ !\ j\|^2)$ **by** *auto*
also have $\ldots \leq nat\ (\prod i{<}m.\ \prod j{<}m.\ \|F\ !\ j\|^2)$
proof (*intro nat-mono prod-mono ballI conjI prod-nonneg*)
fix i
assume i: $i \in \{..{<}m\}$
hence *id*: $\{..{<}m\} = \{..{<}i\} \cup \{i\ ..{<}\ m\}$ **by** *auto*
have $(\prod j{<}i.\ \|F\ !\ j\|^2) = (\prod j{<}i.\ \|F\ !\ j\|^2) * 1$ **by** *simp*
also have $\ldots \leq (\prod j{<}i.\ \|F\ !\ j\|^2) * (\prod j = i..{<}m.\ \|F\ !\ j\|^2)$
by (*rule mult-left-mono*[*OF prod-ge-1 prod-nonneg*], *insert 1*, *auto*)
also have $\ldots = (\prod j{<}m.\ \|F\ !\ j\|^2)$ **unfolding** *id*
by (*subst prod.union-disjoint*, *auto*)
finally show $(\prod j{<}i.\ \|F\ !\ j\|^2) \leq (\prod j{<}m.\ \|F\ !\ j\|^2)$ **.**
qed *auto*
also have $(\prod i{<}m.\ \prod j{<}m.\ \|F\ !\ j\|^2) = (\prod i{<}m.\ \|F\ !\ i\|^2)\ \hat{}m$
unfolding *prod-constant* **by** *simp*
finally have D: $D\ F \leq nat\ ((\prod i{<}m.\ \|F\ !\ i\|^2)\ \hat{}\ m)$ (**is** - $\leq nat\ ?e$) **.**
have e: $?e \geq 0$ **by** (*intro zero-le-power prod-nonneg*, *auto*)
let $?prod = (\prod i{<}m.\ real\text{-}of\text{-}int\ \|F\ !\ i\|^2)$
let $?sum = (\sum i{<}m.\ log\ b\ (of\text{-}int\ \|F\ !\ i\|^2))$
have *prod0*: $?prod > 0$ **by** (*rule prod-pos*, *insert 1*, *force*)
have *prod1*: $?prod \geq 1$ **by** (*rule prod-ge-1*, *insert 1*, *force*)
from D **have** $real\ (D\ F) \leq real\ (nat\ ?e)$ **by** *blast*
also have $\ldots = of\text{-}int\ ?e$ **using** e **by** *simp*
also have $\ldots = ?prod\hat{}m$ **by** *simp*
finally have $log\ b\ (real\ (D\ F)) \leq log\ b\ (?prod\hat{}m)$
by (*subst log-le-cancel-iff*[*OF b1*], *insert D1*, *auto*)
have $real\ (logD\ F) = real\ (nat\ \lfloor log\ b\ (real\ (D\ F))\rfloor)$
unfolding *logD-def b-def* **using** *assms* **by** *auto*
also have $\ldots \leq log\ b\ (real\ (D\ F))$ **using** *b1 D1* **by** *auto*
also have $\ldots \leq log\ b\ (?prod\hat{}m)$ **by** *fact*
also have $\ldots = m * log\ b\ ?prod$ **unfolding** *log-nat-power*[*OF prod0*] **by** *simp*
also have $\ldots = m * ?sum$
by (*subst log-prod*, *insert 1 b1*, *force+*)
finally have *main*: $logD\ F \leq m * ?sum$ **.**

have $real\ (LLL\text{-}measure\ (i,\ Fr,\ Gr)) = real\ (2 * logD\ F + m - i)$
unfolding *LLL-measure-def split invD*(*1*) **by** *simp*
also have $\ldots \leq 2 * real\ (logD\ F) + m$ **using** *invD* **by** *simp*
also have $\ldots \leq 2 * m * ?sum + m$ **using** *main* **by** *auto*
finally show *?thesis* **unfolding** *b-def id* **by** *simp*
qed
end

end

lemma *basis-reduction-main*: **fixes** F G **assumes** *LLL-invariant L* α *state F G*
  and *basis-reduction-main* α *m state* = *state'*
  and α: $\alpha \geq 4/3$
shows $\exists$ F' G'. *LLL-invariant L* α *state'* F' G' $\wedge$ *fst state'* = *m*
proof (*cases m* = *0*)
  case *True*
  from *assms*(*2*)[*unfolded True basis-reduction-main.simps*[*of - 0 state*]]
  have *state'*: *state'* = *state* **by** *auto*
  obtain *i Fr Gr* **where** *state*: *state* = (*i*,*Fr*,*Gr*) **by** (*cases state*, *auto*)
  from *LLL-invD*[*OF assms*(*1*)[*unfolded state*]] *True* **have** *i*: *i* = *0* **by** *auto*
  show *?thesis* **using** *assms*(*1*) **unfolding** *state' state i True* **by** *auto*
next
  case *ne*: *False*
  note [*simp*] = *basis-reduction-main.simps*
  show *?thesis* **using** *assms*(*1*−*2*)
  proof (*induct state arbitrary*: *F G rule*: *wf-induct*[*OF wf-measure*[*of LLL-measure* α]])
    case (*1 state F G*)
    note *inv* = *1*(*2*)
    note *IH* = *1*(*1*)[*rule-format*]
    note *res* = *1*(*3*)
    obtain *i Fr1 Gr1* **where** *state*: *state* = (*i*,*Fr1*,*Gr1*) **by** (*cases state*, *auto*)
    note *inv* = *inv*[*unfolded state*]
    note *res* = *res*[*unfolded state*]
    show *?case*
    proof (*cases i* < *m*)
      case *True*
      with *inv* **have** *i*: *i* < *m* **unfolding** *LLL-invariant-def* **by** *auto*
      obtain *state''* **where** *b*: *basis-reduction-step* α (*i*, *Fr1*, *Gr1*) = *state''* **by** *auto*
      from *res True b*
      have *res*: *basis-reduction-main* α *m state''* = *state'* **by** *simp*
      note *bsr* = *basis-reduction-step*[*OF* α *inv i b*]
      from *bsr*(*1*) **obtain** F' G' **where** *inv*: *LLL-invariant L* α *state''* F' G' **by** *auto*
      from *bsr*(*2*) **have** (*state''* ,*state*) $\in$ *measure* (*LLL-measure* α) **by** (*auto simp*: *state*)
      from *IH*[*OF this inv*] *b res state* **show** *?thesis* **by** *auto*
    next
      case *False*
      define *G1* **where** *Gr*: *G1* = *of-list-repr Fr1*
      note *inv* = *inv*[*unfolded LLL-invariant-def split Gr*[*symmetric*] *Let-def*]
      from *False res* **have** *state'*: *state'* = (*i*, *Fr1*, *Gr1*) **by** *simp*
      from *False inv* **have** *i*: *i* = *m* **unfolding** *LLL-invariant-def* **by** *auto*
      show *?thesis* **using** *1*(*2*) **unfolding** *state' state i* **by** *auto*
    qed
  qed

qed

context fixes α :: *rat* **and** *F*
  assumes α: $\alpha \geq 4/3$
    and *lin-dep*: *gs.lin-indpt-list* (*RAT F*)
    and *len*: *length F* = *m*
begin

lemma *basis-reduction-part-1*: **assumes** *basis-reduction-part-1 n* α *F* = *state*
  shows $\exists$ *F′ G′*. *LLL-invariant* (*lattice-of F*) α *state F′ G′* $\wedge$ *fst state* = *m*
proof −
  let *?F* = *RAT F*
  define *Fr0*::*f-repr* **where** *Fr0* = ([], *F*)
  have *FrF*: *RAT* (*snd Fr0*) = *?F* **unfolding** *Fr0-def* **by** *auto*
  from *lin-dep*
  have *F*: *set F* $\subseteq$ *carrier-vec n*
    unfolding *gs.lin-indpt-list-def* **by** *auto*
  have *repr*: *f-repr 0 Fr0 F* **unfolding** *list-repr-def Fr0-def* **by** *auto*
  obtain *G* **where** *gs*: *snd* (*gram-schmidt-int n F*) = *G* (**is** *snd ?gs* = *G*) **by** *force*
  from *gram-schmidt.mn*[*OF lin-dep - gs*[*unfolded gram-schmidt-int-def gram-schmidt-wit-def*]] *len*
  have *mn*: *m* $\leq$ *n* **by** *auto*
  have *G′*:*length* (*RAT F*) = *m m* $\leq$ *m m* $\leq$ *n*
    set ?F $\subseteq$ *carrier-vec n* **using** *F len mn* **by** *auto*
  define *Gr0* **where** *Gr0* = *gram-schmidt-triv n* (*RAT F*)
  let *?Gr0* = ([], *Gr0*)
  have *RAT-carr*:*set* (*RAT F*) $\subseteq$ *Rn* **using** *F len* **by** *auto*
  have *take*: *RAT F* = *take m* (*RAT F*) **using** *len* **by** *auto*
  from *gram-schmidt.partial-connect*[*OF G′*
      gs[*unfolded gram-schmidt-int-def gram-schmidt-wit-def*] *take RAT-carr*]
  have *gso-init*:*Gr0* = *map* (λ *x*. (*x*, *sq-norm x*)) (*map* (*GSO F*) [*0*..<*m*])
    unfolding *Gr0-def FrF GSO-def gram-schmidt-triv* **using** *len* **by** *auto*
  from *gram-schmidt-int-connect*[*OF lin-dep gs len*]
  have *gso0*: *g-repr 0 ?Gr0 F* **unfolding** *gso-init g-repr-def list-repr-def gs* **by** *auto*
  have *inv*: *LLL-invariant* (*lattice-of F*) α (*0*, *Fr0*, *?Gr0*) *F G*
   by (*rule LLL-invI*[*OF repr gso0 gs refl - - - lin-dep*], *auto simp*:*gs.weakly-reduced-def len*)
  obtain *i Fr1 Gr1* **where** *br*:*state* = (*i*, *Fr1*, *Gr1*) **by**(*cases state*,*auto*)
  note $*$ = *assms*[*unfolded basis-reduction-part-1-def o-def Let-def*,*folded Gr0-def Fr0-def*,*unfolded len*]
  from *basis-reduction-main*[*OF inv* $*$ α]
  show *?thesis* **by** *auto*
qed

lemma *basis-reduction-part-2*: **assumes** *res*: *basis-reduction-part-2 n* α *F* = *state*

  shows $\exists$ *F′ G′*. *LLL-invariant* (*lattice-of F*) α *state F′ G′* $\wedge$
      fst state = *0* $\wedge$ *gs.strictly-reduced m* α *G′* (*gs*.μ (*RAT F′*))

proof –
  obtain *i Fr Gr* **where** *state*: *state* = (*i*,*Fr*,*Gr*) **by** (*cases state*, *auto*)
  obtain *state′* **where** *1*: *basis-reduction-part-1 n* α *F* = *state′* **by** *auto*
  from *res*[*unfolded basis-reduction-part-2-def 1 state*]
  have *2*: *basis-reduction-part-2-main state′* = (*i*,*Fr*,*Gr*) **by** *auto*
  from *basis-reduction-part-1*[*OF 1*] **obtain** *H Hs*
    where *Linv*: *LLL-invariant* (*lattice-of F*) α *state′ H Hs*
      and *n*: *fst state′* = *m* **by** *auto*
  from *basis-reduction-part-2-main*[*OF Linv n 2*] *state*
  show *?thesis* **by** *auto*
qed

lemma *weakly-reduce-basis*: **assumes** *res*: *weakly-reduce-basis n* α *F* = (*F′*, *G′*)
  shows *lattice-of F* = *lattice-of F′* (**is** *?g1*)
  gs.weakly-reduced α *m G′* (**is** *?g2*)
  G′ = *gram-schmidt n* (*RAT F′*) (**is** *?g3*)
  gs.lin-indpt-list (*RAT F′*) (**is** *?g4*)
  length F′ = *m* (**is** *?g5*)
proof –
  obtain *i Fr Gr* **where** *1*: *basis-reduction-part-1 n* α *F* = (*i*, *Fr*, *Gr*) (**is** *?main*
= -)
    by (*cases ?main*) *auto*
  from *basis-reduction-part-1*[*OF 1*] **obtain** *F1 G1*
    where *Linv*: *LLL-invariant* (*lattice-of F*) α (*i*, *Fr*, *Gr*) *F1 G1*
     and *i-n*: *i* = *m* **by** *auto*
  from *res*[*unfolded weakly-reduce-basis-def 1*] **have** *R*: *F′* = *of-list-repr Fr*
    and *Rs*: *G′* = *map fst* (*of-list-repr Gr*) **by** *auto*
  note *inv* = *LLL-invD*[*OF Linv*]
  from *inv*(*1*) *R* **have** *RH*: *F′* = *F1* **unfolding** *of-list-repr-def* **by** *auto*
  with *inv* **have** *Hs*: *G1* = *snd* (*gram-schmidt-int n F′*) **by** *auto*
  from *inv*(*9*)[*unfolded g-repr-def*]
  have *list-repr i Gr* (*map* ($\lambda x.$ (*x*, $\|x\|^2$)) (*map* (*GSO F1*) [*0..<m*])) **by** *auto*
  from *Rs*[*unfolded of-list-repr*[*OF this*]] **have** *Rs*: *G′* = *map* (*GSO F1*) [*0..<m*]
by (*auto simp*: *o-def*)
  also have … = *G1* **unfolding** *LLL-connect*[*OF Linv*] **unfolding** *GSO-def* **by**
simp
  finally have *RsHs*: *G′* = *G1* **by** *auto*
  from *RsHs inv*(*4*,*5*,*6*,*10*) *Rs R Hs RH i-n* **show** *?g1 ?g2 ?g3 ?g4 ?g5* **by** (*auto*
simp: *snd-gram-schmidt-int*)
qed

lemma *strictly-reduce-basis*: **fixes** *F′ G′* **assumes** *res*: *strictly-reduce-basis n* α *F*
= (*F′*, *G′*)
  shows *lattice-of F* = *lattice-of F′* (**is** *?g1*)
  gs.strictly-reduced m α *G′* (*gs.*μ (*RAT F′*)) (**is** *?g2*)
  G′ = *gram-schmidt n* (*RAT F′*) (**is** *?g3*)
  gs.lin-indpt-list (*RAT F′*) (**is** *?g4*)
  length F′ = *m* (**is** *?g5*)
proof –

obtain *i Fr Gr* **where** *2*: *basis-reduction-part-2 n* α *F* = (*i*, *Fr*, *Gr*) (**is** *?main* = -)
by (*cases ?main*) *auto*
from *basis-reduction-part-2*[*OF 2*] **obtain** *F1 G1*
where *Linv*: *LLL-invariant* (*lattice-of F*) α (*i*, *Fr*, *Gr*) *F1 G1*
and *red*: *gs.strictly-reduced m* α *G1* (*gs.*μ (*RAT F1*))
and *i0*: *i* = *0* **by** *auto*
from *res*[*unfolded strictly-reduce-basis-def 2*] **have** *R*: F' = *of-list-repr Fr*
and *Rs*: G' = *map fst* (*of-list-repr Gr*) **by** *auto*
note *inv* = *LLL-invD*[*OF Linv*]
from *inv*(*1*) *R* **have** *RH*: F' = *F1* **unfolding** *of-list-repr-def* **by** *auto*
with *inv* **have** *Hs*: *G1* = *snd* (*gram-schmidt-int n* F') **by** *auto*
from *inv*(*9*)[*unfolded g-repr-def*]
have *list-repr i Gr* (*map* ($\lambda x.$ (x, $\|x\|^2$)) (*map* (*GSO F1*) [*0..<m*])) **by** *auto*
from *Rs*[*unfolded of-list-repr*[*OF this*]] **have** *Rs*: G' = *map* (*GSO F1*) [*0..<m*] **by** (*auto simp*: *o-def*)
also have ... = *G1* **unfolding** *LLL-connect*[*OF Linv*] **unfolding** *GSO-def* **by** *simp*
finally have *RsHs*: G' = *G1* **by** *auto*
from *RsHs inv*(*4*,*5*,*10*) *Rs R Hs RH red* **show** *?g1 ?g2 ?g3 ?g4 ?g5* **by** (*auto simp*: *snd-gram-schmidt-int*)
qed

lemma *short-vector*: **assumes** *short-vector* α *F* = *v*
and *m0*: $m \neq 0$
shows $v \in$ *carrier-vec n*
$v \in$ *lattice-of F* $-$ {0_v *n*}
$h \in$ *lattice-of F* $-$ {0_v *n*} $\Longrightarrow$ *rat-of-int* (*sq-norm v*) $\leq \alpha$ ^ (*m* $-$ *1*) $*$ *rat-of-int* (*sq-norm h*)
$v \neq 0_v$ *j*
proof $-$
let *?L* = *lattice-of F*
have *a1*: $\alpha \geq 1$ **using** α **by** *auto*
obtain *F1 G1* **where** *weak*: *weakly-reduce-basis n* α *F* = (*F1*, *G1*) **by** *force*
from *weakly-reduce-basis*[*OF weak*] *len* **have**
L: *lattice-of F1* = *?L*
and *red*: *gs.weakly-reduced* α *m G1*
and *Gs*: *G1* = *gram-schmidt n* (*RAT F1*)
and *basis*: *gs.lin-indpt-list* (*RAT F1*)
and *lenH*: *length F1* = *m*
and *H*: *set F1* $\subseteq$ *carrier-vec n*
by (*auto simp*: *gs.lin-indpt-list-def*)
from *lin-dep* **have** *G*: *set F* $\subseteq$ *carrier-vec n* **unfolding** *gs.lin-indpt-list-def* **by** *auto*
with *m0 len* **have** *dim-vec* (*hd F*) = *n* **by** (*cases F*, *auto*)
note *res* = *assms*[*unfolded short-vector-def this weak*]
from *res m0 lenH* **have** *v*: *v* = *F1* ! *0* **by** (*cases F1*, *auto*)
from *gs.main-connect*[*OF basis refl*] *Gs*
have *gs*: *snd* (*gs.main* (*RAT F1*)) = *G1* **by** *auto*

let *?r* = *rat-of-int*
let *?rv* = *map-vec ?r*
let *?F* = *RAT F1*
let *?h* = *?rv h*
{ assume *h*:$h \in$ *?L* $-$ $\{0_v\ n\}$ (**is** *?h-req*)
from *h*[*folded L*] **have** *h*: $h \in$ *lattice-of F1 h* $\neq 0_v\ n$ **by** *auto*
{
assume *f*: *?h* = $0_v\ n$
have *?h* = *?rv* ($0_v\ n$) **unfolding** *f* **by** (*intro eq-vecI*, *auto*)
hence $h = 0_v\ n$
using *of-int-hom.vec-hom-zero-iff* [*of h*] *of-int-hom.vec-hom-inj* **by** *auto*
with *h* **have** *False* **by** *simp*
} hence *h0*: *?h* $\neq 0_v\ n$ **by** *auto*
with *lattice-of-of-int*[*OF H h*(*1*)]
have *?h* $\in$ *gs.lattice-of ?F* $-$ $\{0_v\ n\}$ **by** *auto*
}
from *gs.weakly-reduced-imp-short-vector*[*OF basis - gs red this a1*] *lenH*
show $h \in$ *?L* $-$ $\{0_v\ n\} \Longrightarrow$ *?r* (*sq-norm v*) $\leq \alpha$ ^ (*m* $-$ *1*) $*$ *?r* (*sq-norm h*)
unfolding *L v* **by** (*auto simp*: *sq-norm-of-int*)
from *m0 H lenH* **show** *vn*: $v \in$ *carrier-vec n* **unfolding** *v* **by** (*cases F1*, *auto*)
have *vL*: $v \in$ *?L* **unfolding** *L*[*symmetric*] *v* **using** *m0 H lenH*
by (*intro basis-in-latticeI*, *cases F1*, *auto*)
{
assume $v = 0_v\ n$
hence *hd ?F* = $0_v\ n$ **unfolding** *v* **using** *m0 lenH* **by** (*cases F1*, *auto*)
with *gs.lin-indpt-list-nonzero*[*OF basis*] **have** *False* **using** *m0 lenH* **by** (*cases F1*, *auto*)
}
with *vL* **show** *v*: $v \in$ *?L* $-$ $\{0_v\ n\}$ **by** *auto*
have *jn*:$0_v\ j \in$ *carrier-vec n* $\Longrightarrow j = n$ **unfolding** *zero-vec-def carrier-vec-def* **by** *auto*
with *v vn* **show** $v \neq 0_v\ j$ **by** *auto*
qed
end
end
end

References

[1] J. v. z. Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 2nd edition, 2003.

[2] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[3] R. Thiemann and A. Yamada. Formalizing Jordan normal forms in Isabelle/HOL. In *CPP 2016*, pages 88–99. ACM, 2016.