Linear Inequalities*

Ralph Bottesch[1], Alban Reynaud[2], and René Thiemann[1]

[1]University of Innsbruck
[2]ENS Lyon

October 13, 2025

Abstract

We formalize results about linear inqualities, mainly from Schrijver's book [3]. The main results are the proof of the fundamental theorem on linear inequalities, Farkas' lemma, Carathéodory's theorem, the Farkas-Minkowsky-Weyl theorem, the decomposition theorem of polyhedra, and Meyer's result that the integer hull of a polyhedron is a polyhedron itself. Several theorems include bounds on the appearing numbers, and in particular we provide an a-priori bound on mixed-integer solutions of linear inequalities.

Contents

*Supported by FWF (Austrian Science Fund) project Y757.

1 Introduction

The motivation for this formalization is the aim of developing a verified theory solver for linear integer arithmetic. Such a solver can be a combination of a simplex-implementation within a branch-and-bound approach, that might also utilize Gomory cuts [1, Section 4 of the extended version]. However, the branch-and-bound algorithm does not terminate in general, since the search space in infinite. To solve this latter problem, one can use results of Papadimitriou: he showed that whenever a set of linear inequalities has an integer solution, then it also has a small solution, where the bound on such a solution can be computed easily from the input [2].

In this entry, we therefore formalize several results on linear inequalities which are required to obtain the desired bound, by following the proofs of Schrijver's textbook [3, Sections 7 and 16].

We start with basic definitions and results on cones, convex hulls, and polyhedra. Next, we verify the fundamental theorem of linear inequalities, which in our formalization shows the equivalence of four statements to describe a cone. From this theorem, one easily derives Farkas' Lemma and Carathéodory's theorem. Moreover we verify the Farkas-Minkowsky-Weyl theorem, that a convex cone is polyhedral if and only if it is finitely generated, and use this result to obtain the decomposition theorem for polyhedra, i.e., that a polyhedron can always be decomposed into a polytope and a finitely generated cone. For most of the previously mentioned results, we include bounds, so that in particular we have a quantitative version of the decomposition theorem, which provides bounds on the vectors that construct the polytope and the cone, and where these bounds are computed directly from the input polyhedron that should be decomposed.

We further prove the decomposition theorem also for the integer hull of a polyhedron, using the same bounds, which gives rise to small integer solutions for linear inequalities. We finally formalize a direct proof for the more general case of mixed integer solutions, where we also permit both strict and non-strict linear inequalities.

Theorem 1. *Consider* $A_1 \in \mathbb{Z}^{m_1 \times n}, b_1 \in \mathbb{Z}^{m_1}, A_2 \in \mathbb{Z}^{m_2 \times n}, b_2 \in \mathbb{Z}^{m_2}$*. Let* β *be a bound on* A_1, b_1, A_2, b_2*, i.e.,* $\beta \geq |z|$ *for all numbers* z *that occur within* A_1, b_1, A_2, b_2*. Let* $n = n_1 + n_2$*. Then if* $x \in \mathbb{Z}^{n_1} \times \mathbb{R}^{n_2} \subseteq \mathbb{R}^n$ *is a mixed integer solution of the linear inequalities, i.e.,* $A_1 x \leq b_1$ *and* $A_2 x < b_2$*, then there also exists a mixed integer solution* $y \in \mathbb{Z}^{n_1} \times \mathbb{R}^{n_2}$ *where* $|y_i| \leq (n+1) \cdot \sqrt{n^n} \cdot \beta^n$ *for each entry* y_i *of* y*.*

The verified bound in Theorem 1 in particular implies that integer-satisfiability of linear-inqualities with integer coefficients is in NP.

2 Missing Lemmas on Vectors and Matrices

We provide some results on vector spaces which should be merged into Jordan-Normal-Form/Matrix.

```
theory Missing-Matrix
  imports Jordan-Normal-Form.Matrix
begin

lemma orthogonalD': assumes orthogonal vs
  and v ∈ set vs and w ∈ set vs
shows (v · w = 0) = (v ≠ w)
proof −
  from assms(2) obtain i where v: v = vs ! i and i: i < length vs by (auto simp:
set-conv-nth)
  from assms(3) obtain j where w: w = vs ! j and j: j < length vs by (auto
simp: set-conv-nth)
  from orthogonalD[OF assms(1) i j, folded v w] orthogonalD[OF assms(1) i i,
folded v v]
  show ?thesis using v w by auto
qed

lemma zero-mat-mult-vector[simp]: x ∈ carrier-vec nc ⟹ 0_m nr nc *_v x = 0_v
nr
  by (intro eq-vecI, auto)

lemma add-diff-cancel-right-vec:
  a ∈ carrier-vec n ⟹ (b :: 'a :: cancel-ab-semigroup-add vec) ∈ carrier-vec n ⟹
    (a + b) − b = a
  by (intro eq-vecI, auto)

lemma elements-four-block-mat-id:
  assumes c: A ∈ carrier-mat nr1 nc1 B ∈ carrier-mat nr1 nc2
    C ∈ carrier-mat nr2 nc1 D ∈ carrier-mat nr2 nc2
  shows
    elements-mat (four-block-mat A B C D) =
```

```
    elements-mat A ∪ elements-mat B ∪ elements-mat C ∪ elements-mat D
    (is elements-mat ?four = ?X)
proof
  show elements-mat ?four ⊆ ?X
    by (rule elements-four-block-mat[OF c])
  have 4: ?four ∈ carrier-mat (nr1 + nr2) (nc1 + nc2) using c by auto
  {
    fix x
    assume x ∈ ?X
    then consider (A) x ∈ elements-mat A
      | (B) x ∈ elements-mat B
      | (C) x ∈ elements-mat C
      | (D) x ∈ elements-mat D by auto
    hence x ∈ elements-mat ?four
    proof (cases)
      case A
      from elements-matD[OF this] obtain i j
        where ∗: i < nr1 j < nc1 and x: x = A $$ (i,j)
        using c by auto
      from elements-matI[OF 4, of i j x] ∗ c
      show ?thesis unfolding x by auto
    next
      case B
      from elements-matD[OF this] obtain i j
        where ∗: i < nr1 j < nc2 and x: x = B $$ (i,j)
        using c by auto
      from elements-matI[OF 4, of i nc1 + j x] ∗ c
      show ?thesis unfolding x by auto
    next
      case C
      from elements-matD[OF this] obtain i j
        where ∗: i < nr2 j < nc1 and x: x = C $$ (i,j)
        using c by auto
      from elements-matI[OF 4, of nr1 + i j x] ∗ c
      show ?thesis unfolding x by auto
    next
      case D
      from elements-matD[OF this] obtain i j
        where ∗: i < nr2 j < nc2 and x: x = D $$ (i,j)
        using c by auto
      from elements-matI[OF 4, of nr1 + i nc1 + j x] ∗ c
      show ?thesis unfolding x by auto
    qed
  }
  thus elements-mat ?four ⊇ ?X by blast
qed


lemma elements-mat-append-rows: A ∈ carrier-mat nr n =⇒ B ∈ carrier-mat nr2
```

n ⟹
  elements-mat (*A* $@_r$ *B*) = *elements-mat* *A* ∪ *elements-mat* *B*
  unfolding *append-rows-def*
  by (*subst elements-four-block-mat-id*, *auto*)

lemma *elements-mat-uminus*[*simp*]: *elements-mat* (−*A*) = *uminus* ‘ *elements-mat* *A*
  unfolding *elements-mat-def* **by** *auto*

lemma *vec-set-uminus*[*simp*]: *vec-set* (−*A*) = *uminus* ‘ *vec-set* *A*
  unfolding *vec-set-def* **by** *auto*

definition *append-cols* :: ′*a* :: *zero mat* ⇒ ′*a mat* ⇒ ′*a mat* (**infixr** ‹$@_c$› *65*)
where
  $A\ @_c\ B = (A^T\ @_r\ B^T)^T$

lemma *carrier-append-cols*[*simp*, *intro*]:
  A ∈ *carrier-mat nr nc1* ⟹
   B ∈ *carrier-mat nr nc2* ⟹ (*A* $@_c$ *B*) ∈ *carrier-mat nr* (*nc1* + *nc2*)
  unfolding *append-cols-def* **by** *auto*

lemma *elements-mat-transpose-mat*[*simp*]: *elements-mat* (A^T) = *elements-mat* *A*
  unfolding *elements-mat-def* **by** *auto*

lemma *elements-mat-append-cols*: *A* ∈ *carrier-mat n nc* ⟹ *B* ∈ *carrier-mat n nc1*
  ⟹ *elements-mat* (*A* $@_c$ *B*) = *elements-mat* *A* ∪ *elements-mat* *B*
  unfolding *append-cols-def elements-mat-transpose-mat*
  by (*subst elements-mat-append-rows*, *auto*)

lemma *vec-first-index*:
  assumes *v*: *dim-vec v* ≥ *n*
    and *i*: *i* < *n*
  shows (*vec-first v n*) $ *i* = *v* $ *i*
  unfolding *vec-first-def* **using** *assms* **by** *simp*

lemma *vec-last-index*:
  assumes *v*: *v* ∈ *carrier-vec* (*n* + *m*)
    and *i*: *i* < *m*
  shows (*vec-last v m*) $ *i* = *v* $ (*n* + *i*)
  unfolding *vec-last-def* **using** *assms* **by** *simp*

lemma *vec-first-add*:
  assumes *dim-vec x* ≥ *n*
    and *dim-vec y* ≥ *n*
  shows*vec-first* (*x* + *y*) *n* = *vec-first x n* + *vec-first y n*
  unfolding *vec-first-def* **using** *assms* **by** *auto*

lemma *vec-first-zero*[*simp*]: *m* ≤ *n* ⟹ *vec-first* (0_v *n*) *m* = 0_v *m*

unfolding *vec-first-def* **by** *auto*

lemma *vec-first-smult*:
$[\![m \leq n; x \in \textit{carrier-vec } n]\!] \Longrightarrow \textit{vec-first } (c \cdot_v x)\ m = c \cdot_v \textit{vec-first } x\ m$
unfolding *vec-first-def* **by** *auto*

lemma *elements-mat-mat-of-row*[*simp*]: *elements-mat* (*mat-of-row v*) = *vec-set v*
by (*auto simp*: *mat-of-row-def elements-mat-def vec-set-def*)

lemma *vec-set-append-vec*[*simp*]: *vec-set* (v $@_v$ w) = *vec-set v* ∪ *vec-set w*
by (*metis list-of-vec-append set-append set-list-of-vec*)

lemma *vec-set-vNil*[*simp*]: set_v *vNil* = {} **using** *set-list-of-vec* **by** *force*

lemma *diff-smult-distrib-vec*: $((x :: 'a::ring) - y) \cdot_v v = x \cdot_v v - y \cdot_v v$
unfolding *smult-vec-def minus-vec-def*
by (*rule eq-vecI*, *auto simp*: *left-diff-distrib*)

lemma *add-diff-eq-vec*: **fixes** *y* :: *'a* :: *group-add vec*
shows $y \in$ *carrier-vec n* $\Longrightarrow x \in$ *carrier-vec n* $\Longrightarrow z \in$ *carrier-vec n* $\Longrightarrow y + (x - z) = y + x - z$
by (*intro eq-vecI*, *auto simp*: *add-diff-eq*)

definition *mat-of-col v* = $(\textit{mat-of-row } v)^T$

lemma *elements-mat-mat-of-col*[*simp*]: *elements-mat* (*mat-of-col v*) = *vec-set v*
unfolding *mat-of-col-def* **by** *auto*

lemma *mat-of-col-dim*[*simp*]: *dim-row* (*mat-of-col v*) = *dim-vec v*
dim-col (*mat-of-col v*) = *1*
mat-of-col v ∈ *carrier-mat* (*dim-vec v*) *1*
unfolding *mat-of-col-def* **by** *auto*

lemma *col-mat-of-col*[*simp*]: *col* (*mat-of-col v*) *0* = *v*
unfolding *mat-of-col-def* **by** *auto*

lemma *mult-mat-of-col*: $A \in$ *carrier-mat nr nc* $\Longrightarrow v \in$ *carrier-vec nc* $\Longrightarrow$
$A *$ *mat-of-col v* = *mat-of-col* ($A *_v v$)
by (*intro mat-col-eqI*, *auto*)

lemma *mat-mult-append-cols*: **fixes** *A* :: *'a* :: *comm-semiring-0 mat*
assumes *A*: $A \in$ *carrier-mat nr nc1*
and *B*: $B \in$ *carrier-mat nr nc2*
and *v1*: $v1 \in$ *carrier-vec nc1*
and *v2*: $v2 \in$ *carrier-vec nc2*
shows $(A\ @_c\ B) *_v (v1\ @_v\ v2) = A *_v v1 + B *_v v2$
proof −
have $(A\ @_c\ B) *_v (v1\ @_v\ v2) = (A\ @_c\ B) *_v$ *col* (*mat-of-col* ($v1\ @_v\ v2$)) *0* **by**

auto
  also have … = *col* ((*A* $@_c$ *B*) ∗ *mat-of-col* (*v1* $@_v$ *v2*)) *0* **by** *auto*
  also have (*A* $@_c$ *B*) ∗ *mat-of-col* (*v1* $@_v$ *v2*) = ((*A* $@_c$ *B*) ∗ *mat-of-col* (*v1* $@_v$ *v2*))$^{T\,T}$
    by *auto*
  also have ((*A* $@_c$ *B*) ∗ *mat-of-col* (*v1* $@_v$ *v2*))^T =
        (*mat-of-row* (*v1* $@_v$ *v2*))$^{T\,T}$ ∗ (A^T $@_r$ B^T)$^{T\,T}$
    unfolding *append-cols-def mat-of-col-def*
  proof (*rule transpose-mult*, *force*, *unfold transpose-carrier-mat*, *rule mat-of-row-carrier*)
    have A^T ∈ *carrier-mat nc1 nr* **using** *A* **by** *auto*
    moreover have B^T ∈ *carrier-mat nc2 nr* **using** *B* **by** *auto*
    ultimately have A^T $@_r$ B^T ∈ *carrier-mat* (*nc1* + *nc2*) *nr* **by** *auto*
    hence *dim-row* (A^T $@_r$ B^T) = *nc1* + *nc2* **by** *auto*
    thus *v1* $@_v$ *v2* ∈ *carrier-vec* (*dim-row* (A^T $@_r$ B^T)) **using** *v1 v2* **by** *auto*
  qed
  also have … = (*mat-of-row* (*v1* $@_v$ *v2*)) ∗ (A^T $@_r$ B^T) **by** *auto*
  also have … = *mat-of-row v1* ∗ A^T + *mat-of-row v2* ∗ B^T
    using *mat-of-row-mult-append-rows*[*OF v1 v2*] *A B* **by** *auto*
  also have …^{T} = (*mat-of-row v1* ∗ A^T)^T + (*mat-of-row v2* ∗ B^T)^T
    using *transpose-add A B* **by** *auto*
  also have (*mat-of-row v1* ∗ A^T)^T = $A^{T\,T}$ ∗ ((*mat-of-row v1*)^T)
    using *transpose-mult A v1 transpose-carrier-mat mat-of-row-carrier*(*1*)
    by *metis*
  also have (*mat-of-row v2* ∗ B^T)^T = $B^{T\,T}$ ∗ ((*mat-of-row v2*)^T)
    using *transpose-mult B v2 transpose-carrier-mat mat-of-row-carrier*(*1*)
    by *metis*
  also have $A^{T\,T}$ ∗ ((*mat-of-row v1*)^T) + $B^{T\,T}$ ∗ ((*mat-of-row v2*)^T) =
        A ∗ *mat-of-col v1* + *B* ∗ *mat-of-col v2*
    unfolding *mat-of-col-def* **by** *auto*
  also have *col* … *0* = *col* (*A* ∗ *mat-of-col v1*) *0* + *col* (*B* ∗ *mat-of-col v2*) *0*
    using *assms* **by** *auto*
  also have … = *col* (*mat-of-col* (*A* $*_v$ *v1*)) *0* + *col* (*mat-of-col* (*B* $*_v$ *v2*)) *0*
    using *mult-mat-of-col assms* **by** *auto*
  also have … = *A* $*_v$ *v1* + *B* $*_v$ *v2* **by** *auto*
  finally show *?thesis* **by** *auto*
qed

lemma *vec-first-append*:
  assumes *v* ∈ *carrier-vec n*
  shows *vec-first* (*v* $@_v$ *w*) *n* = *v*
proof −
  have *v* $@_v$ *w* = *vec-first* (*v* $@_v$ *w*) *n* $@_v$ *vec-last* (*v* $@_v$ *w*) (*dim-vec w*)
    using *vec-first-last-append assms* **by** *simp*
  thus *?thesis* **using** *append-vec-eq*[*OF assms*] **by** *simp*
qed

lemma *vec-le-iff-diff-le-0*: **fixes** *a* :: *'a* :: *ordered-ab-group-add vec*
  shows (*a* ≤ *b*) = (*a* − *b* ≤ 0_v (*dim-vec a*))
  unfolding *less-eq-vec-def* **by** *auto*

definition *mat-row-first A n* ≡ *mat n* (*dim-col A*) (λ (*i*, *j*). *A* $$ (*i*, *j*))

definition *mat-row-last A n* ≡ *mat n* (*dim-col A*) (λ (*i*, *j*). *A* $$ (*dim-row A* − *n* + *i*, *j*))

lemma *mat-row-first-carrier*[*simp*]: *mat-row-first A n* ∈ *carrier-mat n* (*dim-col A*)
  unfolding *mat-row-first-def* **by** *simp*

lemma *mat-row-first-dim*[*simp*]:
  dim-row (*mat-row-first A n*) = *n*
  dim-col (*mat-row-first A n*) = *dim-col A*
  unfolding *mat-row-first-def* **by** *simp-all*

lemma *mat-row-last-carrier*[*simp*]: *mat-row-last A n* ∈ *carrier-mat n* (*dim-col A*)
  unfolding *mat-row-last-def* **by** *simp*

lemma *mat-row-last-dim*[*simp*]:
  dim-row (*mat-row-last A n*) = *n*
  dim-col (*mat-row-last A n*) = *dim-col A*
  unfolding *mat-row-last-def* **by** *simp-all*

lemma *mat-row-first-nth*[*simp*]: *i* < *n* ⟹ *row* (*mat-row-first A n*) *i* = *row A i*
  unfolding *mat-row-first-def row-def* **by** *fastforce*

lemma *append-rows-nth*:
  assumes *A* ∈ *carrier-mat nr1 nc*
    and *B* ∈ *carrier-mat nr2 nc*
  shows *i* < *nr1* ⟹ *row* (*A* $@_r$ *B*) *i* = *row A i*
    and ⟦ *i* ≥ *nr1*; *i* < *nr1* + *nr2* ⟧ ⟹ *row* (*A* $@_r$ *B*) *i* = *row B* (*i* − *nr1*)
  unfolding *append-rows-def* **using** *row-four-block-mat assms* **by** *auto*

lemma *mat-of-row-last-nth*[*simp*]:
  i < *n* ⟹ *row* (*mat-row-last A n*) *i* = *row A* (*dim-row A* − *n* + *i*)
  unfolding *mat-row-last-def row-def* **by** *auto*

lemma *mat-row-first-last-append*:
  assumes *dim-row A* = *m* + *n*
  shows (*mat-row-first A m*) $@_r$ (*mat-row-last A n*) = *A*
proof (*rule eq-rowI*)
  show *dim-row* (*mat-row-first A m* $@_r$ *mat-row-last A n*) = *dim-row A*
    unfolding *append-rows-def* **using** *assms* **by** *fastforce*
  show *dim-col* (*mat-row-first A m* $@_r$ *mat-row-last A n*) = *dim-col A*
    unfolding *append-rows-def* **by** *fastforce*
  fix *i*
  assume *i*: *i* < *dim-row A*
  show *row* (*mat-row-first A m* $@_r$ *mat-row-last A n*) *i* = *row A i*
  proof *cases*
    assume *i*: *i* < *m*

thus *?thesis* **using** *append-rows-nth*(*1*)[*OF mat-row-first-carrier*[*of A m*]
mat-row-last-carrier[*of A n*] *i*] **by** *simp*
next
assume *i*′: ¬ *i* < *m*
thus *?thesis* **using** *append-rows-nth*(*2*)[*OF mat-row-first-carrier*[*of A m*]
mat-row-last-carrier[*of A n*]] *i assms* **by** *simp*
qed
qed

definition *mat-col-first A n* ≡ (*mat-row-first* A^T *n*)^T

definition *mat-col-last A n* ≡ (*mat-row-last* A^T *n*)^T

lemma *mat-col-first-carrier*[*simp*]: *mat-col-first A n* ∈ *carrier-mat* (*dim-row A*) *n*
unfolding *mat-col-first-def* **by** *fastforce*

lemma *mat-col-first-dim*[*simp*]:
dim-row (*mat-col-first A n*) = *dim-row A*
dim-col (*mat-col-first A n*) = *n*
unfolding *mat-col-first-def* **by** *simp-all*

lemma *mat-col-last-carrier*[*simp*]: *mat-col-last A n* ∈ *carrier-mat* (*dim-row A*) *n*
unfolding *mat-col-last-def* **by** *fastforce*

lemma *mat-col-last-dim*[*simp*]:
dim-row (*mat-col-last A n*) = *dim-row A*
dim-col (*mat-col-last A n*) = *n*
unfolding *mat-col-last-def* **by** *simp-all*

lemma *mat-col-first-nth*[*simp*]:
⟦ *i* < *n*; *i* < *dim-col A* ⟧ ⟹ *col* (*mat-col-first A n*) *i* = *col A i*
unfolding *mat-col-first-def* **by** *force*

lemma *append-cols-nth*:
assumes *A* ∈ *carrier-mat nr nc1*
and *B* ∈ *carrier-mat nr nc2*
shows *i* < *nc1* ⟹ *col* (*A* $@_c$ *B*) *i* = *col A i*
and ⟦ *i* ≥ *nc1*; *i* < *nc1* + *nc2* ⟧ ⟹ *col* (*A* $@_c$ *B*) *i* = *col B* (*i* − *nc1*)
unfolding *append-cols-def append-rows-def* **using** *row-four-block-mat assms*
by *auto*

lemma *mat-of-col-last-nth*[*simp*]:
⟦ *i* < *n*; *i* < *dim-col A* ⟧ ⟹ *col* (*mat-col-last A n*) *i* = *col A* (*dim-col A* − *n* +
i)
unfolding *mat-col-last-def* **by** *auto*

lemma *mat-col-first-last-append*:
assumes *dim-col A* = *m* + *n*
shows (*mat-col-first A m*) $@_c$ (*mat-col-last A n*) = *A*

unfolding *append-cols-def mat-col-first-def mat-col-last-def*
using *mat-row-first-last-append*[*of* A^T] *assms* **by** *simp*

lemma *mat-of-row-dim-row-1*: (*dim-row A* = *1*) = (*A* = *mat-of-row* (*row A 0*))
proof
show *dim-row A* = *1* ⟹ *A* = *mat-of-row* (*row A 0*) **by** *force*
show *A* = *mat-of-row* (*row A 0*) ⟹ *dim-row A* = *1* **using** *mat-of-row-dim*(*1*)
by *metis*
qed

lemma *mat-of-col-dim-col-1*: (*dim-col A* = *1*) = (*A* = *mat-of-col* (*col A 0*))
proof
show *dim-col A* = *1* ⟹ *A* = *mat-of-col* (*col A 0*)
unfolding *mat-of-col-def* **by** *auto*
show *A* = *mat-of-col* (*col A 0*) ⟹ *dim-col A* = *1* **by** (*metis mat-of-col-dim*(*2*))
qed

definition *vec-of-scal* :: *'a* ⇒ *'a vec* **where** *vec-of-scal x* ≡ *vec 1* (λ *i*. *x*)

lemma *vec-of-scal-dim*[*simp*]:
dim-vec (*vec-of-scal x*) = *1*
vec-of-scal x ∈ *carrier-vec 1*
unfolding *vec-of-scal-def* **by** *auto*

lemma *index-vec-of-scal*[*simp*]: (*vec-of-scal x*) $ *0* = *x*
unfolding *vec-of-scal-def* **by** *auto*

lemma *row-mat-of-col*[*simp*]: *i* < *dim-vec v* ⟹ *row* (*mat-of-col v*) *i* = *vec-of-scal* (*v* $ *i*)
unfolding *mat-of-col-def* **by** *auto*

lemma *vec-of-scal-dim-1*: (*v* ∈ *carrier-vec 1*) = (*v* = *vec-of-scal* (*v* $ *0*))
by(*standard*, *auto simp del*: *One-nat-def*, *metis vec-of-scal-dim*(*2*))

lemma *mult-mat-of-row-vec-of-scal*: **fixes** *x* :: *'a* :: *comm-ring-1*
shows *mat-of-col v* $*_v$ *vec-of-scal x* = *x* $\cdot_v$ *v*
by (*auto simp add*: *scalar-prod-def*)

lemma *smult-pos-vec*[*simp*]: **fixes** *l* :: *'a* :: *linordered-ring-strict*
assumes *l*: *l* > *0*
shows (*l* $\cdot_v$ *v* ≤ 0_v *n*) = (*v* ≤ 0_v *n*)
proof (*cases dim-vec v* = *n*)
case *True*
have *i* < *n* ⟹ ((*l* $\cdot_v$ *v*) $ *i* ≤ *0*) ⟷ *v* $ *i* ≤ *0* **for** *i* **using** *True*
mult-le-cancel-left-pos[*OF l*, *of - 0*] **by** *simp*
thus *?thesis* **using** *True* **unfolding** *less-eq-vec-def* **by** *auto*
qed (*auto simp*: *less-eq-vec-def*)

lemma *finite-elements-mat*[*simp*]: *finite* (*elements-mat A*)

```
  unfolding elements-mat-def by (rule finite-set)

lemma finite-vec-set[simp]: finite (vec-set A)
  unfolding vec-set-def by auto

lemma lesseq-vecI: assumes v ∈ carrier-vec n w ∈ carrier-vec n
  ⋀ i. i < n ⟹ v $ i ≤ w $ i
shows v ≤ w
  using assms unfolding less-eq-vec-def by auto

lemma lesseq-vecD: assumes w ∈ carrier-vec n
  and v ≤ w
  and i < n
shows v $ i ≤ w $ i
  using assms unfolding less-eq-vec-def by auto

lemma vec-add-mono: fixes a :: 'a :: ordered-ab-semigroup-add vec
  assumes dim: dim-vec b = dim-vec d
    and ab: a ≤ b
    and cd: c ≤ d
  shows a + c ≤ b + d
proof −
  have ⋀ i. i < dim-vec d ⟹ (a + c) $ i ≤ (b + d) $ i
  proof −
    fix i
    assume id: i < dim-vec d
    have ic: i < dim-vec c using id cd unfolding less-eq-vec-def by auto
    have ib: i < dim-vec b using id dim by auto
    have ia: i < dim-vec a using ib ab unfolding less-eq-vec-def by auto
    have a $ i ≤ b $ i using ab ia ib unfolding less-eq-vec-def by auto
     moreover have c $ i ≤ d $ i using cd ic id unfolding less-eq-vec-def by
auto
    ultimately have abcdi: a $ i + c $ i ≤ b $ i + d $ i using add-mono by auto
    have (a + c) $ i = a $ i + c $ i using index-add-vec(1) ic by auto
    also have ... ≤ b $ i + d $ i using abcdi by auto
    also have b $ i + d $ i = (b + d) $ i using index-add-vec(1) id by auto
    finally show (a + c) $ i ≤ (b + d) $ i by auto
  qed
  then show a + c ≤ b + d unfolding less-eq-vec-def
    using dim index-add-vec(2) cd less-eq-vec-def by auto
qed

lemma smult-nneg-npos-vec: fixes l :: 'a :: ordered-semiring-0
  assumes l: l ≥ 0
    and v: v ≤ 0_v n
  shows l ·_v v ≤ 0_v n
proof −
  {
    fix i
```

```
    assume i: i < n
    then have vi: v $ i ≤ 0 using v unfolding less-eq-vec-def by simp
    then have (l ·v v) $ i = l * v $ i using v i unfolding less-eq-vec-def by auto
    also have l * v $ i ≤ 0 by (rule mult-nonneg-nonpos[OF l vi])
    finally have (l ·v v) $ i ≤ 0 by auto
  }
  then show ?thesis using v unfolding less-eq-vec-def by auto
qed

lemma smult-vec-nonneg-eq: fixes c :: 'a :: field
  shows c ≠ 0 ⟹ (c ·v x = c ·v y) = (x = y)
proof −
  have c ≠ 0 ⟹ c ·v x = c ·v y ⟹ x = y
    by (metis smult-smult-assoc[of 1 / c c] nonzero-divide-eq-eq one-smult-vec)
  thus c ≠ 0 ⟹ ?thesis by auto
qed

lemma distinct-smult-nonneg: fixes c :: 'a :: field
  assumes c: c ≠ 0
  shows distinct lC ⟹ distinct (map ((·v) c) lC)
proof (induction lC)
  case (Cons v lC)
  from Cons.prems have v ∉ set lC by fastforce
  hence c ·v v ∉ set (map ((·v) c) lC) using smult-vec-nonneg-eq[OF c] by fastforce
  moreover have map ((·v) c) (v # lC) = c ·v v # map ((·v) c) lC by simp
  ultimately show ?case using Cons.IH Cons.prems by simp
qed auto

lemma exists-vec-append: (∃ x ∈ carrier-vec (n + m). P x) ⟷ (∃ x1 ∈ carrier-vec n. ∃ x2 ∈ carrier-vec m. P (x1 @v x2))
proof
  assume ∃x ∈ carrier-vec (n + m). P x
  from this obtain x where xcarr: x ∈ carrier-vec (n+m) and Px: P x by auto
  have x = vec n (λ i. x $ i) @v vec m (λ i. x $ (n + i))
    by (rule eq-vecI, insert xcarr, auto)
  hence P x = P (vec n (λ i. x $ i) @v vec m (λ i. x $ (n + i))) by simp
  also have 1: ... using xcarr Px calculation by blast
  finally show ∃x1∈carrier-vec n. ∃x2∈carrier-vec m. P (x1 @v x2) using 1 vec-carrier by blast
next
  assume (∃ x1 ∈ carrier-vec n. ∃ x2 ∈ carrier-vec m. P (x1 @v x2))
  from this obtain x1 x2 where x1: x1 ∈ carrier-vec n
    and x2: x2 ∈ carrier-vec m and P12: P (x1 @v x2) by auto
  define x where x = x1 @v x2
  have xcarr: x ∈ carrier-vec (n+m) using x1 x2 by (simp add: x-def)
  have P x using P12 xcarr using x-def by blast
  then show (∃ x ∈ carrier-vec (n + m). P x) using xcarr by auto
qed
```

end

3 Missing Lemmas on Vector Spaces

We provide some results on vector spaces which should be merged into other AFP entries.

```
theory Missing-VS-Connect
  imports
    Jordan-Normal-Form.VS-Connect
    Missing-Matrix
    Polynomial-Factorization.Missing-List
begin

context vec-space
begin
lemma span-diff: assumes A: A ⊆ carrier-vec n
  and a: a ∈ span A and b: b ∈ span A
shows a − b ∈ span A
proof −
  from A a have an: a ∈ carrier-vec n by auto
  from A b have bn: b ∈ carrier-vec n by auto
  have a + (−1 ·v b) ∈ span A
    by (rule span-add1[OF A a], insert b A, auto)
  also have a + (−1 ·v b) = a − b using an bn by auto
  finally show ?thesis by auto
qed

lemma finsum-scalar-prod-sum':
  assumes f: f ∈ U → carrier-vec n
    and w: w ∈ carrier-vec n
  shows w · finsum V f U = sum (λu. w · f u) U
  by (subst comm-scalar-prod[OF w], (insert f, auto)[1],
      subst finsum-scalar-prod-sum[OF f w],
      insert f, intro sum.cong[OF refl] comm-scalar-prod[OF - w], auto)

lemma lincomb-scalar-prod-left: assumes W ⊆ carrier-vec n v ∈ carrier-vec n
  shows lincomb a W · v = (∑ w∈W. a w * (w · v))
  unfolding lincomb-def
  by (subst finsum-scalar-prod-sum, insert assms, auto intro!: sum.cong)

lemma lincomb-scalar-prod-right: assumes W ⊆ carrier-vec n v ∈ carrier-vec n
  shows v · lincomb a W = (∑ w∈W. a w * (v · w))
  unfolding lincomb-def
  by (subst finsum-scalar-prod-sum', insert assms, auto intro!: sum.cong)

lemma lin-indpt-empty[simp]: lin-indpt {}
  using lin-dep-def by auto
```

lemma *span-carrier-lin-indpt-card-n*:
assumes *W* ⊆ *carrier-vec n card W* = *n lin-indpt W*
shows *span W* = *carrier-vec n*
using *assms basis-def dim-is-n dim-li-is-basis fin-dim-li-fin* **by** *simp*

lemma *ortho-span*: **assumes** *W*: *W* ⊆ *carrier-vec n*
and *X*: *X* ⊆ *carrier-vec n*
and *ortho*: ⋀ *w x*. *w* ∈ *W* ⟹ *x* ∈ *X* ⟹ *w* • *x* = *0*
and *w*: *w* ∈ *span W* **and** *x*: *x* ∈ *X*
shows *w* • *x* = *0*
proof −
from *w W* **obtain** *c V* **where** *finite V* **and** *VW*: *V* ⊆ *W* **and** *w*: *w* = *lincomb c V*
by (*meson in-spanE*)
show *?thesis* **unfolding** *w*
by (*subst lincomb-scalar-prod-left*, *insert W VW X x ortho*, *auto intro*!: *sum.neutral*)
qed

lemma *ortho-span′*: **assumes** *W*: *W* ⊆ *carrier-vec n*
and *X*: *X* ⊆ *carrier-vec n*
and *ortho*: ⋀ *w x*. *w* ∈ *W* ⟹ *x* ∈ *X* ⟹ *x* • *w* = *0*
and *w*: *w* ∈ *span W* **and** *x*: *x* ∈ *X*
shows *x* • *w* = *0*
proof −
from *w W* **obtain** *c V* **where** *finite V* **and** *VW*: *V* ⊆ *W* **and** *w*: *w* = *lincomb c V*
by (*meson in-spanE*)
show *?thesis* **unfolding** *w*
by (*subst lincomb-scalar-prod-right*, *insert W VW X x ortho*, *auto intro*!: *sum.neutral*)
qed

lemma *ortho-span-span*: **assumes** *W*: *W* ⊆ *carrier-vec n*
and *X*: *X* ⊆ *carrier-vec n*
and *ortho*: ⋀ *w x*. *w* ∈ *W* ⟹ *x* ∈ *X* ⟹ *w* • *x* = *0*
and *w*: *w* ∈ *span W* **and** *x*: *x* ∈ *span X*
shows *w* • *x* = *0*
by (*rule ortho-span*[*OF W - ortho-span′*[*OF X W - -*] *w x*], *insert W X ortho*, *auto*)

lemma *lincomb-in-span*[*intro*]:
assumes *X*: *X*⊆ *carrier-vec n*
shows *lincomb a X* ∈ *span X*
proof(*cases finite X*)
case *False* **hence** *lincomb a X* = 0_v *n* **using** *X*
by (*simp add*: *lincomb-def*)
thus *?thesis* **using** *X* **by** *force*
qed (*insert X*, *auto*)

lemma *generating-card-n-basis*: **assumes** *X*: *X* ⊆ *carrier-vec n*
  and *span*: *carrier-vec n* ⊆ *span X*
  and *card*: *card X* = *n*
shows *basis X*
proof −
  have *fin*: *finite X*
  proof (*cases n* = *0*)
    case *False*
    with *card* **show** *finite X* **by** (*meson card.infinite*)
  next
    case *True*
    with *X* **have** *X* ⊆ *carrier-vec 0* **by** *auto*
    also have ... = {0_v *0*} **by** *auto*
    finally have *X* ⊆ {0_v *0*} **.**
    from *finite-subset*[*OF this*] **show** *finite X* **by** *auto*
  qed
  from *X* **have** *span X* ⊆ *carrier-vec n* **by** *auto*
  with *span* **have** *span*: *span X* = *carrier-vec n* **by** *auto*
  from *dim-is-n card* **have** *card*: *card X* ≤ *dim* **by** *auto*
  from *dim-gen-is-basis*[*OF fin X span card*] **show** *basis X* **.**
qed

lemma *lincomb-list-append*:
  assumes *Ws*: *set Ws* ⊆ *carrier-vec n*
  shows *set Vs* ⊆ *carrier-vec n* ⟹ *lincomb-list f* (*Vs* @ *Ws*) =
    lincomb-list f Vs + *lincomb-list* (λ *i*. *f* (*i* + *length Vs*)) *Ws*
proof (*induction Vs arbitrary*: *f*)
  case *Nil* **show** *?case* **by**(*simp add*: *lincomb-list-carrier*[*OF Ws*])
next
  case (*Cons x Vs*)
  have *lincomb-list f* (*x* # (*Vs* @ *Ws*)) = *f 0* $\cdot_v$ *x* + *lincomb-list* (*f* ∘ *Suc*) (*Vs* @ *Ws*)
    by (*rule lincomb-list-Cons*)
  also have *lincomb-list* (*f* ∘ *Suc*) (*Vs* @ *Ws*) =
      lincomb-list (*f* ∘ *Suc*) *Vs* + *lincomb-list* (λ *i*. (*f* ∘ *Suc*) (*i* + *length Vs*)) *Ws*
    using *Cons* **by** *auto*
  also have (λ *i*. (*f* ∘ *Suc*) (*i* + *length Vs*)) = (λ *i*. *f* (*i* + *length* (*x* # *Vs*))) **by** *simp*
  also have *f 0* $\cdot_v$ *x* + ((*lincomb-list* (*f* ∘ *Suc*) *Vs*) + *lincomb-list* ... *Ws*) =
      (*f 0* $\cdot_v$ *x* + (*lincomb-list* (*f* ∘ *Suc*) *Vs*)) + *lincomb-list* ... *Ws*
    using *assoc-add-vec Cons.prems Ws lincomb-list-carrier* **by** *auto*
  finally show *?case* **using** *lincomb-list-Cons* **by** *auto*
qed

lemma *lincomb-list-snoc*[*simp*]:
  shows *set Vs* ⊆ *carrier-vec n* ⟹ *x* ∈ *carrier-vec n* ⟹
    lincomb-list f (*Vs* @ [*x*]) = *lincomb-list f Vs* + *f* (*length Vs*) $\cdot_v$ *x*
  using *lincomb-list-append* **by** *auto*

```
lemma lincomb-list-smult:
  set Vs ⊆ carrier-vec n ⟹ lincomb-list (λ i. a ∗ c i) Vs = a ·v lincomb-list c Vs
proof (induction Vs rule: rev-induct)
  case (snoc x Vs)
  have x: x ∈ carrier-vec n and Vs: set Vs ⊆ carrier-vec n using snoc.prems by
auto
  have lincomb-list (λ i. a ∗ c i) (Vs @ [x]) =
        lincomb-list (λ i. a ∗ c i) Vs + (a ∗ c (length Vs)) ·v x
    using x Vs by auto
  also have lincomb-list (λ i. a ∗ c i) Vs = a ·v lincomb-list c Vs
    by(rule snoc.IH[OF Vs])
  also have (a ∗ c (length Vs)) ·v x = a ·v (c (length Vs) ·v x)
    using smult-smult-assoc x by auto
  also have a ·v lincomb-list c Vs + ... = a ·v (lincomb-list c Vs + c (length Vs)
·v x)
    using smult-add-distrib-vec[of - n - a] lincomb-list-carrier[OF Vs] x by simp
  also have lincomb-list c Vs + c (length Vs) ·v x = lincomb-list c (Vs @ [x])
    using Vs x by auto
  finally show ?case by auto
qed simp

lemma lincomb-list-index:
  assumes i: i < n
  shows set Xs ⊆ carrier-vec n ⟹
        lincomb-list c Xs $ i = sum (λ j. c j ∗ (Xs ! j) $ i) {0..<length Xs}
proof (induction Xs rule: rev-induct)
  case (snoc x Xs)
  hence x: x ∈ carrier-vec n and Xs: set Xs ⊆ carrier-vec n by auto
  hence lincomb-list c (Xs @ [x]) = lincomb-list c Xs + c (length Xs) ·v x by auto
  also have ... $ i = lincomb-list c Xs $ i + (c (length Xs) ·v x) $ i
    using i index-add-vec(1) x by simp
  also have (c (length Xs) ·v x) $ i = c (length Xs) ∗ x $ i using i x by simp
  also have x $ i= (Xs @ [x]) ! (length Xs) $ i by simp
  also have lincomb-list c Xs $ i = (∑ j = 0..<length Xs. c j ∗ Xs ! j $ i)
    by (rule snoc.IH[OF Xs])
  also have ... = (∑ j = 0..<length Xs. c j ∗ (Xs @ [x]) ! j $ i)
   by (rule R.finsum-restrict, force, rule restrict-ext, auto simp: append-Cons-nth-left)
  finally show ?case
    using sum.atLeast0-lessThan-Suc[of λ j. c j ∗ (Xs @ [x]) ! j $ i length Xs]
    by fastforce
qed (simp add: i)

end
end
```

4 Basis Extension

We prove that every linear indepent set/list of vectors can be extended into a basis. Similarly, from every set of vectors one can extract a linear independent set of vectors that spans the same space.

theory *Basis-Extension*
  imports
    LLL-Basis-Reduction.Gram-Schmidt-2
begin

context *cof-vec-space*
begin

lemma *lin-indpt-list-length-le-n*: **assumes** *lin-indpt-list xs*
  shows *length xs* ≤ *n*
proof −
  from *assms*[*unfolded lin-indpt-list-def*]
  have *xs*: *set xs* ⊆ *carrier-vec n* **and** *dist*: *distinct xs* **and** *lin*: *lin-indpt* (*set xs*)
by *auto*
  from *dist* **have** *card* (*set xs*) = *length xs* **by** (*rule distinct-card*)
  moreover have *card* (*set xs*) ≤ *n*
    using *lin xs dim-is-n li-le-dim*(*2*) **by** *auto*
  ultimately show *?thesis* **by** *auto*
qed

lemma *lin-indpt-list-length-eq-n*: **assumes** *lin-indpt-list xs*
  and *length xs* = *n*
shows *span* (*set xs*) = *carrier-vec n basis* (*set xs*)
proof −
  from *assms*[*unfolded lin-indpt-list-def*]
  have *xs*: *set xs* ⊆ *carrier-vec n* **and** *dist*: *distinct xs* **and** *lin*: *lin-indpt* (*set xs*)
by *auto*
  from *dist* **have** *card* (*set xs*) = *length xs* **by** (*rule distinct-card*)
  with *assms* **have** *card* (*set xs*) = *n* **by** *auto*
  with *lin xs* **show** *span* (*set xs*) = *carrier-vec n basis* (*set xs*) **using** *dim-is-n*
   by (*metis basis-def dim-basis dim-li-is-basis fin-dim finite-basis-exists gen-ge-dim li-le-dim*(*1*))+
qed

lemma *expand-to-basis*: **assumes** *lin*: *lin-indpt-list xs*
  shows ∃ *ys*. *set ys* ⊆ *set* (*unit-vecs n*) ∧ *lin-indpt-list* (*xs* @ *ys*) ∧ *length* (*xs* @ *ys*) = *n*
proof −
  define *y* **where** *y* = *n* − *length xs*
  from *lin* **have** *length xs* ≤ *n* **by** (*rule lin-indpt-list-length-le-n*)
  hence *length xs* + *y* = *n* **unfolding** *y-def* **by** *auto*
  thus ∃ *ys*. *set ys* ⊆ *set* (*unit-vecs n*) ∧ *lin-indpt-list* (*xs* @ *ys*) ∧ *length* (*xs* @ *ys*) = *n*

```
    using lin
  proof (induct y arbitrary: xs)
    case (0 xs)
    thus ?case by (intro exI[of - Nil], auto)
  next
    case (Suc y xs)
    hence length xs < n by auto
    from Suc(3)[unfolded lin-indpt-list-def]
    have xs: set xs ⊆ carrier-vec n and dist: distinct xs and lin: lin-indpt (set xs)
by auto
    from distinct-card[OF dist] Suc(2) have card: card (set xs) < n by auto
    have span (set xs) ≠ carrier-vec n using card dim-is-n xs basis-def dim-basis
lin by auto
    with span-closed[OF xs] have span (set xs) ⊂ carrier-vec n by auto
    also have carrier-vec n = span (set (unit-vecs n))
      unfolding span-unit-vecs-is-carrier ..
    finally have sub: span (set xs) ⊂ span (set (unit-vecs n)) .
    have ∃ u. u ∈ set (unit-vecs n) ∧ u ∉ span (set xs)
      using span-subsetI[OF xs, of set (unit-vecs n)] sub by force
    then obtain u where uu: u ∈ set (unit-vecs n) and usxs: u ∉ span (set xs)
by auto
    then have u: u ∈ carrier-vec n unfolding unit-vecs-def by auto
    let ?xs = xs @ [u]
    from span-mem[OF xs, of u] usxs have uxs: u ∉ set xs by auto
    with dist have dist: distinct ?xs by auto
    have lin: lin-indpt (set ?xs) using lin-dep-iff-in-span[OF xs lin u uxs] usxs by
auto
    from lin dist u xs have lin: lin-indpt-list ?xs unfolding lin-indpt-list-def by
auto
    from Suc(2) have length ?xs + y = n by auto
    from Suc(1)[OF this lin] obtain ys where
      set ys ⊆ set (unit-vecs n) lin-indpt-list (?xs @ ys) length (?xs @ ys) = n by
auto
    thus ?case using uu
      by (intro exI[of - u # ys], auto)
  qed
qed

definition basis-extension xs = (SOME ys.
  set ys ⊆ set (unit-vecs n) ∧ lin-indpt-list (xs @ ys) ∧ length (xs @ ys) = n)

lemma basis-extension: assumes lin-indpt-list xs
  shows set (basis-extension xs) ⊆ set (unit-vecs n)
    lin-indpt-list (xs @ basis-extension xs)
    length (xs @ basis-extension xs) = n
  using someI-ex[OF expand-to-basis[OF assms], folded basis-extension-def] by
auto

lemma exists-lin-indpt-sublist: assumes X: X ⊆ carrier-vec n
```

```
  shows ∃ Ls. lin-indpt-list Ls ∧ span (set Ls) = span X ∧ set Ls ⊆ X
proof –
  let ?T = ?thesis
  have (∃ Ls. lin-indpt-list Ls ∧ span (set Ls) ⊆ span X ∧ set Ls ⊆ X ∧ length
Ls = k) ∨ ?T for k
  proof (induct k)
    case 0
    have lin-indpt {} by (simp add: lindep-span)
    thus ?case using span-is-monotone by (auto simp: lin-indpt-list-def)
  next
    case (Suc k)
    show ?case
    proof (cases ?T)
      case False
      with Suc obtain Ls where lin: lin-indpt-list Ls
        and span: span (set Ls) ⊆ span X and Ls: set Ls ⊆ X  and len: length Ls
= k by auto
      from Ls X have LsC: set Ls ⊆ carrier-vec n by auto
      show ?thesis
      proof (cases X ⊆ span (set Ls))
        case True
        hence span X ⊆ span (set Ls) using LsC X by (metis span-subsetI)
        with span have span (set Ls) = span X by auto
        hence ?T by (intro exI[of - Ls] conjI True lin Ls)
        thus ?thesis by auto
      next
        case False
         with span obtain x where xX: x ∈ X and xSLs: x ∉ span (set Ls) by
auto
        from Ls X have LsC: set Ls ⊆ carrier-vec n by auto
        from span-mem[OF this, of x] xSLs have xLs: x ∉ set Ls by auto
        let ?Ls = x # Ls
        show ?thesis
        proof (intro disjI1 exI[of - ?Ls] conjI)
          show length ?Ls = Suc k using len by auto
          show lin-indpt-list ?Ls using lin xSLs xLs unfolding lin-indpt-list-def
            using lin-dep-iff-in-span[OF LsC - - xLs] xX X by auto
          show set ?Ls ⊆ X using xX Ls by auto
          from span-is-monotone[OF this]
          show span (set ?Ls) ⊆ span X .
        qed
      qed
    qed auto
  qed
  from this[of n + 1] lin-indpt-list-length-le-n show ?thesis by fastforce
qed

lemma exists-lin-indpt-subset: assumes X ⊆ carrier-vec n
  shows ∃ Ls. lin-indpt Ls ∧ span (Ls) = span X ∧ Ls ⊆ X
```

proof −
  from *exists-lin-indpt-sublist*[*OF assms*]
  obtain *Ls* **where** *lin-indpt-list Ls* ∧ *span* (*set Ls*) = *span X* ∧ *set Ls* ⊆ *X* **by** *auto*
  thus *?thesis* **by** (*intro exI*[*of - set Ls*], *auto simp*: *lin-indpt-list-def*)
qed
end

end

5 Sum of Vector Sets

We use Isabelle's Set-Algebra theory to be able to write V + W for sets of vectors V and W, and prove some obvious properties about them.

theory *Sum-Vec-Set*
  imports
    Missing-Matrix
    HOL−Library.Set-Algebras
begin

lemma *add-0-right-vecset*:
  assumes (*A* :: ′*a* :: *monoid-add vec set*) ⊆ *carrier-vec n*
  shows $A + \{0_v\ n\} = A$
  unfolding *set-plus-def* **using** *assms* **by** *force*

lemma *add-0-left-vecset*:
  assumes (*A* :: ′*a* :: *monoid-add vec set*) ⊆ *carrier-vec n*
  shows $\{0_v\ n\} + A = A$
  unfolding *set-plus-def* **using** *assms* **by** *force*

lemma *assoc-add-vecset*:
  assumes (*A* :: ′*a* :: *semigroup-add vec set*) ⊆ *carrier-vec n*
    and *B* ⊆ *carrier-vec n*
    and *C* ⊆ *carrier-vec n*
  shows *A* + (*B* + *C*) = (*A* + *B*) + *C*
proof −
  {
    fix *x*
    assume *x* ∈ *A* + (*B* + *C*)
    then obtain *a b c* **where** *x* = *a* + (*b* + *c*) **and** ∗: *a* ∈ *A b* ∈ *B c* ∈ *C*
      unfolding *set-plus-def* **by** *auto*
    with *assms* **have** *x* = (*a* + *b*) + *c* **using** *assoc-add-vec*[*of a n b c*] **by** *force*
    with ∗ **have** *x* ∈ (*A* + *B*) + *C* **by** *auto*
  }
  moreover
  {
    fix *x*

assume $x \in (A + B) + C$
then obtain $a\ b\ c$ **where** $x = (a + b) + c$ **and** $*$: $a \in A\ b \in B\ c \in C$
unfolding *set-plus-def* **by** *auto*
with *assms* **have** $x = a + (b + c)$ **using** *assoc-add-vec*[*of a n b c*] **by** *force*
with $*$ **have** $x \in A + (B + C)$ **by** *auto*
}
ultimately show *?thesis* **by** *blast*
qed

lemma *sum-carrier-vec*[*intro*]: $A \subseteq$ *carrier-vec n* $\Longrightarrow B \subseteq$ *carrier-vec n* $\Longrightarrow A + B \subseteq$ *carrier-vec n*
unfolding *set-plus-def* **by** *force*

lemma *comm-add-vecset*:
assumes (A :: *'a* :: *ab-semigroup-add vec set*) $\subseteq$ *carrier-vec n*
and $B \subseteq$ *carrier-vec n*
shows $A + B = B + A$
unfolding *set-plus-def* **using** *comm-add-vec assms* **by** *blast*

end

6 Integral and Bounded Matrices and Vectors

We define notions of integral vectors and matrices and bounded vectors and matrices and prove some preservation lemmas. Moreover, we prove two bounds on determinants.

theory *Integral-Bounded-Vectors*
imports
Missing-VS-Connect
Sum-Vec-Set
LLL-Basis-Reduction.Gram-Schmidt-2
begin

lemma *sq-norm-unit-vec*[*simp*]: **assumes** i: $i < n$
shows $\|$*unit-vec n i*$\|^2 = (1$ :: *'a* :: {*comm-ring-1*,*conjugatable-ring*})
proof −
from i **have** *id*: $[0..<n] = [0..<i]$ @ $[i]$ @ $[$*Suc i* $..< n]$
by (*metis append-Cons append-Nil diff-zero length-upt list-trisect*)
show *?thesis* **unfolding** *sq-norm-vec-def unit-vec-def*
by (*auto simp*: *o-def id*, *subst* (*1 2*) *sum-list-0*, *auto*)
qed

definition *Ints-vec* (‹$\mathbb{Z}_v$›) **where**
$\mathbb{Z}_v = \{x.\ \forall\ i <$ *dim-vec* $x.\ x\ \$\ i \in \mathbb{Z}\}$

definition *indexed-Ints-vec* **where**

indexed-Ints-vec $I = \{x.\ \forall\ i < \textit{dim-vec}\ x.\ i \in I \longrightarrow x\ \$\ i \in \mathbb{Z}\}$

lemma *indexed-Ints-vec-UNIV*: $\mathbb{Z}_v = \textit{indexed-Ints-vec UNIV}$
unfolding *Ints-vec-def indexed-Ints-vec-def* **by** *auto*

lemma *indexed-Ints-vec-subset*: $\mathbb{Z}_v \subseteq \textit{indexed-Ints-vec}\ I$
unfolding *Ints-vec-def indexed-Ints-vec-def* **by** *auto*

lemma *Ints-vec-vec-set*: $v \in \mathbb{Z}_v = (\textit{vec-set}\ v \subseteq \mathbb{Z})$
unfolding *Ints-vec-def vec-set-def* **by** *auto*

definition *Ints-mat* (‹$\mathbb{Z}_m$›) **where**
$\mathbb{Z}_m = \{A.\ \forall\ i < \textit{dim-row}\ A.\ \forall\ j < \textit{dim-col}\ A.\ A\ \$\$\ (i,j) \in \mathbb{Z}\}$

lemma *Ints-mat-elements-mat*: $A \in \mathbb{Z}_m = (\textit{elements-mat}\ A \subseteq \mathbb{Z})$
unfolding *Ints-mat-def elements-mat-def* **by** *force*

lemma *minus-in-Ints-vec-iff*[*simp*]: $(-x) \in \mathbb{Z}_v \longleftrightarrow (x :: {'a} :: \textit{ring-1 vec}) \in \mathbb{Z}_v$
unfolding *Ints-vec-vec-set* **by** (*auto simp*: *minus-in-Ints-iff*)

lemma *minus-in-Ints-mat-iff*[*simp*]: $(-A) \in \mathbb{Z}_m \longleftrightarrow (A :: {'a} :: \textit{ring-1 mat}) \in \mathbb{Z}_m$
unfolding *Ints-mat-elements-mat* **by** (*auto simp*: *minus-in-Ints-iff*)

lemma *Ints-vec-rows-Ints-mat*[*simp*]: $\textit{set}\ (\textit{rows}\ A) \subseteq \mathbb{Z}_v \longleftrightarrow A \in \mathbb{Z}_m$
unfolding *rows-def Ints-vec-def Ints-mat-def* **by** *force*

lemma *unit-vec-integral*[*simp,intro*]: $\textit{unit-vec}\ n\ i \in \mathbb{Z}_v$
unfolding *Ints-vec-def* **by** (*auto simp*: *unit-vec-def*)

lemma *diff-indexed-Ints-vec*:
$x \in \textit{carrier-vec}\ n \Longrightarrow y \in \textit{carrier-vec}\ n \Longrightarrow x \in \textit{indexed-Ints-vec}\ I \Longrightarrow y \in \textit{indexed-Ints-vec}\ I$
$\Longrightarrow x - y \in \textit{indexed-Ints-vec}\ I$
unfolding *indexed-Ints-vec-def* **by** *auto*

lemma *smult-indexed-Ints-vec*: $x \in \mathbb{Z} \Longrightarrow v \in \textit{indexed-Ints-vec}\ I \Longrightarrow x \cdot_v v \in \textit{indexed-Ints-vec}\ I$
unfolding *indexed-Ints-vec-def smult-vec-def* **by** *simp*

lemma *add-indexed-Ints-vec*:
$x \in \textit{carrier-vec}\ n \Longrightarrow y \in \textit{carrier-vec}\ n \Longrightarrow x \in \textit{indexed-Ints-vec}\ I \Longrightarrow y \in \textit{indexed-Ints-vec}\ I$
$\Longrightarrow x + y \in \textit{indexed-Ints-vec}\ I$
unfolding *indexed-Ints-vec-def* **by** *auto*

lemma (**in** *vec-space*) *lincomb-indexed-Ints-vec*: **assumes** *cI*: $\bigwedge x.\ x \in C \Longrightarrow c\ x \in \mathbb{Z}$
and *C*: $C \subseteq \textit{carrier-vec}\ n$
and *CI*: $C \subseteq \textit{indexed-Ints-vec}\ I$

shows *lincomb c C* ∈ *indexed-Ints-vec I*
proof −
  from *C* **have** *id*: *dim-vec* (*lincomb c C*) = *n* **by** *auto*
  show *?thesis* **unfolding** *indexed-Ints-vec-def mem-Collect-eq id*
  proof (*intro allI impI*)
    fix *i*
    assume *i*: *i* < *n* **and** *iI*: *i* ∈ *I*
    have *lincomb c C* $ *i* = (∑ *x*∈*C*. *c x* ∗ *x* $ *i*)
      by (*rule lincomb-index*[*OF i C*])
    also have ... ∈ ℤ
      by (*intro Ints-sum Ints-mult cI*, *insert i iI CI*[*unfolded indexed-Ints-vec-def*] *C*, *force*+)
    finally show *lincomb c C* $ *i* ∈ ℤ **.**
  qed
qed

definition *Bounded-vec* (*b* :: ′*a* :: *linordered-idom*) = {*x* . ∀ *i* < *dim-vec x* . *abs* (*x* $ *i*) ≤ *b*}

lemma *Bounded-vec-vec-set*: *v* ∈ *Bounded-vec b* ⟷ (∀ *x* ∈ *vec-set v*. *abs x* ≤ *b*)
  unfolding *Bounded-vec-def vec-set-def* **by** *auto*

definition *Bounded-mat* (*b* :: ′*a* :: *linordered-idom*) =
  {*A* . (∀ *i* < *dim-row A* . ∀ *j* < *dim-col A*. *abs* (*A* $$ (*i*,*j*)) ≤ *b*)}

lemma *Bounded-mat-elements-mat*: *A* ∈ *Bounded-mat b* ⟷ (∀ *x* ∈ *elements-mat A*. *abs x* ≤ *b*)
  unfolding *Bounded-mat-def elements-mat-def* **by** *auto*

lemma *Bounded-vec-rows-Bounded-mat*[*simp*]: *set* (*rows A*) ⊆ *Bounded-vec B* ⟷ *A* ∈ *Bounded-mat B*
  unfolding *rows-def Bounded-vec-def Bounded-mat-def* **by** *force*

lemma *unit-vec-Bounded-vec*[*simp*,*intro*]: *unit-vec n i* ∈ *Bounded-vec* (*max 1 Bnd*)
  unfolding *Bounded-vec-def unit-vec-def* **by** *auto*

lemma *unit-vec-int-bounds*: *set* (*unit-vecs n*) ⊆ $\mathbb{Z}_v$ ∩ *Bounded-vec* (*of-int* (*max 1 Bnd*))
  unfolding *unit-vecs-def* **by** (*auto simp*: *Bounded-vec-def*)

lemma *Bounded-matD*: **assumes** *A* ∈ *Bounded-mat b*
  A ∈ *carrier-mat nr nc*
shows *i* < *nr* ⟹ *j* < *nc* ⟹ *abs* (*A* $$ (*i*,*j*)) ≤ *b*
  using *assms* **unfolding** *Bounded-mat-def* **by** *auto*

lemma *Bounded-vec-mono*: *b* ≤ *B* ⟹ *Bounded-vec b* ⊆ *Bounded-vec B*
  unfolding *Bounded-vec-def* **by** *auto*

lemma *Bounded-mat-mono*: *b* ≤ *B* ⟹ *Bounded-mat b* ⊆ *Bounded-mat B*

unfolding *Bounded-mat-def* **by** *force*

lemma *finite-Bounded-vec-Max*:
assumes *A*: $A \subseteq$ *carrier-vec n*
and *fin*: *finite A*
shows $A \subseteq$ *Bounded-vec* (*Max* { *abs* (*a* \$ *i*) | *a i*. $a \in A \wedge i < n$})
proof
let *?B* = { *abs* (*a* \$ *i*) | *a i*. $a \in A \wedge i < n$}
have *fin*: *finite ?B*
by (*rule finite-subset*[*of* - (λ (*a*,*i*). *abs* (*a* \$ *i*)) ‘ (*A* × {*0* ..< *n*})], *insert fin*, *auto*)
fix *a*
assume *a*: $a \in A$
show $a \in$ *Bounded-vec* (*Max ?B*)
unfolding *Bounded-vec-def*
by (*standard*, *intro allI impI Max-ge*[*OF fin*], *insert a A*, *force*)
qed

definition *is-det-bound* :: (*nat* ⇒ ′*a* :: *linordered-idom* ⇒ ′*a*) ⇒ *bool* **where**
is-det-bound f = (∀ *A n x*. $A \in$ *carrier-mat n n* ⟶ $A \in$ *Bounded-mat x* ⟶ *abs* (*det A*) ≤ *f n x*)

lemma *is-det-bound-ge-zero*: **assumes** *is-det-bound f*
and $x \geq 0$
shows *f n x* ≥ *0*
using *assms*(*1*)[*unfolded is-det-bound-def*, *rule-format*, *of* 0_m *n n n x*]
using *assms*(*2*) **unfolding** *Bounded-mat-def* **by** *auto*

definition *det-bound-fact* :: *nat* ⇒ ′*a* :: *linordered-idom* ⇒ ′*a* **where**
det-bound-fact n x = *fact n* ∗ (*x*^*n*)

lemma *det-bound-fact*: *is-det-bound det-bound-fact*
unfolding *is-det-bound-def*
proof (*intro allI impI*)
fix *A* :: ′*a* :: *linordered-idom mat* **and** *n x*
assume *A*: $A \in$ *carrier-mat n n*
and *x*: $A \in$ *Bounded-mat x*
show *abs* (*det A*) ≤ *det-bound-fact n x*
proof −
have *abs* (*det A*) = *abs* (∑ *p* | *p permutes* {*0*..<*n*}. *signof p* ∗ (∏ *i* = *0*..<*n*. *A* \$\$ (*i*, *p i*)))
unfolding *det-def*′[*OF A*] **..**
also have ... ≤ (∑ *p* | *p permutes* {*0*..<*n*}. *abs* (*signof p* ∗ (∏ *i* = *0*..<*n*. *A* \$\$ (*i*, *p i*))))
by (*rule sum-abs*)
also have ... = (∑ *p* | *p permutes* {*0*..<*n*}. (∏ *i* = *0*..<*n*. *abs* (*A* \$\$ (*i*, *p i*))))
by (*rule sum.cong*[*OF refl*], *auto simp*: *abs-mult abs-prod sign-def simp flip*: *of-int-abs*)

```
    also have ... ≤ (∑ p | p permutes {0..<n}. (∏ i = 0..<n. x))
      by (intro sum-mono prod-mono conjI Bounded-matD[OF x A], auto)
    also have ... = fact n * x^n by (auto simp add: card-permutations)
    finally show abs (det A) ≤ det-bound-fact n x unfolding det-bound-fact-def
by auto
  qed
qed


lemma (in gram-schmidt-fs) Gramian-determinant-det: assumes A: A ∈ carrier-mat n n
  shows Gramian-determinant (rows A) n = det A * det A
proof –
  have [simp]: mat-of-rows n (rows A) = A using A
    by (intro eq-matI, auto)
  show ?thesis using A
  unfolding Gramian-determinant-def
  by (subst Gramian-matrix-alt-def, force, simp add: Let-def, subst det-mult[of - n],
    auto simp: det-transpose)
qed

lemma (in gram-schmidt-fs-lin-indpt) det-bound-main: assumes rows: rows A = fs
  and A: A ∈ carrier-mat n n
  and n0: n > 0
  and Bnd: A ∈ Bounded-mat c
shows
  (abs (det A))^2 ≤ of-nat n ^ n * c ^ (2 * n)
proof –
  from n0 A Bnd have abs (A $$ (0,0)) ≤ c by (auto simp: Bounded-mat-def)
  hence c0: c ≥ 0 by auto
  from n0 A rows have fs: set fs ≠ {} by (auto simp: rows-def)
  from rows A have len: length fs = n by auto
  have (abs (det A))^2 = det A * det A unfolding power2-eq-square by simp
  also have ... = d n using Gramian-determinant-det[OF A] unfolding rows by simp
  also have ... = (∏ j<n. ‖gso j‖²)
    by (rule Gramian-determinant(1), auto simp: len)
  also have ... ≤ (∏ j<n. N)
    by (rule prod-mono, insert N-gso, auto simp: len)
  also have ... = N^n by simp
  also have ... ≤ (of-nat n * c^2)^n
  proof (rule power-mono)
    show 0 ≤ N using N-ge-0 len n0 by auto
    show N ≤ of-nat n * c^2 unfolding N-def
    proof (intro Max.boundedI, force, use fs in force, clarify)
      fix f
      assume f ∈ set fs
```

```
    from this[folded rows] obtain i where i: i < n and f: f = row A i
      using A unfolding rows-def by auto
    have ‖f‖^2 = (∑ x←list-of-vec (row A i). x^2)
      unfolding f sq-norm-vec-def power2-eq-square by simp
    also have list-of-vec (row A i) = map (λ j. A $$ (i,j)) [0..<n]
      using i A by (intro nth-equalityI, auto)
    also have sum-list (map power2 (map (λj. A $$ (i, j)) [0..<n])) ≤
        sum-list (map (λ j. c^2) [0..<n]) unfolding map-map o-def
    proof (intro sum-list-mono)
      fix j
      assume j ∈ set [0 ..< n]
      hence j: j < n by auto
      from Bnd i j A have |A $$ (i, j)| ≤ c by (auto simp: Bounded-mat-def)
      thus (A $$ (i, j))^2 ≤ c^2
        by (meson abs-ge-zero order-trans power2-le-iff-abs-le)
    qed
    also have ... = (∑ j <n. c^2)
      unfolding interv-sum-list-conv-sum-set-nat by auto
    also have ... = of-nat n * c^2 by auto
    finally show ‖f‖^2 ≤ of-nat n * c^2 .
  qed
 qed
 also have ... = (of-nat n)^n * (c^2 ^ n) by (auto simp: algebra-simps)
 also have ... =  of-nat n ^n * c^(2 * n) unfolding power-mult[symmetric]
   by (simp add: ac-simps)
 finally show ?thesis .
qed


lemma det-bound-hadamard-squared: fixes A::'a :: trivial-conjugatable-linordered-field
mat
 assumes A: A ∈ carrier-mat n n
   and Bnd: A ∈ Bounded-mat c
 shows (abs (det A))^2 ≤ of-nat n ^ n * c ^(2 * n)
proof (cases n > 0)
 case n: True
 from n A Bnd have abs (A $$ (0,0)) ≤ c by (auto simp: Bounded-mat-def)
 hence c0: c ≥ 0 by auto
 let ?us = map (row A) [0 ..< n]
 interpret gso: gram-schmidt-fs n ?us .
 have len: length ?us = n by simp
 have us: set ?us ⊆ carrier-vec n using A by auto
 let ?vs = map gso.gso [0..<n]
 show ?thesis
 proof (cases carrier-vec n ⊆ gso.span (set ?us))
   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
```

```
    show ?thesis unfolding zero using c0 by simp
  next
    case True
    with us len have basis: gso.basis-list ?us unfolding gso.basis-list-def by auto
    note in-dep = gso.basis-list-imp-lin-indpt-list[OF basis]
    interpret gso: gram-schmidt-fs-lin-indpt n ?us
      by (standard) (use in-dep gso.lin-indpt-list-def in auto)
    from gso.det-bound-main[OF - A n Bnd]
    show ?thesis using A by (auto simp: rows-def)
  qed
next
  case False
  with A show ?thesis by auto
qed

definition det-bound-hadamard :: nat ⇒ int ⇒ int where
  det-bound-hadamard n c = (sqrt-int-floor ((int n * c^2)^n))

lemma det-bound-hadamard-altdef[code]:
  det-bound-hadamard n c = (if n = 1 ∨ even n then int n ^ (n div 2) * (abs c)^n
else sqrt-int-floor ((int n * c^2)^n))
proof (cases n = 1 ∨ even n)
  case False
  thus ?thesis unfolding det-bound-hadamard-def by auto
next
  case True
  define thesis where thesis = ?thesis
  have thesis ⟷ sqrt-int-floor ((int n * c^2)^n) = int n ^ (n div 2) * abs c^n
    using True unfolding thesis-def det-bound-hadamard-def by auto
  also have (int n * c^2)^n = int n^n * c^(2 * n)
    unfolding power-mult[symmetric] power-mult-distrib by (simp add: ac-simps)
  also have int n^n = int n ^ (2 * (n div 2)) using True by auto
  also have ... * c^(2 * n) = (int n ^ (n div 2) * c^n)^2
    unfolding power-mult-distrib power-mult[symmetric] by (simp add: ac-simps)
  also have sqrt-int-floor ... = int n ^ (n div 2) * |c| ^ n
   unfolding sqrt-int-floor of-int-power real-sqrt-abs of-int-abs[symmetric] floor-of-int
     abs-mult power-abs by simp
  finally have thesis by auto
  thus ?thesis unfolding thesis-def by auto
qed

lemma det-bound-hadamard: is-det-bound det-bound-hadamard
  unfolding is-det-bound-def
proof (intro allI impI)
  fix A :: int mat and n c
  assume A: A ∈ carrier-mat n n and BndA: A ∈ Bounded-mat c
  let ?h = rat-of-int
  let ?hA = map-mat ?h A
  let ?hc = ?h c
```

from *A* **have** *hA*: *?hA* ∈ *carrier-mat n n* **by** *auto*
from *BndA* **have** *Bnd*: *?hA* ∈ *Bounded-mat ?hc*
  unfolding *Bounded-mat-def*
  by (*auto*, *unfold of-int-abs*[*symmetric*] *of-int-le-iff*, *auto*)
have *sqrt*: *sqrt* ((*real n* ∗ (*real-of-int c*)^2) ^ *n*) ≥ *0*
  by *simp*
from *det-bound-hadamard-squared*[*OF hA Bnd*, *unfolded of-int-hom.hom-det of-int-abs*[*symmetric*]]
have *?h* (|*det A*|^*2*) ≤ *?h* (*int n* ^ *n* ∗ *c* ^ (*2* ∗ *n*)) **by** *simp*
from *this*[*unfolded of-int-le-iff*]
have |*det A*|^*2* ≤ *int n* ^ *n* ∗ *c* ^ (*2* ∗ *n*) **.**
also have . . . = (*int n* ∗ *c*^*2*)^*n* **unfolding** *power-mult power-mult-distrib* **by**
simp
finally have |*det A*|^2 ≤ (*int n* ∗ *c*^{2}) ^ *n* **by** *simp*
hence *sqrt-int-floor* (|*det A*|^2) ≤ *sqrt-int-floor* ((*int n* ∗ *c*^{2}) ^ *n*)
  unfolding *sqrt-int-floor* **by** (*intro floor-mono real-sqrt-le-mono*, *linarith*)
also have *sqrt-int-floor* (|*det A*|^2) = |*det A*| **by** (*simp del*: *of-int-abs add*: *of-int-abs*[*symmetric*])
finally show |*det A*| ≤ *det-bound-hadamard n c* **unfolding** *det-bound-hadamard-def*
by *simp*
qed

lemma *n-pow-n-le-fact-square*: *n* ^ *n* ≤ (*fact n*)^*2*
proof −
  define *ii* **where** *ii* (*i* :: *nat*) = (*n* + *1* − *i*) **for** *i*
  have *id*: *ii* ‘ {*1*..*n*} = {*1*..*n*} **unfolding** *ii-def*
  proof (*auto*, *goal-cases*)
    case (*1 i*)
    hence *i*: *i* = (−) (*Suc n*) (*ii i*) **unfolding** *ii-def* **by** *auto*
    show *?case* **by** (*subst i*, *rule imageI*, *insert 1*, *auto simp*: *ii-def*)
  qed
  have (*fact n*) = (∏{*1*..*n*})
    by (*simp add*: *fact-prod*)
  hence (*fact n*)^*2* = ((∏{*1*..*n*}) ∗ (∏{*1*..*n*})) **by** (*auto simp*: *power2-eq-square*)
  also have . . . = ((∏{*1*..*n*}) ∗ *prod* (λ *i*. *i*) (*ii* ‘ {*1*..*n*}))
    by (*rule arg-cong*[*of* - - λ *x*. (- ∗ *x*)], *rule prod.cong*[*OF id*[*symmetric*]], *auto*)
  also have . . . = ((∏{*1*..*n*}) ∗ *prod ii* {*1*..*n*})
    by (*subst prod.reindex*, *auto simp*: *ii-def inj-on-def*)
  also have . . . = (*prod* (λ *i*. *i* ∗ *ii i*) {*1*..*n*})
    by (*subst prod.distrib*, *auto*)
  also have . . . ≥ (*prod* (λ *i*. *n*) {*1*..*n*})
  proof (*intro prod-mono conjI*, *simp*)
    fix *i*
    assume *i*: *i* ∈ {*1* .. *n*}
    let *?j* = *ii i*
    show *n* ≤ *i* ∗ *?j*
    proof (*cases i* = *1* ∨ *i* = *n*)
      case *True*
      thus *?thesis* **unfolding** *ii-def* **by** *auto*
    next
      case *False*

```
      hence min: min i ?j ≥ 2 using i by (auto simp: ii-def)
      have max: n ≤ 2 * max i ?j using i by (auto simp: ii-def)
      also have ... ≤ min i ?j * max i ?j using min
        by (intro mult-mono, auto)
      also have ... = i * ?j by (cases i < ?j, auto simp: ac-simps)
      finally show ?thesis .
    qed
  qed
  finally show ?thesis by simp
qed

lemma sqrt-int-floor-bound: 0 ≤ x ⟹ (sqrt-int-floor x)^2 ≤ x
  unfolding sqrt-int-floor-def
  using root-int-floor-def root-int-floor-pos-lower by auto

lemma det-bound-hadamard-improves-det-bound-fact: assumes c: c ≥ 0
  shows det-bound-hadamard n c ≤ det-bound-fact n c
proof −
 have (det-bound-hadamard n c)^2 ≤ (int n * c²) ^n unfolding det-bound-hadamard-def
    by (rule sqrt-int-floor-bound, auto)
 also have ... = int (n ^n) * c^(2 * n) by (simp add: power-mult power-mult-distrib)
  also have ... ≤ int ((fact n)^2) * c^(2 * n)
   by (intro mult-right-mono, unfold of-nat-le-iff, rule n-pow-n-le-fact-square, auto)
  also have ... = (det-bound-fact n c)^2 unfolding det-bound-fact-def
    by (simp add: power-mult-distrib power-mult[symmetric] ac-simps)
  finally have abs (det-bound-hadamard n c) ≤ abs (det-bound-fact n c)
    unfolding abs-le-square-iff .
  hence det-bound-hadamard n c ≤ abs (det-bound-fact n c) by simp
   also have ... = det-bound-fact n c unfolding det-bound-fact-def using c by
auto
  finally show ?thesis .
qed

context
begin
private fun syl :: int ⇒ nat ⇒ int mat where
  syl c 0 = mat 1 1 (λ -. c)
| syl c (Suc n) = (let A = syl c n in
     four-block-mat A A (−A) A)

private lemma syl: assumes c: c ≥ 0
  shows syl c n ∈ Bounded-mat c ∧ syl c n ∈ carrier-mat (2^n) (2^n)
    ∧ det (syl c n) = det-bound-hadamard (2^n) c
proof (cases n = 0)
  case True
  thus ?thesis using c
    unfolding det-bound-hadamard-altdef
    by (auto simp: Bounded-mat-def det-single)
next
```

```
  case False
  then obtain m where n: n = Suc m by (cases n, auto)
  show ?thesis unfolding n
  proof (induct m)
    case 0
    show ?case unfolding syl.simps Let-def using c
      apply (subst det-four-block-mat[of - 1]; force?)
      apply (subst det-single,
          auto simp: Bounded-mat-def scalar-prod-def det-bound-hadamard-altdef
power2-eq-square)
      done
  next
    case (Suc m)
    define A where A = syl c (Suc m)
    let ?FB = four-block-mat A A (− A) A
    define n :: nat where n = 2 ^ Suc m
    from Suc[folded A-def n-def]
    have Bnd: A ∈ Bounded-mat c
      and A: A ∈ carrier-mat n n
      and detA: det A = det-bound-hadamard n c
      by auto
    have n2: 2 ^ Suc (Suc m) = 2 ∗ n unfolding n-def by auto
    show ?case unfolding syl.simps(2)[of - Suc m] A-def[symmetric] Let-def n2
    proof (intro conjI)
      show ?FB ∈ carrier-mat (2 ∗ n) (2 ∗ n) using A by auto
     show ?FB ∈ Bounded-mat c using Bnd A unfolding Bounded-mat-elements-mat
        by (subst elements-four-block-mat-id, auto)
      have ev: even n and sum: n div 2 + n div 2 = n unfolding n-def by auto
      have n2: n ∗ 2 = n + n by simp
      have det ?FB = det (A ∗ A − A ∗ − A)
        by (rule det-four-block-mat[OF A A - A], insert A, auto)
      also have A ∗ A − A ∗ − A = A ∗ A + A ∗ A using A by auto
      also have . . . = 2 ·m (A ∗ A) using A by auto
      also have det . . . = 2^n ∗ det (A ∗ A)
        by (subst det-smult, insert A, auto)
      also have det (A ∗ A) = det A ∗ det A by (rule det-mult[OF A A])
      also have 2^n ∗ . . . = det-bound-hadamard (2 ∗ n) c unfolding detA
     unfolding det-bound-hadamard-altdef by (simp add: ev ac-simps power-add[symmetric]
sum n2)
      finally show det ?FB = det-bound-hadamard (2 ∗ n) c .
    qed
  qed
qed

lemma det-bound-hadamard-tight:
    assumes c: c ≥ 0
      and n = 2^m
   shows ∃ A. A ∈ carrier-mat n n ∧ A ∈ Bounded-mat c ∧ det A = det-bound-hadamard
n c
```

by (*rule exI*[*of* - *syl c m*], *insert syl*[*OF c*, *of m*, *folded assms*(*2*)], *auto*)
end

lemma *Ints-matE*: **assumes** $A \in \mathbb{Z}_m$
shows ∃ *B*. *A* = *map-mat of-int B*
proof −
have ∀ *ij*. ∃ *x*. *fst ij* < *dim-row A* ⟶ *snd ij* < *dim-col A* ⟶ *A* $$ *ij* = *of-int x*
using *assms* **unfolding** *Ints-mat-def Ints-def* **by** *auto*
from *choice*[*OF this*] **obtain** *f* **where**
f: ∀ *i j*. *i* < *dim-row A* ⟶ *j* < *dim-col A* ⟶ *A* $$ (*i*,*j*) = *of-int* (*f* (*i*,*j*))
by *auto*
show *?thesis*
by (*intro exI*[*of* - *mat* (*dim-row A*) (*dim-col A*) *f*] *eq-matI*, *insert f*, *auto*)
qed

lemma *is-det-bound-of-int*: **fixes** *A* :: ′*a* :: *linordered-idom mat*
assumes *db*: *is-det-bound db*
and *A*: *A* ∈ *carrier-mat n n*
and $A \in \mathbb{Z}_m \cap$ *Bounded-mat* (*of-int bnd*)
shows *abs* (*det A*) ≤ *of-int* (*db n bnd*)
proof −
from *assms* **have** $A \in \mathbb{Z}_m$ **by** *auto*
from *Ints-matE*[*OF this*] **obtain** *B* **where**
AB: *A* = *map-mat of-int B* **by** *auto*
from *assms* **have** *A* ∈ *Bounded-mat* (*of-int bnd*) **by** *auto*
hence *B* ∈ *Bounded-mat bnd* **unfolding** *AB Bounded-mat-elements-mat*
by (*auto simp flip*: *of-int-abs*)
from *db*[*unfolded is-det-bound-def*, *rule-format*, *OF* - *this*, *of n*] *AB A*
have |*det B*| ≤ *db n bnd* **by** *auto*
thus *?thesis* **unfolding** *AB of-int-hom.hom-det*
by (*simp flip*: *of-int-abs*)
qed

lemma *minus-in-Bounded-vec*[*simp*]:
(−*x*) ∈ *Bounded-vec b* ⟷ *x* ∈ *Bounded-vec b*
unfolding *Bounded-vec-def* **by** *auto*

lemma *sum-in-Bounded-vecI*[*intro*]: **assumes**
xB: *x* ∈ *Bounded-vec B1* **and**
yB: *y* ∈ *Bounded-vec B2* **and**
x: *x* ∈ *carrier-vec n* **and**
y: *y* ∈ *carrier-vec n*
shows *x* + *y* ∈ *Bounded-vec* (*B1* + *B2*)
proof −
from *x y* **have** *id*: *dim-vec* (*x* + *y*) = *n* **by** *auto*
show *?thesis* **unfolding** *Bounded-vec-def mem-Collect-eq id*

```
  proof (intro allI impI)
    fix i
    assume i: i < n
    with x y xB yB have *: abs (x $ i) ≤ B1 abs (y $ i) ≤ B2
      unfolding Bounded-vec-def by auto
    thus |(x + y) $ i| ≤ B1 + B2 using i x y by simp
  qed
qed

lemma (in gram-schmidt) lincomb-card-bound: assumes XBnd: X ⊆ Bounded-vec
Bnd
  and X: X ⊆ carrier-vec n
  and Bnd: Bnd ≥ 0
  and c: ⋀ x. x ∈ X ⟹ abs (c x) ≤ 1
  and card: card X ≤ k
shows lincomb c X ∈ Bounded-vec (of-nat k * Bnd)
proof –
  from X have dim: dim-vec (lincomb c X) = n by auto
  show ?thesis unfolding Bounded-vec-def mem-Collect-eq dim
  proof (intro allI impI)
    fix i
    assume i: i < n
    have abs (lincomb c X $ i) = abs (∑x∈X. c x * x $ i)
      by (subst lincomb-index[OF i X], auto)
    also have ... ≤ (∑x∈X. abs (c x * x $ i)) by auto
    also have ... = (∑x∈X. abs (c x) * abs (x $ i)) by (auto simp: abs-mult)
    also have ... ≤ (∑x∈X. 1 * abs (x $ i))
      by (rule sum-mono[OF mult-right-mono], insert c, auto)
    also have ... = (∑x∈X. abs (x $ i)) by simp
    also have ... ≤ (∑x∈X. Bnd)
      by (rule sum-mono, insert i XBnd[unfolded Bounded-vec-def] X, force)
    also have ... = of-nat (card X) * Bnd by simp
    also have ... ≤ of-nat k * Bnd
      by (rule mult-right-mono[OF - Bnd], insert card, auto)
    finally show abs (lincomb c X $ i) ≤ of-nat k * Bnd by auto
  qed
qed

lemma bounded-vecset-sum:
  assumes Acarr: A ⊆ carrier-vec n
    and Bcarr: B ⊆ carrier-vec n
    and sum: C = A + B
    and Cbnd: ∃ bndC. C ⊆ Bounded-vec bndC
  shows A ≠ {} ⟹ (∃ bndB. B ⊆ Bounded-vec bndB)
    and B ≠ {} ⟹ (∃ bndA. A ⊆ Bounded-vec bndA)
proof –
  {
    fix A B :: 'a vec set
    assume Acarr: A ⊆ carrier-vec n
```

assume *Bcarr*: $B \subseteq$ *carrier-vec n*
assume *sum*: $C = A + B$
assume *Ane*: $A \neq \{\}$
have $\exists$ *bndB*. $B \subseteq$ *Bounded-vec bndB*
proof(*cases* $B = \{\}$)
case *Bne*: *False*
from *Cbnd* **obtain** *bndC* **where** *bndC*: $C \subseteq$ *Bounded-vec bndC* **by** *auto*
from *Ane* **obtain** *a* **where** *aA*: $a \in A$ **and** *acarr*: $a \in$ *carrier-vec n* **using** *Acarr* **by** *auto*
let *?M* = {*abs* (*a* \$ *i*) | *i*. *i* < *n*}
have *finM*: *finite ?M* **by** *simp*
define *nb* **where** *nb* = *abs bndC* + *Max ?M*
{
fix *b*
assume *bB*: $b \in B$ **and** *bcarr*: $b \in$ *carrier-vec n*
have *ab*: $a + b \in$ *Bounded-vec bndC* **using** *aA bB bndC sum* **by** *auto*
{
fix *i*
assume *i-lt-n*: $i < n$
hence *ai-le-max*: *abs*(*a* \$ *i*) ≤ *Max ?M* **using** *acarr finM Max-ge* **by** *blast*
hence *abs*(*a* \$ *i* + *b* \$ *i*) ≤ *abs bndC*
using *ab bcarr acarr index-add-vec*(*1*) *i-lt-n* **unfolding** *Bounded-vec-def* **by** *auto*
hence *abs*(*b* \$ *i*) ≤ *abs bndC* + *abs*(*a* \$ *i*) **by** *simp*
hence *abs*(*b* \$ *i*) ≤ *nb* **using** *i-lt-n bcarr ai-le-max* **unfolding** *nb-def* **by** *simp*
}
hence $b \in$ *Bounded-vec nb* **unfolding** *Bounded-vec-def* **using** *bcarr carrier-vecD* **by** *blast*
}
hence $B \subseteq$ *Bounded-vec nb* **unfolding** *Bounded-vec-def* **using** *Bcarr* **by** *auto*
thus *?thesis* **by** *auto*
qed *auto*
} **note** *theor* = *this*
show $A \neq \{\} \Longrightarrow$ ($\exists$ *bndB*. $B \subseteq$ *Bounded-vec bndB*) **using** *theor*[*OF Acarr Bcarr sum*] **by** *simp*
have *CBA*: $C = B + A$ **unfolding** *sum* **by** (*rule comm-add-vecset*[*OF Acarr Bcarr*])
show $B \neq \{\} \Longrightarrow \exists$ *bndA*. $A \subseteq$ *Bounded-vec bndA* **using** *theor*[*OF Bcarr Acarr CBA*] **by** *simp*
qed

end

7 Cones

We define the notions like cone, polyhedral cone, etc. and prove some basic facts about them.

```
theory Cone
  imports
    Basis-Extension
    Missing-VS-Connect
    Integral-Bounded-Vectors
begin

context gram-schmidt
begin

definition nonneg-lincomb c Vs b = (lincomb c Vs = b ∧ c ‘ Vs ⊆ {x. x ≥ 0})
definition nonneg-lincomb-list c Vs b = (lincomb-list c Vs = b ∧ (∀ i < length
Vs. c i ≥ 0))

definition finite-cone :: 'a vec set ⇒ 'a vec set where
  finite-cone Vs = ({ b. ∃ c. nonneg-lincomb c (if finite Vs then Vs else {}) b})

definition cone :: 'a vec set ⇒ 'a vec set where
  cone Vs = ({ x. ∃ Ws. finite Ws ∧ Ws ⊆ Vs ∧ x ∈ finite-cone Ws})

definition cone-list :: 'a vec list ⇒ 'a vec set where
  cone-list Vs = {b. ∃ c. nonneg-lincomb-list c Vs b}

lemma finite-cone-iff-cone-list: assumes Vs: Vs ⊆ carrier-vec n
  and id: Vs = set Vsl
shows finite-cone Vs = cone-list Vsl
proof –
  have fin: finite Vs unfolding id by auto
  from Vs id have Vsl: set Vsl ⊆ carrier-vec n by auto
  {
    fix c b
    assume b: lincomb c Vs = b and c: c ‘ Vs ⊆ {x. x ≥ 0}
    from lincomb-as-lincomb-list[OF Vsl, of c]
    have b: lincomb-list (λi. if ∃j<i. Vsl ! i = Vsl ! j then 0 else c (Vsl ! i)) Vsl
= b
      unfolding b[symmetric] id by simp
    have ∃ c. nonneg-lincomb-list c Vsl b
      unfolding nonneg-lincomb-list-def
      apply (intro exI conjI, rule b)
      by (insert c, auto simp: set-conv-nth id)
  }
  moreover
  {
    fix c b
    assume b: lincomb-list c Vsl = b and c: (∀ i < length Vsl. c i ≥ 0)
    have nonneg-lincomb (mk-coeff Vsl c) Vs b
      unfolding b[symmetric] nonneg-lincomb-def
      apply (subst lincomb-list-as-lincomb[OF Vsl])
      by (insert c, auto simp: id mk-coeff-def intro!: sum-list-nonneg)
```

hence ∃ *c. nonneg-lincomb c Vs b* **by** *blast*
}
ultimately show *?thesis* **unfolding** *finite-cone-def cone-list-def nonneg-lincomb-def nonneg-lincomb-list-def* **using** *fin* **by** *auto*
qed

lemma *cone-alt-def*: **assumes** *Vs*: $Vs \subseteq carrier\text{-}vec\ n$
shows $cone\ Vs = (\{\ x.\ \exists\ Ws.\ set\ Ws \subseteq Vs \wedge x \in cone\text{-}list\ Ws\})$
unfolding *cone-def*
proof (*intro Collect-cong iffI*)
fix *x*
assume $\exists\ Ws.\ finite\ Ws \wedge Ws \subseteq Vs \wedge x \in finite\text{-}cone\ Ws$
then obtain *Ws* **where** ∗: $finite\ Ws\ Ws \subseteq Vs\ x \in finite\text{-}cone\ Ws$ **by** *auto*
from *finite-list*[*OF* ∗(*1*)] **obtain** *Wsl* **where** *id*: $Ws = set\ Wsl$ **by** *auto*
from *finite-cone-iff-cone-list*[*OF* - *this*] ∗(*2*−*3*) *Vs*
have $x \in cone\text{-}list\ Wsl$ **by** *auto*
with ∗(*2*) *id* **show** $\exists\ Wsl.\ set\ Wsl \subseteq Vs \wedge x \in cone\text{-}list\ Wsl$ **by** *blast*
next
fix *x*
assume $\exists\ Wsl.\ set\ Wsl \subseteq Vs \wedge x \in cone\text{-}list\ Wsl$
then obtain *Wsl* **where** $set\ Wsl \subseteq Vs\ x \in cone\text{-}list\ Wsl$ **by** *auto*
thus $\exists\ Ws.\ finite\ Ws \wedge Ws \subseteq Vs \wedge x \in finite\text{-}cone\ Ws$ **using** *Vs*
by (*intro exI*[*of* - *set Wsl*], *subst finite-cone-iff-cone-list*, *auto*)
qed

lemma *cone-mono*: $Vs \subseteq Ws \Longrightarrow cone\ Vs \subseteq cone\ Ws$
unfolding *cone-def* **by** *blast*

lemma *finite-cone-mono*: **assumes** *fin*: *finite Ws*
and *Ws*: $Ws \subseteq carrier\text{-}vec\ n$
and *sub*: $Vs \subseteq Ws$
shows $finite\text{-}cone\ Vs \subseteq finite\text{-}cone\ Ws$
proof
fix *b*
assume $b \in finite\text{-}cone\ Vs$
then obtain *c* **where** *b*: $b = lincomb\ c\ Vs$ **and** *c*: $c\ `\ Vs \subseteq \{x.\ x \geq 0\}$
unfolding *finite-cone-def nonneg-lincomb-def* **using** *finite-subset*[*OF sub fin*]
by *auto*
define *d* **where** $d = (\lambda\ v.\ if\ v \in Vs\ then\ c\ v\ else\ 0)$
from *c* **have** *d*: $d\ `\ Ws \subseteq \{x.\ x \geq 0\}$ **unfolding** *d-def* **by** *auto*
have $lincomb\ d\ Ws = lincomb\ d\ (Ws - Vs) + lincomb\ d\ Vs$
by (*rule lincomb-vec-diff-add*[*OF Ws sub fin*], *auto*)
also have $lincomb\ d\ Vs = lincomb\ c\ Vs$
by (*rule lincomb-cong*, *insert Ws sub*, *auto simp*: *d-def*)
also have $lincomb\ d\ (Ws - Vs) = 0_v\ n$
by (*rule lincomb-zero*, *insert Ws sub*, *auto simp*: *d-def*)
also have $0_v\ n + lincomb\ c\ Vs = lincomb\ c\ Vs$ **using** *Ws sub* **by** *auto*
also have $\ldots = b$ **unfolding** *b* **by** *simp*
finally

have *b* = *lincomb d Ws* **by** *auto*
then show *b* ∈ *finite-cone Ws* **using** *d fin*
unfolding *finite-cone-def nonneg-lincomb-def* **by** *auto*
qed

lemma *finite-cone-carrier*: *A* ⊆ *carrier-vec n* ⟹ *finite-cone A* ⊆ *carrier-vec n*
unfolding *finite-cone-def nonneg-lincomb-def* **by** *auto*

lemma *cone-carrier*: *A* ⊆ *carrier-vec n* ⟹ *cone A* ⊆ *carrier-vec n*
using *finite-cone-carrier* **unfolding** *cone-def* **by** *blast*

lemma *cone-iff-finite-cone*: **assumes** *A*: *A* ⊆ *carrier-vec n*
and *fin*: *finite A*
shows *cone A* = *finite-cone A*
proof
show *finite-cone A* ⊆ *cone A* **unfolding** *cone-def* **using** *fin* **by** *auto*
show *cone A* ⊆ *finite-cone A* **unfolding** *cone-def* **using** *fin finite-cone-mono*[*OF fin A*] **by** *auto*
qed

lemma *set-in-finite-cone*:
assumes *Vs*: *Vs* ⊆ *carrier-vec n*
and *fin*: *finite Vs*
shows *Vs* ⊆ *finite-cone Vs*
proof
fix *x*
assume *x*: *x* ∈ *Vs*
show *x* ∈ *finite-cone Vs* **unfolding** *finite-cone-def*
proof
let *?c* = λ *y*. *if x* = *y then 1 else 0* :: *'a*
have *Vsx*: *Vs* − {*x*} ⊆ *carrier-vec n* **using** *Vs* **by** *auto*
have *lincomb ?c Vs* = *x* + *lincomb ?c* (*Vs* − {*x*})
using *lincomb-del2 x Vs fin* **by** *auto*
also have *lincomb ?c* (*Vs* − {*x*}) = 0_v *n* **using** *lincomb-zero Vsx* **by** *auto*
also have *x* + 0_v *n* = *x* **using** *M.r-zero Vs x* **by** *auto*
finally have *lincomb ?c Vs* = *x* **by** *auto*
moreover have *?c* ‘ *Vs* ⊆ {*z*. *z* ≥ *0*} **by** *auto*
ultimately show ∃ *c*. *nonneg-lincomb c* (*if finite Vs then Vs else* {}) *x*
unfolding *nonneg-lincomb-def*
using *fin* **by** *auto*
qed
qed

lemma *set-in-cone*:
assumes *Vs*: *Vs* ⊆ *carrier-vec n*
shows *Vs* ⊆ *cone Vs*
proof
fix *x*
assume *x*: *x* ∈ *Vs*

```
  show x ∈ cone Vs unfolding cone-def
  proof (intro CollectI exI)
    have x ∈ carrier-vec n using Vs x by auto
    then have x ∈ finite-cone {x} using set-in-finite-cone by auto
    then show finite {x} ∧ {x} ⊆ Vs ∧ x ∈ finite-cone {x} using x by auto
  qed
qed

lemma zero-in-finite-cone:
  assumes Vs: Vs ⊆ carrier-vec n
  shows 0v n ∈ finite-cone Vs
proof –
  let ?Vs = (if finite Vs then Vs else {})
  have lincomb (λ x. 0 :: 'a) ?Vs = 0v n using lincomb-zero Vs by auto
  moreover have (λ x. 0 :: 'a) ` ?Vs ⊆ {y. y ≥ 0} by auto
  ultimately show ?thesis unfolding finite-cone-def nonneg-lincomb-def by blast
qed

lemma lincomb-in-finite-cone:
  assumes x = lincomb l W
    and finite W
    and ∀ i ∈ W . l i ≥ 0
    and W ⊆ carrier-vec n
  shows x ∈ finite-cone W
   using cone-iff-finite-cone assms unfolding finite-cone-def nonneg-lincomb-def
by auto

lemma lincomb-in-cone:
  assumes x = lincomb l W
    and finite W
    and ∀ i ∈ W . l i ≥ 0
    and W ⊆ carrier-vec n
  shows x ∈ cone W
   using cone-iff-finite-cone assms unfolding finite-cone-def nonneg-lincomb-def
by auto

lemma zero-in-cone: 0v n ∈ cone Vs
proof –
  have finite {} by auto
  moreover have {} ⊆ cone Vs by auto
  moreover have 0v n ∈ finite-cone {} using zero-in-finite-cone by auto
  ultimately show ?thesis unfolding cone-def by blast
qed

lemma cone-smult:
  assumes a: a ≥ 0
    and Vs: Vs ⊆ carrier-vec n
    and x: x ∈ cone Vs
  shows a ·v x ∈ cone Vs
```

proof −
  from *x Vs* **obtain** *Ws c* **where** *Ws*: *Ws* ⊆ *Vs* **and** *fin*: *finite Ws* **and**
    nonneg-lincomb c Ws x
    unfolding *cone-def finite-cone-def* **by** *auto*
  then have *nonneg-lincomb* (λ *w*. *a* ∗ *c w*) *Ws* (*a* $\cdot_v$ *x*)
    unfolding *nonneg-lincomb-def* **using** *a lincomb-distrib Vs* **by** *auto*
  then show *?thesis* **using** *Ws fin* **unfolding** *cone-def finite-cone-def* **by** *auto*
qed

lemma *finite-cone-empty*[*simp*]: *finite-cone* {} = {0_v *n*}
  by (*auto simp*: *finite-cone-def nonneg-lincomb-def*)

lemma *cone-empty*[*simp*]: *cone* {} = {0_v *n*}
  unfolding *cone-def* **by** *simp*

lemma *cone-elem-sum*:
  assumes *Vs*: *Vs* ⊆ *carrier-vec n*
    and *x*: *x* ∈ *cone Vs*
    and *y*: *y* ∈ *cone Vs*
  shows *x* + *y* ∈ *cone Vs*
proof −
  obtain *Xs* **where** *Xs*: *Xs* ⊆ *Vs* **and** *fin-Xs*: *finite Xs*
    and *Xs-cone*: *x* ∈ *finite-cone Xs*
    using *Vs x* **unfolding** *cone-def* **by** *auto*
  obtain *Ys* **where** *Ys*: *Ys* ⊆ *Vs* **and** *fin-Ys*: *finite Ys*
    and *Ys-cone*: *y* ∈ *finite-cone Ys*
    using *Vs y* **unfolding** *cone-def*
    by *auto*
  have *x* ∈ *finite-cone* (*Xs* ∪ *Ys*) **and** *y* ∈ *finite-cone* (*Xs* ∪ *Ys*)
    using *finite-cone-mono fin-Xs fin-Ys Xs Ys Vs Xs-cone Ys-cone*
    by (*blast*, *blast*)
  then obtain *cx cy* **where** *nonneg-lincomb cx* (*Xs* ∪ *Ys*) *x*
    and *nonneg-lincomb cy* (*Xs* ∪ *Ys*) *y*
    unfolding *finite-cone-def* **using** *fin-Xs fin-Ys* **by** *auto*
  hence *nonneg-lincomb* (λ *v*. *cx v* + *cy v*) (*Xs* ∪ *Ys*) (*x* + *y*)
    unfolding *nonneg-lincomb-def*
    using *lincomb-sum*[*of Xs* ∪ *Ys cx cy*] *fin-Xs fin-Ys Xs Ys Vs*
    by *fastforce*
  hence *x* + *y* ∈ *finite-cone* (*Xs* ∪ *Ys*)
    unfolding *finite-cone-def* **using** *fin-Xs fin-Ys* **by** *auto*
  thus *?thesis* **unfolding** *cone-def* **using** *fin-Xs fin-Ys Xs Ys* **by** *auto*
qed

lemma *cone-cone*:
  assumes *Vs*: *Vs* ⊆ *carrier-vec n*
  shows *cone* (*cone Vs*) = *cone Vs*
proof
  show *cone Vs* ⊆ *cone* (*cone Vs*)

```
    by (rule set-in-cone[OF cone-carrier[OF Vs]])
next
  show cone (cone Vs) ⊆ cone Vs
  proof
    fix x
    assume x: x ∈ cone (cone Vs)
    then obtain Ws c where Ws: set Ws ⊆ cone Vs
      and c: nonneg-lincomb-list c Ws x
      using cone-alt-def Vs cone-carrier unfolding cone-list-def by auto

    have set Ws ⊆ cone Vs ⟹ nonneg-lincomb-list c Ws x ⟹ x ∈ cone Vs
    proof (induction Ws arbitrary: x c)
      case Nil
      hence x = 0_v n unfolding nonneg-lincomb-list-def by auto
      thus x ∈ cone Vs using zero-in-cone by auto
    next
      case (Cons a Ws)
      have a ∈ cone Vs using Cons.prems(1) by auto
      moreover have c 0 ≥ 0
        using Cons.prems(2) unfolding nonneg-lincomb-list-def by fastforce
      ultimately have c 0 ·_v a ∈ cone Vs using cone-smult Vs by auto
      moreover have lincomb-list (c ∘ Suc) Ws ∈ cone Vs
        using Cons unfolding nonneg-lincomb-list-def by fastforce
      moreover have x = c 0 ·_v a + lincomb-list (c ∘ Suc) Ws
        using Cons.prems(2) unfolding nonneg-lincomb-list-def
        by auto
      ultimately show x ∈ cone Vs using cone-elem-sum Vs by auto
    qed

    thus x ∈ cone Vs using Ws c by auto
  qed
qed

lemma cone-smult-basis:
  assumes Vs: Vs ⊆ carrier-vec n
    and l: l ‘ Vs ⊆ {x. x > 0}
  shows cone {l v ·_v v | v . v ∈ Vs} = cone Vs
proof
  have {l v ·_v v |v. v ∈ Vs} ⊆ cone Vs
  proof
    fix x
    assume x ∈ {l v ·_v v | v. v ∈ Vs}
    then obtain v where v ∈ Vs and x = l v ·_v v by auto
    thus x ∈ cone Vs using
        set-in-cone[OF Vs] cone-smult[OF - Vs, of l v v] l by fastforce
  qed
  thus cone {l v ·_v v | v. v ∈ Vs} ⊆ cone Vs
    using cone-mono cone-cone[OF Vs] by blast
next
```

have *lVs*: {*l v* $\cdot_v$ *v* | *v*. *v* ∈ *Vs*} ⊆ *carrier-vec n* **using** *Vs* **by** *auto*
have *Vs* ⊆ *cone* {*l v* $\cdot_v$ *v* | *v*. *v* ∈ *Vs*}
proof
fix *v* **assume** *v*: *v* ∈ *Vs*
hence *l v* $\cdot_v$ *v* ∈ *cone* {*l v* $\cdot_v$ *v* | *v*. *v* ∈ *Vs*} **using** *set-in-cone*[*OF lVs*] **by** *auto*
moreover have *1* / *l v* > *0* **using** *l v* **by** *auto*
ultimately have (*1* / *l v*) $\cdot_v$ (*l v* $\cdot_v$ *v*) ∈ *cone* {*l v* $\cdot_v$ *v* | *v*. *v* ∈ *Vs*}
using *cone-smult*[*OF - lVs*] **by** *auto*
also have (*1* / *l v*) $\cdot_v$ (*l v* $\cdot_v$ *v*) = *v* **using** *l v*
by(*auto simp add*: *smult-smult-assoc*)
finally show *v* ∈ *cone* {*l v* $\cdot_v$ *v* | *v*. *v* ∈ *Vs*} **by** *auto*
qed
thus *cone Vs* ⊆ *cone* {*l v* $\cdot_v$ *v* | *v*. *v* ∈ *Vs*}
using *cone-mono cone-cone*[*OF lVs*] **by** *blast*
qed

lemma *cone-add-cone*:
assumes *C*: *C* ⊆ *carrier-vec n*
shows *cone C* + *cone C* = *cone C*
proof
note *CC* = *cone-carrier*[*OF C*]
have *cone C* = *cone C* + {0_v *n*} **by** (*subst add-0-right-vecset*[*OF CC*], *simp*)
also have ... ⊆ *cone C* + *cone C*
by (*rule set-plus-mono2*, *insert zero-in-cone*, *auto*)
finally show *cone C* ⊆ *cone C* + *cone C* **by** *auto*
from *cone-elem-sum*[*OF C*]
show *cone C* + *cone C* ⊆ *cone C*
by (*auto elim*!: *set-plus-elim*)
qed

lemma *orthogonal-cone*:
assumes *X*: *X* ⊆ *carrier-vec n*
and *W*: *W* ⊆ *carrier-vec n*
and *finX*: *finite X*
and *spanLW*: *span* (*set Ls* ∪ *W*) = *carrier-vec n*
and *ortho*: ⋀ *w x*. *w* ∈ *W* ⟹ *x* ∈ *set Ls* ⟹ *w* • *x* = *0*
and *WWs*: *W* = *set Ws*
and *spanL*: *span* (*set Ls*) = *span X*
and *LX*: *set Ls* ⊆ *X*
and *lin-Ls-Bs*: *lin-indpt-list* (*Ls* @ *Bs*)
and *len-Ls-Bs*: *length* (*Ls* @ *Bs*) = *n*
shows *cone* (*X* ∪ *set Bs*) ∩ {*x* ∈ *carrier-vec n*. ∀ *w*∈*W*. *w* • *x* = *0*} = *cone X*
⋀ *x*. ∀ *w*∈*W*. *w* • *x* = *0* ⟹ *Z* ⊆ *X* ⟹ *B* ⊆ *set Bs* ⟹ *x* = *lincomb c* (*Z* ∪
B)
⟹ *x* = *lincomb c* (*Z* − *B*)
proof −
from *WWs* **have** *finW*: *finite W* **by** *auto*
define *Y* **where** *Y* = *X* ∪ *set Bs*
from *lin-Ls-Bs*[*unfolded lin-indpt-list-def*] **have**

```
  Ls: set Ls ⊆ carrier-vec n and
  Bs: set Bs ⊆ carrier-vec n and
  distLsBs: distinct (Ls @ Bs) and
  lin: lin-indpt (set (Ls @ Bs))  by auto
have LW: set Ls ∩ W = {}
proof (rule ccontr)
  assume ¬ ?thesis
  then obtain x where xX: x ∈ set Ls and xW: x ∈ W by auto
  from ortho[OF xW xX] have x · x = 0 by auto
  hence sq-norm x = 0 by (auto simp: sq-norm-vec-as-cscalar-prod)
  with vs-zero-lin-dep[OF - lin] xX Ls Bs show False by auto
qed
have Y: Y ⊆ carrier-vec n using X Bs unfolding Y-def by auto
have CLB: carrier-vec n = span (set (Ls @ Bs))
  using lin-Ls-Bs len-Ls-Bs lin-indpt-list-length-eq-n by blast
also have ... ⊆ span Y
  by (rule span-is-monotone, insert LX, auto simp: Y-def)
finally have span: span Y = carrier-vec n using Y by auto
have finY: finite Y using finX finW unfolding Y-def by auto
{
  fix x Z B d
  assume xX: ∀ w∈W. w · x = 0 and ZX: Z ⊆ X and B: B ⊆ set Bs and
    xd: x = lincomb d (Z ∪ B)
  from ZX B X Bs have ZB: Z ∪ B ⊆ carrier-vec n by auto
  with xd have x: x ∈ carrier-vec n by auto
  from xX W have w0: w ∈ W ⟹ w · x = 0 for w by auto
  from finite-in-span[OF - - x[folded spanLW]] Ls X W finW finX
  obtain c where xc: x = lincomb c (set Ls ∪ W) by auto
  have x = lincomb c (set Ls ∪ W) unfolding xc by auto
  also have ... = lincomb c (set Ls) + lincomb c W
    by (rule lincomb-union, insert X LX W LW finW, auto)
  finally have xsum: x = lincomb c (set Ls) + lincomb c W .
  {
    fix w
    assume wW: w ∈ W
    with W have w: w ∈ carrier-vec n by auto
    from w0[OF wW, unfolded xsum]
    have 0 = w · (lincomb c (set Ls) + lincomb c W) by simp
    also have ... = w · lincomb c (set Ls) + w · lincomb c W
      by (rule scalar-prod-add-distrib[OF w], insert Ls W, auto)
    also have w · lincomb c (set Ls) = 0 using ortho[OF wW]
      by (subst lincomb-scalar-prod-right[OF Ls w], auto)
    finally have w · lincomb c W = 0 by simp
  }
  hence lincomb c W · lincomb c W = 0 using W
    by (subst lincomb-scalar-prod-left, auto)
  hence sq-norm (lincomb c W) = 0
    by (auto simp: sq-norm-vec-as-cscalar-prod)
  hence 0: lincomb c W = 0_v n using lincomb-closed[OF W, of c] by simp
```

have *xc*: $x = lincomb\ c\ (set\ Ls)$ **unfolding** *xsum 0* **using** *Ls* **by** *auto*
hence *xL*: $x \in span\ (set\ Ls)$ **by** *auto*
let $?X = Z - B$
have $lincomb\ d\ ?X \in span\ X$ **using** *finite-subset*[*OF - finX*, *of ?X*] *X ZX* **by** *auto*
from *finite-in-span*[*OF finite-set Ls this*[*folded spanL*]]
obtain *e* **where** *ed*: $lincomb\ e\ (set\ Ls) = lincomb\ d\ ?X$ **by** *auto*
from *B finite-subset*[*OF B*] **have** *finB*: *finite B* **by** *auto*
from *B Bs* **have** *BC*: $B \subseteq carrier\text{-}vec\ n$ **by** *auto*
define *f* **where** *f* =
(λ *x*. *if* $x \in set\ Bs$ *then if* $x \in B$ *then d x else 0 else if* $x \in set\ Ls$ *then e x else undefined*)
have $x = lincomb\ d\ (?X \cup B)$ **unfolding** *xd* **by** *auto*
also have ... $= lincomb\ d\ ?X + lincomb\ d\ B$
by (*rule lincomb-union*[*OF - - - finite-subset*[*OF - finX*]], *insert ZX X finB B Bs*, *auto*)
finally have *xd*: $x = lincomb\ d\ ?X + lincomb\ d\ B$ **.**
also have ... $= lincomb\ e\ (set\ Ls) + lincomb\ d\ B$ **unfolding** *ed* **by** *auto*
also have $lincomb\ e\ (set\ Ls) = lincomb\ f\ (set\ Ls)$
by (*rule lincomb-cong*[*OF - Ls*], *insert distLsBs*, *auto simp*: *f-def*)
also have $lincomb\ d\ B = lincomb\ f\ B$
by (*rule lincomb-cong*[*OF - BC*], *insert B*, *auto simp*: *f-def*)
also have $lincomb\ f\ B = lincomb\ f\ (B \cup (set\ Bs - B))$
by (*subst lincomb-clean*, *insert finB Bs B*, *auto simp*: *f-def*)
also have $B \cup (set\ Bs - B) = set\ Bs$ **using** *B* **by** *auto*
finally have $x = lincomb\ f\ (set\ Ls) + lincomb\ f\ (set\ Bs)$ **by** *auto*
also have $lincomb\ f\ (set\ Ls) + lincomb\ f\ (set\ Bs) = lincomb\ f\ (set\ (Ls\ @\ Bs))$
by (*subst lincomb-union*[*symmetric*], *insert Ls distLsBs Bs*, *auto*)
finally have $x = lincomb\ f\ (set\ (Ls\ @\ Bs))$ **.**
hence *f*: $f \in set\ (Ls\ @\ Bs) \to_E UNIV \wedge lincomb\ f\ (set\ (Ls\ @\ Bs)) = x$
by (*auto simp*: *f-def split*: *if-splits*)
from *finite-in-span*[*OF finite-set Ls xL*] **obtain** *g* **where**
xg: $x = lincomb\ g\ (set\ Ls)$ **by** *auto*
define *h* **where** *h* = (λ *x*. *if* $x \in set\ Bs$ *then 0 else if* $x \in set\ Ls$ *then g x else undefined*)
have $x = lincomb\ h\ (set\ Ls)$ **unfolding** *xg*
by (*rule lincomb-cong*[*OF - Ls*], *insert distLsBs*, *auto simp*: *h-def*)
also have ... $= lincomb\ h\ (set\ Ls) + 0_v\ n$ **using** *Ls* **by** *auto*
also have $0_v\ n = lincomb\ h\ (set\ Bs)$
by (*rule lincomb-zero*[*symmetric*, *OF Bs*], *auto simp*: *h-def*)
also have $lincomb\ h\ (set\ Ls) + lincomb\ h\ (set\ Bs) = lincomb\ h\ (set\ (Ls\ @\ Bs))$
by (*subst lincomb-union*[*symmetric*], *insert Ls Bs distLsBs*, *auto*)
finally have $x = lincomb\ h\ (set\ (Ls\ @\ Bs))$ **.**
hence *h*: $h \in set\ (Ls\ @\ Bs) \to_E UNIV \wedge lincomb\ h\ (set\ (Ls\ @\ Bs)) = x$
by (*auto simp*: *h-def split*: *if-splits*)
have *basis*: *basis* (*set* (*Ls* @ *Bs*)) **using** *lin-Ls-Bs*[*unfolded lin-indpt-list-def*] *len-Ls-Bs*
using *CLB basis-def* **by** *blast*
from *Ls Bs* **have** $set\ (Ls\ @\ Bs) \subseteq carrier\text{-}vec\ n$ **by** *auto*

```
    from basis[unfolded basis-criterion[OF finite-set this], rule-format, OF x] f h
    have fh: f = h by auto
    hence ⋀ x. x ∈ set Bs ⟹ f x = 0 unfolding h-def by auto
    hence ⋀ x. x ∈ B ⟹ d x = 0 unfolding f-def using B by force
    thus x = lincomb d ?X unfolding xd
      by (subst (2) lincomb-zero, insert BC ZB X, auto intro!: M.r-zero)
  } note main = this
  have cone Y ∩ {x ∈ carrier-vec n. ∀ w∈W. w · x = 0} = cone X (is ?I = -)
  proof
    {
      fix x
      assume xX: x ∈ cone X
      with cone-carrier[OF X] have x: x ∈ carrier-vec n by auto
      have X ⊆ Y unfolding Y-def by auto
      from cone-mono[OF this] xX have xY: x ∈ cone Y by auto
      from cone-iff-finite-cone[OF X finX] xX have x ∈ finite-cone X by auto
      from this[unfolded finite-cone-def nonneg-lincomb-def] finX obtain c
        where x = lincomb c X by auto
      with finX X have x ∈ span X by auto
      with spanL have x ∈ span (set Ls) by auto
      from finite-in-span[OF - Ls this] obtain c where
        xc: x = lincomb c (set Ls) by auto
      {
        fix w
        assume wW: w ∈ W
        hence w: w ∈ carrier-vec n using W by auto
        have w · x = 0 unfolding xc using ortho[OF wW]
          by (subst lincomb-scalar-prod-right[OF Ls w], auto)
      }
      with xY x have x ∈ ?I by blast
    }
    thus cone X ⊆ ?I by blast
    {
      fix x
      let ?X = X − set Bs
      assume x ∈ ?I
      with cone-carrier[OF Y] cone-iff-finite-cone[OF Y finY]
      have xY: x ∈ finite-cone Y and x: x ∈ carrier-vec n
        and w0: ⋀ w. w ∈ W ⟹ w · x = 0 by auto
      from xY[unfolded finite-cone-def nonneg-lincomb-def] finY obtain d
        where xd: x = lincomb d Y and nonneg: d ‘ Y ⊆ Collect ((≤) 0) by auto
      from main[OF - - - xd[unfolded Y-def]] w0
      have x = lincomb d ?X by auto
      hence nonneg-lincomb d ?X x unfolding nonneg-lincomb-def
        using nonneg[unfolded Y-def] by auto
      hence x ∈ finite-cone ?X using finX
        unfolding finite-cone-def by auto
      hence x ∈ cone X using finite-subset[OF - finX, of ?X] unfolding cone-def
by blast
```

```
    }
    then show ?I ⊆ cone X by auto
  qed
  thus cone (X ∪ set Bs) ∩ {x ∈ carrier-vec n. ∀ w∈W. w · x = 0} = cone X
unfolding Y-def .
qed

definition polyhedral-cone (A :: 'a mat) = { x . x ∈ carrier-vec n ∧ A *v x ≤ 0v
(dim-row A)}

lemma polyhedral-cone-carrier: assumes A ∈ carrier-mat nr n
  shows polyhedral-cone A ⊆ carrier-vec n
  using assms unfolding polyhedral-cone-def by auto

lemma cone-in-polyhedral-cone:
  assumes CA: C ⊆ polyhedral-cone A
    and A: A ∈ carrier-mat nr n
  shows cone C ⊆ polyhedral-cone A
proof
  interpret nr: gram-schmidt nr TYPE ('a).
  from polyhedral-cone-carrier[OF A] assms(1)
  have C: C ⊆ carrier-vec n by auto
  fix x
  assume x: x ∈ cone C
  then have xn: x ∈ carrier-vec n
    using cone-carrier[OF C] by auto
  from x[unfolded cone-alt-def[OF C] cone-list-def nonneg-lincomb-list-def]
  obtain ll Ds where l0: lincomb-list ll Ds = x and l1: ∀ i<length Ds. 0 ≤ ll i
    and DsC: set Ds ⊆ C
    by auto
  from DsC C have Ds: set Ds ⊆ carrier-vec n by auto

  have A *v x = A *v (lincomb-list ll Ds) using l0 by auto
  also have ... = nr.lincomb-list ll (map (λ d. A *v d) Ds)
  proof –
    have one: ∀ w∈set Ds. dim-vec w = n using DsC C by auto
     have two: ∀ w∈set (map ((*v) A) Ds). dim-vec w = nr using A DsC C by
auto
    show A *v lincomb-list ll Ds = nr.lincomb-list ll (map ((*v) A) Ds)
      unfolding lincomb-list-as-mat-mult[OF one] nr.lincomb-list-as-mat-mult[OF
two] length-map
     proof (subst assoc-mult-mat-vec[symmetric, OF A], force+, rule arg-cong[of -
- λ x. x *v -])
      show A * mat-of-cols n Ds = mat-of-cols nr (map ((*v) A) Ds)
        unfolding mat-of-cols-def
        by (intro eq-matI, insert A Ds[unfolded set-conv-nth],
            (force intro!: arg-cong[of - - λ x. row A - · x])+)
    qed
  qed
```

```
  also have … ≤ 0_v nr
  proof (intro lesseq-vecI[of - nr])
    have ∗: set (map ((∗_v) A) Ds) ⊆ carrier-vec nr using A Ds by auto
    show Carr: nr.lincomb-list ll (map ((∗_v) A) Ds) ∈ carrier-vec nr
      by (intro nr.lincomb-list-carrier[OF ∗])
    fix i
    assume i: i < nr
    from CA[unfolded polyhedral-cone-def] A
    have l2: x ∈ C ⟹ A ∗_v x ≤ 0_v nr for x by auto
    show nr.lincomb-list ll (map ((∗_v) A) Ds) $ i ≤ 0_v nr $ i
     unfolding subst nr.lincomb-list-index[OF i ∗] length-map index-zero-vec(1)[OF
i]
    proof (intro sum-nonpos mult-nonneg-nonpos)
      fix j
      assume j ∈ {0..<length Ds}
      hence j: j < length Ds by auto
      from j show 0 ≤ ll j using l1 by auto
      from j have Ds ! j ∈ C using DsC by auto
      from l2[OF this] have l2: A ∗_v Ds ! j ≤ 0_v nr by auto
      from lesseq-vecD[OF - this i] i have (A ∗_v Ds ! j) $ i ≤ 0 by auto
      thus map ((∗_v) A) Ds ! j $ i ≤ 0 using j i by auto
    qed
  qed auto
  finally show x ∈ polyhedral-cone A
    unfolding polyhedral-cone-def using A xn by auto
qed

lemma bounded-cone-is-zero:
  assumes Ccarr: C ⊆ carrier-vec n and bnd: cone C ⊆ Bounded-vec bnd
  shows cone C = {0_v n}
proof(rule ccontr)
  assume ¬ ?thesis
  then obtain v where vC: v ∈ cone C and vnz: v ≠ 0_v n
    using zero-in-cone assms by auto
  have vcarr: v ∈ carrier-vec n using vC Ccarr cone-carrier by blast
  from vnz vcarr obtain i where i-le-n: i < dim-vec v and vinz: v $ i ≠ 0 by
force
  define M where M = (1 / (v $ i) ∗ (bnd + 1))
  have abs-ge-bnd: abs (M ∗ (v $ i)) > bnd unfolding M-def by (simp add: vinz)
  have aMvC: (abs M) ·_v v ∈ cone C using cone-smult[OF - Ccarr vC] abs-ge-bnd
by simp
  have ¬(abs (abs M ∗ (v $ i)) ≤ bnd) using abs-ge-bnd by simp
  hence (abs M) ·_v v ∉ Bounded-vec bnd unfolding Bounded-vec-def using i-le-n
aMvC by auto
  thus False using aMvC bnd by auto
qed

lemma cone-of-cols: fixes A :: 'a mat and b :: 'a vec
  assumes A: A ∈ carrier-mat n nr and b: b ∈ carrier-vec n
```

```
  shows b ∈ cone (set (cols A)) ⟷ (∃ x. x ≥ 0v nr ∧ A *v x = b)
proof –
  let ?C = set (cols A)
  from A have C: ?C ⊆ carrier-vec n and C': ∀ w∈set (cols A). dim-vec w = n
    unfolding cols-def by auto
  have id: finite ?C = True length (cols A) = nr using A by auto
  have Aid: mat-of-cols n (cols A) = A using A unfolding mat-of-cols-def
    by (intro eq-matI, auto)
  show ?thesis
     unfolding cone-iff-finite-cone[OF C finite-set] finite-cone-iff-cone-list[OF C
refl]
    unfolding cone-list-def nonneg-lincomb-list-def mem-Collect-eq id
    unfolding lincomb-list-as-mat-mult[OF C'] id Aid
  proof –
    {
      fix x
      assume x≥0v nr A *v x = b
      hence ∃ c. A *v vec nr c = b ∧ (∀ i<nr. 0 ≤ c i) using A b
        by (intro exI[of - λ i. x $ i], auto simp: less-eq-vec-def intro!: arg-cong[of -
- (*v) A])
    }
    moreover
    {
      fix c
      assume A *v vec nr c = b (∀ i<nr. 0 ≤ c i)
      hence ∃ x. x≥0v nr ∧ A *v x = b
        by (intro exI[of - vec nr c], auto simp: less-eq-vec-def)
    }
    ultimately show (∃ c. A *v vec nr c = b ∧ (∀ i<nr. 0 ≤ c i)) = (∃ x≥0v nr.
A *v x = b) by blast
  qed
qed

end
end
```

8 Convex Hulls

We define the notion of convex hull of a set or list of vectors and derive basic properties thereof.

```
theory Convex-Hull
  imports Cone
begin

context gram-schmidt
begin

definition convex-lincomb c Vs b = (nonneg-lincomb c Vs b ∧ sum c Vs = 1)
```

definition *convex-lincomb-list c Vs b* = (*nonneg-lincomb-list c Vs b* ∧ *sum c* {*0..<length Vs*} = *1*)

definition *convex-hull Vs* = {*x*. ∃ *Ws c*. *finite Ws* ∧ *Ws* ⊆ *Vs* ∧ *convex-lincomb c Ws x*}

lemma *convex-hull-carrier*[*intro*]: *Vs* ⊆ *carrier-vec n* ⟹ *convex-hull Vs* ⊆ *carrier-vec n*
  unfolding *convex-hull-def convex-lincomb-def nonneg-lincomb-def* **by** *auto*

lemma *convex-hull-mono*: *Vs* ⊆ *Ws* ⟹ *convex-hull Vs* ⊆ *convex-hull Ws*
  unfolding *convex-hull-def* **by** *auto*

lemma *convex-lincomb-empty*[*simp*]: ¬ (*convex-lincomb c* {} *x*)
  unfolding *convex-lincomb-def* **by** *simp*

lemma *set-in-convex-hull*:
  assumes *A* ⊆ *carrier-vec n*
  shows *A* ⊆ *convex-hull A*
proof
  fix *a*
  assume *a* ∈ *A*
  hence *acarr*: *a* ∈ *carrier-vec n* **using** *assms* **by** *auto*
  hence *convex-lincomb* (λ *x*. *1*) {*a*} *a* **unfolding** *convex-lincomb-def*
    by (*auto simp*: *nonneg-lincomb-def lincomb-def*)
  then show *a* ∈ *convex-hull A* **using** ‹*a* ∈ *A*› **unfolding** *convex-hull-def* **by** *auto*
qed

lemma *convex-hull-empty*[*simp*]:
  convex-hull {} = {}
  A ⊆ *carrier-vec n* ⟹ *convex-hull A* = {} ⟷ *A* = {}
proof −
  show *convex-hull* {} = {} **unfolding** *convex-hull-def* **by** *auto*
  then show *A* ⊆ *carrier-vec n* ⟹ *convex-hull A* = {} ⟷ *A* = {}
    using *set-in-convex-hull*[*of A*] **by** *auto*
qed

lemma *convex-hull-bound*: **assumes** *XBnd*: *X* ⊆ *Bounded-vec Bnd*
  and *X*: *X* ⊆ *carrier-vec n*
shows *convex-hull X* ⊆ *Bounded-vec Bnd*
proof
  fix *x*
  assume *x* ∈ *convex-hull X*
  from *this*[*unfolded convex-hull-def*]
  obtain *Y c* **where** *fin*: *finite Y* **and** *YX*: *Y* ⊆ *X* **and** *cx*: *convex-lincomb c Y x* **by** *auto*
  from *cx*[*unfolded convex-lincomb-def nonneg-lincomb-def*]
  have *x*: *x* = *lincomb c Y* **and** *sum*: *sum c Y* = *1* **and** *c0*: ⋀ *y*. *y* ∈ *Y* ⟹ *c y*

$\geq$ *0* **by** *auto*
  from *YX X XBnd* **have** *Y*: $Y \subseteq$ *carrier-vec n* **and** *YBnd*: $Y \subseteq$ *Bounded-vec Bnd* **by** *auto*
  from *x Y* **have** *dim*: *dim-vec x* = *n* **by** *auto*
  show $x \in$ *Bounded-vec Bnd* **unfolding** *Bounded-vec-def mem-Collect-eq dim*
  proof (*intro allI impI*)
    fix *i*
    assume *i*: $i < n$
    have *abs* (*x* \$ *i*) = *abs* ($\sum x \in Y$. *c x* ∗ *x* \$ *i*) **unfolding** *x*
      by (*subst lincomb-index*[*OF i Y*], *auto*)
    also have ... $\leq$ ($\sum x \in Y$. *abs* (*c x* ∗ *x* \$ *i*)) **by** *auto*
    also have ... = ($\sum x \in Y$. *abs* (*c x*) ∗ *abs* (*x* \$ *i*)) **by** (*simp add*: *abs-mult*)
    also have ... $\leq$ ($\sum x \in Y$. *abs* (*c x*) ∗ *Bnd*)
      by (*intro sum-mono mult-left-mono*, *insert YBnd*[*unfolded Bounded-vec-def*] *i Y*, *force+*)
    also have ... = ($\sum x \in Y$. *abs* (*c x*)) ∗ *Bnd*
      by (*simp add*: *sum-distrib-right*)
    also have ($\sum x \in Y$. *abs* (*c x*)) = ($\sum x \in Y$. *c x*)
      by (*rule sum.cong*, *insert c0*, *auto*)
    also have ... = *1* **by** *fact*
    finally show |*x* \$ *i*| $\leq$ *Bnd* **by** *auto*
  qed
qed

definition *convex-hull-list Vs* = {*x*. ∃ *c*. *convex-lincomb-list c Vs x*}

lemma *lincomb-list-elem*:
  set Vs $\subseteq$ *carrier-vec n* $\Longrightarrow$
    lincomb-list (λ *j*. *if i=j then 1 else 0*) *Vs* = (*if i* < *length Vs then Vs* ! *i else* 0_v *n*)
proof (*induction Vs rule*: *rev-induct*)
  case (*snoc x Vs*)
  have *x*: $x \in$ *carrier-vec n* **and** *Vs*: *set Vs* $\subseteq$ *carrier-vec n* **using** *snoc.prems* **by** *auto*
  let *?f* = λ *j*. *if i* = *j then 1 else 0*
  have *lincomb-list ?f* (*Vs* @ [*x*]) = *lincomb-list ?f Vs* + *?f* (*length Vs*) $\cdot_v$ *x*
    using *x Vs* **by** *simp*
  also have ... = (*if i* < *length* (*Vs* @ [*x*]) *then* (*Vs* @ [*x*]) ! *i else* 0_v *n*) (**is** *?goal*)
    using *less-linear*[*of i length Vs*]
  proof (*elim disjE*)
    assume *i*: *i* < *length Vs*
    have *lincomb-list* (λ*j*. *if i* = *j then 1 else 0*) *Vs* = *Vs* ! *i*
      using *snoc.IH*[*OF Vs*] *i* **by** *auto*
    moreover have (*if i* = *length Vs then 1 else 0*) $\cdot_v$ *x* = 0_v *n* **using** *i x* **by** *auto*
    moreover have (*if i* < *length* (*Vs* @ [*x*]) *then* (*Vs* @ [*x*]) ! *i else* 0_v *n*) = *Vs* ! *i*
      using *i append-Cons-nth-left* **by** *fastforce*
    ultimately show *?goal* **using** *Vs i lincomb-list-carrier M.r-zero* **by** *metis*
  next
    assume *i*: *i* = *length Vs*

```
    have lincomb-list (λj. if i = j then 1 else 0) Vs = 0_v n
      using snoc.IH[OF Vs] i by auto
    moreover have (if i = length Vs then 1 else 0) ·_v x = x using i x by auto
    moreover have (if i < length (Vs @ [x]) then (Vs @ [x]) ! i else 0_v n) = x
      using i append-Cons-nth-left by simp
    ultimately show ?goal using x by simp
  next
    assume i: i > length Vs
    have lincomb-list (λj. if i = j then 1 else 0) Vs = 0_v n
      using snoc.IH[OF Vs] i by auto
    moreover have (if i = length Vs then 1 else 0) ·_v x = 0_v n using i x by auto
    moreover have (if i < length (Vs @ [x]) then (Vs @ [x]) ! i else 0_v n) = 0_v n
      using i by simp
    ultimately show ?goal by simp
  qed
  finally show ?case by auto
qed simp

lemma set-in-convex-hull-list: fixes Vs :: 'a vec list
  assumes set Vs ⊆ carrier-vec n
  shows set Vs ⊆ convex-hull-list Vs
proof
  fix x assume x ∈ set Vs
  then obtain i where i: i < length Vs
    and x: x = Vs ! i using set-conv-nth[of Vs] by auto
  let ?f = λ j. if i = j then 1 else 0 :: 'a
  have lincomb-list ?f Vs = x using i x lincomb-list-elem[OF assms] by auto
  moreover have ∀ j < length Vs. ?f j ≥ 0 by auto
  moreover have sum ?f {0..<length Vs} = 1 using i by simp
  ultimately show x ∈ convex-hull-list Vs
    unfolding convex-hull-list-def convex-lincomb-list-def nonneg-lincomb-list-def
    by auto
qed

lemma convex-hull-list-combination:
  assumes Vs: set Vs ⊆ carrier-vec n
    and x: x ∈ convex-hull-list Vs
    and y: y ∈ convex-hull-list Vs
    and l0: 0 ≤ l and l1: l ≤ 1
  shows l ·_v x + (1 − l) ·_v y ∈ convex-hull-list Vs
proof −
  from x obtain cx where x: lincomb-list cx Vs = x and cx0: ∀ i < length Vs.
cx i ≥ 0
    and cx1: sum cx {0..<length Vs} = 1
    unfolding convex-hull-list-def convex-lincomb-list-def nonneg-lincomb-list-def
    by auto
  from y obtain cy where y: lincomb-list cy Vs = y and cy0: ∀ i < length Vs.
cy i ≥ 0
    and cy1: sum cy {0..<length Vs} = 1
```

unfolding *convex-hull-list-def convex-lincomb-list-def nonneg-lincomb-list-def*
by *auto*
let *?c* = λ *i*. *l* ∗ *cx i* + (*1* − *l*) ∗ *cy i*
have *set Vs* ⊆ *carrier-vec n* ⟹
lincomb-list ?c Vs = *l* $\cdot_v$ *lincomb-list cx Vs* + (*1* − *l*) $\cdot_v$ *lincomb-list cy Vs*
proof (*induction Vs rule*: *rev-induct*)
case (*snoc v Vs*)
have *v*: *v* ∈ *carrier-vec n* **and** *Vs*: *set Vs* ⊆ *carrier-vec n*
using *snoc.prems* **by** *auto*
have *lincomb-list ?c* (*Vs* @ [*v*]) = *lincomb-list ?c Vs* + *?c* (*length Vs*) $\cdot_v$ *v*
using *snoc.prems* **by** *auto*
also have *lincomb-list ?c Vs* =
l $\cdot_v$ *lincomb-list cx Vs* + (*1* − *l*) $\cdot_v$ *lincomb-list cy Vs*
by (*rule snoc.IH*[*OF Vs*])
also have *?c* (*length Vs*) $\cdot_v$ *v* =
l $\cdot_v$ (*cx* (*length Vs*) $\cdot_v$ *v*) + (*1* − *l*) $\cdot_v$ (*cy* (*length Vs*) $\cdot_v$ *v*)
using *add-smult-distrib-vec smult-smult-assoc* **by** *metis*
also have *l* $\cdot_v$ *lincomb-list cx Vs* + (*1* − *l*) $\cdot_v$ *lincomb-list cy Vs* + . . . =
l $\cdot_v$ (*lincomb-list cx Vs* + *cx* (*length Vs*) $\cdot_v$ *v*) +
(*1* − *l*) $\cdot_v$ (*lincomb-list cy Vs* + *cy* (*length Vs*) $\cdot_v$ *v*)
using *lincomb-list-carrier*[*OF Vs*] *v*
by (*simp add*: *M.add.m-assoc M.add.m-lcomm smult-r-distr*)
finally show *?case* **using** *Vs v* **by** *simp*
qed *simp*
hence *lincomb-list ?c Vs* = *l* $\cdot_v$ *x* + (*1* − *l*) $\cdot_v$ *y* **using** *Vs x y* **by** *simp*
moreover have ∀ *i* < *length Vs*. *?c i* ≥ *0* **using** *cx0 cy0 l0 l1* **by** *simp*
moreover have *sum ?c* {*0*..<*length Vs*} = *1*
proof(*simp add*: *sum.distrib*)
have (∑ *i* = *0*..<*length Vs*. (*1* − *l*) ∗ *cy i*) = (*1* − *l*) ∗ *sum cy* {*0*..<*length Vs*}
using *sum-distrib-left* **by** *metis*
moreover have (∑ *i* = *0*..<*length Vs*. *l* ∗ *cx i*) = *l* ∗ *sum cx* {*0*..<*length Vs*}
using *sum-distrib-left* **by** *metis*
ultimately show (∑ *i* = *0*..<*length Vs*. *l* ∗ *cx i*) + (∑ *i* = *0*..<*length Vs*. (*1* − *l*) ∗ *cy i*) = *1*
using *cx1 cy1* **by** *simp*
qed
ultimately show *?thesis*
unfolding *convex-hull-list-def convex-lincomb-list-def nonneg-lincomb-list-def*
by *auto*
qed

lemma *convex-hull-list-mono*:
assumes *set Ws* ⊆ *carrier-vec n*
shows *set Vs* ⊆ *set Ws* ⟹ *convex-hull-list Vs* ⊆ *convex-hull-list Ws*
proof (*standard*, *induction Vs*)
case *Nil*
from *Nil*(*2*) **show** *?case* **unfolding** *convex-hull-list-def convex-lincomb-list-def* **by** *auto*

```
next
  case (Cons v Vs)
  have v: v ∈ set Ws and Vs: set Vs ⊆ set Ws using Cons.prems(1) by auto
   hence v1: v ∈ convex-hull-list Ws using set-in-convex-hull-list[OF assms] by
auto
  from Cons.prems(2) obtain c
    where x: lincomb-list c (v # Vs) = x and c0: ∀ i < length Vs + 1. c i ≥ 0
      and c1: sum c {0..<length Vs + 1} = 1
    unfolding convex-hull-list-def convex-lincomb-list-def nonneg-lincomb-list-def
    by auto
  have x: x = c 0 ·v v + lincomb-list (c ∘ Suc) Vs using Vs v assms x by auto

  show ?case proof (cases)
    assume P: c 0 = 1
    hence sum (c ∘ Suc) {0..<length Vs} = 0
      using sum.atLeast0-lessThan-Suc-shift c1
      by (metis One-nat-def R.show-r-zero add.right-neutral add-Suc-right)
    moreover have ⋀ i. i ∈ {0..<length Vs} ⟹ (c ∘ Suc) i ≥ 0
      using c0 by simp
    ultimately have ∀ i ∈ {0..<length Vs}. (c ∘ Suc) i = 0
      using sum-nonneg-eq-0-iff by blast
    hence ⋀ i. i < length Vs ⟹ (c ∘ Suc) i ·v Vs ! i = 0v n
      using Vs assms by (simp add: subset-code(1))
    hence lincomb-list (c ∘ Suc) Vs = 0v n
      using lincomb-list-eq-0 by simp
    hence x = v using P x v assms by auto
    thus ?case using v1 by auto

  next

    assume P: c 0 ≠ 1
    have c1: c 0 + sum (c ∘ Suc) {0..<length Vs} = 1
      using sum.atLeast0-lessThan-Suc-shift[of c] c1 by simp
   have sum (c ∘ Suc) {0..<length Vs} ≥ 0 by (rule sum-nonneg, insert c0, simp)
    hence c 0 < 1 using P c1 by auto
    let ?c' = λ i. 1 / (1 − c 0) ∗ (c ∘ Suc) i
     have sum ?c' {0..<length Vs} = 1 / (1 − c 0) ∗ sum (c ∘ Suc) {0..<length
Vs}
      using c1 P sum-distrib-left by metis
    hence sum ?c' {0..<length Vs} = 1 using P c1 by simp
    moreover have ∀ i < length Vs. ?c' i ≥ 0 using c0 ‹c 0 < 1› by simp
    ultimately have c': lincomb-list ?c' Vs ∈ convex-hull-list Ws
      using Cons.IH[OF Vs]
        convex-hull-list-def convex-lincomb-list-def nonneg-lincomb-list-def
      by blast
    have lincomb-list ?c' Vs = 1 / (1 − c 0) ·v lincomb-list (c ∘ Suc) Vs
      by(rule lincomb-list-smult, insert Vs assms, auto)
    hence (1 − c 0) ·v lincomb-list ?c' Vs = lincomb-list (c ∘ Suc) Vs
      using P by auto
```

hence *x* = *c 0* $\cdot_v$ *v* + (*1* − *c 0*) $\cdot_v$ *lincomb-list ?c′ Vs* **using** *x* **by** *auto*
thus *x* ∈ *convex-hull-list Ws*
using *convex-hull-list-combination*[*OF assms v1 c′*] *c0* ‹*c 0* < *1*›
by *simp*
qed
qed

lemma *convex-hull-list-eq-set*:
set Vs ⊆ *carrier-vec n* ⟹ *set Vs* = *set Ws* ⟹ *convex-hull-list Vs* = *convex-hull-list Ws*
using *convex-hull-list-mono* **by** *blast*

lemma *find-indices-empty*: (*find-indices x Vs* = []) = (*x* ∉ *set Vs*)
proof (*induction Vs rule*: *rev-induct*)
case (*snoc v Vs*)
show *?case*
proof
assume *find-indices x* (*Vs* @ [*v*]) = []
hence *x* ≠ *v* ∧ *find-indices x Vs* = [] **by** *auto*
thus *x* ∉ *set* (*Vs* @ [*v*]) **using** *snoc* **by** *simp*
next
assume *x* ∉ *set* (*Vs* @ [*v*])
hence *x* ≠ *v* ∧ *find-indices x Vs* = [] **using** *snoc* **by** *auto*
thus *find-indices x* (*Vs* @ [*v*]) = [] **by** *simp*
qed
qed *simp*

lemma *distinct-list-find-indices*:
shows ⟦ *i* < *length Vs*; *Vs* ! *i* = *x*; *distinct Vs* ⟧ ⟹ *find-indices x Vs* = [*i*]
proof (*induction Vs rule*: *rev-induct*)
case (*snoc v Vs*)
have *dist*: *distinct Vs* **and** *xVs*: *v* ∉ *set Vs* **using** *snoc.prems*(*3*) **by**(*simp-all*)
show *?case*
proof (*cases*)
assume *i*: *i* = *length Vs*
hence *x* = *v* **using** *snoc.prems*(*2*) **by** *auto*
thus *?case* **using** *xVs find-indices-empty i*
by *fastforce*
next
assume *i* ≠ *length Vs*
hence *i*: *i* < *length Vs* **using** *snoc.prems*(*1*) **by** *simp*
hence *Vsi*: *Vs* ! *i* = *x* **using** *snoc.prems*(*2*) *append-Cons-nth-left* **by** *fastforce*
hence *x* ≠ *v* **using** *snoc.prems*(*3*) *i* **by** *auto*
thus *?case* **using** *snoc.IH*[*OF i Vsi dist*] **by** *simp*
qed
qed *auto*

lemma *finite-convex-hull-iff-convex-hull-list*: **assumes** *Vs*: *Vs* ⊆ *carrier-vec n*
and *id′*: *Vs* = *set Vsl′*

```
shows convex-hull Vs = convex-hull-list Vsl′
proof −
  have fin: finite Vs unfolding id′ by auto
  from finite-distinct-list fin obtain Vsl
    where id: Vs = set Vsl and dist: distinct Vsl by auto
  from Vs id have Vsl: set Vsl ⊆ carrier-vec n by auto
  {
    fix c :: nat ⇒ ′a
    have distinct Vsl =⇒(∑ x∈set Vsl. sum-list (map c (find-indices x Vsl))) =
                         sum c {0..<length Vsl}
    proof (induction Vsl rule: rev-induct)
      case (snoc v Vsl)
      let ?coef = λ x. sum-list (map c (find-indices x (Vsl @ [v])))
      let ?coef′ = λ x. sum-list (map c (find-indices x Vsl))
      have dist: distinct Vsl using snoc.prems by simp
      have sum ?coef (set (Vsl @ [v])) = sum-list (map ?coef (Vsl @ [v]))
        by (rule sum.distinct-set-conv-list[OF snoc.prems, of ?coef])
      also have . . . = sum-list (map ?coef Vsl) + ?coef v by simp
      also have sum-list (map ?coef Vsl) = sum ?coef (set Vsl)
        using sum.distinct-set-conv-list[OF dist, of ?coef] by auto
      also have . . . = sum ?coef′ (set Vsl)
      proof (intro R.finsum-restrict[of ?coef] restrict-ext, standard)
        fix x
        assume x ∈ set Vsl
        then obtain i where i: i < length Vsl and x: x = Vsl ! i
          using in-set-conv-nth[of x Vsl] by blast
        hence (Vsl @ [v]) ! i = x by (simp add: append-Cons-nth-left)
        hence ?coef x = c i
          using distinct-list-find-indices[OF - - snoc.prems] i by fastforce
        also have c i = ?coef′ x
          using distinct-list-find-indices[OF i - dist] x by simp
        finally show ?coef x = ?coef′ x by auto
      qed
      also have . . . = sum c {0..<length Vsl} by (rule snoc.IH[OF dist])
      also have ?coef v = c (length Vsl)
        using distinct-list-find-indices[OF - - snoc.prems, of length Vsl v]
          nth-append-length by simp
      finally show ?case using sum.atLeast0-lessThan-Suc by simp
    qed simp
  } note sum-sumlist = this
  {
    fix b
    assume b ∈ convex-hull-list Vsl
    then obtain c where b: lincomb-list c Vsl = b and c: (∀ i < length Vsl. c i
≥ 0)
      and c1: sum c {0..<length Vsl} = 1
      unfolding convex-hull-list-def convex-lincomb-list-def nonneg-lincomb-list-def
      by auto
    have convex-lincomb (mk-coeff Vsl c) Vs b
```

```
      unfolding b[symmetric] convex-lincomb-def nonneg-lincomb-def
      apply (subst lincomb-list-as-lincomb[OF Vsl])
    by (insert c c1, auto simp: id mk-coeff-def dist sum-sumlist intro!: sum-list-nonneg)
    hence b ∈ convex-hull Vs
      unfolding convex-hull-def convex-lincomb-def using fin by blast
  }
  moreover
  {
    fix b
    assume b ∈ convex-hull Vs
    then obtain c Ws where Ws: Ws ⊆ Vs and b: lincomb c Ws = b
      and c: c ` Ws ⊆ {x. x ≥ 0} and c1: sum c Ws = 1
      unfolding convex-hull-def convex-lincomb-def nonneg-lincomb-def by auto
    let ?d = λ x. if x ∈ Ws then c x else 0
    have lincomb ?d Vs = lincomb c Ws + lincomb (λ x. 0) (Vs − Ws)
      using lincomb-union2[OF - - Diff-disjoint[of Ws Vs], of c λ x. 0]
        fin Vs Diff-partition[OF Ws] by metis
    also have lincomb (λ x. 0) (Vs − Ws) = 0_v n
      using lincomb-zero[of Vs − Ws λ x. 0] Vs by auto
    finally have lincomb ?d Vs = b using b lincomb-closed Vs Ws by auto
    moreover have ?d ` Vs ⊆ {t. t ≥ 0} using c by auto
    moreover have sum ?d Vs = 1 using  c1 R.extend-sum[OF fin Ws] by auto
    ultimately have ∃ c. convex-lincomb c Vs b
      unfolding convex-lincomb-def nonneg-lincomb-def by blast
  }
  moreover
  {
    fix b
    assume ∃ c. convex-lincomb c Vs b
    then obtain c where b: lincomb c Vs = b and c: c ` Vs ⊆ {x. x ≥ 0}
      and c1: sum c Vs = 1
      unfolding convex-lincomb-def nonneg-lincomb-def by auto
    from lincomb-as-lincomb-list-distinct[OF Vsl dist, of c]
    have b: lincomb-list (λi. c (Vsl ! i)) Vsl = b
      unfolding b[symmetric] id by simp
    have 1 = sum c (set Vsl) using c1 id by auto
     also have ... = sum-list (map c Vsl) by(rule sum.distinct-set-conv-list[OF
dist])
    also have ... = sum ((!) (map c Vsl)) {0..<length Vsl}
      using sum-list-sum-nth length-map by metis
    also have ... = sum (λ i. c (Vsl ! i)) {0..<length Vsl} by simp
    finally have sum-1: (∑ i = 0..<length Vsl. c (Vsl ! i)) = 1 by simp

    have ∃ c. convex-lincomb-list c Vsl b
      unfolding convex-lincomb-list-def nonneg-lincomb-list-def
      by (intro exI[of - λi. c (Vsl ! i)] conjI b sum-1)
        (insert c, force simp: set-conv-nth id)
    hence b ∈ convex-hull-list Vsl unfolding convex-hull-list-def by auto
  }
```

ultimately have *convex-hull Vs* = *convex-hull-list Vsl* **by** *auto*
also have *convex-hull-list Vsl* = *convex-hull-list Vsl′*
using *convex-hull-list-eq-set*[*OF Vsl*, *of Vsl′*] *id id′* **by** *simp*
finally show *?thesis* **by** *simp*
qed

definition *convex S* = (*convex-hull S* = *S*)

lemma *convex-convex-hull*: *convex S* ⟹ *convex-hull S* = *S*
unfolding *convex-def* **by** *auto*

lemma *convex-hull-convex-hull-listD*: **assumes** *A*: *A* ⊆ *carrier-vec n*
and *x*: *x* ∈ *convex-hull A*
shows ∃ *as*. *set as* ⊆ *A* ∧ *x* ∈ *convex-hull-list as*
proof −
from *x*[*unfolded convex-hull-def*]
obtain *X c* **where** *finX*: *finite X* **and** *XA*: *X* ⊆ *A* **and** *convex-lincomb c X x* **by** *auto*
hence *x*: *x* ∈ *convex-hull X* **unfolding** *convex-hull-def* **by** *auto*
from *finite-list*[*OF finX*] **obtain** *xs* **where** *X*: *X* = *set xs* **by** *auto*
from *finite-convex-hull-iff-convex-hull-list*[*OF - this*] *x XA A* **have** *x*: *x* ∈ *convex-hull-list xs* **by** *auto*
thus *?thesis* **using** *XA* **unfolding** *X* **by** *auto*
qed

lemma *convex-hull-convex-sum*: **assumes** *A*: *A* ⊆ *carrier-vec n*
and *x*: *x* ∈ *convex-hull A*
and *y*: *y* ∈ *convex-hull A*
and *a*: *0* ≤ *a a* ≤ *1*
shows $a \cdot_v x + (1 - a) \cdot_v y$ ∈ *convex-hull A*
proof −
from *convex-hull-convex-hull-listD*[*OF A x*] **obtain** *xs* **where** *xs*: *set xs* ⊆ *A*
and *x*: *x* ∈ *convex-hull-list xs* **by** *auto*
from *convex-hull-convex-hull-listD*[*OF A y*] **obtain** *ys* **where** *ys*: *set ys* ⊆ *A*
and *y*: *y* ∈ *convex-hull-list ys* **by** *auto*
have *fin*: *finite* (*set* (*xs* @ *ys*)) **by** *auto*
have *sub*: *set* (*xs* @ *ys*) ⊆ *A* **using** *xs ys* **by** *auto*
from *convex-hull-list-mono*[*of xs* @ *ys xs*] *x sub A* **have** *x*: *x* ∈ *convex-hull-list* (*xs* @ *ys*) **by** *auto*
from *convex-hull-list-mono*[*of xs* @ *ys ys*] *y sub A* **have** *y*: *y* ∈ *convex-hull-list* (*xs* @ *ys*) **by** *auto*
from *convex-hull-list-combination*[*OF - x y a*]
have $a \cdot_v x + (1 - a) \cdot_v y$ ∈ *convex-hull-list* (*xs* @ *ys*) **using** *sub A* **by** *auto*
from *finite-convex-hull-iff-convex-hull-list*[*of - xs* @ *ys*] *this sub A*
have $a \cdot_v x + (1 - a) \cdot_v y$ ∈ *convex-hull* (*set* (*xs* @ *ys*)) **by** *auto*
with *convex-hull-mono*[*OF sub*]
show $a \cdot_v x + (1 - a) \cdot_v y$ ∈ *convex-hull A* **by** *auto*
qed

lemma *convexI*: **assumes** *S*: *S* ⊆ *carrier-vec n*
  and *step*: ⋀ *a x y*. *x* ∈ *S* ⟹ *y* ∈ *S* ⟹ *0* ≤ *a* ⟹ *a* ≤ *1* ⟹ *a* $\cdot_v$ *x* + (*1* − *a*) $\cdot_v$ *y* ∈ *S*
shows *convex S*
  unfolding *convex-def*
proof (*standard*, *standard*)
  fix *z*
  assume *z* ∈ *convex-hull S*
  from *this*[*unfolded convex-hull-def*] **obtain** *W c* **where** *finite W* **and** *WS*: *W* ⊆ *S*
    and *convex-lincomb c W z* **by** *auto*
  then show *z* ∈ *S*
  proof (*induct W arbitrary*: *c z*)
    case *empty*
    thus *?case* **unfolding** *convex-lincomb-def* **by** *auto*
  next
    case (*insert w W c z*)
    have *convex-lincomb c* (*insert w W*) *z* **by** *fact*
    hence *zl*: *z* = *lincomb c* (*insert w W*) **and** *nonneg*: ⋀ *w*. *w* ∈ *W* ⟹ *0* ≤ *c w*
      and *cw*: *c w* ≥ *0*
      and *sum*: *sum c* (*insert w W*) = *1*
      unfolding *convex-lincomb-def nonneg-lincomb-def* **by** *auto*
    have *zl*: *z* = *c w* $\cdot_v$ *w* + *lincomb c W* **unfolding** *zl*
      by (*rule lincomb-insert2*, *insert insert S*, *auto*)
    have *sum*: *c w* + *sum c W* = *1* **unfolding** *sum*[*symmetric*]
      by (*subst sum.insert*, *insert insert*, *auto*)
    have *W*: *W* ⊆ *carrier-vec n* **and** *w*: *w* ∈ *carrier-vec n* **using** *S insert* **by** *auto*
    show *?case*
    proof (*cases sum c W* = *0*)
      case *True*
      with *nonneg* **have** *c0*: ⋀ *w*. *w* ∈ *W* ⟹ *c w* = *0*
        using *insert*(*1*) *sum-nonneg-eq-0-iff* **by** *auto*
      with *sum* **have** *cw*: *c w* = *1* **by** *auto*
      have *lin0*: *lincomb c W* = 0_v *n*
        by (*intro lincomb-zero W*, *insert c0*, *auto*)
      have *z* = *w* **unfolding** *zl cw lin0* **using** *w* **by** *simp*
      with *insert*(*4*) **show** *?thesis* **by** *simp*
    next
      case *False*
      have *sum c W* ≥ *0* **using** *nonneg* **by** (*metis sum-nonneg*)
      with *False* **have** *pos*: *sum c W* > *0* **by** *auto*
      define *b* **where** *b* = (λ *w*. *inverse* (*sum c W*) ∗ *c w*)
      have *convex-lincomb b W* (*lincomb b W*)
        unfolding *convex-lincomb-def nonneg-lincomb-def b-def*
      proof (*intro conjI refl*)
        show (λ*w*. *inverse* (*sum c W*) ∗ *c w*) ‘ *W* ⊆ *Collect* ((≤) *0*) **using** *nonneg pos* **by** *auto*
        show (∑ *w*∈*W*. *inverse* (*sum c W*) ∗ *c w*) = *1* **unfolding** *sum-distrib-left*[*symmetric*] **using** *False* **by** *auto*

```
      qed
      from insert(3)[OF - this] insert
      have IH: lincomb b W ∈ S by auto
      have lin: lincomb c W = sum c W ·v lincomb b W
        unfolding b-def
          by (subst lincomb-smult[symmetric, OF W], rule lincomb-cong[OF - W],
insert False, auto)
      from sum cw pos have sum: sum c W = 1 − c w and cw1: c w ≤ 1 by auto
      show ?thesis unfolding zl lin unfolding sum
        by (rule step[OF - IH cw cw1], insert insert, auto)
    qed
  qed
next
  show S ⊆ convex-hull S using S by (rule set-in-convex-hull)
qed

lemma convex-hulls-are-convex: assumes A ⊆ carrier-vec n
  shows convex (convex-hull A)
  by (intro convexI convex-hull-convex-sum convex-hull-carrier assms)

lemma convex-hull-sum: assumes A: A ⊆ carrier-vec n and B: B ⊆ carrier-vec
n
  shows convex-hull (A + B) = convex-hull A + convex-hull B
proof
  note cA = convex-hull-carrier[OF A]
  note cB = convex-hull-carrier[OF B]
  have convex (convex-hull A + convex-hull B)
  proof (intro convexI sum-carrier-vec convex-hull-carrier A B)
    fix a :: 'a and x1 x2
    assume x1 ∈ convex-hull A + convex-hull B x2 ∈ convex-hull A + convex-hull
B
    then obtain y1 y2 z1 z2 where
      x12: x1 = y1 + z1 x2 = y2 + z2 and
      y12: y1 ∈ convex-hull A y2 ∈ convex-hull A and
      z12: z1 ∈ convex-hull B z2 ∈ convex-hull B
      unfolding set-plus-def by auto
    from y12 z12 cA cB have carr:
      y1 ∈ carrier-vec n y2 ∈ carrier-vec n
      z1 ∈ carrier-vec n z2 ∈ carrier-vec n
      by auto
    assume a: 0 ≤ a a ≤ 1
    have A: a ·v y1 + (1 − a) ·v y2 ∈ convex-hull A using y12 a A by (metis
convex-hull-convex-sum)
    have B: a ·v z1 + (1 − a) ·v z2 ∈ convex-hull B using z12 a B by (metis
convex-hull-convex-sum)
    have a ·v x1 + (1 − a) ·v x2 = (a ·v y1 + a ·v z1) + ((1 − a) ·v y2 + (1 −
a) ·v z2) unfolding x12
      using carr by (auto simp: smult-add-distrib-vec)
    also have ... = (a ·v y1 + (1 − a) ·v y2) + (a ·v z1 + (1 − a) ·v z2) using
```

carr
by (*intro eq-vecI*, *auto*)
finally show *a* $\cdot_v$ *x1* + (*1* − *a*) $\cdot_v$ *x2* ∈ *convex-hull A* + *convex-hull B*
using *A B* **by** *auto*
qed
from *convex-convex-hull*[*OF this*]
have *id*: *convex-hull* (*convex-hull A* + *convex-hull B*) = *convex-hull A* + *convex-hull B* **.**
show *convex-hull* (*A* + *B*) ⊆ *convex-hull A* + *convex-hull B*
by (*subst id*[*symmetric*], *rule convex-hull-mono*[*OF set-plus-mono2*]; *intro set-in-convex-hull A B*)
show *convex-hull A* + *convex-hull B* ⊆ *convex-hull* (*A* + *B*)
proof
fix *x*
assume *x* ∈ *convex-hull A* + *convex-hull B*
then obtain *y z* **where** *x*: *x* = *y* + *z* **and** *y*: *y* ∈ *convex-hull A* **and** *z*: *z* ∈ *convex-hull B*
by (*auto simp*: *set-plus-def*)
from *convex-hull-convex-hull-listD*[*OF A y*] **obtain** *ys* **where** *ysA*: *set ys* ⊆ *A* **and**
y: *y* ∈ *convex-hull-list ys* **by** *auto*
from *convex-hull-convex-hull-listD*[*OF B z*] **obtain** *zs* **where** *zsB*: *set zs* ⊆ *B* **and**
z: *z* ∈ *convex-hull-list zs* **by** *auto*
from *y*[*unfolded convex-hull-list-def convex-lincomb-list-def nonneg-lincomb-list-def*]
obtain *c* **where** *yid*: *y* = *lincomb-list c ys*
and *conv-c*: (∀ *i*<*length ys*. *0* ≤ *c i*) ∧ *sum c* {*0*..<*length ys*} = *1*
by *auto*
from *z*[*unfolded convex-hull-list-def convex-lincomb-list-def nonneg-lincomb-list-def*]
obtain *d* **where** *zid*: *z* = *lincomb-list d zs*
and *conv-d*: (∀ *i*<*length zs*. *0* ≤ *d i*) ∧ *sum d* {*0*..<*length zs*} = *1*
by *auto*
from *ysA A* **have** *ys*: *set ys* ⊆ *carrier-vec n* **by** *auto*
from *zsB B* **have** *zs*: *set zs* ⊆ *carrier-vec n* **by** *auto*
have [*intro*, *simp*]: *lincomb-list x ys* ∈ *carrier-vec n* **for** *x* **using** *lincomb-list-carrier*[*OF ys*] **.**
have [*intro*, *simp*]: *lincomb-list x zs* ∈ *carrier-vec n* **for** *x* **using** *lincomb-list-carrier*[*OF zs*] **.**
have *dim*[*simp*]: *dim-vec* (*lincomb-list d zs*) = *n* **by** *auto*
from *yid* **have** *y*: *y* ∈ *carrier-vec n* **by** *auto*
from *zid* **have** *z*: *z* ∈ *carrier-vec n* **by** *auto*
{
fix *x*
assume *x* ∈ *set* (*map* ((+) *y*) *zs*)
then obtain *z* **where** *x* = *y* + *z* **and** *z* ∈ *set zs* **by** *auto*
then obtain *j* **where** *j*: *j* < *length zs* **and** *x*: *x* = *y* + *zs* ! *j* **unfolding** *set-conv-nth* **by** *auto*
hence *mem*: *zs* ! *j* ∈ *set zs* **by** *auto*
hence *zsj*: *zs* ! *j* ∈ *carrier-vec n* **using** *zs* **by** *auto*

```
      let ?list = (map (λ y. y + zs ! j) ys)
      let ?set = set ?list
      have set: ?set ⊆ carrier-vec n using ys A zsj by auto
      have lin-map: lincomb-list c ?list ∈ carrier-vec n
        by (intro lincomb-list-carrier[OF set])
      have y + (zs ! j) = lincomb-list c ?list
      unfolding yid using zsj lin-map lincomb-list-index[OF - set] lincomb-list-index[OF
- ys]
       by (intro eq-vecI, auto simp: field-simps sum-distrib-right[symmetric] conv-c)
      hence convex-lincomb-list c ?list (y + (zs ! j))
        unfolding convex-lincomb-list-def nonneg-lincomb-list-def using conv-c by
auto
      hence y + (zs ! j) ∈ convex-hull-list ?list unfolding convex-hull-list-def by
auto
      with finite-convex-hull-iff-convex-hull-list[OF set refl]
      have (y + zs ! j) ∈ convex-hull ?set by auto
      also have ... ⊆ convex-hull (A + B)
        by (rule convex-hull-mono, insert mem ys ysA zsB, force simp: set-plus-def)
      finally have x ∈ convex-hull (A + B) unfolding x .
    } note step1 = this
    {
      let ?list = map ((+) y) zs
      let ?set = set ?list
      have set: ?set ⊆ carrier-vec n using zs B y by auto
      have lin-map: lincomb-list d ?list ∈ carrier-vec n
        by (intro lincomb-list-carrier[OF set])
      have [simp]: i < n ⟹ (∑ j = 0..<length zs. d j * (y + zs ! j) $ i) =
        (∑ j = 0..<length zs. d j * (y $ i + zs ! j $ i)) for i
        by (rule sum.cong, insert zs[unfolded set-conv-nth] y, auto)
      have y + z = lincomb-list d ?list
      unfolding zid using y zs lin-map lincomb-list-index[OF - set] lincomb-list-index[OF
- zs]
          set lincomb-list-carrier[OF zs, of d] zs[unfolded set-conv-nth]
       by (intro eq-vecI, auto simp: field-simps sum-distrib-right[symmetric] conv-d)
      hence convex-lincomb-list d ?list x unfolding x
        unfolding convex-lincomb-list-def nonneg-lincomb-list-def using conv-d by
auto
      hence x ∈ convex-hull-list ?list unfolding convex-hull-list-def by auto
      with finite-convex-hull-iff-convex-hull-list[OF set refl]
      have x ∈ convex-hull ?set by auto
      also have ... ⊆ convex-hull (convex-hull (A + B))
        by (rule convex-hull-mono, insert step1, auto)
      also have ... = convex-hull (A + B)
      by (rule convex-convex-hull[OF convex-hulls-are-convex], intro sum-carrier-vec
A B)
      finally show x ∈ convex-hull (A + B) .
    }
  qed
qed
```

lemma *convex-hull-in-cone*:
  convex-hull $C \subseteq$ *cone* C
  unfolding *convex-hull-def cone-def convex-lincomb-def finite-cone-def* **by** *auto*

lemma *convex-cone*:
  assumes C: $C \subseteq$ *carrier-vec n*
  shows *convex* (*cone* C)
  unfolding *convex-def*
  using *convex-hull-in-cone set-in-convex-hull*[*OF cone-carrier*[*OF C*]] *cone-cone*[*OF C*]
  by *blast*

end
end

9 Normal Vectors

We provide a function for the normal vector of a half-space (given as n-1 linearly independent vectors). We further provide a function that returns a list of normal vectors that span the orthogonal complement of some subspace of R^n. Bounds for all normal vectors are provided.

theory *Normal-Vector*
  imports
    Integral-Bounded-Vectors
    Basis-Extension
begin

context *gram-schmidt*
begin

lemma *ortho-sum-in-span*:
  assumes W: $W \subseteq$ *carrier-vec n*
    and X: $X \subseteq$ *carrier-vec n*
    and *ortho*: $\bigwedge w\ x.\ w \in W \Longrightarrow x \in X \Longrightarrow x \cdot w = 0$
    and *inspan*: *lincomb l1 X* + *lincomb l2 W* $\in$ *span X*
  shows *lincomb l2 W* = 0_v *n*
proof (*rule ccontr*)
  let *?v* = *lincomb l2 W*
  have *vcarr*: *?v* $\in$ *carrier-vec n* **using** *W* **by** *auto*
  have *vspan*: *?v* $\in$ *span W* **using** *W* **by** *auto*
  assume ¬*?thesis*
  from *this* **have** *vnz*: *?v* $\neq$ 0_v *n* **by** *auto*
  let *?x* = *lincomb l1 X*
  have *xcarr*: *?x* $\in$ *carrier-vec n* **using** *X* **by** *auto*
  have *xspan*: *?x* $\in$ *span X* **using** *X xcarr* **by** *auto*
  have *0* $\neq$ *sq-norm ?v* **using** *vnz vcarr* **by** *simp*
  also have *sq-norm ?v* = *0* + *?v* · *?v* **by** (*simp add*: *sq-norm-vec-as-cscalar-prod*)

```
  also have ... = ?x · ?v + ?v · ?v
    by (subst (2) ortho-span-span[OF X W ortho], insert X W, auto)
  also have ... = (?x + ?v ) · ?v using xcarr vcarr
    using add-scalar-prod-distrib by force
  also have ... = 0
    by (rule ortho-span-span[OF X W ortho inspan vspan])
  finally show False by simp
qed


lemma ortho-lin-indpt: assumes W: W ⊆ carrier-vec n
  and X: X ⊆ carrier-vec n
  and ortho: ⋀ w x. w ∈ W ⟹ x ∈ X ⟹ x · w = 0
  and linW: lin-indpt W
  and linX: lin-indpt X
shows lin-indpt (W ∪ X)
proof (rule ccontr)
  assume ¬?thesis
  from this obtain c where zerocomb:lincomb c (W ∪ X) = 0_v n
    and notallz: ∃ v ∈ (W ∪ X). c v ≠ 0
    using assms fin-dim fin-dim-li-fin finite-lin-indpt2 infinite-Un le-sup-iff
    by metis
  have zero-nin-W: 0_v n ∉ W using assms by (metis vs-zero-lin-dep)
  have WXinters: W ∩ X = {}
  proof (rule ccontr)
    assume ¬?thesis
    from this obtain v where v: v∈ W ∩ X by auto
    hence v·v=0 using ortho by auto
    moreover have v ∈ carrier-vec n using assms v by auto
    ultimately have v=0_v n using sq-norm-vec-as-cscalar-prod[of v] by auto
    then show False using zero-nin-W v by auto
  qed
  have finX: finite X using X linX by (simp add: fin-dim-li-fin)
  have finW: finite W using W linW by (simp add: fin-dim-li-fin)
  have split: lincomb c (W ∪ X) = lincomb c X + lincomb c W
    using lincomb-union[OF W X WXinters finW finX]
    by (simp add: M.add.m-comm W X)
  hence lincomb c X + lincomb c W ∈ span X using zerocomb
    using local.span-zero by auto
  hence z1: lincomb c W = 0_v n
    using ortho-sum-in-span[OF W X ortho] by simp
  hence z2: lincomb c X = 0_v n using split zerocomb X by simp
  have or: (∃ v ∈ W. c v ≠ 0) ∨ (∃ v∈ X. c v ≠ 0) using notallz by auto
  have ex1: ∃ v ∈ W. c v ≠ 0 ⟹ False using linW
    using finW lin-dep-def z1 by blast
  have ex2: ∃ v∈ X. c v ≠ 0 ⟹ False using linX
    using finX lin-dep-def z2 by blast
  show False using ex1 ex2 or by auto
qed
```

definition *normal-vector* :: ′*a vec set* ⇒ ′*a vec* **where**
normal-vector W = (*let ws* = (*SOME ws. set ws* = *W* ∧ *distinct ws*);
m = *length ws*;
B = (λ *j*. *mat m m* (λ(*i*, *j*′). *ws* ! *i* $ (*if j*′ < *j then j*′ *else Suc j*′)))
in vec n (λ *j*. (−*1*) ^ (*m*+*j*) ∗ *det* (*B j*)))

lemma *normal-vector*: **assumes** *fin*: *finite W*
and *card*: *Suc* (*card W*) = *n*
and *lin*: *lin-indpt W*
and *W*: *W* ⊆ *carrier-vec n*
shows *normal-vector W* ∈ *carrier-vec n*
normal-vector W ≠ 0_v *n*
w ∈ *W* ⟹ *w* · *normal-vector W* = *0*
w ∈ *W* ⟹ *normal-vector W* · *w* = *0*
lin-indpt (*insert* (*normal-vector W*) *W*)
normal-vector W ∉ *W*
is-det-bound db ⟹ *W* ⊆ $\mathbb{Z}_v$ ∩ *Bounded-vec* (*of-int Bnd*) ⟹ *normal-vector W* ∈ $\mathbb{Z}_v$ ∩ *Bounded-vec* (*of-int* (*db* (*n*−*1*) *Bnd*))
proof −
define *ws* **where** *ws* = (*SOME ws. set ws* = *W* ∧ *distinct ws*)
from *finite-distinct-list*[*OF fin*]
have ∃ *ws. set ws* = *W* ∧ *distinct ws* **by** *auto*
from *someI-ex*[*OF this*, *folded ws-def*] **have** *id*: *set ws* = *W* **and** *dist*: *distinct ws* **by** *auto*
have *len*: *length ws* = *card W* **using** *distinct-card*[*OF dist*] *id* **by** *auto*
let *?n* = *length ws*
define *B* **where** *B* = (λ *j*. *mat ?n ?n* (λ(*i*, *j*′). *ws* ! *i* $ (*if j*′ < *j then j*′ *else Suc j*′)))
define *nv* **where** *nv* = *vec n* (λ *j*. (−*1*) ^ (*?n*+*j*) ∗ *det* (*B j*))
have *nv2*: *normal-vector W* = *nv* **unfolding** *normal-vector-def Let-def*
ws-def[*symmetric*] *B-def nv-def* **..**
define *A* **where** *A* = (λ *w*. *mat-of-rows n* (*ws* @ [*w*]))
from *len id card* **have** *len*: *Suc ?n* = *n* **by** *auto*
have *A*: *A w* ∈ *carrier-mat n n* **for** *w* **using** *id W len* **unfolding** *A-def* **by** *auto*
{
fix *w* :: ′*a vec*
assume *w*: *w* ∈ *carrier-vec n*
from *len* **have** *n1*[*simp*]: *n* − *Suc 0* = *?n* **by** *auto*
{
fix *j*
assume *j*: *j* < *n*
have *mat-delete* (*A w*) *?n j* = *B j*
unfolding *mat-delete-def A-def mat-of-rows-def B-def*
by (*rule eq-matI*, *insert j len*, *auto simp*: *nth-append*)
} **note** *B* = *this*
have *det* (*A w*) = (∑*j*<*n*. (*A w*) $$ (*length ws*, *j*) ∗ *cofactor* (*A w*) *?n j*)
by (*subst laplace-expansion-row*[*OF A*, *of ?n*], *insert len*, *auto*)
also have ... = (∑*j*<*n*. *w* $ *j* ∗ (−*1*) ^ (*?n*+*j*) ∗ *det* (*mat-delete* (*A w*) *?n j*))

by (*rule sum.cong*, *auto simp*: *A-def mat-of-rows-def cofactor-def*)
also have ... = ($\sum$*j*<*n*. *w* $ *j* ∗ (−*1*)^(*?n*+*j*) ∗ *det* (*B j*))
by (*rule sum.cong*[*OF refl*], *subst B*, *auto*)
also have ... = ($\sum$*j*<*n*. *w* $ *j* ∗ *nv* $ *j*)
by (*rule sum.cong*[*OF refl*], *auto simp*: *nv-def*)
also have ... = *w* · *nv* **unfolding** *scalar-prod-def* **unfolding** *nv-def*
by (*rule sum.cong*, *auto*)
finally have *det* (*A w*) = *w* · *nv* **.**
} **note** *det-scalar* = *this*
have *nv*: *nv* ∈ *carrier-vec n* **unfolding** *nv-def* **by** *auto*
{
fix *w*
assume *wW*: *w* ∈ *W*
with *W* **have** *w*: *w* ∈ *carrier-vec n* **by** *auto*
from *wW id* **obtain** *i* **where** *i*: *i* < *?n* **and** *ws*: *ws* ! *i* = *w* **unfolding** *set-conv-nth* **by** *auto*
from *det-scalar*[*OF w*] **have** *det* (*A w*) = *w* · *nv* **.**
also have *det* (*A w*) = *0*
by (*subst det-identical-rows*[*OF A*, *of i ?n*], *insert i ws len*, *auto simp*: *A-def mat-of-rows-def nth-append*)
finally have *w* · *nv* = *0* **..**
note *this this*[*unfolded comm-scalar-prod*[*OF w nv*]]
} **note** *ortho* = *this*
have *nv0*: *nv* ≠ 0_v *n*
proof
assume *nv*: *nv* = 0_v *n*
define *bs* **where** *bs* = *basis-extension ws*
define *w* **where** *w* = *hd bs*
have *lin-indpt-list ws* **using** *dist lin W* **unfolding** *lin-indpt-list-def id* **by** *auto*
from *basis-extension*[*OF this*, *folded bs-def*] *len*
have *lin*: *lin-indpt-list* (*ws* @ *bs*) **and** *length bs* = *1* **and** *bsc*: *set bs* ⊆ *carrier-vec n*
by (*auto simp*: *unit-vecs-def*)
hence *bs*: *bs* = [*w*] **unfolding** *w-def* **by** (*cases bs*, *auto*)
with *bsc* **have** *w*: *w* ∈ *carrier-vec n* **by** *auto*
note *lin* = *lin*[*unfolded bs*]
from *lin-indpt-list-length-eq-n*[*OF lin*] *len*
have *basis*: *basis* (*set* (*ws* @ [*w*])) **by** *auto*
from *w det-scalar nv* **have** *det0*: *det* (*A w*) = *0* **by** *auto*
with *basis-det-nonzero*[*OF basis*] *len* **show** *False*
unfolding *A-def* **by** *auto*
qed
let *?nv* = *normal-vector W*
from *ortho nv nv0*
show *nv*: *?nv* ∈ *carrier-vec n*
and *ortho*: ⋀ *w*. *w* ∈ *W* ⟹ *w* · *?nv* = *0*
⋀ *w*. *w* ∈ *W* ⟹ *?nv* · *w* = *0*
and *n0*: *?nv* ≠ 0_v *n* **unfolding** *nv2* **by** *auto*
from *n0 nv* **have** *sq-norm ?nv* ≠ *0* **by** *auto*

```
  hence nvnv: ?nv · ?nv ≠ 0 by (auto simp: sq-norm-vec-as-cscalar-prod)
  show nvW: ?nv ∉ W using nvnv ortho by blast
  have ?nv ∉ span W using W ortho nvnv nv
    using orthocompl-span by blast
  with lin-dep-iff-in-span[OF W lin nv nvW]
  show lin-indpt (insert ?nv W) by auto
  {
    assume db: is-det-bound db
    assume W ⊆ ℤv ∩ Bounded-vec (of-int Bnd)
    hence wsI: set ws ⊆ ℤv ∩ Bounded-vec (of-int Bnd) unfolding id by auto
    have ws: set ws ⊆ carrier-vec n using W unfolding id by auto
    from wsI ws have wsI: i < ?n ⟹ ws ! i ∈ ℤv ∩ Bounded-vec (of-int Bnd) ∩
carrier-vec n for i
      using len wsI unfolding set-conv-nth by auto
    have ints: i < ?n ⟹ j < n ⟹ ws ! i $ j ∈ ℤ for i j
      using wsI[of i, unfolded Ints-vec-def] by force
    have bnd: i < ?n ⟹ j < n ⟹ abs (ws ! i $ j) ≤ of-int Bnd for i j
      using wsI[unfolded Bounded-vec-def, of i] by auto
    {
      fix i
      assume i: i < n
      have ints-nv: nv $ i ∈ ℤ unfolding nv-def using wsI len ws
        by (auto simp: i B-def set-conv-nth intro!: Ints-mult Ints-det ints)
      have B i ∈ ℤm ∩ Bounded-mat (of-int Bnd)
        unfolding B-def using len ws i bnd ints-nv
       apply (simp add: Ints-mat-def Ints-vec-def Bounded-mat-def, intro allI impI)
        subgoal for ii j using ints[of ii j] ints[of ii Suc j]
          by auto
        done
      from is-det-bound-of-int[OF db - this, of ?n]
      have |nv $ i| ≤ of-int (db (n − 1) Bnd)
        unfolding nv-def using wsI len ws i
        by (auto simp: B-def abs-mult bnd)
      note ints-nv this
    }
    with nv nv2 show ?nv ∈ ℤv ∩ Bounded-vec (of-int (db (n − 1) Bnd))
      unfolding Ints-vec-def Bounded-vec-def by auto
  }
qed

lemma normal-vector-span:
  assumes card: Suc (card D) = n
    and D: D ⊆ carrier-vec n and fin: finite D and lin: lin-indpt D
  shows span D = { x. x ∈ carrier-vec n ∧ x · normal-vector D = 0}
proof −
  note nv = normal-vector[OF fin card lin D]
  {
    fix x
    assume xspan: x ∈ span D
```

```
    from finite-in-span[OF fin D xspan] obtain c where
      x · normal-vector D = lincomb c D · normal-vector D by auto
    also have ... = (∑ w∈D. c w * (w · normal-vector D))
      by (rule lincomb-scalar-prod-left, insert D nv, auto)
    also have ... = 0
     apply (rule sum.neutral) using nv(1,2,3) D comm-scalar-prod[of normal-vector
D] by fastforce
     finally have x ∈ carrier-vec n  x · normal-vector D = 0 using xspan D by
auto
  }
  moreover
  {
    let ?n = normal-vector D
    fix x
    assume x: x ∈ carrier-vec n and xscal: x · normal-vector D = 0
    let ?B = (insert (normal-vector D) D)
    have card ?B = n using card card-insert-disjoint[OF fin nv(6)] by auto
    moreover have B: ?B ⊆ carrier-vec n using D nv by auto
    ultimately have span ?B = carrier-vec n
      by (intro span-carrier-lin-indpt-card-n, insert nv(5), auto)
    hence xspan: x ∈ span ?B using x by auto
    obtain c where x = lincomb c ?B using finite-in-span[OF - B xspan] fin by
auto
    hence 0 = lincomb c ?B · normal-vector D using xscal by auto
    also have ... = (∑ w∈ ?B. c w * (w · normal-vector D))
      by (subst lincomb-scalar-prod-left, insert B, auto)
    also have ... = (∑ w∈ D. c w * (w · normal-vector D)) + c ?n * (?n · ?n)
      by (subst sum.insert[OF fin nv(6)], auto)
    also have (∑ w∈ D. c w * (w · normal-vector D)) = 0
       apply(rule sum.neutral) using nv(1,3) comm-scalar-prod[OF nv(1)] D by
fastforce
    also have ?n · ?n = sq-norm ?n using sq-norm-vec-as-cscalar-prod[of ?n] by
simp
    finally have c ?n * sq-norm ?n = 0 by simp
    hence ncoord: c ?n = 0 using nv(1−5) by auto
    have x = lincomb c ?B by fact
    also have ... = lincomb c D
      apply (subst lincomb-insert2[OF fin D - nv(6,1)]) using ncoord nv(1) D by
auto
    finally have x ∈ span D using fin by auto
  }
  ultimately show ?thesis by auto
qed

definition normal-vectors :: 'a vec list ⇒ 'a vec list where
  normal-vectors ws = (let us = basis-extension ws
    in map (λ i. normal-vector (set (ws @ us) − {us ! i})) [0..<length us])

lemma normal-vectors:
```

```
  assumes lin: lin-indpt-list ws
  shows set (normal-vectors ws) ⊆ carrier-vec n
    w ∈ set ws ⟹ nv ∈ set (normal-vectors ws) ⟹ nv · w = 0
    w ∈ set ws ⟹ nv ∈ set (normal-vectors ws) ⟹ w · nv = 0
    lin-indpt-list (ws @ normal-vectors ws)
    length ws + length (normal-vectors ws) = n
    set ws ∩ set (normal-vectors ws) = {}
    is-det-bound db ⟹ set ws ⊆ ℤv ∩ Bounded-vec (of-int Bnd) ⟹
      set (normal-vectors ws) ⊆ ℤv ∩ Bounded-vec (of-int (db (n−1) (max 1 Bnd)))
proof −
  define us where us = basis-extension ws
  from basis-extension[OF assms, folded us-def]
  have units: set us ⊆ set (unit-vecs n)
    and lin: lin-indpt-list (ws @ us)
    and len: length (ws @ us) = n
    by auto
  from lin-indpt-list-length-eq-n[OF lin len]
  have span: span (set (ws @ us)) = carrier-vec n by auto
  from lin[unfolded lin-indpt-list-def]
  have wsus: set (ws @ us) ⊆ carrier-vec n
    and dist: distinct (ws @ us)
    and lin′: lin-indpt (set (ws @ us)) by auto
  let ?nv = normal-vectors ws
  note nv-def = normal-vectors-def[of ws, unfolded Let-def, folded us-def]
  let ?m = length ws
  let ?n = length us
  have lnv[simp]: length ?nv = length us unfolding nv-def by auto
  {
    fix i
    let ?V = set (ws @ us) − {us ! i}
    assume i: i < ?n
    hence nvi: ?nv ! i = normal-vector ?V unfolding nv-def by auto
    from i have us ! i ∈ set us by auto
    with wsus have u: us ! i ∈ carrier-vec n by auto
    have id: ?V ∪ {us ! i} = set (ws @ us) using i by auto
    have V: ?V ⊆ carrier-vec n using wsus by auto
    have finV: finite ?V by auto
    have Suc (card ?V) = card (insert (us ! i) ?V)
      by (subst card-insert-disjoint[OF finV], auto)
    also have insert (us ! i) ?V = set (ws @ us) using i by auto
    finally have cardV: Suc (card ?V) = n
      using len distinct-card[OF dist] by auto
    from subset-li-is-li[OF lin′] have linV: lin-indpt ?V by auto
    from lin-dep-iff-in-span[OF - linV u, unfolded id] wsus lin′
    have usV: us ! i ∉ span ?V by auto
    note nv = normal-vector[OF finV cardV linV V, folded nvi]
    from normal-vector-span[OF cardV V - linV, folded nvi] comm-scalar-prod[OF
- nv(1)]
    have span: span ?V = {x ∈ carrier-vec n. ?nv ! i · x = 0}
```

```
      by auto
    from nv(1,2) have sq-norm (?nv ! i) ≠ 0 by auto
    hence nvi: ?nv ! i · ?nv ! i ≠ 0
      by (auto simp: sq-norm-vec-as-cscalar-prod)
    from span nvi have nvspan: ?nv ! i ∉ span ?V by auto
    from u usV[unfolded span] have ?nv ! i · us ! i ≠ 0 by blast
    note nv nvi this span usV nvspan
  } note nvi = this
  show nv: set ?nv ⊆ carrier-vec n
    unfolding set-conv-nth using nvi(1) by auto
  {
    fix w nv
    assume w: w ∈ set ws
    with dist have wus: w ∉ set us by auto
    assume n: nv ∈ set ?nv
    with w wus show nv · w = 0
      unfolding set-conv-nth[of normal-vectors -] by (auto intro!: nvi(4)[of - w])
    thus w · nv = 0 using comm-scalar-prod[of w n nv] w nv n wsus by auto
  } note scalar-0 = this
  show length ws + length ?nv = n using len by simp
  {
    let ?oi = of-int :: int ⇒ 'a
    assume wsI: set ws ⊆ ℤv ∩ Bounded-vec (?oi Bnd) and db: is-det-bound db
    {
      fix nv
      assume nv ∈ set ?nv
     then obtain i where nv: nv = ?nv ! i and i: i < ?n unfolding set-conv-nth
by auto
      from order.trans[OF units unit-vec-int-bounds]
        wsI have set (ws @ us) − {us ! i} ⊆ ℤv ∩ Bounded-vec (?oi (max 1 Bnd))
using
          Bounded-vec-mono[of ?oi Bnd ?oi (max 1 Bnd), unfolded of-int-le-iff]
        by auto
      from nvi(7)[OF i db this] nv
      have nv ∈ ℤv ∩ Bounded-vec (?oi (db (n − 1) (max 1 Bnd)))
        by auto
    }
    thus set ?nv ⊆ ℤv ∩ Bounded-vec (?oi (db (n − 1) (max 1 Bnd))) by auto
  }
  have dist-nv: distinct ?nv unfolding distinct-conv-nth lnv
  proof (intro allI impI)
    fix i j
    assume i: i < ?n and j: j < ?n and ij: i ≠ j
    with dist have usj: us ! j ∈ set (ws @ us) − {us ! i}
      by (simp, auto simp: distinct-conv-nth set-conv-nth)
    from nvi(4)[OF i this] nvi(9)[OF j]
    show ?nv ! i ≠ ?nv ! j by auto
  qed
  show disj: set ws ∩ set ?nv = {}
```

```
  proof (rule ccontr)
    assume ¬ ?thesis
    then obtain w where w: w ∈ set ws w ∈ set ?nv by auto
    from scalar-0[OF this] this(1) have sq-norm w = 0
      by (auto simp: sq-norm-vec-as-cscalar-prod)
    with w wsus have w = 0v n by auto
    with vs-zero-lin-dep[OF wsus lin'] w(1) show False by auto
  qed
  have dist': distinct (ws @ ?nv) using dist disj dist-nv by auto
  show lin-indpt-list (ws @ ?nv) unfolding lin-indpt-list-def
  proof (intro conjI dist')
    show set: set (ws @ ?nv) ⊆ carrier-vec n using nv wsus by auto
    hence ws: set ws ⊆ carrier-vec n by auto
    have lin-nv: lin-indpt (set ?nv)
    proof
      assume lin-dep (set ?nv)
      from finite-lin-dep[OF finite-set this nv]
       obtain a v where comb: lincomb a (set ?nv) = 0v n and vnv: v ∈ set ?nv
and av0: a v ≠ 0 by auto
      from vnv[unfolded set-conv-nth] obtain i where i: i < ?n and vi: v = ?nv
! i by auto
      define b where b = (λ w. a w / a v)
      define c where c = (λ w. −1 ∗ b w)
      define x where x = lincomb b (set ?nv − {v})
      define w where w = lincomb c (set ?nv − {v})
      have w: w ∈ carrier-vec n unfolding w-def using nv by auto
      have x: x ∈ carrier-vec n unfolding x-def using nv by auto
      from arg-cong[OF comb, of λ x. (1/ a v) ·v x]
      have 0v n = 1 / a v ·v lincomb a (set ?nv) by auto
      also have . . . = lincomb b (set ?nv)
        by (subst lincomb-smult[symmetric, OF nv], auto simp: b-def)
      also have . . . = b v ·v v + lincomb b (set ?nv − {v})
        by (subst lincomb-del2[OF - nv - vnv], auto)
      also have b v ·v v = v using av0 unfolding b-def by auto
      finally have v + lincomb b (set ?nv − {v}) − lincomb b (set ?nv − {v}) =
          0v n − lincomb b (set ?nv − {v}) (is ?l = ?r) by simp
      also have ?l = v
        by (rule add-diff-cancel-right-vec, insert vnv nv, auto)
      also have ?r = w unfolding w-def c-def
        by (subst lincomb-smult, unfold x-def[symmetric], insert nv x, auto)
      finally have vw: v = w .
      have u: us ! i ∈ carrier-vec n using i wsus by auto
      have nv': set ?nv − {?nv ! i} ⊆ carrier-vec n using nv by auto
      have ?nv ! i · us ! i = 0
        unfolding vi[symmetric] vw unfolding w-def vi
        unfolding lincomb-scalar-prod-left[OF nv' u]
      proof (rule sum.neutral, intro ballI)
        fix x
        assume x ∈ set ?nv − {?nv ! i}
```

then obtain *j* **where** *j*: *j* < *?n* **and** *x*: *x* = *?nv* ! *j* **and** *ij*: *i* ≠ *j* **unfolding** *set-conv-nth* **by** *auto*
from *dist*[*simplified*] *ij i j* **have** *us* ! *i* ≠ *us* ! *j* **unfolding** *distinct-conv-nth* **by** *auto*
with *i* **have** *us* ! *i* ∈ *set* (*ws* @ *us*) − {*us* ! *j*} **by** *auto*
from *nvi*(*3*−*4*)[*OF j this*]
show *c x* ∗ (*x* · *us* ! *i*) = *0* **unfolding** *x* **by** *auto*
qed
with *nvi*(*9*)[*OF i*] **show** *False* **..**
qed
from *subset-li-is-li*[*OF lin′*] **have** *lin-indpt* (*set ws*) **by** *auto*
from *ortho-lin-indpt*[*OF nv ws scalar-0 lin-nv this*]
have *lin-indpt* (*set ?nv* ∪ *set ws*) **.**
also have *set ?nv* ∪ *set ws* = *set* (*ws* @ *?nv*) **by** *auto*
finally show *lin-indpt* (*set* (*ws* @ *?nv*)) **.**
qed
qed

definition *pos-norm-vec* :: *′a vec set* ⇒ *′a vec* ⇒ *′a vec* **where**
pos-norm-vec D x = (*let c′* = *normal-vector D*;
c = (*if c′* · *x* > *0 then c′ else* −*c′*) *in c*)

lemma *pos-norm-vec*:
assumes *D*: *D* ⊆ *carrier-vec n* **and** *fin*: *finite D* **and** *lin*: *lin-indpt D*
and *card*: *Suc* (*card D*) = *n*
and *c-def*: *c* = *pos-norm-vec D x*
shows *c* ∈ *carrier-vec n span D* = { *x*. *x* ∈ *carrier-vec n* ∧ *x* · *c* = *0*}
x ∉ *span D* ⟹ *x* ∈ *carrier-vec n* ⟹ *c* · *x* > *0*
c ∈ {*normal-vector D*, −*normal-vector D*}
proof −
have *n*: *normal-vector D* ∈ *carrier-vec n* **using** *normal-vector assms* **by** *auto*
show *cnorm*: *c* ∈ {*normal-vector D*, −*normal-vector D*} **unfolding** *c-def pos-norm-vec-def Let-def* **by** *auto*
then show *c*: *c* ∈ *carrier-vec n* **using** *assms normal-vector* **by** *auto*
have *span D* = { *x*. *x* ∈ *carrier-vec n* ∧ *x* · *normal-vector D* = *0*}
using *normal-vector-span*[*OF card D fin lin*] **by** *auto*
also have ... = { *x*. *x* ∈ *carrier-vec n* ∧ *x* · *c* = *0*} **using** *cnorm c* **by** *auto*
finally show *span-char*: *span D* = { *x*. *x* ∈ *carrier-vec n* ∧ *x* · *c* = *0*} **by** *auto*
{
assume *x*: *x* ∉ *span D x* ∈ *carrier-vec n*
hence *c* · *x* ≠ *0* **using** *comm-scalar-prod*[*OF c*] **unfolding** *span-char* **by** *auto*
hence *normal-vector D* · *x* ≠ *0* **using** *cnorm n x* **by** *auto*
with *x* **have** *b*: ¬ (*normal-vector D* · *x* > *0*) ⟹ (−*normal-vector D*) · *x* > *0*
using *assms n* **by** *auto*
then show *c* · *x* > *0* **unfolding** *c-def pos-norm-vec-def Let-def*
by (*auto split*: *if-splits*)
}
qed

end

end

10 Dimension of Spans

We define the notion of dimension of a span of vectors and prove some natural results about them. The definition is made as a function, so that no interpretation of locales like subspace is required.

theory *Dim-Span*
  imports *Missing-VS-Connect*
begin

context *vec-space*
begin
definition *dim-span W* = *Max* (*card* ‘ {*V*. *V* ⊆ *carrier-vec n* ∧ *V* ⊆ *span W* ∧ *lin-indpt V*})

lemma fixes *V W* :: *'a vec set*
  shows
    card-le-dim-span:
    V ⊆ *carrier-vec n* ⟹ *V* ⊆ *span W* ⟹ *lin-indpt V* ⟹ *card V* ≤ *dim-span W* **and**
    card-eq-dim-span-imp-same-span:
    W ⊆ *carrier-vec n* ⟹ *V* ⊆ *span W* ⟹ *lin-indpt V* ⟹ *card V* = *dim-span W* ⟹ *span V* = *span W* **and**
    same-span-imp-card-eq-dim-span:
    V ⊆ *carrier-vec n* ⟹ *W* ⊆ *carrier-vec n* ⟹ *span V* = *span W* ⟹ *lin-indpt V* ⟹ *card V* = *dim-span W* **and**
    dim-span-cong:
    span V = *span W* ⟹ *dim-span V* = *dim-span W* **and**
    ex-basis-span:
    V ⊆ *carrier-vec n* ⟹ ∃ *W*. *W* ⊆ *carrier-vec n* ∧ *lin-indpt W* ∧ *span V* = *span W* ∧ *dim-span V* = *card W*
proof −
  show *cong*: ⋀ *V W*. *span V* = *span W* ⟹ *dim-span V* = *dim-span W* **unfolding** *dim-span-def* **by** *auto*
  {
    fix *W* :: *'a vec set*
    let *?M* = {*V*. *V* ⊆ *carrier-vec n* ∧ *V* ⊆ *span W* ∧ *lin-indpt V*}
    have *card* ‘ *?M* ⊆ {*0* .. *n*}
    proof
      fix *k*
      assume *k* ∈ *card* ‘ *?M*
      then obtain *V* **where** *V*: *V* ⊆ *carrier-vec n* ∧ *V* ⊆ *span W* ∧ *lin-indpt V*
        and *k*: *k* = *card V*
        by *auto*
      from *V* **have** *card V* ≤ *n* **using** *dim-is-n li-le-dim* **by** *auto*

with *k* **show** $k \in \{0 .. n\}$ **by** *auto*
qed
from *finite-subset*[*OF this*]
have *fin*: *finite* (*card* ‘ *?M*) **by** *auto*
have {} ∈ *?M* **by** (*auto simp*: *span-empty span-zero*)
from *imageI*[*OF this*, *of card*]
have *0* ∈ *card* ‘ *?M* **by** *auto*
hence *Mempty*: *card* ‘ *?M* ≠ {} **by** *auto*
from *Max-ge*[*OF fin*, *folded dim-span-def*]
show ⋀ *V* :: ′*a vec set*. *V* ⊆ *carrier-vec n* ⟹ *V* ⊆ *span W* ⟹ *lin-indpt V* ⟹ *card V* ≤ *dim-span W*
by *auto*
note *this fin Mempty*
} **note** *part1* = *this*
{
fix *V W* :: ′*a vec set*
assume *W*: *W* ⊆ *carrier-vec n*
and *VsW*: *V* ⊆ *span W* **and** *linV*: *lin-indpt V* **and** *card*: *card V* = *dim-span W*
from *W VsW* **have** *V*: *V* ⊆ *carrier-vec n* **using** *span-mem*[*OF W*] **by** *auto*
from *Max-in*[*OF part1*(*2*,*3*), *folded dim-span-def*, *of W*]
obtain *WW* **where** *WW*: *WW* ⊆ *carrier-vec n* *WW* ⊆ *span W* *lin-indpt WW*
and *id*: *dim-span W* = *card WW* **by** *auto*
show *span V* = *span W*
proof (*rule ccontr*)
from *VsW V W* **have** *sub*: *span V* ⊆ *span W* **using** *span-subsetI* **by** *metis*
assume *span V* ≠ *span W*
with *sub* **obtain** *w* **where** *wW*: *w* ∈ *span W* **and** *wsV*: *w* ∉ *span V* **by** *auto*
from *wW W* **have** *w*: *w* ∈ *carrier-vec n* **by** *auto*
from *linV V* **have** *finV*: *finite V* **using** *fin-dim fin-dim-li-fin* **by** *blast*
from *wsV span-mem*[*OF V*, *of w*] **have** *wV*: *w* ∉ *V* **by** *auto*
let *?X* = *insert w V*
have *card ?X* = *Suc* (*card V*) **using** *wV finV* **by** *simp*
hence *gt*: *card ?X* > *dim-span W* **unfolding** *card* **by** *simp*
have *linX*: *lin-indpt ?X* **using** *lin-dep-iff-in-span*[*OF V linV w wV*] *wsV* **by** *auto*
have *XW*: *?X* ⊆ *span W* **using** *wW VsW* **by** *auto*
from *part1*(*1*)[*OF - XW linX*] *w V* **have** *card ?X* ≤ *dim-span W* **by** *auto*
with *gt* **show** *False* **by** *auto*
qed
} **note** *card-dim-span* = *this*
{
fix *V* :: ′*a vec set*
assume *V*: *V* ⊆ *carrier-vec n*
from *Max-in*[*OF part1*(*2*,*3*), *folded dim-span-def*, *of V*]
obtain *W* **where** *W*: *W* ⊆ *carrier-vec n* *W* ⊆ *span V* *lin-indpt W*
and *idW*: *card W* = *dim-span V* **by** *auto*
show ∃ *W*. *W* ⊆ *carrier-vec n* ∧ *lin-indpt W* ∧ *span V* = *span W* ∧ *dim-span V* = *card W*

proof (*intro exI*[*of* - *W*] *conjI W idW*[*symmetric*])
from *card-dim-span*[*OF V*(*1*) *W*(*2*−*3*) *idW*] **show** *span V* = *span W* **by** *auto*
qed
}
{
fix *V W*
assume *V*: *V* ⊆ *carrier-vec n*
and *W*: *W* ⊆ *carrier-vec n*
and *span*: *span V* = *span W*
and *lin*: *lin-indpt V*
from *Max-in*[*OF part1*(*2*,*3*), *folded dim-span-def*, *of W*]
obtain *WW* **where** *WW*: *WW* ⊆ *carrier-vec n WW* ⊆ *span W lin-indpt WW*
and *idWW*: *card WW* = *dim-span W* **by** *auto*
from *card-dim-span*[*OF W WW*(*2*−*3*) *idWW*] *span*
have *spanWW*: *span WW* = *span V* **by** *auto*
from *span* **have** *V* ⊆ *span W* **using** *span-mem*[*OF V*] **by** *auto*
from *part1*(*1*)[*OF V this lin*] **have** *VW*: *card V* ≤ *dim-span W* **.**
have *finWW*: *finite WW* **using** *WW* **by** (*simp add*: *fin-dim-li-fin*)
have *finV*: *finite V* **using** *lin V* **by** (*simp add*: *fin-dim-li-fin*)
from *replacement*[*OF finWW finV V WW*(*3*) *WW*(*2*)[*folded span*], *unfolded idWW*]
obtain *C* :: ′*a vec set*
where *le*: *int* (*card C*) ≤ *int* (*card V*) − *int* (*dim-span W*) **by** *auto*
from *le* **have** *int* (*dim-span W*) + *int* (*card C*) ≤ *int* (*card V*) **by** *linarith*
hence *dim-span W* + *card C* ≤ *card V* **by** *linarith*
with *VW* **show** *card V* = *dim-span W* **by** *auto*
}
qed

lemma *dim-span-le-n*: **assumes** *W*: *W* ⊆ *carrier-vec n* **shows** *dim-span W* ≤ *n*
proof −
from *ex-basis-span*[*OF W*] **obtain** *V* **where**
V: *V* ⊆ *carrier-vec n*
and *lin*: *lin-indpt V*
and *dim*: *dim-span W* = *card V*
by *auto*
show *?thesis* **unfolding** *dim* **using** *lin V*
using *dim-is-n li-le-dim* **by** *auto*
qed

lemma *dim-span-insert*: **assumes** *W*: *W* ⊆ *carrier-vec n*
and *v*: *v* ∈ *carrier-vec n* **and** *vs*: *v* ∉ *span W*
shows *dim-span* (*insert v W*) = *Suc* (*dim-span W*)
proof −
from *ex-basis-span*[*OF W*] **obtain** *V* **where**
V: *V* ⊆ *carrier-vec n*
and *lin*: *lin-indpt V*
and *span*: *span W* = *span V*

and *dim*: *dim-span W* = *card V*
by *auto*
from *V vs*[*unfolded span*] **have** *vV*: $v \notin V$ **using** *span-mem*[*OF V*] **by** *blast*
from *lin-dep-iff-in-span*[*OF V lin v vV*] *vs span*
have *lin'*: *lin-indpt* (*insert v V*) **by** *auto*
have *finV*: *finite V* **using** *lin V* **using** *fin-dim fin-dim-li-fin* **by** *blast*
have *card* (*insert v V*) = *Suc* (*card V*) **using** *finV vV* **by** *auto*
hence *cvV*: *card* (*insert v V*) = *Suc* (*dim-span W*) **using** *dim* **by** *auto*
have *span* (*insert v V*) = *span* (*insert v W*)
using *span V W v* **by** (*metis bot-least insert-subset insert-union span-union-is-sum*)
from *same-span-imp-card-eq-dim-span*[*OF - - this lin'*] *cvV v V W*
show *?thesis* **by** *auto*
qed
end
end

11 The Fundamental Theorem of Linear Inequalities

The theorem states that for a given set of vectors A and vector b, either b is in the cone of a linear independent subset of A, or there is a hyperplane that contains span(A,b)-1 linearly independent vectors of A that separates A from b. We prove this theorem and derive some consequences, e.g., Caratheodory's theorem that b is the cone of A iff b is in the cone of a linear independent subset of A.

theory *Fundamental-Theorem-Linear-Inequalities*
imports
Cone
Normal-Vector
Dim-Span
begin

context *gram-schmidt*
begin

The mentions equivances A-D are:

- A: b is in the cone of vectors A,
- B: b is in the cone of a subset of linear independent of vectors A,
- C: there is no separating hyperplane of b and the vectors A, which contains dim many linear independent vectors of A
- D: there is no separating hyperplane of b and the vectors A

lemma *fundamental-theorem-of-linear-inequalities-A-imp-D*:

assumes *A*: $A \subseteq$ *carrier-vec n*
  and *fin*: *finite A*
  and *b*: $b \in$ *cone A*
shows $\nexists$ *c*. $c \in$ *carrier-vec n* $\wedge$ ($\forall\ a_i \in A.\ c \cdot a_i \geq 0$) $\wedge$ $c \cdot b < 0$
proof
  assume $\exists$ *c*. $c \in$ *carrier-vec n* $\wedge$ ($\forall\ a_i \in A.\ c \cdot a_i \geq 0$) $\wedge$ $c \cdot b < 0$
  then obtain *c* **where** *c*: $c \in$ *carrier-vec n*
    and *ai*: $\bigwedge$ *ai*. $ai \in A \Longrightarrow c \cdot ai \geq 0$
    and *cb*: $c \cdot b < 0$ **by** *auto*
  from *b*[*unfolded cone-def nonneg-lincomb-def finite-cone-def*]
  obtain *l AA* **where** *bc*: b = *lincomb l AA* **and** *l*: *l* ‘ $AA \subseteq \{x.\ x \geq 0\}$ **and** *AA*: $AA \subseteq A$ **by** *auto*
  from *cone-carrier*[*OF A*] *b* **have** *b*: $b \in$ *carrier-vec n* **by** *auto*
  have $0 \leq (\sum ai{\in}AA.\ l\ ai * (c \cdot ai))$
    by (*intro sum-nonneg mult-nonneg-nonneg*, *insert l ai AA*, *auto*)
  also have ... $= (\sum ai{\in}AA.\ l\ ai * (ai \cdot c))$
    by (*rule sum.cong*, *insert c A AA comm-scalar-prod*, *force*+)
  also have ... $= (\sum ai{\in}AA.\ ((l\ ai \cdot_v ai) \cdot c))$
    by (*rule sum.cong*, *insert smult-scalar-prod-distrib c A AA*, *auto*)
  also have ... $= b \cdot c$ **unfolding** *bc lincomb-def*
    by (*subst finsum-scalar-prod-sum*[*symmetric*], *insert c A AA*, *auto*)
  also have ... $= c \cdot b$ **using** *comm-scalar-prod b c* **by** *auto*
  also have ... < 0 **by** *fact*
  finally show *False* **by** *simp*
qed

The difficult direction is that C implies B. To this end we follow the proof that at least one of B and the negation of C is satisfied.

context
  fixes *a* :: *nat* $\Rightarrow$ *'a vec*
    and *b* :: *'a vec*
    and *m* :: *nat*
  assumes *a*: *a* ‘ $\{0 ..< m\} \subseteq$ *carrier-vec n*
    and *inj-a*: *inj-on a* $\{0 ..< m\}$
    and *b*: $b \in$ *carrier-vec n*
    and *full-span*: *span* (*a* ‘ $\{0 ..< m\}$) = *carrier-vec n*
begin

private definition *goal* = (($\exists$ *I*. $I \subseteq \{0 ..< m\}$ $\wedge$ *card* (*a* ‘ *I*) = *n* $\wedge$ *lin-indpt* (*a* ‘ *I*) $\wedge$ $b \in$ *finite-cone* (*a* ‘ *I*))
  $\vee$ ($\exists$ *c I*. $I \subseteq \{0 ..< m\}$ $\wedge$ $c \in$ {*normal-vector* (*a* ‘ *I*), $-$ *normal-vector* (*a* ‘ *I*)} $\wedge$ *Suc* (*card* (*a* ‘ *I*)) = *n*
    $\wedge$ *lin-indpt* (*a* ‘ *I*) $\wedge$ ($\forall\ i < m.\ c \cdot a\ i \geq 0$) $\wedge$ $c \cdot b < 0$))

private lemma *card-a-I*[*simp*]: $I \subseteq \{0 ..< m\} \Longrightarrow$ *card* (*a* ‘ *I*) = *card I*
  by (*smt inj-a card-image inj-on-image-eq-iff subset-image-inj subset-refl subset-trans*)

private lemma *in-a-I*[*simp*]: $I \subseteq \{0 ..< m\} \Longrightarrow i < m \Longrightarrow$ (*a i* $\in$ *a* ‘ *I*) = ($i \in$

I)
  using inj-a
  by (meson atLeastLessThan-iff image-eqI inj-on-image-mem-iff zero-le)

private definition valid-I = { I. card I = n ∧ lin-indpt (a ‘ I) ∧ I ⊆ {0 ..< m}}

private definition cond **where** cond I I′ l c h k ≡
  b = lincomb l (a ‘ I) ∧
  h ∈ I ∧ l (a h) < 0 ∧ (∀ h′. h′ ∈ I ⟶ h′ < h ⟶ l (a h′) ≥ 0) ∧
  c ∈ carrier-vec n ∧ span (a ‘ (I − {h})) = { x. x ∈ carrier-vec n ∧ c · x = 0} ∧ c · b < 0 ∧
  k < m ∧ c · a k < 0 ∧ (∀ k′. k′ < k ⟶ c · a k′ ≥ 0) ∧
  I′ = insert k (I − {h})

private definition step-rel = Restr { (I′′, I). ∃ l c h k. cond I I′′ l c h k } valid-I

private lemma finite-step-rel: finite step-rel
proof (rule finite-subset)
  show step-rel ⊆ (Pow {0 ..< m} × Pow {0 ..< m}) **unfolding** step-rel-def valid-I-def **by** auto
qed auto

private lemma acyclic-imp-goal: acyclic step-rel ⟹ goal
proof (rule ccontr)
  assume ngoal: ¬ goal
  {
    fix I
    assume I: I ∈ valid-I
    hence Im: I ⊆ {0..<m} **and**
      lin: lin-indpt (a ‘ I) **and**
      cardI: card I = n
      by (auto simp: valid-I-def)
    let ?D = (a ‘ I)
    have finD: finite ?D **using** Im infinite-super **by** blast
    have carrD: ?D ⊆ carrier-vec n **using** a Im **by** auto
    have cardD: card ?D = n **using** cardI Im **by** simp
    have spanD: span ?D = carrier-vec n
      by (intro span-carrier-lin-indpt-card-n lin cardD carrD)
    obtain lamb **where** b-is-lincomb: b = lincomb lamb (a ‘ I)
      using finite-in-span[OF fin carrD, of b] **using** spanD b carrD fin-dim lin **by** auto
    define h **where** h = (LEAST h. h ∈ I ∧ lamb (a h) < 0)
    have ∃ I′. (I′, I) ∈ step-rel
    proof (cases ∀ i∈ I . lamb (a i) ≥ 0)
      case cond1-T: True
      have goal **unfolding** goal-def
        by (intro disjI1 exI[of - I] conjI lin cardI
            lincomb-in-finite-cone[OF b-is-lincomb finD - carrD], insert cardI Im

cond1-T, *auto*)
with *ngoal* **show** *?thesis* **by** *auto*
next
case *cond1-F*: *False*
hence ∃ *h*. *h* ∈ *I* ∧ *lamb* (*a h*) < *0* **by** *fastforce*
from *LeastI-ex*[*OF this*, *folded h-def*] **have** *h*: *h* ∈ *I lamb* (*a h*) < *0* **by** *auto*
from *not-less-Least*[*of* - λ *h*. *h* ∈ *I* ∧ *lamb* (*a h*) < *0*, *folded h-def*]
have *h-least*: ∀ *k*. *k* ∈ *I* ⟶ *k* < *h* ⟶ *lamb* (*a k*) ≥ *0* **by** *fastforce*
obtain *I′* **where** *I′-def*: *I′* = *I* − {*h*} **by** *auto*
obtain *c* **where** *c-def*: *c* = *pos-norm-vec* (*a* ‘ *I′*) (*a h*) **by** *auto*
let *?D′* = *a* ‘ *I′*
have *I′m*: *I′* ⊆ {*0*..<*m*} **using** *Im I′-def* **by** *auto*
have *carrD′*: *?D′* ⊆ *carrier-vec n* **using** *a Im I′-def* **by** *auto*
have *finD′*: *finite* (*?D′*) **using** *Im I′-def subset-eq-atLeast0-lessThan-finite* **by** *auto*
have *D′subs*: *?D′* ⊆ *?D* **using** *I′-def* **by** *auto*
have *linD′*: *lin-indpt* (*?D′*) **using** *lin I′-def Im D′subs subset-li-is-li* **by** *auto*
have *D′strictsubs*: *?D* = *?D′* ∪ {*a h*} **using** *h I′-def* **by** *auto*
have *h-nin-I*: *h* ∉ *I′* **using** *h I′-def* **by** *auto*
have *ah-nin-D′*: *a h* ∉ *?D′* **using** *h inj-a Im h-nin-I* **by** (*subst in-a-I*, *auto simp*: *I′-def*)
have *cardD′*: *Suc* (*card* (*?D′*)) = *n* **using** *cardD ah-nin-D′ D′strictsubs finD′* **by** *simp*
have *ah-carr*: *a h* ∈ *carrier-vec n* **using** *h a Im* **by** *auto*
note *pnv* = *pos-norm-vec*[*OF carrD′ finD′ linD′ cardD′ c-def*]
have *ah-nin-span*: *a h* ∉ *span ?D′*
using *D′strictsubs lin-dep-iff-in-span*[*OF carrD′ linD′ ah-carr ah-nin-D′*] *lin* **by** *auto*
have *cah-ge-zero*:*c* · *a h* > *0* **and** *c* ∈ *carrier-vec n*
and *cnorm*: *span ?D′* = {*x* ∈ *carrier-vec n*. *x* · *c* = *0*}
using *ah-carr ah-nin-span pnv* **by** *auto*
have *ccarr*: *c* ∈ *carrier-vec n* **by** *fact*
have *b* · *c* = *lincomb lamb* (*a* ‘ *I*) · *c* **using** *b-is-lincomb* **by** *auto*
also have ... = (∑ *w*∈ *?D*. *lamb w* ∗ (*w* · *c*))
using *lincomb-scalar-prod-left*[*OF carrD*, *of c lamb*] *pos-norm-vec ccarr* **by** *blast*
also have ... = *lamb* (*a h*) ∗ (*a h* · *c*) + (∑ *w*∈ *?D′*. *lamb w* ∗ (*w* · *c*))
using *sum.insert*[*OF finD′ ah-nin-D′*, *of lamb*] *D′strictsubs ah-nin-D′ finD′* **by** *auto*
also have (∑ *w*∈ *?D′*. *lamb w* ∗ (*w* · *c*)) = *0*
apply (*rule sum.neutral*)
using *span-mem*[*OF carrD′*, *unfolded cnorm*] **by** *simp*
also have *lamb* (*a h*) ∗ (*a h* · *c*) + *0* < *0*
using *cah-ge-zero h*(*2*) *comm-scalar-prod*[*OF ah-carr ccarr*]
by (*auto intro*: *mult-neg-pos*)
finally have *cb-le-zero*: *c* · *b* < *0* **using** *comm-scalar-prod*[*OF b ccarr*] **by** *auto*

show *?thesis*

proof (*cases* ∀ *i* < *m* . *c* · *a i* ≥ *0*)
case *cond2-T*: *True*
have *goal*
unfolding *goal-def*
by (*intro disjI2 exI*[*of - c*] *exI*[*of - I′*] *conjI cb-le-zero linD′ cond2-T cardD′ I′m pnv*(*4*))
with *ngoal* **show** *?thesis* **by** *auto*
next
case *cond2-F*: *False*
define *k* **where** *k* = (*LEAST k. k* < *m* ∧ *c* · *a k* < *0*)
let *?I′′* = *insert k I′*
show *?thesis* **unfolding** *step-rel-def*
proof (*intro exI*[*of - ?I′′*], *standard*, *unfold mem-Collect-eq split*, *intro exI*)
from *LeastI-ex*[*OF*]
have ∃ *k. k* < *m* ∧ *c* · *a k* < *0* **using** *cond2-F* **by** *fastforce*
from *LeastI-ex*[*OF this*, *folded k-def*] **have** *k*: *k* < *m c* · *a k* < *0* **by** *auto*
show *cond I ?I′′ lamb c h k* **unfolding** *cond-def I′-def*[*symmetric*] *cnorm*
proof(*intro conjI cb-le-zero b-is-lincomb h ccarr h-least refl k*)
show {*x* ∈ *carrier-vec n. x* · *c* = *0*} = {*x* ∈ *carrier-vec n. c* · *x* = *0*}
using *comm-scalar-prod*[*OF ccarr*] **by** *auto*
from *not-less-Least*[*of -* λ *k. k* < *m* ∧ *c* · *a k* < *0*, *folded k-def*]
have ∀ *k′* < *k* . *k′* > *m* ∨ *c* · *a k′* ≥ *0* **using** *k*(*1*) *less-trans not-less* **by** *blast*
then show ∀ *k′* < *k* . *c* · *a k′* ≥ *0* **using** *k*(*1*) **by** *auto*
qed

have *?I′′* ∈ *valid-I* **unfolding** *valid-I-def*
proof(*standard*, *intro conjI*)
from *k a* **have** *ak-carr*: *a k* ∈ *carrier-vec n* **by** *auto*
have *ak-nin-span*: *a k* ∉ *span ?D′* **using** *k*(*2*) *cnorm comm-scalar-prod*[*OF ak-carr ccarr*] **by** *auto*
hence *ak-nin-D′*: *a k* ∉ *?D′* **using** *span-mem*[*OF carrD′*] **by** *auto*
from *lin-dep-iff-in-span*[*OF carrD′ linD′ ak-carr ak-nin-D′*]
show *lin-indpt* (*a* ‘ *?I′′*) **using** *ak-nin-span* **by** *auto*
show *?I′′* ⊆ {*0..<m*} **using** *I′m k* **by** *auto*
show *card* (*insert k I′*) = *n* **using** *cardD′ ak-nin-D′ finD′*
by (*metis* ‹*insert k I′* ⊆ {*0..<m*}› *card-a-I card-insert-disjoint image-insert*)
qed
then show (*?I′′*, *I*) ∈ *valid-I* × *valid-I* **using** *I* **by** *auto*

qed
qed
qed
} **note** *step* = *this*
{
from *exists-lin-indpt-subset*[*OF a*, *unfolded full-span*]
obtain *A* **where** *lin*: *lin-indpt A* **and** *span*: *span A* = *carrier-vec n* **and** *Am*: *A* ⊆ *a* ‘ {*0* ..<*m*} **by** *auto*

from *Am a* **have** *A*: *A* ⊆ *carrier-vec n* **by** *auto*
from *lin span A* **have** *card*: *card A* = *n*
using *basis-def dim-basis dim-is-n fin-dim-li-fin* **by** *auto*
from *A Am* **obtain** *I* **where** *A*: *A* = *a* ' *I* **and** *I*: *I* ⊆ {*0* ..< *m*} **by** (*metis subset-imageE*)
have *I* ∈ *valid-I* **using** *I card lin* **unfolding** *valid-I-def A* **by** *auto*
hence ∃ *I*. *I* ∈ *valid-I* **..**
}
note *init* = *this*
have *step-valid*: (*I'*,*I*) ∈ *step-rel* ⟹ *I'* ∈ *valid-I* **for** *I I'* **unfolding** *step-rel-def* **by** *auto*
have ¬ (*wf step-rel*)
proof
from *init* **obtain** *I* **where** *I*: *I* ∈ *valid-I* **by** *auto*
assume *wf step-rel*
from *this*[*unfolded wf-eq-minimal*, *rule-format*, *OF I*] *step step-valid* **show** *False* **by** *blast*
qed
with *wf-iff-acyclic-if-finite*[*OF finite-step-rel*]
have ¬ *acyclic step-rel* **by** *auto*
thus *acyclic step-rel* ⟹ *False* **by** *auto*
qed

private lemma *acyclic-step-rel*: *acyclic step-rel*
proof (*rule ccontr*)
assume ¬ *?thesis*
hence ¬ *acyclic* ($step\text{-}rel^{-1}$) **by** *auto*

then obtain *I* **where** (*I*, *I*) ∈ (*step-rel*^*−1*)^+ **unfolding** *acyclic-def* **by** *blast*
from *this*[*unfolded trancl-power*]
obtain *len* **where** (*I*, *I*) ∈ (*step-rel*^*−1*) ^^ *len* **and** *len0*: *len* > *0* **by** *blast*

from *this*[*unfolded relpow-fun-conv*] **obtain** *Is* **where**
stepsIs: ⋀ *i*. *i* < *len* ⟹ (*Is* (*Suc i*), *Is i*) ∈ *step-rel*
and *IsI*: *Is 0* = *I Is len* = *I* **by** *auto*
{
fix *i*
assume *i* ≤ *len* **hence** *i* − *1* < *len* **using** *len0* **by** *auto*
from *stepsIs*[*unfolded step-rel-def*, *OF this*]
have *Is i* ∈ *valid-I* **by** (*cases i*, *auto*)
} **note** *Is-valid* = *this*
from *stepsIs*[*unfolded step-rel-def*]
have ∀ *i*. ∃ *l c h k*. *i* < *len* ⟶ *cond* (*Is i*) (*Is* (*Suc i*)) *l c h k* **by** *auto*

from *choice*[*OF this*] **obtain** *ls* **where** ∀ *i*. ∃ *c h k*. *i* < *len* ⟶ *cond* (*Is i*) (*Is* (*Suc i*)) (*ls i*) *c h k* **by** *auto*
from *choice*[*OF this*] **obtain** *cs* **where** ∀ *i*. ∃ *h k*. *i* < *len* ⟶ *cond* (*Is i*) (*Is* (*Suc i*)) (*ls i*) (*cs i*) *h k* **by** *auto*

from *choice*[*OF this*] **obtain** *hs* **where** ∀ *i*. ∃ *k*. *i* < *len* ⟶ *cond* (*Is i*) (*Is* (*Suc i*)) (*ls i*) (*cs i*) (*hs i*) *k* **by** *auto*
from *choice*[*OF this*] **obtain** *ks* **where**
cond: ⋀ *i*. *i* < *len* ⟹ *cond* (*Is i*) (*Is* (*Suc i*)) (*ls i*) (*cs i*) (*hs i*) (*ks i*) **by** *auto*

let *?R* = {*hs i* | *i*. *i* < *len*}
define *r* **where** *r* = *Max ?R*
from *cond*[*OF len0*] **have** *hs 0* ∈ *I* **using** *IsI* **unfolding** *cond-def* **by** *auto*
hence *R0*: *hs 0* ∈ *?R* **using** *len0* **by** *auto*
have *finR*: *finite ?R* **by** *auto*
from *Max-in*[*OF finR*] *R0*
have *rR*: *r* ∈ *?R* **unfolding** *r-def*[*symmetric*] **by** *auto*
then obtain *p* **where** *rp*: *r* = *hs p* **and** *p*: *p* < *len* **by** *auto*
from *Max-ge*[*OF finR*, *folded r-def*]
have *rLarge*: *i* < *len* ⟹ *hs i* ≤ *r* **for** *i* **by** *auto*
have *exq*: ∃ *q*. *ks q* = *r* ∧ *q* < *len*
proof (*rule ccontr*)
assume *neg*: ¬*?thesis*
show *False*
proof(*cases r* ∈ *I*)
case *True*
have *1*: *j*∈{*Suc p*..*len*} ⟹ *r* ∉ *Is j* **for** *j*
proof(*induction j rule*: *less-induct*)
case (*less j*)
from *less*(*2*) **have** *j-bounds*: *j* = *Suc p* ∨ *j* > *Suc p* **by** *auto*
from *less*(*2*) **have** *j-len*: *j* ≤ *len* **by** *auto*
have *pj-cond*: *j* = *Suc p* ⟹ *cond* (*Is p*) (*Is j*) (*ls p*) (*cs p*) (*hs p*) (*ks p*) **using** *cond p* **by** *blast*
have *r-neq-ksp*: *r* ≠ *ks p* **using** *p neg* **by** *auto*
have *j* = *Suc p* ⟹ *Is j* = *insert* (*ks p*) (*Is p* − {*r*})
using *rp cond pj-cond cond-def*[*of Is p Is j - - r*] **by** *blast*
hence *c1*: *j* = *Suc p* ⟹ *r* ∉ *Is j* **using** *r-neq-ksp* **by** *simp*
have *IH*: ⋀*t*. *t* < *j* ⟹ *t* ∈ {*Suc p*..*len*} ⟹ *r* ∉ *Is t* **by** *fact*
have *r-neq-kspj*: *j* > *Suc p* ∧ *j* ≤ *len* ⟹ *r* ≠ *ks* (*j*−*1*) **using** *j-len neg IH* **by** *auto*
have *jsucj-cond*: *j* > *Suc p* ∧ *j* ≤ *len* ⟹ *Is j* = *insert* (*ks* (*j*−*1*)) (*Is* (*j*−*1*) − {*hs* (*j*−*1*)})
using *cond-def*[*of Is* (*j*−*1*) *Is j*] *cond*
by (*metis* (*no-types*, *lifting*) *Suc-less-eq2 diff-Suc-1 le-simps*(*3*))
hence *j* > *Suc p* ∧ *j* ≤ *len* ⟹ *r* ∉ *insert* (*ks* (*j*−*1*)) (*Is* (*j*−*1*))
using *IH r-neq-kspj* **by** *auto*
hence *j* > *Suc p* ∧ *j* ≤ *len* ⟹ *r* ∉ *Is j* **using** *jsucj-cond* **by** *simp*
then show *?case* **using** *j-bounds j-len c1* **by** *blast*
qed
then show *?thesis* **using** *neg IsI*(*2*) *True p* **by** *auto*
next
case *False*
have *2*: *j*∈{*0*..*p*} ⟹ *r* ∉ *Is j* **for** *j*
proof(*induction j rule*: *less-induct*)

case(*less j*)
from *less*(*2*) **have** *j-bound*: $j \le p$ **by** *auto*
have *r-nin-Is0*: $r \notin Is\ 0$ **using** *IsI*(*1*) *False* **by** *simp*
have *IH*: $\bigwedge t.\ t < j \wedge t \in \{0..p\} \Longrightarrow r \notin Is\ t$ **using** *less.IH* **by** *blast*
have *j-neq-ksjpred*: $j > 0 \Longrightarrow r \neq ks\ (j - 1)$ **using** *neg j-bound p* **by** *auto*
have *Is-jpredj*: $j > 0 \Longrightarrow Is\ j = insert\ (ks\ (j-1))\ (Is\ (j-1) - \{hs\ (j-1)\})$
using *cond-def*[*of Is* (*j−1*) *Is j* - - *hs* (*j−1*) *ks* (*j−1*)] *cond*
by (*metis* (*full-types*) *One-nat-def Suc-pred diff-le-self j-bound le-less-trans p*)
have $j > 0 \Longrightarrow r \notin insert\ (ks\ (j-1))\ (Is\ (j-1))$
using *j-neq-ksjpred IH j-bound* **by** *fastforce*
hence $j > 0 \Longrightarrow r \notin Is\ j$ **using** *Is-jpredj* **by** *blast*
then show *?case* **using** *j-bound r-nin-Is0* **by** *blast*
qed
have *3*: $r \in Is\ p$ **using** *rp cond cond-def p* **by** *blast*
then show *?thesis* **using** *2 3* **by** *auto*
qed
qed
then obtain *q* **where** *q1*: $ks\ q = r$ **and** *q-len*: $q < len$ **by** *blast*

{
fix *t i1 i2*
assume $i1 < len\ i2 < len\ t < m$
assume $t \in Is\ i1\ t \notin Is\ i2$
have $\exists j < len.\ t = hs\ j$
proof (*rule ccontr*)
assume ¬ *?thesis*
hence *hst*: $\bigwedge j.\ j < len \Longrightarrow hs\ j \neq t$ **by** *auto*
have *main*: $t \notin Is\ (i + k) \Longrightarrow i + k \le len \Longrightarrow t \notin Is\ k$ **for** *i k*
proof (*induct i*)
case (*Suc i*)
hence *i*: $i + k < len$ **by** *auto*
from *cond*[*OF this*, *unfolded cond-def*]
have $Is\ (Suc\ i + k) = insert\ (ks\ (i + k))\ (Is\ (i + k) - \{hs\ (i + k)\})$ **by** *auto*
from *Suc*(*2*)[*unfolded this*] *hst*[*OF i*] **have** $t \notin Is\ (i + k)$ **by** *auto*
from *Suc*(*1*)[*OF this*] *i* **show** *?case* **by** *auto*
qed *auto*
from *main*[*of i2 0*] ‹$i2 < len$› ‹$t \notin Is\ i2$› **have** $t \notin Is\ 0$ **by** *auto*
with *IsI* **have** $t \notin Is\ len$ **by** *auto*
with *main*[*of len − i1 i1*] ‹$i1 < len$› **have** $t \notin Is\ i1$ **by** *auto*
with ‹$t \in Is\ i1$› **show** *False* **by** *blast*
qed
} **note** *innotin* = *this*

{
fix *i*
assume *i*: $i \in \{Suc\ r..<m\}$
{

```
      assume i-in-Isp: i ∈ Is p
      have i ∈ Is q
      proof (rule ccontr)
        have i-range: i < m using i by simp
        assume ¬ ?thesis
        then have ex: ∃ j < len. i = hs j
          using innotin[OF p q-len i-range i-in-Isp] by simp
        then obtain j where j-hs: i = hs j by blast
        have i>r using i by simp
        then show False using j-hs p rLarge ex by force
      qed
    }
    hence (i ∈ Is p) ⟹ (i ∈ Is q) by blast
  } note bla = this
  have blin: b = lincomb (ls p) (a ` (Is p)) using cond-def p cond by blast
  have carrDp: (a ` (Is p)) ⊆ carrier-vec n  using Is-valid valid-I-def a p
    by (smt image-subset-iff less-imp-le-nat mem-Collect-eq subsetD)
  have carrcq: cs q ∈ carrier-vec n using cond cond-def q-len by simp
  have ineq1: (cs q) · b < 0 using cond-def q-len cond by blast
  let ?Isp-lt-r = {x ∈ Is p . x < r}
  let ?Isp-gt-r = {x ∈ Is p . x > r}
  have Is-disj: ?Isp-lt-r ∩ ?Isp-gt-r = {} using Is-valid by auto
  have ?Isp-lt-r ⊆ Is p by simp
  hence Isp-lt-0m: ?Isp-lt-r ⊆ {0..<m} using valid-I-def Is-valid p less-imp-le-nat
by blast
  have ?Isp-gt-r ⊆ Is p by simp
  hence Isp-gt-0m: ?Isp-gt-r ⊆ {0..<m} using valid-I-def Is-valid p less-imp-le-nat
by blast
  let ?Dp-lt = a ` ?Isp-lt-r
  let ?Dp-ge = a ` ?Isp-gt-r
  {
    fix A B
    assume Asub: A ⊆ {0..<m} ∪ {0..<Suc r}
    assume Bsub: B ⊆ {0..<m} ∪ {0..<Suc r}
    assume ABinters: A ∩ B = {}
    have r ∈ Is p using rp p cond unfolding cond-def by simp
    hence r-lt-m: r < m using p Is-valid[of p] unfolding valid-I-def by auto
    hence 1: A ⊆ {0..<m} using Asub by auto
    have 2: B ⊆ {0..<m} using r-lt-m Bsub by auto
    have a ` A ∩ a ` B = {}
      using inj-on-image-Int[OF inj-a 1 2] ABinters by auto
  } note inja = this

  have (Is p ∩ {0..<r}) ∩ (Is p ∩ {r}) = {} by auto
  hence a ` (Is p ∩ {0..<r} ∪ Is p ∩ {r}) = a ` (Is p ∩ {0..<r}) ∪ a ` (Is p ∩
{r})
    using inj-a by auto
  moreover have Is p ∩ {0..<r} ∪ Is p ∩ {r} ⊆ {0..<m} ∪ {0..<Suc r} by auto
  moreover have Is p ∩ {Suc r..<m} ⊆ {0..<m} ∪ {0..<Suc r} by auto
```

```
  moreover have (Is p ∩ {0..<r} ∪ Is p ∩ {r}) ∩ (Is p ∩ {Suc r..<m}) = {} by
auto
  ultimately have one: (a ‘ (Is p ∩ {0..<r}) ∪ a ‘ (Is p ∩ {r})) ∩ a ‘ (Is p ∩
{Suc r..<m}) = {}
    using inja[of Is p ∩ {0..<r} ∪ Is p ∩ {r} Is p ∩ {Suc r..<m}] by auto
  have split: Is p = Is p ∩ {0..<r} ∪ Is p ∩ {r} ∪ Is p ∩ {Suc r ..< m}
    using rp p Is-valid[of p] unfolding valid-I-def by auto
  have gtr: (∑ w ∈ (a ‘ (Is p ∩ {Suc r ..< m})). ((ls p) w) ∗ (cs q · w)) = 0
  proof (rule sum.neutral, clarify)
    fix w
    assume w1: w ∈ Is p and w2: w∈{Suc r..<m}
    have w-in-q:  w ∈ Is q using bla[OF w2] w1 by blast
    moreover have hs q ≤ r using rR rLarge using q-len by blast
    ultimately have w ≠ hs q using w2 by simp
    hence w ∈ Is q −{hs q} using w1 w-in-q by auto
    moreover have Is q − {hs q} ⊆ {0..<m}
      using q-len Is-valid[of q] unfolding valid-I-def by auto
   ultimately have a w ∈ span ( a ‘ (Is q − {hs q})) using a by (intro span-mem,
auto)
    moreover have cs q ∈ carrier-vec n ∧ span (a ‘ (Is q − {hs q})) =
      { x. x ∈ carrier-vec n ∧ cs q · x = 0}
      using cond[of q] q-len unfolding cond-def by auto
    ultimately have (cs q) · (a w) = 0 using a w2 by simp
    then show ls p (a w) ∗ (cs q · a w) = 0 by simp
  qed
  note pp = cond[OF p, unfolded cond-def rp[symmetric]]
  note qq = cond[OF q-len, unfolded cond-def q1]
  have (cs q) · b = (cs q) · lincomb (ls p) (a ‘ (Is p)) using blin by auto
  also have . . . = (∑ w ∈ (a ‘ (Is p)). ((ls p) w) ∗ (cs q · w))
    by (subst lincomb-scalar-prod-right[OF carrDp carrcq], simp)
  also have . . . = (∑ w ∈ (a ‘ (Is p ∩ {0..<r}) ∪ a ‘ (Is p ∩ {r}) ∪ a ‘ (Is p ∩
{Suc r..<m})).
     ((ls p) w) ∗ (cs q · w))
    by (subst (1) split, rule sum.cong, auto)
  also have . . . = (∑ w ∈ (a ‘ (Is p ∩ {0..<r})). ((ls p) w) ∗ (cs q · w))
      + (∑ w ∈ (a ‘ (Is p ∩ {r})). ((ls p) w) ∗ (cs q · w))
     + (∑ w ∈ (a ‘ (Is p ∩ {Suc r ..< m})). ((ls p) w) ∗ (cs q · w))
    apply (subst sum.union-disjoint[OF - - one])
      apply (force+)[2]
    apply (subst sum.union-disjoint)
       apply (force+)[2]
     apply (rule inja)
    by auto
  also have . . . = (∑ w ∈ (a ‘ (Is p ∩ {0..<r})). ((ls p) w) ∗ (cs q · w))
      + (∑ w ∈ (a ‘ (Is p ∩ {r})). ((ls p) w) ∗ (cs q · w))
    using sum.neutral gtr by simp
  also have . . . > 0 + 0
  proof (intro add-le-less-mono sum-nonneg mult-nonneg-nonneg)
    {
```

fix *x*
assume *x*: $x \in a$ ‘ $(Is\ p \cap \{0..<r\})$
show $0 \leq ls\ p\ x$ **using** *pp x* **by** *auto*
show $0 \leq cs\ q \cdot x$ **using** *qq x* **by** *auto*
}
have $r \in Is\ p$ **using** *pp* **by** *blast*
hence a ‘ $(Is\ p \cap \{r\}) = \{a\ r\}$ **by** *auto*
hence *id*: $(\sum w{\in}a$ ‘ $(Is\ p \cap \{r\}).\ ls\ p\ w * (cs\ q \cdot w)) = ls\ p\ (a\ r) * (cs\ q \cdot a\ r)$
by *simp*
show $0 < (\sum w{\in}a$ ‘ $(Is\ p \cap \{r\}).\ ls\ p\ w * (cs\ q \cdot w))$
unfolding *id*
proof (*rule mult-neg-neg*)
show $ls\ p\ (a\ r) < 0$ **using** *pp* **by** *auto*
show $cs\ q \cdot a\ r < 0$ **using** *qq* **by** *auto*
qed
qed
finally have $cs\ q \cdot b > 0$ **by** *simp*
moreover have $cs\ q \cdot b < 0$ **using** *qq* **by** *blast*
ultimately show *False* **by** *auto*
qed

lemma *fundamental-theorem-neg-C-or-B-in-context*:
assumes *W*: $W = a$ ‘ $\{0\ ..<\ m\}$
shows $(\exists\ U.\ U \subseteq W \wedge card\ U = n \wedge lin\text{-}indpt\ U \wedge b \in finite\text{-}cone\ U) \vee$
$(\exists\, c\ U.\ U \subseteq W \wedge$
$c \in \{normal\text{-}vector\ U, -\ normal\text{-}vector\ U\} \wedge$
$Suc\ (card\ U) = n \wedge lin\text{-}indpt\ U \wedge (\forall\, w \in W.\ 0 \leq c \cdot w) \wedge c \cdot b < 0)$
using *acyclic-imp-goal*[*unfolded goal-def*, *OF acyclic-step-rel*]
proof
assume $\exists\, I.\ I{\subseteq}\{0..<m\} \wedge card\ (a$ ‘ $I) = n \wedge lin\text{-}indpt\ (a$ ‘ $I) \wedge b \in finite\text{-}cone$
$(a$ ‘ $I)$
thus *?thesis* **unfolding** *W* **by** (*intro disjI1*, *blast*)
next
assume $\exists\, c\ I.\ I \subseteq \{0..<m\} \wedge$
$c \in \{normal\text{-}vector\ (a$ ‘ $I), -\ normal\text{-}vector\ (a$ ‘ $I)\} \wedge$
$Suc\ (card\ (a$ ‘ $I)) = n \wedge lin\text{-}indpt\ (a$ ‘ $I) \wedge (\forall\, i{<}m.\ 0 \leq c \cdot a\ i) \wedge c \cdot b$
< 0
then obtain *c I* **where** $I \subseteq \{0..<m\} \wedge$
$c \in \{normal\text{-}vector\ (a$ ‘ $I), -\ normal\text{-}vector\ (a$ ‘ $I)\} \wedge$
$Suc\ (card\ (a$ ‘ $I)) = n \wedge lin\text{-}indpt\ (a$ ‘ $I) \wedge (\forall\, i{<}m.\ 0 \leq c \cdot a\ i) \wedge c \cdot b$
< 0 **by** *auto*
thus *?thesis* **unfolding** *W*
by (*intro disjI2 exI*[*of - c*] *exI*[*of - a* ‘ *I*], *auto*)
qed

end

lemma *fundamental-theorem-of-linear-inequalities-C-imp-B-full-dim*:
assumes *A*: $A \subseteq carrier\text{-}vec\ n$

and *fin*: *finite A*
and *span*: *span A* = *carrier-vec n*
and *b*: *b* ∈ *carrier-vec n*
and *C*: ∄ *c B*. *B* ⊆ *A* ∧ *c* ∈ {*normal-vector B*, − *normal-vector B*} ∧ *Suc* (*card B*) = *n*
∧ *lin-indpt B* ∧ (∀ a_i ∈ *A*. *c* • a_i ≥ *0*) ∧ *c* • *b* < *0*
shows ∃ *B* ⊆ *A*. *lin-indpt B* ∧ *card B* = *n* ∧ *b* ∈ *finite-cone B*
proof −
from *finite-distinct-list*[*OF fin*] **obtain** *as* **where** *Aas*: *A* = *set as* **and** *dist*: *distinct as* **by** *auto*
define *m* **where** *m* = *length as*
define *a* **where** *a* = (λ *i*. *as* ! *i*)
have *inj*: *inj-on a* {*0*..< (*m* :: *nat*)}
and *id*: *A* = *a* ' {*0*..<*m*}
unfolding *m-def a-def Aas* **using** *inj-on-nth*[*OF dist*] **unfolding** *set-conv-nth*
by *auto*
from *fundamental-theorem-neg-C-or-B-in-context*[*OF - inj b*, *folded id*, *OF A span refl*] *C*
show *?thesis* **by** *blast*
qed

lemma *fundamental-theorem-of-linear-inequalities-full-dim*: **fixes** *A* :: *'a vec set*
defines *HyperN* ≡ {*b*. *b* ∈ *carrier-vec n* ∧ (∄ *B c*. *B* ⊆ *A* ∧ *c* ∈ {*normal-vector B*, − *normal-vector B*}
∧ *Suc* (*card B*) = *n* ∧ *lin-indpt B* ∧ (∀ a_i ∈ *A*. *c* • a_i ≥ *0*) ∧ *c* • *b* < *0*)}
defines *HyperA* ≡ {*b*. *b* ∈ *carrier-vec n* ∧ (∄ *c*. *c* ∈ *carrier-vec n* ∧ (∀ a_i ∈ *A*. *c* • a_i ≥ *0*) ∧ *c* • *b* < *0*)}
defines *lin-indpt-cone* ≡ ⋃ { *finite-cone B* | *B*. *B* ⊆ *A* ∧ *card B* = *n* ∧ *lin-indpt B*}
assumes *A*: *A* ⊆ *carrier-vec n*
and *fin*: *finite A*
and *span*: *span A* = *carrier-vec n*
shows
cone A = *lin-indpt-cone*
cone A = *HyperN*
cone A = *HyperA*
proof −
have *lin-indpt-cone* ⊆ *cone A* **unfolding** *lin-indpt-cone-def cone-def* **using** *fin finite-cone-mono A*
by *auto*
moreover have *cone A* ⊆ *HyperA*
proof
fix *c*
assume *cA*: *c* ∈ *cone A*
from *fundamental-theorem-of-linear-inequalities-A-imp-D*[*OF A fin this*] *cone-carrier*[*OF A*] *cA*
show *c* ∈ *HyperA* **unfolding** *HyperA-def* **by** *auto*
qed

moreover have *HyperA* ⊆ *HyperN*
proof
fix *c*
assume *c* ∈ *HyperA*
hence *False*: ⋀ *v*. *v* ∈ *carrier-vec n* ⟹ (∀ a_i∈*A*. *0* ≤ *v* · a_i) ⟹ *v* · *c* < *0* ⟹ *False*
and *c*: *c* ∈ *carrier-vec n* **unfolding** *HyperA-def* **by** *auto*
show *c* ∈ *HyperN*
unfolding *HyperN-def*
proof (*standard*, *intro conjI c notI*, *clarify*, *goal-cases*)
case (*1 W nv*)
with *A fin* **have** *fin*: *finite W* **and** *W*: *W* ⊆ *carrier-vec n* **by** (*auto intro*: *finite-subset*)
show *?case* **using** *False*[*of nv*] *1 normal-vector*[*OF fin - - W*] **by** *auto*
qed
qed
moreover have *HyperN* ⊆ *lin-indpt-cone*
proof
fix *b*
assume *b* ∈ *HyperN*
from *this*[*unfolded HyperN-def*]
fundamental-theorem-of-linear-inequalities-C-imp-B-full-dim[*OF A fin span*, *of b*]
show *b* ∈ *lin-indpt-cone* **unfolding** *lin-indpt-cone-def* **by** *auto*
qed
ultimately show
cone A = *lin-indpt-cone*
cone A = *HyperN*
cone A = *HyperA*
by *auto*
qed

lemma *fundamental-theorem-of-linear-inequalities-C-imp-B*:
assumes *A*: *A* ⊆ *carrier-vec n*
and *fin*: *finite A*
and *b*: *b* ∈ *carrier-vec n*
and *C*: ∄ *c A′*. *c* ∈ *carrier-vec n*
∧ *A′* ⊆ *A* ∧ *Suc* (*card A′*) = *dim-span* (*insert b A*)
∧ (∀ *a* ∈ *A′*. *c* · *a* = *0*)
∧ *lin-indpt A′* ∧ (∀ a_i ∈ *A*. *c* · a_i ≥ *0*) ∧ *c* · *b* < *0*
shows ∃ *B* ⊆ *A*. *lin-indpt B* ∧ *card B* = *dim-span A* ∧ *b* ∈ *finite-cone B*
proof −
from *exists-lin-indpt-sublist*[*OF A*] **obtain** *A′* **where**
lin: *lin-indpt-list A′* **and** *span*: *span* (*set A′*) = *span A* **and** *A′A*: *set A′* ⊆ *A* **by** *auto*
hence *linA′*: *lin-indpt* (*set A′*) **unfolding** *lin-indpt-list-def* **by** *auto*
from *A′A A* **have** *A′*: *set A′* ⊆ *carrier-vec n* **by** *auto*
have *dim-spanA*: *dim-span A* = *card* (*set A′*)
by (*rule sym*, *rule same-span-imp-card-eq-dim-span*[*OF A′ A span linA′*])

show *?thesis*
proof (*cases b* ∈ *span A*)
case *False*
with *span* **have** *b* ∉ *span* (*set A′*) **by** *auto*
with *lin* **have** *linAb*: *lin-indpt-list* (*A′* @ [*b*]) **unfolding** *lin-indpt-list-def*
using *lin-dep-iff-in-span*[*OF A′ - b*] *span-mem*[*OF A′*, *of b*] *b* **by** *auto*
interpret *gso*: *gram-schmidt-fs-lin-indpt n A′* @ [*b*]
by (*standard*, *insert linAb*[*unfolded lin-indpt-list-def*], *auto*)
let *?m* = *length A′*
define *c* **where** *c* = − *gso.gso ?m*
have *c*: *c* ∈ *carrier-vec n* **using** *gso.gso-carrier*[*of ?m*] **unfolding** *c-def* **by** *auto*
from *gso.gso-times-self-is-norm*[*of ?m*]
have *b* · *gso.gso ?m* = *sq-norm* (*gso.gso ?m*) **unfolding** *c-def* **using** *b c* **by** *auto*
also have . . . > *0* **using** *gso.sq-norm-pos*[*of ?m*] **by** *auto*
finally have *cb*: *c* · *b* < *0* **using** *b c comm-scalar-prod*[*OF b c*] **unfolding** *c-def* **by** *auto*
{
fix *a*
assume *a* ∈ *A*
hence *a* ∈ *span* (*set A′*) **unfolding** *span* **using** *span-mem*[*OF A*] **by** *auto*
from *finite-in-span*[*OF - A′ this*]
obtain *l* **where** *a* = *lincomb l* (*set A′*) **by** *auto*
hence *c* · *a* = *c* · *lincomb l* (*set A′*) **by** *simp*
also have . . . = *0*
by (*subst lincomb-scalar-prod-right*[*OF A′ c*], *rule sum.neutral*, *insert A′*, *unfold set-conv-nth*,
insert gso.gso-scalar-zero[*of ?m*] *c*, *auto simp*: *c-def nth-append*)
finally have *c* · *a* = *0* **.**
} **note** *cA* = *this*
have ∃ *c A′*. *c* ∈ *carrier-vec n* ∧ *A′* ⊆ *A* ∧ *Suc* (*card A′*) = *dim-span* (*insert b A*)
∧ (∀ *a* ∈ *A′*. *c* · *a* = *0*) ∧ *lin-indpt A′* ∧ (∀ a_i ∈ *A*. *c* · a_i ≥ *0*) ∧ *c* · *b* < *0*
proof (*intro exI*[*of - c*] *exI*[*of - set A′*] *conjI A′A linA′ cb c*)
show ∀ *a*∈*set A′*. *c* · *a* = *0* ∀ a_i∈*A*. *0* ≤ *c* · a_i **using** *cA A′A* **by** *auto*
have *dim-span* (*insert b A*) = *Suc* (*dim-span A*)
by (*rule dim-span-insert*[*OF A b False*])
also have . . . = *Suc* (*card* (*set A′*)) **unfolding** *dim-spanA* **..**
finally show *Suc* (*card* (*set A′*)) = *dim-span* (*insert b A*) **..**
qed
with *C* **have** *False* **by** *blast*
thus *?thesis* **..**
next
case *bspan*: *True*
define *N* **where** *N* = *normal-vectors A′*
from *normal-vectors*[*OF lin*, *folded N-def*]
have *N*: *set N* ⊆ *carrier-vec n* **and**
orthA′N: ⋀ *w nv*. *w* ∈ *set A′* ⟹ *nv* ∈ *set N* ⟹ *nv* · *w* = *0* **and**

linAN: *lin-indpt-list* $(A' @ N)$ **and**
lenAN: *length* $(A' @ N) = n$ **and**
disj: *set* $A' \cap$ *set* $N = \{\}$ **by** *auto*
from *linAN lenAN* **have** *full-span'*: *span* (*set* $(A' @ N)$) = *carrier-vec* n
using *lin-indpt-list-length-eq-n* **by** *blast*
hence *full-span''*: *span* (*set* $A' \cup$ *set* N) = *carrier-vec* n **by** *auto*
from A N A' **have** *AN*: $A \cup$ *set* $N \subseteq$ *carrier-vec* n **and** *A'N*: *set* $(A' @ N) \subseteq$ *carrier-vec* n **by** *auto*
hence *span* ($A \cup$ *set* N) $\subseteq$ *carrier-vec* n **by** (*simp add*: *span-is-subset2*)
with *A'A span-is-monotone*[*of set* $(A' @ N)$ $A \cup$ *set* N, *unfolded full-span'*]
have *full-span*: *span* ($A \cup$ *set* N) = *carrier-vec* n **unfolding** *set-append* **by** *fast*
from *fin* **have** *finAN*: *finite* ($A \cup$ *set* N) **by** *auto*
note *fundamental* = *fundamental-theorem-of-linear-inequalities-full-dim*[*OF AN finAN full-span*]
show *?thesis*
proof (*cases* $b \in$ *cone* ($A \cup$ *set* N))
case *True*
from *this*[*unfolded fundamental*(*1*)] **obtain** C **where** *CAN*: $C \subseteq A \cup$ *set* N **and** *cardC*: *card* $C = n$
and *linC*: *lin-indpt* C
and *bC*: $b \in$ *finite-cone* C **by** *auto*
have *finC*: *finite* C **using** *finite-subset*[*OF CAN*] *fin* **by** *auto*
from *CAN* A N **have** C: $C \subseteq$ *carrier-vec* n **by** *auto*
from *bC*[*unfolded finite-cone-def nonneg-lincomb-def*] *finC* **obtain** c
where *bC*: b = *lincomb* c C **and** *nonneg*: $\bigwedge b.\ b \in C \Longrightarrow c\ b \geq 0$ **by** *auto*
let *?C* = C − *set* N
show *?thesis*
proof (*intro exI*[*of* - *?C*] *conjI*)
from *subset-li-is-li*[*OF linC*] **show** *lin-indpt ?C* **by** *auto*
show *CA*: *?C* $\subseteq A$ **using** *CAN* **by** *auto*
have *bc*: b = *lincomb* c (*?C* $\cup$ ($C \cap$ *set* N)) **unfolding** *bC*
by (*rule arg-cong*[*of* - - *lincomb* -], *auto*)
have b = *lincomb* c (*?C* − $C \cap$ *set* N)
proof (*rule orthogonal-cone*(*2*)[*OF A N fin full-span'' orthA'N refl span A'A linAN lenAN - CA - bc*])
show $\forall$ *w*$\in$*set* N. $w \cdot b = 0$
using *ortho-span'*[*OF A' N - bspan*[*folded span*]] *orthA'N* **by** *auto*
qed *auto*
also have *?C* − $C \cap$ *set* N = *?C* **by** *auto*
finally have b = *lincomb* c *?C* **.**
with *nonneg* **have** *nonneg-lincomb* c *?C* b **unfolding** *nonneg-lincomb-def* **by** *auto*
thus $b \in$ *finite-cone ?C* **unfolding** *finite-cone-def* **using** *finite-subset*[*OF CA fin*] **by** *auto*
have *Cid*: $C \cap$ *set* $N \cup$ *?C* = C **by** *auto*
have *length* A' + *length* $N = n$ **by** *fact*
also have ... = *card* ($C \cap$ *set* $N \cup$ *?C*) **using** *Cid cardC* **by** *auto*
also have ... = *card* ($C \cap$ *set* N) + *card ?C*
by (*subst card-Un-disjoint*, *insert finC*, *auto*)

```
      also have ... ≤ length N + card ?C
        by (rule add-right-mono, rule order.trans, rule card-mono[OF finite-set[of
N]],
            auto intro: card-length)
      also have length A′ = card (set A′) using lin[unfolded lin-indpt-list-def]
          distinct-card[of A′] by auto
      finally have le: dim-span A ≤ card ?C using dim-spanA by auto
      have CA: ?C ⊆ span A using CA C in-own-span[OF A] by auto
      have linC: lin-indpt ?C using subset-li-is-li[OF linC] by auto
      show card ?C = dim-span A
        using card-le-dim-span[OF - CA linC] le C by force
    qed
  next
    case False
    from False[unfolded fundamental(2)] b
    obtain C c where
      CAN: C ⊆ A ∪ set N and
      cardC: Suc (card C) = n and
      linC: lin-indpt C and
      contains: (∀ a_i∈A ∪ set N. 0 ≤ c · a_i) and
      cb: c · b < 0 and
      nv: c ∈ {normal-vector C, − normal-vector C}
      by auto
    from CAN A N have C: C ⊆ carrier-vec n by auto
    from cardC have cardCn: card C < n by auto
    from finite-subset[OF CAN] fin have finC: finite C by auto
    let ?C = C − set N
    note nv′ = normal-vector(1−4)[OF finC cardC linC C]
    from nv′ nv have c: c ∈ carrier-vec n by auto
    have ∃ c A′. c ∈ carrier-vec n ∧ A′ ⊆ A ∧ Suc (card A′) = dim-span (insert
b A)
        ∧ (∀ a ∈ A′. c · a = 0) ∧ lin-indpt A′ ∧ (∀ a_i ∈ A. c · a_i ≥ 0) ∧ c · b
< 0
    proof (intro exI[of - c] exI[of - ?C] conjI cb c)
      show CA: ?C ⊆ A using CAN by auto
      show ∀ a_i∈A. 0 ≤ c · a_i using contains by auto
      show lin′: lin-indpt ?C using subset-li-is-li[OF linC] by auto
      show sC0: ∀ a∈ ?C. c · a = 0 using nv′ nv C by auto
      have Cid: C ∩ set N ∪ ?C = C by auto
      have dim-span (set A′) = card (set A′)
        by (rule sym, rule same-span-imp-card-eq-dim-span[OF A′ A′ refl linA′])
      also have ... = length A′
        using lin[unfolded lin-indpt-list-def] distinct-card[of A′] by auto
      finally have dimA′: dim-span (set A′) = length A′ .
    from bspan have span (insert b A) = span A using b A using already-in-span
by auto
      from dim-span-cong[OF this[folded span]] dimA′
      have dimbA: dim-span (insert b A) = length A′ by simp
      also have ... = Suc (card ?C)
```

```
        proof (rule ccontr)
          assume neq: length A' ≠ Suc (card ?C)
          have length A' + length N = n by fact
          also have ... = Suc (card (C ∩ set N ∪ ?C)) using Cid cardC by auto
          also have ... = Suc (card (C ∩ set N) + card ?C)
            by (subst card-Un-disjoint, insert finC, auto)
           finally have id: length A' + length N = Suc (card (C ∩ set N) + card
?C) .
          have le1: card (C ∩ set N) ≤ length N
            by (metis Int-lower2 List.finite-set card-length card-mono inf.absorb-iff2
le-inf-iff)
         from CA C A have CsA: ?C ⊆ span (set A') unfolding span by (meson
in-own-span order.trans)
          from card-le-dim-span[OF - this lin'] C
          have le2: card ?C ≤ length A' unfolding dimA' by auto
          from id le1 le2 neq
          have id2: card ?C = length A' by linarith+
          from card-eq-dim-span-imp-same-span[OF A' CsA lin' id2[folded dimA']]
          have span ?C = span A unfolding span by auto
          with bspan have b ∈ span ?C by auto
          from orthocompl-span[OF - - c this] C sC0
          have c · b = 0 by auto
          with cb show False by simp
        qed
        finally show Suc (card ?C) = dim-span (insert b A) by simp
      qed
      with assms(4) have False by blast
      thus ?thesis ..
    qed
  qed
qed

lemma fundamental-theorem-of-linear-inequalities: fixes A :: 'a vec set
  defines HyperN ≡ {b. b ∈ carrier-vec n ∧ (∄ c B. c ∈ carrier-vec n ∧ B ⊆ A
      ∧ Suc (card B) = dim-span (insert b A) ∧ lin-indpt B
      ∧ (∀ a ∈ B. c · a = 0)
      ∧ (∀ a_i ∈ A. c · a_i ≥ 0) ∧ c · b < 0)}
  defines HyperA ≡ {b. b ∈ carrier-vec n ∧ (∄ c. c ∈ carrier-vec n ∧ (∀ a_i ∈ A.
c · a_i ≥ 0) ∧ c · b < 0)}
  defines lin-indpt-cone ≡ ⋃ { finite-cone B | B. B ⊆ A ∧ card B = dim-span A
∧ lin-indpt B}
  assumes A: A ⊆ carrier-vec n
    and fin: finite A
  shows
    cone A = lin-indpt-cone
    cone A = HyperN
    cone A = HyperA
proof −
  have lin-indpt-cone ⊆ cone A
```

unfolding *lin-indpt-cone-def cone-def* **using** *fin finite-cone-mono A* **by** *auto*
moreover have *cone A* ⊆ *HyperA*
using *fundamental-theorem-of-linear-inequalities-A-imp-D*[*OF A fin*] *cone-carrier*[*OF A*]
unfolding *HyperA-def* **by** *blast*
moreover have *HyperA* ⊆ *HyperN* **unfolding** *HyperA-def HyperN-def* **by** *blast*
moreover have *HyperN* ⊆ *lin-indpt-cone*
proof
fix *b*
assume *b* ∈ *HyperN*
from *this*[*unfolded HyperN-def*]
fundamental-theorem-of-linear-inequalities-C-imp-B[*OF A fin*, *of b*]
show *b* ∈ *lin-indpt-cone* **unfolding** *lin-indpt-cone-def* **by** *blast*
qed
ultimately show
cone A = *lin-indpt-cone*
cone A = *HyperN*
cone A = *HyperA*
by *auto*
qed

corollary *Caratheodory-theorem*: **assumes** *A*: *A* ⊆ *carrier-vec n*
shows *cone A* = ⋃ {*finite-cone B* |*B*. *B* ⊆ *A* ∧ *lin-indpt B*}
proof
show ⋃ {*finite-cone B* |*B*. *B* ⊆ *A* ∧ *lin-indpt B*} ⊆ *cone A* **unfolding** *cone-def*
using *fin*[*OF fin-dim - subset-trans*[*OF - A*]] **by** *auto*
{
fix *a*
assume *a* ∈ *cone A*
from *this*[*unfolded cone-def*] **obtain** *B* **where**
finB: *finite B* **and** *BA*: *B* ⊆ *A* **and** *a*: *a* ∈ *finite-cone B* **by** *auto*
from *BA A* **have** *B*: *B* ⊆ *carrier-vec n* **by** *auto*
hence *a* ∈ *cone B* **using** *finB a* **by** (*simp add*: *cone-iff-finite-cone*)
with *fundamental-theorem-of-linear-inequalities*(*1*)[*OF B finB*]
obtain *C* **where** *CB*: *C* ⊆ *B* **and** *a*: *a* ∈ *finite-cone C* **and** *lin-indpt C* **by** *auto*
with *BA* **have** *a* ∈ ⋃ {*finite-cone B* |*B*. *B* ⊆ *A* ∧ *lin-indpt B*} **by** *auto*
}
thus ⋃ {*finite-cone B* |*B*. *B* ⊆ *A* ∧ *lin-indpt B*} ⊇ *cone A* **by** *blast*
qed
end
end

12 Farkas' Lemma

We prove two variants of Farkas' lemma. Note that type here is more general than in the versions of Farkas' Lemma which are in the AFP-entry Farkas-Lemma, which is restricted to rational matrices. However, there δ-rationals are supported, which are not present here.

```
theory Farkas-Lemma
  imports Fundamental-Theorem-Linear-Inequalities
begin

context gram-schmidt
begin

lemma Farkas-Lemma: fixes A :: 'a mat and b :: 'a vec
  assumes A: A ∈ carrier-mat n nr and b: b ∈ carrier-vec n
  shows (∃ x. x ≥ 0_v nr ∧ A *_v x = b) ⟷ (∀ y. y ∈ carrier-vec n ⟶ A^T *_v
y ≥ 0_v nr ⟶ y · b ≥ 0)
proof −
  let ?C = set (cols A)
  from A have C: ?C ⊆ carrier-vec n and C': ∀ w∈set (cols A). dim-vec w = n
    unfolding cols-def by auto
  have (∃ x. x ≥ 0_v nr ∧ A *_v x = b) = (b ∈ cone ?C)
    using cone-of-cols[OF A b] by simp
  also have ... = (∄ y. y ∈ carrier-vec n ∧ (∀ a_i∈?C. 0 ≤ y · a_i) ∧ y · b < 0)
   unfolding fundamental-theorem-of-linear-inequalities(3)[OF C finite-set] mem-Collect-eq
    using b by auto
  also have ... = (∀ y. y ∈ carrier-vec n ⟶ (∀ a_i∈?C. 0 ≤ y · a_i) ⟶ y · b ≥
0)
    by auto
  also have ... = (∀ y. y ∈ carrier-vec n ⟶ A^T *_v y ≥ 0_v nr ⟶ y · b ≥ 0)
  proof (intro all-cong imp-cong refl)
    fix y :: 'a vec
    assume y: y ∈ carrier-vec n
    have (∀ a_i∈ ?C. 0 ≤ y · a_i) = (∀ a_i∈ ?C. 0 ≤ a_i · y)
      by (intro ball-cong[OF refl], subst comm-scalar-prod[OF y], insert C, auto)
    also have ... = (0_v nr ≤ A^T *_v y)
      unfolding less-eq-vec-def using C A y by (auto simp: cols-def)
    finally show (∀ a_i∈set (cols A). 0 ≤ y · a_i) = (0_v nr ≤ A^T *_v y) .
  qed
  finally show ?thesis .
qed

lemma Farkas-Lemma':
  fixes A :: 'a mat and b :: 'a vec
  assumes A: A ∈ carrier-mat nr nc and b: b∈ carrier-vec nr
  shows (∃ x. x∈ carrier-vec nc ∧ A *_v x ≤ b)
        ⟷ (∀ y. y ≥ 0_v nr ∧ A^T *_v y = 0_v nc ⟶ y · b ≥ 0)
proof −
  define B where B = (1_m nr) @_c (A @_c −A)
  define b' where b' = 0_v nc @_v (b @_v −b)
  define n where n = nr + (nc + nc)
  have id0: 0_v (nr + (nc + nc)) = 0_v nr @_v (0_v nc @_v 0_v nc) by (intro eq-vecI,
auto)
  have idcarr: (1_m nr) ∈ carrier-mat nr nr by auto
```

have B: $B \in$ *carrier-mat nr n* **unfolding** *B-def n-def* **using** A **by** *auto*
have $(\exists x \in$ *carrier-vec nc*. $A *_v x \leq b) =$
$(\exists x1 \in$ *carrier-vec nr*. $\exists x2 \in$ *carrier-vec nc*. $\exists x3 \in$ *carrier-vec nc*.
$x1 \geq 0_v\ nr \wedge x2 \geq 0_v\ nc \wedge x3 \geq 0_v\ nc \wedge B *_v (x1\ @_v\ (x2\ @_v\ x3)) = b)$
proof
assume $\exists x \in$ *carrier-vec nc*. $A *_v x \leq b$
from *this* **obtain** x **where** *Axb*: $A *_v x \leq b$ **and** *xcarr*: $x \in$ *carrier-vec nc* **by** *auto*
have *bmAx*: $b - A *_v x \in$ *carrier-vec nr* **using** A b *xcarr* **by** *simp*
define *x1* **where** $x1 = b - A *_v x$
have *x1*: $x1 \in$ *carrier-vec nr* **using** *bmAx* **unfolding** *x1-def* **by** (*simp add*: *xcarr*)
define *x2* **where** $x2 = vec\ (dim\text{-}vec\ x)\ (\lambda i.\ if\ x\ \$\ i \geq 0\ then\ x\ \$\ i\ else\ 0)$
have *x2*: $x2 \in$ *carrier-vec nc* **using** *xcarr* **unfolding** *x2-def* **by** *simp*
define *x3* **where** $x3 = vec\ (dim\text{-}vec\ x)\ (\lambda i.\ if\ x\ \$\ i < 0\ then\ -x\ \$\ i\ else\ 0)$
have *x3*: $x3 \in$ *carrier-vec nc* **using** *xcarr* **unfolding** *x3-def* **by** *simp*
have *x2x3carr*: $x2\ @_v\ x3 \in$ *carrier-vec* $(nc + nc)$ **using** *x2 x3* **by** *simp*
have *x2x3x*: $x2 - x3 = x$ **unfolding** *x2-def x3-def* **by** *auto*
have $A *_v x - b \leq 0_v\ nr$ **using** *vec-le-iff-diff-le-0 b*
by (*metis A Axb carrier-matD*(*1*) *dim-mult-mat-vec*)
hence *x1lez*: $x1 \geq 0_v\ nr$ **using** *x1* **unfolding** *x1-def*
by (*smt A Axb carrier-matD*(*1*) *carrier-vecD diff-ge-0-iff-ge dim-mult-mat-vec*

index-minus-vec(*1*) *index-zero-vec*(*1*) *index-zero-vec*(*2*) *less-eq-vec-def*)
have *x2lez*: $x2 \geq 0_v\ nc$ **using** *x2 less-eq-vec-def* **unfolding** *x2-def* **by** *fastforce*
have *x3lez*: $x3 \geq 0_v\ nc$ **using** *x3 less-eq-vec-def* **unfolding** *x3-def* **by** *fastforce*
have *B1*: $(1_m\ nr) *_v x1 = b - A *_v x$ **using** *xcarr x1* **unfolding** *x1-def* **by** *simp*
have $A *_v x2 + (-A) *_v x3 = A *_v x2 + A *_v (-x3)$ **using** *x2 x3 A* **by** *auto*
also have $\ldots = A *_v (x2 + (-x3))$ **using** *A x2 x3*
by (*metis mult-add-distrib-mat-vec uminus-carrier-vec*)
also have $\ldots = A *_v x$ **using** *x2x3x minus-add-uminus-vec x2 x3* **by** *fastforce*
finally have *B2*:$A *_v x2 + (-A) *_v x3 = A *_v x$ **by** *auto*
have $B *_v (x1\ @_v\ (x2\ @_v\ x3)) = (1_m\ nr) *_v x1 + (A *_v x2 + (-A) *_v x3)$ (**is** $\ldots = ?p1 + ?p2$)
using *x1 x2 x3 A mat-mult-append-cols* **unfolding** *B-def*
by (*subst mat-mult-append-cols*[*OF - - x1 x2x3carr*], *auto simp add*: *mat-mult-append-cols*)
also have $?p1 = b - A *_v x$ **using** *B1* **unfolding** *x1-def* **by** *auto*
also have $?p2 = A *_v x$ **using** *B2* **by** *simp*
finally have *res*: $B *_v (x1\ @_v\ (x2\ @_v\ x3)) = b$ **using** *A xcarr b* **by** *auto*
show $\exists x \in$ *carrier-vec nc*. $A *_v x \leq b \Longrightarrow \exists x1 \in$ *carrier-vec nr*. $\exists x2 \in$ *carrier-vec nc*. $\exists x3 \in$ *carrier-vec nc*.
$0_v\ nr \leq x1 \wedge 0_v\ nc \leq x2 \wedge 0_v\ nc \leq x3 \wedge B *_v (x1\ @_v\ x2\ @_v\ x3) = b$
using *x1 x2 x3 x1lez x2lez x3lez res* **by** *auto*
next
assume $\exists x1 \in$ *carrier-vec nr*. $\exists x2 \in$ *carrier-vec nc*. $\exists x3 \in$ *carrier-vec nc*.
$x1 \geq 0_v\ nr \wedge x2 \geq 0_v\ nc \wedge x3 \geq 0_v\ nc \wedge B *_v (x1\ @_v\ (x2\ @_v\ x3)) = b$
from *this* **obtain** *x1 x2 x3* **where** *x1*: $x1 \in$ *carrier-vec nr* **and** *x1lez*: $x1 \geq 0_v\ nr$

and *x2*: $x2 \in$ *carrier-vec nc* **and** *x2lez*: $x2 \geq 0_v\ nc$
and *x3*: $x3 \in$ *carrier-vec nc* **and** *x3lez*: $x3 \geq 0_v\ nc$
and *clc*: $B *_v (x1\ @_v\ (x2\ @_v\ x3)) = b$ **by** *auto*
have *x2x3carr*: $x2\ @_v\ x3 \in$ *carrier-vec* $(nc + nc)$ **using** *x2 x3* **by** *simp*
define *x* **where** $x = x2 - x3$
have *xcarr*: $x \in$ *carrier-vec nc* **using** *x2 x3* **unfolding** *x-def* **by** *simp*
have $A *_v x2 + (-A) *_v x3 = A *_v x2 + A *_v (-x3)$ **using** *x2 x3 A* **by** *auto*
also have ... $= A *_v (x2 + (-x3))$ **using** *A x2 x3*
by (*metis mult-add-distrib-mat-vec uminus-carrier-vec*)
also have ... $= A *_v x$ **using** *minus-add-uminus-vec x2 x3* **unfolding** *x-def* **by** *fastforce*
finally have *B2*: $A *_v x2 + (-A) *_v x3 = A *_v x$ **by** *auto*
have *Axcarr*: $A *_v x \in$ *carrier-vec nr* **using** *A xcarr* **by** *auto*
have $b = B *_v (x1\ @_v\ (x2\ @_v\ x3))$ **using** *clc* **by** *auto*
also have ... $= (1_m\ nr) *_v x1 + (A *_v x2 + (-A) *_v x3)$ (**is** ... = *?p1* + *?p2*)
using *x1 x2 x3 A mat-mult-append-cols* **unfolding** *B-def*
by (*subst mat-mult-append-cols*[*OF - - x1 x2x3carr*], *auto simp add*: *mat-mult-append-cols*)
also have *?p2* $= A *_v x$ **using** *B2* **by** *simp*
finally have *res*: $b = (1_m\ nr) *_v x1 + A *_v x$ **using** *A xcarr b* **by** *auto*
hence $b = x1 + A *_v x$ **using** *x1 A b* **by** *simp*
hence $b - A *_v x = x1$ **using** *x1 A b* **by** *auto*
hence $b - A *_v x \geq 0_v\ nr$ **using** *x1lez* **by** *auto*
hence $A *_v x \leq b$ **using** *Axcarr*
by (*smt* ‹$b - A *_v x = x1$› ‹$b = x1 + A *_v x$› *carrier-vecD comm-add-vec index-zero-vec*(*2*)
minus-add-minus-vec minus-cancel-vec vec-le-iff-diff-le-0 x1)
then show ∃ *x1*∈*carrier-vec nr*. ∃ *x2*∈*carrier-vec nc*. ∃ *x3*∈*carrier-vec nc*.
$0_v\ nr \leq x1 \wedge 0_v\ nc \leq x2 \wedge 0_v\ nc \leq x3 \wedge B *_v (x1\ @_v\ x2\ @_v\ x3) = b \Longrightarrow$
∃ *x*∈*carrier-vec nc*. $A *_v x \leq b$ **using** *xcarr* **by** *blast*
qed
also have ... = (∃ *x1* ∈ *carrier-vec nr*. ∃ *x2* ∈ *carrier-vec nc*. ∃ *x3* ∈ *carrier-vec nc*.
$(x1\ @_v\ (x2\ @_v\ x3)) \geq 0_v\ n \wedge B *_v (x1\ @_v\ (x2\ @_v\ x3)) = b$)
by (*metis append-vec-le id0 n-def zero-carrier-vec*)
also have ... = (∃ *x* ∈ *carrier-vec n*. $x \geq 0_v\ n \wedge B *_v x = b$)
unfolding *n-def exists-vec-append* **by** *auto*
also have ... = (∃ $x \geq 0_v\ n$. $B *_v x = b$) **unfolding** *less-eq-vec-def* **by** *fastforce*
also have ... = (∀ *y*. $y \in$ *carrier-vec nr* $\longrightarrow B^T *_v y \geq 0_v\ n \longrightarrow y \cdot b \geq 0$)
by (*rule gram-schmidt.Farkas-Lemma*[*OF B b*])
also have ... = (∀ *y*. $y \in$ *carrier-vec nr* $\longrightarrow (y \geq 0_v\ nr \wedge A^T *_v y = 0_v\ nc) \longrightarrow y \cdot b \geq 0$)
proof (*intro all-cong imp-cong refl*)
fix *y* :: *'a vec*
assume *y*: $y \in$ *carrier-vec nr*
have *idtcarr*: $(1_m\ nr)^T \in$ *carrier-mat nr nr* **by** *auto*
have *Atcarr*: $A^T \in$ *carrier-mat nc nr* **using** *A* **by** *auto*
have *mAtcarr*: $(-A)^T \in$ *carrier-mat nc nr* **using** *A* **by** *auto*

have *AtAtcarr*: A^T $@_r$ $(-A)^T$ $\in$ *carrier-mat* $(nc + nc)$ *nr* **using** *A* **by** *auto*
have $B^T *_v y = ((1_m\ nr)^T\ @_r\ A^T\ @_r\ (-A)^T) *_v y$ **unfolding** *B-def*
by (*simp add*: *append-cols-def*)
also have $\ldots = ((1_m\ nr)^T *_v y)\ @_v\ (A^T *_v y)\ @_v\ ((-A)^T *_v y)$
using *mat-mult-append*[*OF Atcarr mAtcarr y*] *mat-mult-append y Atcarr idtcarr mAtcarr*
by (*metis AtAtcarr*)
finally have *eq*: $B^T *_v y = ((1_m\ nr)^T *_v y)\ @_v\ (A^T *_v y)\ @_v\ ((-A)^T *_v y)$ **by** *auto*
have $(B^T *_v y \geq 0_v\ n) = (0_v\ n \leq (1_m\ nr)^T *_v y\ @_v\ A^T *_v y\ @_v\ (-\ A)^T *_v y)$ **unfolding** *eq* **by** *simp*
also have $\ldots = (((1_m\ nr)^T *_v y)\ @_v\ (A^T *_v y)\ @_v\ ((-A)^T *_v y) \geq 0_v\ nr\ @_v\ 0_v\ nc\ @_v\ 0_v\ nc)$
using *id0* **by** (*metis eq n-def*)
also have $\ldots = (y \geq 0_v\ nr \wedge A^T *_v y \geq 0_v\ nc \wedge ((-A)^T *_v y) \geq 0_v\ nc)$
by (*metis Atcarr append-vec-le mult-mat-vec-carrier one-mult-mat-vec transpose-one y zero-carrier-vec*)
also have $\ldots = (y \geq 0_v\ nr \wedge A^T *_v y \geq 0_v\ nc \wedge -(A^T *_v y) \geq 0_v\ nc)$
by (*metis A Atcarr carrier-matD*(*2*) *carrier-vecD transpose-uminus uminus-mult-mat-vec y*)
also have $\ldots = (y \geq 0_v\ nr \wedge A^T *_v y \geq 0_v\ nc \wedge (A^T *_v y) \leq 0_v\ nc)$
by (*metis* (*mono-tags*, *lifting*) *A Atcarr carrier-matD*(*2*) *carrier-vecD index-zero-vec*(*2*)
mAtcarr mult-mat-vec-carrier transpose-uminus uminus-mult-mat-vec uminus-uminus-vec
vec-le-iff-diff-le-0 y zero-minus-vec)
also have $\ldots = (y \geq 0_v\ nr \wedge A^T *_v y = 0_v\ nc)$ **by** *auto*
finally show $(B^T *_v y \geq 0_v\ n) = (y \geq 0_v\ nr \wedge A^T *_v y = 0_v\ nc)$ **.**
qed
finally show *?thesis* **by** (*auto simp*: *less-eq-vec-def*)
qed

end
end

13 The Theorem of Farkas, Minkowsky and Weyl

We prove the theorem of Farkas, Minkowsky and Weyl that a cone is finitely generated iff it is polyhedral. Moreover, we provide quantative bounds via determinant bounds.

theory *Farkas-Minkowsky-Weyl*
imports *Fundamental-Theorem-Linear-Inequalities*
begin

context *gram-schmidt*
begin

We first prove the one direction of the theorem for the case that the span of the vectors is the full n-dimensional space.

lemma *farkas-minkowsky-weyl-theorem-1-full-dim*:
  assumes *X*: $X \subseteq$ *carrier-vec n*
    and *fin*: *finite X*
    and *span*: *span X = carrier-vec n*
  shows $\exists$ *nr A*. $A \in$ *carrier-mat nr n* $\wedge$ *cone X = polyhedral-cone A*
  $\wedge$ (*is-det-bound db* $\longrightarrow X \subseteq \mathbb{Z}_v \cap$ *Bounded-vec* (*of-int Bnd*) $\longrightarrow A \in \mathbb{Z}_m \cap$ *Bounded-mat* (*of-int* (*db* (*n*−*1*) *Bnd*)))
proof −
  define *cond* **where** *cond* = (λ *W*. *Suc* (*card W*) = *n* $\wedge$ *lin-indpt W* $\wedge$ $W \subseteq X$)
  let *?oi = of-int* :: *int* $\Rightarrow$ *'a*
  {
    fix *W*
    assume *cond W*
    hence ∗: *finite W Suc* (*card W*) = *n lin-indpt W W* $\subseteq$ *carrier-vec n* **and** *WX*: $W \subseteq X$ **unfolding** *cond-def*
      using *finite-subset*[*OF - fin*] *X* **by** *auto*
    note *nv = normal-vector*[*OF* ∗]
    hence *normal-vector W* $\in$ *carrier-vec n* $\bigwedge$ *w*. $w \in W \Longrightarrow$ *normal-vector W* · *w* = *0*
      normal-vector W $\neq 0_v$ *n is-det-bound db* $\Longrightarrow X \subseteq \mathbb{Z}_v \cap$ *Bounded-vec* (*?oi Bnd*) $\Longrightarrow$ *normal-vector W* $\in \mathbb{Z}_v \cap$ *Bounded-vec* (*?oi* (*db* (*n* − *1*) *Bnd*))
      using *WX* **by** *blast*+
  } **note** *condD = this*
  define *Ns* **where** *Ns* = { *normal-vector W* | *W*. *cond W* $\wedge$ ($\forall$ $w \in X$. *normal-vector W* · $w \geq 0$) }
      $\cup$ { − *normal-vector W* | *W*. *cond W* $\wedge$ ($\forall$ $w \in X$. (− *normal-vector W*) · $w \geq 0$)}
  have *Ns* $\subseteq$ *normal-vector* ‘ {*W* . $W \subseteq X$} $\cup$ (λ *W*. − *normal-vector W*) ‘ {*W*. $W \subseteq X$} **unfolding** *Ns-def cond-def* **by** *blast*
  moreover have *finite* ... **using** ‹*finite X*› **by** *auto*
  ultimately have *finite Ns* **by** (*metis finite-subset*)
  from *finite-list*[*OF this*] **obtain** *ns* **where** *ns*: *set ns = Ns* **by** *auto*
  have *Ns*: *Ns* $\subseteq$ *carrier-vec n* **unfolding** *Ns-def* **using** *condD* **by** *auto*
  define *A* **where** *A = mat-of-rows n ns*
  define *nr* **where** *nr = length ns*
  have *A*: − $A \in$ *carrier-mat nr n* **unfolding** *A-def nr-def* **by** *auto*
  show *?thesis*
  proof (*intro exI conjI impI*, *rule A*)
    have *not-conj*: ¬ ($a \wedge b$) $\longleftrightarrow$ ($a \longrightarrow$ ¬ b) **for** *a b* **by** *auto*
    have *id*: *Ns* = { *nv* . $\exists$ *W*. $W \subseteq X \wedge nv \in$ {*normal-vector W*, − *normal-vector W*} $\wedge$
          Suc (*card W*) = *n* $\wedge$ *lin-indpt W* $\wedge$ ($\forall$ $a_i \in X$. *0* $\leq$ *nv* · a_i)}
      unfolding *Ns-def cond-def* **by** *auto*
    have *polyhedral-cone* (− *A*) = { *b*. $b \in$ *carrier-vec n* $\wedge$ (− *A*) $*_v$ *b* $\leq 0_v$ *nr*} **unfolding** *polyhedral-cone-def*
      using *A* **by** *auto*
    also have ... = {*b*. $b \in$ *carrier-vec n* $\wedge$ ($\forall$ *i* < *nr*. *row* (− *A*) *i* · *b* $\leq$ *0*)}
      unfolding *less-eq-vec-def* **using** *A* **by** *auto*

also have . . . = {*b*. *b* ∈ *carrier-vec n* ∧ (∀ *i* < *nr*. − (*ns* ! *i*) · *b* ≤ *0*)} **using** *A Ns*[*folded ns*]
by (*intro Collect-cong conj-cong refl all-cong arg-cong*[*of - - λ x. x · - ≤ -*], *force simp*: *A-def mat-of-rows-def nr-def set-conv-nth*)
also have . . . = {*b*. *b* ∈ *carrier-vec n* ∧ (∀ *n* ∈ *Ns*. − *n* · *b* ≤ *0*)}
unfolding *ns*[*symmetric*] *nr-def* **by** (*auto simp*: *set-conv-nth*)
also have . . . = {*b*. *b* ∈ *carrier-vec n* ∧ (∀ *n* ∈ *Ns*. *n* · *b* ≥ *0*)}
by (*intro Collect-cong conj-cong refl ball-cong*, *insert Ns*, *auto*)
also have . . . = *cone X*
unfolding *fundamental-theorem-of-linear-inequalities-full-dim*(*2*)[*OF X fin span*]
by (*intro Collect-cong conj-cong refl*, *unfold not-le*[*symmetric*] *not-ex not-conj not-not id*, *blast*)
finally show *cone X* = *polyhedral-cone* (− *A*) **..**
{
assume *XI*: *X* ⊆ $\mathbb{Z}_v$ ∩ *Bounded-vec* (*?oi Bnd*) **and** *db*: *is-det-bound db*
{
fix *v*
assume *v* ∈ *set* (*rows* (− *A*))
hence −*v* ∈ *set* (*rows A*) **unfolding** *rows-def* **by** *auto*
hence −*v* ∈ *Ns* **unfolding** *A-def* **using** *ns Ns* **by** *auto*
from *this*[*unfolded Ns-def*] **obtain** *W* **where** *cW*: *cond W*
and *v*: −*v* = *normal-vector W* ∨ *v* = *normal-vector W* **by** *auto*
from *cW*[*unfolded cond-def*] **have** *WX*: *W* ⊆ *X* **by** *auto*
from *v* **have** *v*: *v* = *normal-vector W* ∨ *v* = − *normal-vector W*
by (*metis uminus-uminus-vec*)
from *condD*(*4*)[*OF cW db XI*]
have *normal-vector W* ∈ $\mathbb{Z}_v$ ∩ *Bounded-vec* (*?oi* (*db* (*n* − *1*) *Bnd*)) **by** *auto*
hence *v* ∈ $\mathbb{Z}_v$ ∩ *Bounded-vec* (*?oi* (*db* (*n* − *1*) *Bnd*)) **using** *v* **by** *auto*
}
hence *set* (*rows* (− *A*)) ⊆ $\mathbb{Z}_v$ ∩ *Bounded-vec* (*?oi* (*db* (*n* − *1*) *Bnd*)) **by** *blast*
thus − *A* ∈ $\mathbb{Z}_m$ ∩ *Bounded-mat* (*?oi* (*db* (*n* − *1*) *Bnd*)) **by** *simp*
}
qed
qed

We next generalize the theorem to the case where X does not span the full space. To this end, we extend X by unit-vectors until the full space is spanned, and then add the normal-vectors of these unit-vectors which are orthogonal to span X as additional constraints to the resulting matrix.

lemma *farkas-minkowsky-weyl-theorem-1*:
assumes *X*: *X* ⊆ *carrier-vec n*
and *finX*: *finite X*
shows ∃ *nr A*. *A* ∈ *carrier-mat nr n* ∧ *cone X* = *polyhedral-cone A* ∧
(*is-det-bound db* ⟶ *X* ⊆ $\mathbb{Z}_v$ ∩ *Bounded-vec* (*of-int Bnd*) ⟶ *A* ∈ $\mathbb{Z}_m$ ∩ *Bounded-mat* (*of-int* (*db* (*n*−*1*) (*max 1 Bnd*))))
proof −
let *?oi* = *of-int* :: *int* ⇒ ′*a*

from *exists-lin-indpt-sublist*[*OF X*]
obtain *Ls* **where** *lin-Ls*: *lin-indpt-list Ls* **and**
spanL: *span* (*set Ls*) = *span X* **and** *LX*: *set Ls* ⊆ *X* **by** *auto*
define *Ns* **where** *Ns* = *normal-vectors Ls*
define *Bs* **where** *Bs* = *basis-extension Ls*
from *basis-extension*[*OF lin-Ls*, *folded Bs-def*]
have *BU*: *set Bs* ⊆ *set* (*unit-vecs n*)
and *lin-Ls-Bs*: *lin-indpt-list* (*Ls* @ *Bs*)
and *len-Ls-Bs*: *length* (*Ls* @ *Bs*) = *n*
by *auto*
note *nv* = *normal-vectors*[*OF lin-Ls*, *folded Ns-def*]
from *nv*(*1*−*6*) *nv*(*7*)[*of db Bnd*]
have *N*: *set Ns* ⊆ *carrier-vec n*
and *LN'*: *lin-indpt-list* (*Ls* @ *Ns*) *length* (*Ls* @ *Ns*) = *n*
and *ortho*: ⋀ *l w*. *l* ∈ *set Ls* ⟹ *w* ∈ *set Ns* ⟹ *w* · *l* = *0*
and *Ns-bnd*: *is-det-bound db* ⟹ *set Ls* ⊆ $\mathbb{Z}_v$ ∩ *Bounded-vec* (*?oi Bnd*)
⟹ *set Ns* ⊆ $\mathbb{Z}_v$ ∩ *Bounded-vec* (*?oi* (*db* (*n*−*1*) (*max 1 Bnd*)))
by *auto*
from *lin-indpt-list-length-eq-n*[*OF LN'*]
have *spanLN*: *span* (*set Ls* ∪ *set Ns*) = *carrier-vec n* **by** *auto*
let *?Bnd* = *Bounded-vec* (*?oi* (*db* (*n*−*1*) (*max 1 Bnd*)))
let *?Bndm* = *Bounded-mat* (*?oi* (*db* (*n*−*1*) (*max 1 Bnd*)))
define *Y* **where** *Y* = *X* ∪ *set Bs*
from *lin-Ls-Bs*[*unfolded lin-indpt-list-def*] **have**
Ls: *set Ls* ⊆ *carrier-vec n* **and**
Bs: *set Bs* ⊆ *carrier-vec n* **and**
distLsBs: *distinct* (*Ls* @ *Bs*) **and**
lin': *lin-indpt* (*set* (*Ls* @ *Bs*)) **by** *auto*
have *LN*: *set Ls* ∩ *set Ns* = {}
proof (*rule ccontr*)
assume ¬ *?thesis*
then obtain *x* **where** *xX*: *x* ∈ *set Ls* **and** *xW*: *x* ∈ *set Ns* **by** *auto*
from *ortho*[*OF xX xW*] **have** *x* · *x* = *0* **by** *auto*
hence *sq-norm x* = *0* **by** (*auto simp*: *sq-norm-vec-as-cscalar-prod*)
with *xX LX X* **have** *x* = 0_v *n* **by** *auto*
with *vs-zero-lin-dep*[*OF - lin'*] *Ls Bs xX* **show** *False* **by** *auto*
qed
have *Y*: *Y* ⊆ *carrier-vec n* **using** *X Bs* **unfolding** *Y-def* **by** *auto*
have *CLB*: *carrier-vec n* = *span* (*set* (*Ls* @ *Bs*))
using *lin-Ls-Bs len-Ls-Bs lin-indpt-list-length-eq-n* **by** *blast*
also have ... ⊆ *span Y*
by (*rule span-is-monotone*, *insert LX*, *auto simp*: *Y-def*)
finally have *span*: *span Y* = *carrier-vec n* **using** *Y* **by** *auto*
have *finY*: *finite Y* **using** *finX* **unfolding** *Y-def* **by** *auto*
from *farkas-minkowsky-weyl-theorem-1-full-dim*[*OF Y finY span*]
obtain *A nr* **where** *A*: *A* ∈ *carrier-mat nr n* **and** *YA*: *cone Y* = *polyhedral-cone A*
and *Y-Ints*: *is-det-bound db* ⟹ *Y* ⊆ $\mathbb{Z}_v$ ∩ *Bounded-vec* (*?oi* (*max 1 Bnd*))
⟹ *A* ∈ $\mathbb{Z}_m$ ∩ *?Bndm* **by** *blast*

have *fin*: *finite* ({*row A i* | *i*. *i* < *nr*} ∪ *set Ns* ∪ *uminus* ' *set Ns*) **by** *auto*
from *finite-list*[*OF this*] **obtain** *rs* **where** *rs-def*: *set rs* = {*row A i* |*i*. *i* < *nr*} ∪ *set Ns* ∪ *uminus* ' *set Ns* **by** *auto*
from *A N* **have** *rs*: *set rs* ⊆ *carrier-vec n* **unfolding** *rs-def* **by** *auto*
let *?m* = *length rs*
define *B* **where** *B* = *mat-of-rows n rs*
have *B*: *B* ∈ *carrier-mat ?m n* **unfolding** *B-def* **by** *auto*
show *?thesis*
proof (*intro exI conjI impI*, *rule B*)
have *id*: (∀ *r*∈{*rs* ! *i* |*i*. *i* < *?m*}. *P r*) = (∀ *r* < *?m*. *P* (*rs* ! *r*)) **for** *P* **by** *auto*
have *polyhedral-cone B* = { *x* ∈ *carrier-vec n*. *B* $*_v$ *x* ≤ 0_v *?m*} **unfolding** *polyhedral-cone-def*
using *B* **by** *auto*
also have ... = { *x* ∈ *carrier-vec n*. ∀ *i* < *?m*. *row B i* · *x* ≤ *0*}
unfolding *less-eq-vec-def* **using** *B* **by** *auto*
also have ... = { *x* ∈ *carrier-vec n*. ∀ *r* ∈ *set rs*. *r* · *x* ≤ *0*} **using** *rs* **unfolding** *set-conv-nth id*
by (*intro Collect-cong conj-cong refl all-cong arg-cong*[*of - - λ x*. *x* · - ≤ *0*], *auto simp*: *B-def*)
also have ... = {*x* ∈ *carrier-vec n*. ∀ *i* < *nr*. *row A i* · *x* ≤ *0*}
∩ {*x* ∈ *carrier-vec n*. ∀ *w* ∈ *set Ns* ∪ *uminus* ' *set Ns*. *w* · *x* ≤ *0*}
unfolding *rs-def* **by** *blast*
also have {*x* ∈ *carrier-vec n*. ∀ *i* < *nr*. *row A i* · *x* ≤ *0*} = *polyhedral-cone A*
unfolding *polyhedral-cone-def* **using** *A* **by** (*auto simp*: *less-eq-vec-def*)
also have ... = *cone Y* **unfolding** *YA* **..**
also have {*x* ∈ *carrier-vec n*. ∀ *w* ∈ *set Ns* ∪ *uminus* ' *set Ns*. *w* · *x* ≤ *0*}
= {*x* ∈ *carrier-vec n*. ∀ *w* ∈ *set Ns*. *w* · *x* = *0*}
(**is** *?l* = *?r*)
proof
show *?r* ⊆ *?l* **using** *N* **by** *auto*
{
fix *x w*
assume *x* ∈ *?l w* ∈ *set Ns*
with *N* **have** *x*: *x* ∈ *carrier-vec n* **and** *w*: *w* ∈ *carrier-vec n*
and *one*: *w* · *x* ≤ *0* **and** *two*: (−*w*) · *x* ≤ *0* **by** *auto*
from *two* **have** *w* · *x* ≥ *0*
by (*subst* (*asm*) *scalar-prod-uminus-left*, *insert w x*, *auto*)
with *one* **have** *w* · *x* = *0* **by** *auto*
}
thus *?l* ⊆ *?r* **by** *blast*
qed
finally have *polyhedral-cone B* = *cone Y* ∩ {*x* ∈ *carrier-vec n*. ∀ *w*∈*set Ns*. *w* · *x* = *0*} **.**
also have ... = *cone X* **unfolding** *Y-def*
by (*rule orthogonal-cone*(*1*)[*OF X N finX spanLN ortho refl spanL LX lin-Ls-Bs len-Ls-Bs*])
finally show *cone X* = *polyhedral-cone B* **..**
assume *X-I*: *X* ⊆ $\mathbb{Z}_v$ ∩ *Bounded-vec* (*?oi Bnd*) **and** *db*: *is-det-bound db*
with *LX* **have** *set Ls* ⊆ $\mathbb{Z}_v$ ∩ *Bounded-vec* (*?oi Bnd*) **by** *auto*

from *Ns-bnd*[*OF db this*] **have** *N-I-Bnd*: *set Ns* ⊆ $\mathbb{Z}_v$ ∩ *?Bnd* **by** *auto*
from *lin-Ls-Bs* **have** *linLs*: *lin-indpt-list Ls* **unfolding** *lin-indpt-list-def*
using *subset-li-is-li*[*of - set Ls*] **by** *auto*
from *X-I LX* **have** *L-I*: *set Ls* ⊆ $\mathbb{Z}_v$ **by** *auto*
have *Y-I*: *Y* ⊆ $\mathbb{Z}_v$ ∩ *Bounded-vec* (*?oi* (*max 1 Bnd*)) **unfolding** *Y-def* **using** *X-I order.trans*[*OF BU unit-vec-int-bounds*, *of Bnd*]
Bounded-vec-mono[*of ?oi Bnd ?oi* (*max 1 Bnd*)] **by** *auto*
from *Y-Ints*[*OF db Y-I*]
have *A-I-Bnd*: *set* (*rows A*) ⊆ $\mathbb{Z}_v$ ∩ *?Bnd* **by** *auto*
have *set* (*rows B*) = *set* (*rows* (*mat-of-rows n rs*)) **unfolding** *B-def* **by** *auto*
also have ... = *set rs* **using** *rs* **by** *auto*
also have ... = *set* (*rows A*) ∪ *set Ns* ∪ *uminus* ' *set Ns* **unfolding** *rs-def rows-def* **using** *A* **by** *auto*
also have ... ⊆ $\mathbb{Z}_v$ ∩ *?Bnd* **using** *A-I-Bnd N-I-Bnd* **by** *auto*
finally show *B* ∈ $\mathbb{Z}_m$ ∩ *?Bndm* **by** *simp*
qed
qed

Now for the other direction.

lemma *farkas-minkowsky-weyl-theorem-2*:
assumes *A*: *A* ∈ *carrier-mat nr n*
shows ∃ *X*. *X* ⊆ *carrier-vec n* ∧ *finite X* ∧ *polyhedral-cone A* = *cone X*
∧ (*is-det-bound db* ⟶ *A* ∈ $\mathbb{Z}_m$ ∩ *Bounded-mat* (*of-int Bnd*) ⟶ *X* ⊆ $\mathbb{Z}_v$ ∩ *Bounded-vec* (*of-int* (*db* (*n*−*1*) (*max 1 Bnd*))))
proof −
let *?oi* = *of-int* :: *int* ⇒ *'a*
let *?rows-A* = {*row A i* | *i*. *i* < *nr*}
let *?Bnd* = *Bounded-vec* (*?oi* (*db* (*n*−*1*) (*max 1 Bnd*)))
have *rows-A-n*: *?rows-A* ⊆ *carrier-vec n* **using** *row-carrier-vec A* **by** *auto*
hence ∃ *mr B*. *B* ∈ *carrier-mat mr n* ∧ *cone ?rows-A* = *polyhedral-cone B*
∧ (*is-det-bound db* ⟶ *?rows-A* ⊆ $\mathbb{Z}_v$ ∩ *Bounded-vec* (*?oi Bnd*) ⟶ *set* (*rows B*) ⊆ $\mathbb{Z}_v$ ∩ *?Bnd*)
using *farkas-minkowsky-weyl-theorem-1*[*of ?rows-A*] **by** *auto*
then obtain *mr B*
where *mr*: *B* ∈ *carrier-mat mr n* **and** *B*: *cone ?rows-A* = *polyhedral-cone B*
and *Bnd*: *is-det-bound db* ⟹ *?rows-A* ⊆ $\mathbb{Z}_v$ ∩ *Bounded-vec* (*?oi Bnd*) ⟹ *set* (*rows B*) ⊆ $\mathbb{Z}_v$ ∩ *?Bnd*
by *blast*
let *?rows-B* = {*row B i* | *i*. *i* < *mr*}
have *rows-B*: *?rows-B* ⊆ *carrier-vec n* **using** *mr* **by** *auto*
have *cone ?rows-B* = *polyhedral-cone A*
proof
have *?rows-B* ⊆ *polyhedral-cone A*
proof
fix *r*
assume *r* ∈ *?rows-B*
then obtain *j* **where** *r*: *r* = *row B j* **and** *j*: *j* < *mr* **by** *auto*
then have *rn*: *r* ∈ *carrier-vec n* **using** *mr row-carrier* **by** *auto*
moreover have *A* $*_v$ *r* ≤ 0_v *nr* **unfolding** *less-eq-vec-def*

```
      proof (standard, unfold index-zero-vec)
        show dim-vec (A *v r) = nr using A by auto
      next
        show ∀ i< nr. (A *v r) $ i ≤ 0v nr $ i
        proof (standard, rule impI)
          fix i
          assume i: i < nr
          then have row A i ∈ ?rows-A by auto
          then have row A i ∈ cone ?rows-A
            using set-in-cone rows-A-n by blast
          then have row A i ∈ polyhedral-cone B using B by auto
          then have Br: B *v (row A i) ≤ 0v mr
            unfolding polyhedral-cone-def using rows-A-n mr by auto

          then have (A *v r) $ i = (row A i) · r using A i index-mult-mat-vec by
auto
          also have ... = r · (row A i)
            using comm-scalar-prod[OF - rn] row-carrier A by auto
          also have ... = (row B j) · (row A i) using r by auto
          also have ... = (B *v (row A i)) $ j using index-mult-mat-vec mr j by
auto
          also have ... ≤ 0 using Br j unfolding less-eq-vec-def by auto
          also have ... = 0v nr $ i using i by auto
          finally show (A *v r) $ i ≤ 0v nr $ i by auto
        qed
      qed
      then show r ∈ polyhedral-cone A
        unfolding polyhedral-cone-def
        using A rn by auto
    qed
    then show cone ?rows-B ⊆ polyhedral-cone A
      using cone-in-polyhedral-cone A by auto

  next

    show polyhedral-cone A ⊆ cone ?rows-B
    proof (rule ccontr)
      assume ¬ polyhedral-cone A ⊆ cone ?rows-B
      then obtain y where yA: y ∈ polyhedral-cone A
        and yB: y ∉ cone ?rows-B by auto
      then have yn: y ∈ carrier-vec n unfolding polyhedral-cone-def by auto
      have finRB: finite ?rows-B by auto
      from farkas-minkowsky-weyl-theorem-1[OF rows-B finRB]
      obtain nr' A' where A': A' ∈ carrier-mat nr' n and cone: cone ?rows-B =
polyhedral-cone A'
        by blast
      from yB[unfolded cone polyhedral-cone-def] yn A'
      have ¬ (A' *v y ≤ 0v nr') by auto
      then obtain i where i: i < nr' and row A' i · y > 0
```

unfolding *less-eq-vec-def* **using** *A′ yn* **by** *auto*
define *w* **where** *w* = *row A′ i*
have *w*: *w* ∈ *carrier-vec n* **using** *i A′ yn* **unfolding** *w-def* **by** *auto*
from ‹*row A′ i* · *y* > *0*› *comm-scalar-prod*[*OF w yn*] **have** *wy*: *w* · *y* > *0 y* · *w* > *0* **unfolding** *w-def* **by** *auto*
{
fix *b*
assume *b* ∈ *?rows-B*
hence *b* ∈ *cone ?rows-B* **using** *set-in-cone*[*OF rows-B*] **by** *auto*
from *this*[*unfolded cone polyhedral-cone-def*] *A′*
have *b*: *b* ∈ *carrier-vec n* **and** $A' *_v\ b \leq 0_v\ nr'$ **by** *auto*
from *this*(*2*)[*unfolded less-eq-vec-def*, *THEN conjunct2*, *rule-format*, *of i*]
have *w* · *b* ≤ *0* **unfolding** *w-def* **using** *i A′* **by** *auto*
hence *b* · *w* ≤ *0* **using** *comm-scalar-prod*[*OF b w*] **by** *auto*
}
hence *wA*: *w* ∈ *cone ?rows-A* **unfolding** *B polyhedral-cone-def* **using** *mr w*
by (*auto simp*: *less-eq-vec-def*)
from *wy* **have** *yw*: −*y* · *w* < *0*
by (*subst scalar-prod-uminus-left*, *insert yn w*, *auto*)
have *?rows-A* ⊆ *carrier-vec n finite ?rows-A* **using** *assms* **by** *auto*
from *fundamental-theorem-of-linear-inequalities-A-imp-D*[*OF this wA*, *unfolded not-ex*,
rule-format, *of* −*y*] *yn yw*
obtain *i* **where** *i*: *i* < *nr* **and** − *y* · *row A i* < *0* **by** *auto*
hence *y* · *row A i* > *0* **by** (*subst* (*asm*) *scalar-prod-uminus-left*, *insert i assms yn*, *auto*)
hence *row A i* · *y* > *0* **using** *comm-scalar-prod*[*OF - yn*, *of row A i*] *i assms yn* **by** *auto*
with *yA* **show** *False* **unfolding** *polyhedral-cone-def less-eq-vec-def* **using** *i assms* **by** *auto*
qed
qed
moreover have *?rows-B* ⊆ *carrier-vec n*
using *row-carrier-vec mr* **by** *auto*
moreover have *finite ?rows-B* **by** *auto*
moreover {
have *rA*: *set* (*rows A*) = *?rows-A* **using** *A* **unfolding** *rows-def* **by** *auto*
have *rB*: *set* (*rows B*) = *?rows-B* **using** *mr* **unfolding** *rows-def* **by** *auto*
assume $A \in \mathbb{Z}_m$ ∩ *Bounded-mat* (*?oi Bnd*) **and** *db*: *is-det-bound db*
hence *set* (*rows A*) ⊆ $\mathbb{Z}_v$ ∩ *Bounded-vec* (*?oi Bnd*) **by** *simp*
from *Bnd*[*OF db this*[*unfolded rA*]]
have *?rows-B* ⊆ $\mathbb{Z}_v$ ∩ *?Bnd* **unfolding** *rA rB* **.**
}
ultimately show *?thesis* **by** *blast*
qed

lemma *farkas-minkowsky-weyl-theorem*:
(∃ *X*. *X* ⊆ *carrier-vec n* ∧ *finite X* ∧ *P* = *cone X*)
⟷ (∃ *A nr*. *A* ∈ *carrier-mat nr n* ∧ *P* = *polyhedral-cone A*)

using *farkas-minkowsky-weyl-theorem-1 farkas-minkowsky-weyl-theorem-2* **by** *metis*
end
end

14 The Decomposition Theorem

This theory contains a proof of the fact, that every polyhedron can be decomposed into a convex hull of a finite set of points + a finitely generated cone, including bounds on the numbers that are required in the decomposition. We further prove the inverse direction of this theorem (without bounds) and as a corollary, we derive that a polyhedron is bounded iff it is the convex hull of finitely many points, i.e., a polytope.

theory *Decomposition-Theorem*
  imports
    Farkas-Minkowsky-Weyl
    Convex-Hull
begin

context *gram-schmidt*
begin

definition *polytope* $P = (\exists\ V.\ V \subseteq carrier\text{-}vec\ n \wedge finite\ V \wedge P = convex\text{-}hull\ V)$

definition *polyhedron* $A\ b = \{x \in carrier\text{-}vec\ n.\ A *_v x \leq b\}$

lemma *polyhedra-are-convex*:
  assumes *A*: $A \in carrier\text{-}mat\ nr\ n$
    and *b*: $b \in carrier\text{-}vec\ nr$
    and *P*: $P = polyhedron\ A\ b$
  shows *convex P*
proof (*intro convexI*)
  show *Pcarr*: $P \subseteq carrier\text{-}vec\ n$ **using** *assms* **unfolding** *polyhedron-def* **by** *auto*
  fix $a :: 'a$ **and** *x y*
  assume *xy*: $x \in P\ y \in P$ **and** *a*: $0 \leq a\ a \leq 1$
  from *xy*[*unfolded P polyhedron-def*]
  have *x*: $x \in carrier\text{-}vec\ n$ **and** *y*: $y \in carrier\text{-}vec\ n$ **and** *le*: $A *_v x \leq b\ A *_v y \leq b$ **by** *auto*
  show $a \cdot_v x + (1 - a) \cdot_v y \in P$ **unfolding** *P polyhedron-def*
  proof (*intro CollectI conjI*)
    from *x* **have** *ax*: $a \cdot_v x \in carrier\text{-}vec\ n$ **by** *auto*
    from *y* **have** *ay*: $(1 - a) \cdot_v y \in carrier\text{-}vec\ n$ **by** *auto*
    show $a \cdot_v x + (1 - a) \cdot_v y \in carrier\text{-}vec\ n$ **using** *ax ay* **by** *auto*
    show $A *_v (a \cdot_v x + (1 - a) \cdot_v y) \leq b$
    proof (*intro lesseq-vecI*[*OF - b*])
      show $A *_v (a \cdot_v x + (1 - a) \cdot_v y) \in carrier\text{-}vec\ nr$ **using** *A x y* **by** *auto*
      fix *i*
      assume *i*: $i < nr$

from *lesseq-vecD*[*OF b le*(*1*) *i*] *lesseq-vecD*[*OF b le*(*2*) *i*]
have *le*: $(A *_v x)$ \$ $i \leq b$ \$ i $(A *_v y)$ \$ $i \leq b$ \$ i **by** *auto*
have $(A *_v (a \cdot_v x + (1 - a) \cdot_v y))$ \$ $i = a * (A *_v x)$ \$ $i + (1 - a) * (A *_v y)$ \$ i
using *A x y i* **by** (*auto simp*: *scalar-prod-add-distrib*[*of - n*])
also have $\ldots \leq a * b$ \$ $i + (1 - a) * b$ \$ i
by (*rule add-mono*; *rule mult-left-mono*, *insert le a*, *auto*)
also have $\ldots = b$ \$ i **by** (*auto simp*: *field-simps*)
finally show $(A *_v (a \cdot_v x + (1 - a) \cdot_v y))$ \$ $i \leq b$ \$ i **.**
qed
qed
qed

end

locale *gram-schmidt-m* = *n*: *gram-schmidt n f-ty* + *m*: *gram-schmidt m f-ty*
for *n m* :: *nat* **and** *f-ty*
begin

lemma *vec-first-lincomb-list*:
assumes *Xs*: *set Xs* ⊆ *carrier-vec n*
and *nm*: $m \leq n$
shows *vec-first* (*n.lincomb-list c Xs*) *m* =
m.lincomb-list c (*map* (λ *v*. *vec-first v m*) *Xs*)
using *Xs*
proof (*induction Xs arbitrary*: *c*)
case *Nil*
show *?case* **by** (*simp add*: *nm*)
next
case (*Cons x Xs*)
from *Cons.prems* **have** *x*: *x* ∈ *carrier-vec n* **and** *Xs*: *set Xs* ⊆ *carrier-vec n* **by** *auto*

have *vec-first* (*n.lincomb-list c* (*x* # *Xs*)) *m* =
vec-first ($c\ 0 \cdot_v x$ + *n.lincomb-list* (*c* ∘ *Suc*) *Xs*) *m* **by** *auto*
also have … = *vec-first* ($c\ 0 \cdot_v x$) *m* + *vec-first* (*n.lincomb-list* (*c* ∘ *Suc*) *Xs*) *m*
using *vec-first-add*[*of m* $c\ 0 \cdot_v x$] *x n.lincomb-list-carrier*[*OF Xs*, *of c* ∘ *Suc*] *nm*
by *simp*
also have *vec-first* ($c\ 0 \cdot_v x$) *m* = $c\ 0 \cdot_v$ *vec-first x m*
using *vec-first-smult*[*OF nm*, *of x c 0*] *Cons.prems* **by** *auto*
also have *vec-first* (*n.lincomb-list* (*c* ∘ *Suc*) *Xs*) *m* =
m.lincomb-list (*c* ∘ *Suc*) (*map* (λ *v*. *vec-first v m*) *Xs*)
using *Cons* **by** *simp*
also have $c\ 0 \cdot_v$ *vec-first x m* + … =
m.lincomb-list c (*map* (λ *v*. *vec-first v m*) (*x* # *Xs*))

```
    by simp
  finally show ?case by auto
qed

lemma convex-hull-next-dim:
  assumes n = m + 1
    and X: X ⊆ carrier-vec n
    and finite X
    and Xm1: ∀ y ∈ X. y $ m = 1
    and y-dim: y ∈ carrier-vec n
    and y: y $ m = 1
  shows (vec-first y m ∈ m.convex-hull {vec-first y m | y. y ∈ X}) = (y ∈ n.cone
X)
proof −
  from ‹finite X› obtain Xs where Xs: X = set Xs using finite-list by auto
  let ?Y = {vec-first y m | y. y ∈ X}
  let ?Ys = map (λ y. vec-first y m) Xs
  have Ys: ?Y = set ?Ys using Xs by auto

  define x where x = vec-first y m
  {
    have y = vec-first y m @v vec-last y 1
      using ‹n = m + 1› vec-first-last-append y-dim by auto
    also have vec-last y 1 = vec-of-scal (vec-last y 1 $ 0)
      using vec-of-scal-dim-1[of vec-last y 1] by simp
    also have vec-last y 1 $ 0 = y $ m
      using y-dim ‹n = m + 1› vec-last-index[of y m 1 0] by auto
    finally have y = x @v vec-of-scal 1 unfolding x-def using y by simp
  } note xy = this
  {
    assume y ∈ n.cone X
    then obtain c where x: n.nonneg-lincomb c X y
      using n.cone-iff-finite-cone[OF X] ‹finite X›
      unfolding n.finite-cone-def by auto

    have 1 = y $ m by (simp add: y)
    also have y = n.lincomb c X
      using x unfolding n.nonneg-lincomb-def by simp
    also have ... $ m = (∑x∈X. c x * x $ m)
      using n.lincomb-index[OF - X] ‹n = m + 1› by simp
    also have ... = sum c X
      by (rule n.R.finsum-restrict, auto, rule restrict-ext, simp add: Xm1)
    finally have y ∈ n.convex-hull X
      unfolding n.convex-hull-def n.convex-lincomb-def
      using ‹finite X› x by auto
  }
  moreover have n.convex-hull X ⊆ n.cone X
    unfolding n.convex-hull-def n.convex-lincomb-def n.finite-cone-def n.cone-def
    using ‹finite X› by auto
```

```
moreover have n.convex-hull X = n.convex-hull-list Xs
  by (rule n.finite-convex-hull-iff-convex-hull-list[OF X Xs])
moreover {
  assume y ∈ n.convex-hull-list Xs
  then obtain c where c: n.lincomb-list c Xs = y
    and c0: ∀ i < length Xs. c i ≥ 0 and c1: sum c {0..<length Xs} = 1
    unfolding n.convex-hull-list-def n.convex-lincomb-list-def
      n.nonneg-lincomb-list-def by fast
  have m.lincomb-list c ?Ys = vec-first y m
    using c vec-first-lincomb-list[of Xs c] X Xs ‹n = m + 1› by simp
  hence x ∈ m.convex-hull-list ?Ys
    unfolding m.convex-hull-list-def m.convex-lincomb-list-def
      m.nonneg-lincomb-list-def
    using x-def c0 c1 x-def by auto
} moreover {
  assume x ∈ m.convex-hull-list ?Ys
  then obtain c where x: m.lincomb-list c ?Ys = x
    and c0: ∀ i < length Xs. c i ≥ 0
    and c1: sum c {0..<length Xs} = 1
    unfolding m.convex-hull-list-def m.convex-lincomb-list-def
      m.nonneg-lincomb-list-def by auto

  have n.lincomb-list c Xs $ m = (∑ j = 0..<length Xs. c j ∗ Xs ! j $ m)
    using n.lincomb-list-index[of m Xs c] ‹n = m + 1› Xs X by fastforce
  also have ... = sum c {0..<length Xs}
    apply(rule n.R.finsum-restrict, auto, rule restrict-ext)
    by (simp add: Xm1 Xs)
  also have ... = 1 by (rule c1)
  finally have vec-last (n.lincomb-list c Xs) 1 $ 0 = 1
    using vec-of-scal-dim-1 vec-last-index[of n.lincomb-list c Xs m 1 0]
      n.lincomb-list-carrier Xs X ‹n = m + 1› by simp
  hence vec-last (n.lincomb-list c Xs) 1 = vec-of-scal 1
    using vec-of-scal-dim-1 by auto

  moreover have vec-first (n.lincomb-list c Xs) m = x
    using vec-first-lincomb-list ‹n = m + 1› Xs X x by auto

  moreover have n.lincomb-list c Xs =
              vec-first (n.lincomb-list c Xs) m @v vec-last (n.lincomb-list c Xs) 1
    using vec-first-last-append Xs X n.lincomb-list-carrier ‹n = m + 1› by auto

  ultimately have n.lincomb-list c Xs = y using xy by simp

  hence y ∈ n.convex-hull-list Xs
    unfolding n.convex-hull-list-def n.convex-lincomb-list-def
      n.nonneg-lincomb-list-def using c0 c1 by blast
}
moreover have m.convex-hull ?Y = m.convex-hull-list ?Ys
  using m.finite-convex-hull-iff-convex-hull-list[OF - Ys] by fastforce
```

```
  ultimately show ?thesis unfolding x-def by blast
qed

lemma cone-next-dim:
  assumes n = m + 1
    and X: X ⊆ carrier-vec n
    and finite X
    and Xm0: ∀ y ∈ X. y $ m = 0
    and y-dim: y ∈ carrier-vec n
    and y: y $ m = 0
  shows (vec-first y m ∈ m.cone {vec-first y m | y. y ∈ X}) = (y ∈ n.cone X)
proof –
  from ‹finite X› obtain Xs where Xs: X = set Xs using finite-list by auto
  let ?Y = {vec-first y m | y. y ∈ X}
  let ?Ys = map (λ y. vec-first y m) Xs
  have Ys: ?Y = set ?Ys using Xs by auto

  define x where x = vec-first y m
  {
    have y = vec-first y m @v vec-last y 1
      using ‹n = m + 1› vec-first-last-append y-dim by auto
    also have vec-last y 1 = vec-of-scal (vec-last y 1 $ 0)
      using vec-of-scal-dim-1[of vec-last y 1] by simp
    also have vec-last y 1 $ 0 = y $ m
      using y-dim ‹n = m + 1› vec-last-index[of y m 1 0] by auto
    finally have y = x @v vec-of-scal 0 unfolding x-def using y by simp
  } note xy = this

  have n.cone X = n.cone-list Xs
     using n.cone-iff-finite-cone[OF X ‹finite X›] n.finite-cone-iff-cone-list[OF X
Xs]
    by simp
  moreover {
    assume y ∈ n.cone-list Xs
    then obtain c where y: n.lincomb-list c Xs = y and c: ∀ i < length Xs. c i
≥ 0
      unfolding n.cone-list-def n.nonneg-lincomb-list-def by blast
    from y have m.lincomb-list c ?Ys = x
      unfolding x-def
      using vec-first-lincomb-list Xs X ‹n = m + 1› by auto
    hence x ∈ m.cone-list ?Ys using c
      unfolding m.cone-list-def m.nonneg-lincomb-list-def by auto
  } moreover {
    assume x ∈ m.cone-list ?Ys
    then obtain c where x: m.lincomb-list c ?Ys = x and c: ∀ i < length Xs. c
i ≥ 0
      unfolding m.cone-list-def m.nonneg-lincomb-list-def by auto

    have vec-last (n.lincomb-list c Xs) 1 $ 0 = n.lincomb-list c Xs $ m
```

```
      using ‹n = m + 1› n.lincomb-list-carrier X Xs vec-last-index[of - m 1 0]
      by auto
    also have ... = 0
      using n.lincomb-list-index[of m Xs c] Xs X ‹n = m + 1› Xm0 by simp
    also have ... = vec-last y 1 $ 0
      using y y-dim ‹n = m + 1› vec-last-index[of y m 1 0] by auto
    finally have vec-last (n.lincomb-list c Xs) 1 = vec-last y 1 by fastforce

    moreover have vec-first (n.lincomb-list c Xs) m = x
      using vec-first-lincomb-list[of Xs c] x X Xs ‹n = m + 1›
      unfolding x-def by simp

    ultimately have n.lincomb-list c Xs = y unfolding x-def
      using vec-first-last-append[of - m 1] ‹n = m + 1› y-dim
        n.lincomb-list-carrier[of Xs c] Xs X
      by metis
    hence y ∈ n.cone-list Xs
      unfolding n.cone-list-def n.nonneg-lincomb-list-def using c by blast
  }
  moreover have m.cone-list ?Ys = m.cone ?Y
    using m.finite-cone-iff-cone-list[OF - Ys] m.cone-iff-finite-cone[of ?Y]
      ‹finite X› by force
  ultimately show ?thesis unfolding x-def by blast
qed

end

context gram-schmidt
begin

lemma decomposition-theorem-polyhedra-1:
  assumes A: A ∈ carrier-mat nr n
    and b: b ∈ carrier-vec nr
    and P: P = polyhedron A b
  shows ∃ Q X. X ⊆ carrier-vec n ∧ finite X ∧
    Q ⊆ carrier-vec n ∧ finite Q ∧
    P = convex-hull Q + cone X ∧
     (is-det-bound db ⟶ A ∈ ℤm ∩ Bounded-mat (of-int Bnd) ⟶ b ∈ ℤv ∩
Bounded-vec (of-int Bnd) ⟶
      X ⊆ ℤv ∩ Bounded-vec (of-int (db n (max 1 Bnd)))
    ∧ Q ⊆ Bounded-vec (of-int (db n (max 1 Bnd))))
proof −
  let ?oi = of-int :: int ⇒ 'a

  interpret next-dim: gram-schmidt n + 1 TYPE ('a).
  interpret gram-schmidt-m n + 1 n TYPE('a).

  from P[unfolded polyhedron-def] have P ⊆ carrier-vec n by auto
```

```
have mcb: mat-of-col (−b) ∈ carrier-mat nr 1 using b by auto
define M where M = (A @_c mat-of-col (−b)) @_r (0_m 1 n @_c −1_m 1)
have M-top: A @_c mat-of-col (− b) ∈ carrier-mat nr (n + 1)
  by (rule carrier-append-cols[OF A mcb])
have M-bottom: (0_m 1 n @_c −1_m 1) ∈ carrier-mat 1 (n + 1)
  by (rule carrier-append-cols, auto)
have M-dim: M ∈ carrier-mat (nr + 1) (n + 1)
  unfolding M-def
  by (rule carrier-append-rows[OF M-top M-bottom])

{
  fix x :: 'a vec fix t assume x: x ∈ carrier-vec n
  have x @_v vec-of-scal t ∈ next-dim.polyhedral-cone M =
       (A *_v x − t ·_v b ≤ 0_v nr ∧ t ≥ 0)
  proof −
    let ?y = x @_v vec-of-scal t
    have y: ?y ∈ carrier-vec (n + 1) using x by(simp del: One-nat-def)
    have ?y ∈ next-dim.polyhedral-cone M =
         (M *_v ?y ≤ 0_v (nr + 1))
      unfolding next-dim.polyhedral-cone-def using y M-dim by auto
    also have 0_v (nr + 1) = 0_v nr @_v 0_v 1 by auto
    also have M *_v ?y ≤ 0_v nr @_v 0_v 1 =
              ((A @_c mat-of-col (−b)) *_v ?y ≤ 0_v nr ∧
              (0_m 1 n @_c −1_m 1) *_v ?y ≤ 0_v 1)
      unfolding M-def
      by (intro append-rows-le[OF M-top M-bottom - y], auto)
    also have (A @_c mat-of-col(−b)) *_v ?y =
             A *_v x + mat-of-col(−b) *_v vec-of-scal t
      by (rule mat-mult-append-cols[OF A - x],
          auto simp add: b simp del: One-nat-def)
    also have mat-of-col(−b) *_v vec-of-scal t = t ·_v (−b)
      by(rule mult-mat-of-row-vec-of-scal)
    also have A *_v x + t ·_v (−b) = A *_v x − t ·_v b by auto
    also have (0_m 1 n @_c − 1_m 1) *_v (x @_v vec-of-scal t) =
             0_m 1 n *_v x + − 1_m 1 *_v vec-of-scal t
      by(rule mat-mult-append-cols, auto simp add: x simp del: One-nat-def)
    also have ... = − vec-of-scal t using x by (auto simp del: One-nat-def)
    also have (... ≤ 0_v 1) = (t ≥ 0) unfolding less-eq-vec-def by auto
    finally show (?y ∈ next-dim.polyhedral-cone M) =
              (A *_v x − t ·_v b ≤ 0_v nr ∧ t ≥ 0) by auto
  qed
} note M-cone-car = this
from next-dim.farkas-minkowsky-weyl-theorem-2[OF M-dim, of db max 1 Bnd]
obtain X where X: next-dim.polyhedral-cone M = next-dim.cone X and
  fin-X: finite X and X-carrier: X ⊆ carrier-vec (n+1)
 and Bnd: is-det-bound db ⟹ M ∈ ℤ_m ∩ Bounded-mat (?oi (max 1 Bnd)) ⟹
       X ⊆ ℤ_v ∩ Bounded-vec (?oi (db n (max 1 Bnd)))
  by auto
let ?f = λ x. if x $ n = 0 then 1 else 1 / (x $ n)
```

define Y **where** $Y = \{?f\ x \cdot_v x \mid x.\ x \in X\}$
have *finite Y* **unfolding** *Y-def* **using** *fin-X* **by** *auto*
have *Y-carrier*: $Y \subseteq$ *carrier-vec* $(n+1)$ **unfolding** *Y-def* **using** *X-carrier* **by** *auto*
have $?f$ ' $X \subseteq \{y.\ y > 0\}$
proof
fix y
assume $y \in ?f$ ' X
then obtain x **where** x: $x \in X$ **and** y: $y = ?f\ x$ **by** *auto*
show $y \in \{y.\ y > 0\}$
proof *cases*
assume $x \$ n = 0$
thus $y \in \{y.\ y > 0\}$ **using** y **by** *auto*
next
assume P: $x \$ n \neq 0$
have $x =$ *vec-first* $x\ n$ $@_v$ *vec-last* x *1*
using x *X-carrier vec-first-last-append* **by** *auto*
also have *vec-last* x *1* = *vec-of-scal* (*vec-last* x *1* \$ *0*) **by** *auto*
also have *vec-last* x *1* \$ *0* = $x \$ n$
using x *X-carrier* **unfolding** *vec-last-def* **by** *auto*
finally have $x =$ *vec-first* $x\ n$ $@_v$ *vec-of-scal* $(x \$ n)$ **by** *auto*
moreover have $x \in$ *next-dim.polyhedral-cone M*
using x X *X-carrier next-dim.set-in-cone* **by** *auto*
ultimately have $x \$ n \geq 0$ **using** *M-cone-car vec-first-carrier* **by** *metis*
hence $x \$ n > 0$ **using** P **by** *auto*
thus $y \in \{y.\ y > 0\}$ **using** y **by** *auto*
qed
qed
hence Y: *next-dim.cone* Y = *next-dim.polyhedral-cone M* **unfolding** *Y-def*
using *next-dim.cone-smult-basis*[*OF X-carrier*] X **by** *auto*
define *Y0* **where** *Y0* = $\{v \in Y.\ v \$ n = 0\}$
define *Y1* **where** *Y1* = $Y -$ *Y0*
have *Y0-carrier*: *Y0* $\subseteq$ *carrier-vec* $(n + 1)$ **and** *Y1-carrier*: *Y1* $\subseteq$ *carrier-vec* $(n + 1)$
unfolding *Y0-def Y1-def* **using** *Y-carrier* **by** *auto*
have *finite Y0* **and** *finite Y1*
unfolding *Y0-def Y1-def* **using** ‹*finite Y*› **by** *auto*

have *Y1*: $\bigwedge y.\ y \in$ *Y1* $\Longrightarrow y \$ n = 1$
proof −
fix y **assume** y: $y \in$ *Y1*
hence $y \in Y$ **unfolding** *Y1-def* **by** *auto*
then obtain x **where** $x \in X$ **and** x: $y = ?f\ x \cdot_v x$ **unfolding** *Y-def* **by** *auto*
then have $x \$ n \neq 0$ **using** x y *Y1-def Y0-def* **by** *auto*
then have $y = 1 / (x \$ n) \cdot_v x$ **using** x **by** *auto*
then have $y \$ n = 1 / (x \$ n) * x \$ n$ **using** *X-carrier* ‹$x \in X$› **by** *auto*
thus $y \$ n = 1$ **using** ‹$x \$ n \neq 0$› **by** *auto*
qed

```
let ?Z0 = {vec-first y n | y. y ∈ Y0}
let ?Z1 = {vec-first y n | y. y ∈ Y1}
show ?thesis
proof (intro exI conjI impI)
  show ?Z0 ⊆ carrier-vec n by auto
  show ?Z1 ⊆ carrier-vec n by auto
  show finite ?Z0 using ‹finite Y0› by auto
  show finite ?Z1 using ‹finite Y1› by auto
  show P = convex-hull ?Z1 + cone ?Z0
  proof −
    {
      fix x
      assume x ∈ P
      hence xn: x ∈ carrier-vec n and A *_v x ≤ b
        using P unfolding polyhedron-def by auto
      hence A *_v x − 1 ·_v b ≤ 0_v nr
      using vec-le-iff-diff-le-0 A b carrier-vecD mult-mat-vec-carrier one-smult-vec
        by metis
      hence x @_v vec-of-scal 1 ∈ next-dim.polyhedral-cone M
        using M-cone-car[OF xn] by auto
      hence x @_v vec-of-scal 1 ∈ next-dim.cone Y using Y by auto
      hence x @_v vec-of-scal 1 ∈ next-dim.finite-cone Y
        using next-dim.cone-iff-finite-cone[OF Y-carrier ‹finite Y›] by auto
      then obtain c where c: next-dim.nonneg-lincomb c Y (x @_v vec-of-scal 1)
        unfolding next-dim.finite-cone-def using ‹finite Y› by auto
      let ?y = next-dim.lincomb c Y1
      let ?z = next-dim.lincomb c Y0
      have y-dim: ?y ∈ carrier-vec (n + 1) and z-dim: ?z ∈ carrier-vec (n + 1)
        unfolding next-dim.nonneg-lincomb-def
        using Y0-carrier Y1-carrier next-dim.lincomb-closed by simp-all
      hence yz-dim: ?y + ?z ∈ carrier-vec (n + 1) by auto
      have x @_v vec-of-scal 1 = next-dim.lincomb c Y
        using c unfolding next-dim.nonneg-lincomb-def by auto
      also have Y = Y1 ∪ Y0 unfolding Y1-def using Y0-def by blast
      also have next-dim.lincomb c (Y1 ∪ Y0) = ?y + ?z
        using next-dim.lincomb-union2[of Y1 Y0]
          ‹finite Y0› ‹finite Y› Y0-carrier Y-carrier
        unfolding Y1-def by fastforce
      also have ?y + ?z = vec-first (?y + ?z) n @_v vec-last (?y + ?z) 1
        using vec-first-last-append[of ?y + ?z n 1] add-carrier-vec yz-dim
        by simp
      also have vec-last (?y + ?z) 1 = vec-of-scal ((?y + ?z) $ n)
        using vec-of-scal-dim-1 vec-last-index[OF yz-dim, of 0] by auto
      finally have x @_v vec-of-scal 1 =
                   vec-first (?y + ?z) n @_v vec-of-scal ((?y + ?z) $ n) by auto
      hence x = vec-first (?y + ?z) n and
        yz-last: vec-of-scal 1 = vec-of-scal ((?y + ?z) $ n)
        using append-vec-eq yz-dim xn by auto
      hence xyz: x = vec-first ?y n + vec-first ?z n
```

```
    using vec-first-add[of n ?y ?z] y-dim z-dim by simp

  have 1 = ((?y + ?z) $ n) using yz-last index-vec-of-scal
    by (metis (no-types, lifting))
  hence 1 = ?y $ n + ?z $ n using y-dim z-dim by auto
  moreover have zn0: ?z $ n = 0
    using next-dim.lincomb-index[OF - Y0-carrier] Y0-def by auto
  ultimately have yn1: 1 = ?y $ n by auto
  have next-dim.nonneg-lincomb c Y1 ?y
    using c Y1-def
    unfolding next-dim.nonneg-lincomb-def by auto
  hence ?y ∈ next-dim.cone Y1
    using next-dim.cone-iff-finite-cone[OF Y1-carrier] ‹finite Y1›
    unfolding next-dim.finite-cone-def by auto
  hence y: vec-first ?y n ∈ convex-hull ?Z1
    using convex-hull-next-dim[OF - Y1-carrier ‹finite Y1› - y-dim] Y1 yn1
    by simp

  have next-dim.nonneg-lincomb c Y0 ?z using c Y0-def
    unfolding next-dim.nonneg-lincomb-def by blast
  hence ?z ∈ next-dim.cone Y0
    using ‹finite Y0› next-dim.cone-iff-finite-cone[OF Y0-carrier ‹finite Y0›]
    unfolding next-dim.finite-cone-def
    by fastforce
  hence z: vec-first ?z n ∈ cone ?Z0
    using cone-next-dim[OF - Y0-carrier ‹finite Y0› - - zn0] Y0-def
      next-dim.lincomb-closed[OF Y0-carrier] by blast

  from xyz y z have x ∈ convex-hull ?Z1 + cone ?Z0 by blast
} moreover {
  fix x
  assume x ∈ convex-hull ?Z1 + cone ?Z0
  then obtain y z where x = y + z and y: y ∈ convex-hull ?Z1
    and z: z ∈ cone ?Z0 by (auto elim: set-plus-elim)

  have yn: y ∈ carrier-vec n
    using y convex-hull-carrier[OF ‹?Z1 ⊆ carrier-vec n›] by blast
  hence y @_v vec-of-scal 1 ∈ carrier-vec (n + 1)
    using vec-of-scal-dim(2) by fast
  moreover have vec-first (y @_v vec-of-scal 1) n ∈ convex-hull ?Z1
    using vec-first-append[OF yn] y by auto
  moreover have (y @_v vec-of-scal 1) $ n = 1 using yn by simp
  ultimately have y @_v vec-of-scal 1 ∈ next-dim.cone Y1
    using convex-hull-next-dim[OF - Y1-carrier ‹finite Y1›] Y1 by blast
  hence y-cone: y @_v vec-of-scal 1 ∈ next-dim.cone Y
    using next-dim.cone-mono[of Y1 Y] Y1-def by blast

  have zn: z ∈ carrier-vec n using z cone-carrier[of ?Z0] by fastforce
  hence z @_v vec-of-scal 0 ∈ carrier-vec (n + 1)
```

using *vec-of-scal-dim*(*2*) **by** *fast*
moreover have *vec-first* (z @$_v$ *vec-of-scal 0*) $n \in$ *cone ?Z0*
using *vec-first-append*[*OF zn*] *z* **by** *auto*
moreover have (z @$_v$ *vec-of-scal 0*) \$ $n = 0$ **using** *zn* **by** *simp*
ultimately have z @$_v$ *vec-of-scal 0* $\in$ *next-dim.cone Y0*
using *cone-next-dim*[*OF - Y0-carrier* ‹*finite Y0*›] *Y0-def* **by** *blast*
hence *z-cone*: z @$_v$ *vec-of-scal 0* $\in$ *next-dim.cone Y*
using *Y0-def next-dim.cone-mono*[*of Y0 Y*] **by** *blast*

have *xn*: $x \in$ *carrier-vec n* **using** ‹$x = y + z$› *yn zn* **by** *blast*
have x @$_v$ *vec-of-scal 1* = (y @$_v$ *vec-of-scal 1*) + (z @$_v$ *vec-of-scal 0*)
using ‹$x = y + z$› *append-vec-add*[*OF yn zn*]
unfolding *vec-of-scal-def* **by** *auto*
hence x @$_v$ *vec-of-scal 1* $\in$ *next-dim.cone Y*
using *next-dim.cone-elem-sum*[*OF Y-carrier y-cone z-cone*] **by** *simp*
hence $A *_v x - b \le 0_v$ *nr* **using** *M-cone-car*[*OF xn*] *Y* **by** *simp*
hence $A *_v x \le b$ **using** *vec-le-iff-diff-le-0*[*of* $A *_v x$ *b*]
dim-mult-mat-vec[*of A x*] *A* **by** *simp*
hence $x \in P$ **using** *P xn* **unfolding** *polyhedron-def* **by** *blast*
}
ultimately show $P =$ *convex-hull ?Z1* + *cone ?Z0* **by** *blast*
qed

let *?Bnd* = *db n* (*max 1 Bnd*)
assume $A \in \mathbb{Z}_m \cap$ *Bounded-mat* (*?oi Bnd*)
$b \in \mathbb{Z}_v \cap$ *Bounded-vec* (*?oi Bnd*)
and *db*: *is-det-bound db*
hence $*$: $A \in \mathbb{Z}_m$ $A \in$ *Bounded-mat* (*?oi Bnd*) $b \in \mathbb{Z}_v$ $b \in$ *Bounded-vec* (*?oi Bnd*) **by** *auto*
have *elements-mat M* $\subseteq$ *elements-mat A* $\cup$ *vec-set* ($-b$) $\cup$ {*0*,$-$*1*}
unfolding *M-def*
unfolding *elements-mat-append-rows*[*OF M-top M-bottom*]
unfolding *elements-mat-append-cols*[*OF A mcb*]
by (*subst elements-mat-append-cols*, *auto*)
also have ... $\subseteq \mathbb{Z} \cap$ ({*x. abs x* $\le$ *?oi Bnd*} $\cup$ {*0*,$-$*1*})
using $*$[*unfolded Bounded-mat-elements-mat Ints-mat-elements-mat*
Bounded-vec-vec-set Ints-vec-vec-set] **by** *auto*
also have ... $\subseteq \mathbb{Z} \cap$ ({*x. abs x* $\le$ *?oi* (*max 1 Bnd*)}) **by** (*auto simp*: *of-int-max*)
finally have $M \in \mathbb{Z}_m$ $M \in$ *Bounded-mat* (*?oi* (*max 1 Bnd*))
unfolding *Bounded-mat-elements-mat Ints-mat-elements-mat* **by** *auto*
hence $M \in \mathbb{Z}_m \cap$ *Bounded-mat* (*?oi* (*max 1 Bnd*)) **by** *blast*
from *Bnd*[*OF db this*]
have *XBnd*: $X \subseteq \mathbb{Z}_v \cap$ *Bounded-vec* (*?oi ?Bnd*) **.**
{
fix *y*
assume *y*: $y \in Y$
then obtain *x* **where** *y*: $y =$ *?f x* $\cdot_v$ *x* **and** *xX*: $x \in X$ **unfolding** *Y-def* **by** *auto*
with ‹$X \subseteq$ *carrier-vec* (n+*1*)› **have** *x*: $x \in$ *carrier-vec* (n+*1*) **by** *auto*

```
    from XBnd xX have xI: x ∈ ℤv and xB: x ∈ Bounded-vec (?oi ?Bnd) by
auto
    {
      assume y $ n = 0
      hence y = x unfolding y using x by auto
      hence y ∈ ℤv ∩ Bounded-vec (?oi ?Bnd) using xI xB by auto
    } note y0 = this
    {
      assume y $ n ≠ 0
      hence x0: x $ n ≠ 0 using x unfolding y by auto
      from x xI have x $ n ∈ ℤ unfolding Ints-vec-def by auto
      with x0 have abs (x $ n) ≥ 1 by (meson Ints-nonzero-abs-ge1)
      hence abs: abs (1 / (x $ n)) ≤ 1 by simp
      {
        fix a
        have abs ((1 / (x $ n)) ∗ a) = abs (1 / (x $ n)) ∗ abs a
          by simp
        also have ... ≤ 1 ∗ abs a
          by (rule mult-right-mono[OF abs], auto)
        finally have abs ((1 / (x $ n)) ∗ a) ≤ abs a by auto
      } note abs = this
      from x0 have y: y = (1 / (x $ n)) ·v x unfolding y by auto
      have vy: vec-set y = (λ a. (1 / (x $ n)) ∗ a) ‘ vec-set x
        unfolding y by (auto simp: vec-set-def)
      have y ∈ Bounded-vec (?oi ?Bnd) using xB abs
        unfolding Bounded-vec-vec-set vy
        by (smt imageE max.absorb2 max.bounded-iff)
    } note yn0 = this
    note y0 yn0
  } note BndY = this
  from ‹Y ⊆ carrier-vec (n+1)›
  have setvY: y ∈ Y ⟹ setv (vec-first y n) ⊆ setv y for y
    unfolding vec-first-def vec-set-def by auto
  from BndY(1) setvY
  show ?Z0 ⊆ ℤv ∩ Bounded-vec (?oi (db n (max 1 Bnd)))
    by (force simp: Bounded-vec-vec-set Ints-vec-vec-set Y0-def)
  from BndY(2) setvY
  show ?Z1 ⊆ Bounded-vec (?oi (db n (max 1 Bnd)))
    by (force simp: Bounded-vec-vec-set Ints-vec-vec-set Y0-def Y1-def)
 qed
qed

lemma decomposition-theorem-polyhedra-2:
 assumes Q: Q ⊆ carrier-vec n and fin-Q: finite Q
  and X: X ⊆ carrier-vec n and fin-X: finite X
  and P: P = convex-hull Q + cone X
 shows ∃ A b nr. A ∈ carrier-mat nr n ∧ b ∈ carrier-vec nr ∧ P = polyhedron A
b
proof −
```

```
interpret next-dim: gram-schmidt n + 1 TYPE ('a).
interpret gram-schmidt-m n + 1 n TYPE('a).

from fin-Q obtain Qs where Qs: Q = set Qs using finite-list by auto
from fin-X obtain Xs where Xs: X = set Xs using finite-list by auto
define Y where Y = {x @v vec-of-scal 1 | x. x ∈ Q}
define Z where Z = {x @v vec-of-scal 0 | x. x ∈ X}
have fin-Y: finite Y unfolding Y-def using fin-Q by simp
have fin-Z: finite Z unfolding Z-def using fin-X by simp
have Y-dim: Y ⊆ carrier-vec (n + 1)
  unfolding Y-def using Q append-carrier-vec[OF - vec-of-scal-dim(2)[of 1]]
  by blast
have Z-dim: Z ⊆ carrier-vec (n + 1)
  unfolding Z-def using X append-carrier-vec[OF - vec-of-scal-dim(2)[of 0]]
  by blast
have Y-car: Q = {vec-first x n | x. x ∈ Y}
proof (intro equalityI subsetI)
  fix x assume x: x ∈ Q
  hence x @v vec-of-scal 1 ∈ Y unfolding Y-def by blast
  thus x ∈ {vec-first x n | x. x ∈ Y}
    using Q vec-first-append[of x n vec-of-scal 1] x by force
next
  fix x assume x ∈ {vec-first x n | x. x ∈ Y}
  then obtain y where y ∈ Q and x = vec-first (y @v vec-of-scal 1) n
    unfolding Y-def by blast
  thus x ∈ Q using Q vec-first-append[of y] by auto
qed
have Z-car: X = {vec-first x n | x. x ∈ Z}
proof (intro equalityI subsetI)
  fix x assume x: x ∈ X
  hence x @v vec-of-scal 0 ∈ Z unfolding Z-def by blast
  thus x ∈ {vec-first x n | x. x ∈ Z}
    using X vec-first-append[of x n vec-of-scal 0] x by force
next
  fix x assume x ∈ {vec-first x n | x. x ∈ Z}
  then obtain y where y ∈ X and x = vec-first (y @v vec-of-scal 0) n
    unfolding Z-def by blast
  thus x ∈ X using X vec-first-append[of y] by auto
qed
have Y-last: ∀ x ∈ Y. x $ n = 1 unfolding Y-def using Q by auto
have Z-last: ∀ x ∈ Z. x $ n = 0 unfolding Z-def using X by auto

have finite (Y ∪ Z) using fin-Y fin-Z by blast
moreover have Y ∪ Z ⊆ carrier-vec (n + 1) using Y-dim Z-dim by blast
ultimately obtain B nr
  where B: next-dim.cone (Y ∪ Z) = next-dim.polyhedral-cone B
    and B-carrier: B ∈ carrier-mat nr (n + 1)
  using next-dim.farkas-minkowsky-weyl-theorem[of next-dim.cone (Y ∪ Z)]
  by blast
```

define *A* **where** *A* = *mat-col-first B n*
define *b* **where** *b* = *col B n*
have *B-blocks*: *B* = *A* $@_c$ *mat-of-col b*
unfolding *A-def b-def*
using *mat-col-first-last-append*[*of B n 1*] *B-carrier*
mat-of-col-dim-col-1[*of mat-col-last B 1*] **by** *auto*
have *A-carrier*: *A* ∈ *carrier-mat nr n* **unfolding** *A-def* **using** *B-carrier* **by** *force*
have *b-carrier*: *b* ∈ *carrier-vec nr* **unfolding** *b-def* **using** *B-carrier* **by** *force*

{
fix *x* **assume** *x* ∈ *P*
then obtain *y z* **where** *x*: *x* = *y* + *z* **and** *y*: *y* ∈ *convex-hull Q* **and** *z*: *z* ∈ *cone X*
using *P* **by** (*auto elim*: *set-plus-elim*)

have *yn*: *y* ∈ *carrier-vec n* **using** *y convex-hull-carrier*[*OF Q*] **by** *blast*
moreover have *zn*: *z* ∈ *carrier-vec n* **using** *z cone-carrier*[*OF X*] **by** *blast*
ultimately have *xn*: *x* ∈ *carrier-vec n* **using** *x* **by** *blast*

have *yn1*: *y* $@_v$ *vec-of-scal 1* ∈ *carrier-vec* (*n* + *1*)
using *append-carrier-vec*[*OF yn*] *vec-of-scal-dim* **by** *fast*
have *y-last*: (*y* $@_v$ *vec-of-scal 1*) $ *n* = *1* **using** *yn* **by** *force*
have *vec-first* (*y* $@_v$ *vec-of-scal 1*) *n* = *y*
using *vec-first-append*[*OF yn*] **by** *simp*
hence *y* $@_v$ *vec-of-scal 1* ∈ *next-dim.cone Y*
using *convex-hull-next-dim*[*OF - Y-dim fin-Y Y-last yn1 y-last*] *Y-car y* **by** *argo*
hence *y-cone*: *y* $@_v$ *vec-of-scal 1* ∈ *next-dim.cone* (*Y* ∪ *Z*)
using *next-dim.cone-mono*[*of Y Y* ∪ *Z*] **by** *blast*

have *zn1*: *z* $@_v$ *vec-of-scal 0* ∈ *carrier-vec* (*n* + *1*)
using *append-carrier-vec*[*OF zn*] *vec-of-scal-dim* **by** *fast*
have *z-last*: (*z* $@_v$ *vec-of-scal 0*) $ *n* = *0* **using** *zn* **by** *force*
have *vec-first* (*z* $@_v$ *vec-of-scal 0*) *n* = *z*
using *vec-first-append*[*OF zn*] **by** *simp*
hence *z* $@_v$ *vec-of-scal 0* ∈ *next-dim.cone Z*
using *cone-next-dim*[*OF - Z-dim fin-Z Z-last zn1 z-last*] *Z-car z* **by** *argo*
hence *z-cone*: *z* $@_v$ *vec-of-scal 0* ∈ *next-dim.cone* (*Y* ∪ *Z*)
using *next-dim.cone-mono*[*of Z Y* ∪ *Z*] **by** *blast*

from ‹*x* = *y* + *z*›
have *x* $@_v$ *vec-of-scal 1* = (*y* $@_v$ *vec-of-scal 1*) + (*z* $@_v$ *vec-of-scal 0*)
using *append-vec-add*[*OF yn zn*] *vec-of-scal-dim-1*
unfolding *vec-of-scal-def* **by** *auto*
hence *x* $@_v$ *vec-of-scal 1* ∈ *next-dim.cone* (*Y* ∪ *Z*) ∧ *x* ∈ *carrier-vec n*
using *next-dim.cone-elem-sum*[*OF - y-cone z-cone*] *Y-dim Z-dim xn* **by** *auto*
} moreover {
fix *x* **assume** *x* $@_v$ *vec-of-scal 1* ∈ *next-dim.cone* (*Y* ∪ *Z*)
then obtain *c* **where** *x*: *next-dim.lincomb c* (*Y* ∪ *Z*) = *x* $@_v$ *vec-of-scal 1*

and *c*: *c* ‘ (*Y* ∪ *Z*) ⊆ {*t*. *t* ≥ *0*}
using *next-dim.cone-iff-finite-cone Y-dim Z-dim fin-Y fin-Z*
unfolding *next-dim.finite-cone-def next-dim.nonneg-lincomb-def* **by** *auto*

let *?y* = *next-dim.lincomb c Y*
let *?z* = *next-dim.lincomb c Z*
have *xyz*: $x\ @_v$ *vec-of-scal 1* = *?y* + *?z*
using *x next-dim.lincomb-union*[*OF Y-dim Z-dim - fin-Y fin-Z*] *Y-last Z-last*
by *fastforce*

have *y-dim*: *?y* ∈ *carrier-vec* (*n* + *1*) **using** *next-dim.lincomb-closed*[*OF Y-dim*]
by *blast*
have *z-dim*: *?z* ∈ *carrier-vec* (*n* + *1*) **using** *next-dim.lincomb-closed*[*OF Z-dim*]
by *blast*
have $x\ @_v$ *vec-of-scal 1* ∈ *carrier-vec* (*n* + *1*)
using *xyz add-carrier-vec*[*OF y-dim z-dim*] **by** *argo*
hence *x-dim*: *x* ∈ *carrier-vec n*
using *carrier-dim-vec*[*of x n*] *carrier-dim-vec*[*of - n* + *1*]
by *force*

have *z-last*: *?z* $ *n* = *0* **using** *Z-last next-dim.lincomb-index*[*OF - Z-dim*, *of n*]
by *force*
have *?y* $ *n* + *?z* $ *n* = ($x\ @_v$ *vec-of-scal 1*) $ *n*
using *xyz index-add-vec*(*1*) *z-dim* **by** *simp*
also have . . . = *1* **using** *x-dim* **by** *auto*
finally have *y-last*: *?y* $ *n* = *1* **using** *z-last* **by** *algebra*

have *?y* ∈ *next-dim.cone Y*
using *next-dim.cone-iff-finite-cone*[*OF Y-dim*] *fin-Y c*
unfolding *next-dim.finite-cone-def next-dim.nonneg-lincomb-def* **by** *auto*
hence *y-cone*: *vec-first ?y n* ∈ *convex-hull Q*
using *convex-hull-next-dim*[*OF - Y-dim fin-Y Y-last y-dim y-last*] *Y-car*
by *blast*

have *?z* ∈ *next-dim.cone Z*
using *next-dim.cone-iff-finite-cone*[*OF Z-dim*] *fin-Z c*
unfolding *next-dim.finite-cone-def next-dim.nonneg-lincomb-def* **by** *auto*
hence *z-cone*: *vec-first ?z n* ∈ *cone X*
using *cone-next-dim*[*OF - Z-dim fin-Z Z-last z-dim z-last*] *Z-car*
by *blast*

have *x* = *vec-first* ($x\ @_v$ *vec-of-scal 1*) *n* **using** *vec-first-append*[*OF x-dim*] **by** *simp*
also have . . . = *vec-first ?y n* + *vec-first ?z n*
using *xyz vec-first-add*[*of n ?y ?z*] *y-dim z-dim carrier-dim-vec* **by** *auto*
finally have *x* ∈ *P*
using *y-cone z-cone P* **by** *blast*
} moreover {
fix *x* :: *'a vec*

assume *xn*: $x \in$ *carrier-vec n*
hence (x $@_v$ *vec-of-scal 1* $\in$ *next-dim.polyhedral-cone B*) =
($B *_v$ (x $@_v$ *vec-of-scal 1*) $\leq$ 0_v *nr*)
unfolding *next-dim.polyhedral-cone-def* **using** *B-carrier*
using *append-carrier-vec*[*OF - vec-of-scal-dim*(*2*)[*of 1*]] **by** *auto*
also have ... = ((*A* $@_c$ *mat-of-col b*) $*_v$ (x $@_v$ *vec-of-scal 1*) $\leq$ 0_v *nr*)
using *B-blocks* **by** *blast*
also have (*A* $@_c$ *mat-of-col b*) $*_v$ (x $@_v$ *vec-of-scal 1*) =
$A *_v x$ + *mat-of-col b* $*_v$ *vec-of-scal 1*
by (*rule mat-mult-append-cols*, *insert A-carrier b-carrier xn*, *auto simp del*: *One-nat-def*)
also have *mat-of-col b* $*_v$ *vec-of-scal 1* = *b*
using *mult-mat-of-row-vec-of-scal*[*of b 1*] **by** *simp*
also have $A *_v x + b = A *_v x - -b$ **by** *auto*
finally have (x $@_v$ *vec-of-scal 1* $\in$ *next-dim.polyhedral-cone B*) = ($A *_v x \leq -b$)
using *vec-le-iff-diff-le-0*[*of* $A *_v x$ $-b$] *A-carrier* **by** *simp*
}
ultimately have *P* = *polyhedron A* ($-b$)
unfolding *polyhedron-def* **using** *B* **by** *blast*
moreover have $-b \in$ *carrier-vec nr* **using** *b-carrier* **by** *simp*
ultimately show *?thesis* **using** *A-carrier* **by** *blast*
qed

lemma *decomposition-theorem-polyhedra*:
($\exists$ *A b nr*. *A* $\in$ *carrier-mat nr n* $\wedge$ *b* $\in$ *carrier-vec nr* $\wedge$ *P* = *polyhedron A b*) $\longleftrightarrow$
($\exists$ *Q X*. $Q \cup X \subseteq$ *carrier-vec n* $\wedge$ *finite* ($Q \cup X$) $\wedge$ *P* = *convex-hull Q* + *cone X*) (**is** *?l* = *?r*)
proof
assume *?l*
then obtain *A b nr* **where** *A*: *A* $\in$ *carrier-mat nr n*
and *b*: *b* $\in$ *carrier-vec nr* **and** *P*: *P* = *polyhedron A b* **by** *auto*
from *decomposition-theorem-polyhedra-1*[*OF this*] **obtain** *Q X*
where $*$: $X \subseteq$ *carrier-vec n finite X* $Q \subseteq$ *carrier-vec n finite Q P* = *convex-hull Q* + *cone X*
by *meson*
show *?r*
by (*rule exI*[*of - Q*], *rule exI*[*of - X*], *insert* $*$, *auto simp*: *polytope-def*)
next
assume *?r*
then obtain *Q X* **where** *QX-carrier*: $Q \cup X \subseteq$ *carrier-vec n*
and *QX-fin*: *finite* ($Q \cup X$)
and *P*: *P* = *convex-hull Q* + *cone X* **by** *blast*
from *QX-carrier* **have** *Q*: $Q \subseteq$ *carrier-vec n* **and** *X*: $X \subseteq$ *carrier-vec n* **by** *simp-all*
from *QX-fin* **have** *fin-Q*: *finite Q* **and** *fin-X*: *finite X* **by** *simp-all*
show *?l* **using** *decomposition-theorem-polyhedra-2*[*OF Q fin-Q X fin-X P*] **by** *blast*

qed

lemma *polytope-equiv-bounded-polyhedron*:
polytope P ⟷
(∃ *A b nr bnd*. *A* ∈ *carrier-mat nr n* ∧ *b* ∈ *carrier-vec nr* ∧ *P* = *polyhedron A b*
∧ *P* ⊆ *Bounded-vec bnd*)
proof
assume *polyP*: *polytope P*
from *this* **obtain** *Q* **where** *Qcarr*: *Q* ⊆ *carrier-vec n* **and** *finQ*: *finite Q*
and *PconvhQ*: *P* = *convex-hull Q* **unfolding** *polytope-def* **by** *auto*
let *?X* = {}
have *convex-hull Q* + {*0*$_v$ *n*} = *convex-hull Q* **using** *Qcarr add-0-right-vecset*[*of convex-hull Q*]
by (*simp add*: *convex-hull-carrier*)
hence *P* = *convex-hull Q* + *cone ?X* **using** *PconvhQ* **by** *simp*
hence *Q* ∪ *?X* ⊆ *carrier-vec n* ∧ *finite* (*Q* ∪ *?X*) ∧ *P* = *convex-hull Q* + *cone ?X*
using *Qcarr finQ PconvhQ* **by** *simp*
hence ∃ *A b nr*. *A* ∈ *carrier-mat nr n* ∧ *b* ∈ *carrier-vec nr* ∧ *P* = *polyhedron A b*
using *decomposition-theorem-polyhedra* **by** *blast*
hence *Ppolyh*: ∃ *A b nr*. *A* ∈ *carrier-mat nr n* ∧ *b* ∈ *carrier-vec nr* ∧ *P* = *polyhedron A b* **by** *blast*
from *finite-Bounded-vec-Max*[*OF Qcarr finQ*] **obtain** *bnd* **where** *Q* ⊆ *Bounded-vec bnd* **by** *auto*
hence *Pbnd*: *P* ⊆ *Bounded-vec bnd* **using** *convex-hull-bound PconvhQ Qcarr* **by** *auto*
from *Ppolyh Pbnd* **show** ∃ *A b nr bnd*. *A* ∈ *carrier-mat nr n* ∧ *b* ∈ *carrier-vec nr*
∧ *P* = *polyhedron A b* ∧ *P* ⊆ *Bounded-vec bnd* **by** *auto*
next
assume ∃ *A b nr bnd*. *A* ∈ *carrier-mat nr n* ∧ *b* ∈ *carrier-vec nr* ∧ *P* = *polyhedron A b*
∧ *P* ⊆ *Bounded-vec bnd*
from *this* **obtain** *A b nr bnd* **where** *Adim*: *A* ∈ *carrier-mat nr n* **and** *bdim*: *b* ∈ *carrier-vec nr*
and *Ppolyh*: *P* = *polyhedron A b* **and** *Pbnd*: *P* ⊆ *Bounded-vec bnd* **by** *auto*
have ∃ *A b nr*. *A* ∈ *carrier-mat nr n* ∧ *b* ∈ *carrier-vec nr* ∧ *P* = *polyhedron A b*
using *Adim bdim Ppolyh* **by** *blast*
hence ∃ *Q X*. *Q* ∪ *X* ⊆ *carrier-vec n* ∧ *finite* (*Q* ∪ *X*) ∧ *P* = *convex-hull Q* + *cone X*
using *decomposition-theorem-polyhedra* **by** *simp*
from *this* **obtain** *Q X* **where** *QXcarr*: *Q* ∪ *X* ⊆ *carrier-vec n*
and *finQX*: *finite* (*Q* ∪ *X*) **and** *Psum*: *P* = *convex-hull Q* + *cone X* **by** *auto*
from *QXcarr* **have** *Qcarr*: *convex-hull Q* ⊆ *carrier-vec n* **by** (*simp add*: *convex-hull-carrier*)
from *QXcarr* **have** *Xcarr*: *cone X* ⊆ *carrier-vec n* **by** (*simp add*: *gram-schmidt.cone-carrier*)
from *Pbnd* **have** *Pcarr*: *P* ⊆ *carrier-vec n* **using** *Ppolyh* **unfolding** *polyhe-*

dron-def **by** *simp*
  have *P* = *convex-hull Q*
  proof(*cases Q* = {})
    case *True*
    then show *P* = *convex-hull Q* **unfolding** *Psum* **by** (*auto simp*: *set-plus-def*)
  next
    case *False*
    hence *convnotempty*: *convex-hull Q* ≠ {} **using** *QXcarr* **by** *simp*
    have *Pbndex*: ∃ *bnd*. *P* ⊆ *Bounded-vec bnd* **using** *Pbnd*
      using *QXcarr* **by** *auto*
    from *False* **have** (∃ *bndc*. *cone X* ⊆ *Bounded-vec bndc*)
      using *bounded-vecset-sum*[*OF Qcarr Xcarr Psum Pbndex*] *False convnotempty*
by *blast*
    hence *cone X* = {0_v *n*} **using** *bounded-cone-is-zero QXcarr* **by** *auto*
    thus *?thesis* **unfolding** *Psum* **using** *Qcarr* **by** (*auto simp*: *add-0-right-vecset*)
  qed
  thus *polytope P* **using** *finQX QXcarr* **unfolding** *polytope-def* **by** *auto*
qed
end

end

15 Mixed Integer Solutions

We prove that if an integral system of linear inequalities $Ax \leq b \wedge A'x < b'$ has a (mixed)integer solution, then there is also a small (mixed)integer solution, where the numbers are bounded by $(n + 1) * db\,m\,n$ where n is the number of variables, m is a bound on the absolute values of numbers occurring in A, A', b, b', and $db\,m\,n$ is a bound on determinants for matrices of size n with values of at most m.

theory *Mixed-Integer-Solutions*
  imports *Decomposition-Theorem*
begin

definition *less-vec* :: *'a vec* ⇒ (*'a* :: *ord*) *vec* ⇒ *bool* (**infix** ‹$<_v$› *50*) **where**
  v $<_v$ *w* = (*dim-vec v* = *dim-vec w* ∧ (∀ *i* < *dim-vec w*. *v* $ *i* < *w* $ *i*))

lemma *less-vecD*: **assumes** *v* $<_v$ *w* **and** *w* ∈ *carrier-vec n*
  shows *i* < *n* ⟹ *v* $ *i* < *w* $ *i*
  using *assms* **unfolding** *less-vec-def* **by** *auto*

lemma *less-vecI*: **assumes** *v* ∈ *carrier-vec n w* ∈ *carrier-vec n*
  ⋀ *i*. *i* < *n* ⟹ *v* $ *i* < *w* $ *i*
shows *v* $<_v$ *w*
  using *assms* **unfolding** *less-vec-def* **by** *auto*

lemma *less-vec-lesseq-vec*: *v* $<_v$ (*w* :: *'a* :: *preorder vec*) ⟹ *v* ≤ *w*

unfolding *less-vec-def less-eq-vec-def*
by (*auto simp*: *less-le-not-le*)

lemma *floor-less*: $x \notin \mathbb{Z} \Longrightarrow$ *of-int* $\lfloor x \rfloor < x$
using *le-less* **by** *fastforce*

lemma *floor-of-int-eq*[*simp*]: $x \in \mathbb{Z} \Longrightarrow$ *of-int* $\lfloor x \rfloor = x$
by (*metis Ints-cases of-int-floor-cancel*)

locale *gram-schmidt-floor* = *gram-schmidt n f-ty*
for *n* :: *nat* **and** *f-ty* :: *'a* :: {*floor-ceiling*,
trivial-conjugatable-linordered-field} *itself*
begin

lemma *small-mixed-integer-solution-main*: **fixes** A_1 :: *'a mat*
assumes *db*: *is-det-bound db*
and *A1*: $A_1 \in$ *carrier-mat* nr_1 *n*
and *A2*: $A_2 \in$ *carrier-mat* nr_2 *n*
and *b1*: $b_1 \in$ *carrier-vec* nr_1
and *b2*: $b_2 \in$ *carrier-vec* nr_2
and *A1Bnd*: $A_1 \in \mathbb{Z}_m \cap$ *Bounded-mat* (*of-int Bnd*)
and *b1Bnd*: $b_1 \in \mathbb{Z}_v \cap$ *Bounded-vec* (*of-int Bnd*)
and *A2Bnd*: $A_2 \in \mathbb{Z}_m \cap$ *Bounded-mat* (*of-int Bnd*)
and *b2Bnd*: $b_2 \in \mathbb{Z}_v \cap$ *Bounded-vec* (*of-int Bnd*)
and *x*: $x \in$ *carrier-vec n*
and *xI*: $x \in$ *indexed-Ints-vec I*
and *sol-nonstrict*: $A_1 *_v x \le b_1$
and *sol-strict*: $A_2 *_v x <_v b_2$
shows $\exists$ *x*.
$x \in$ *carrier-vec n* $\wedge$
$x \in$ *indexed-Ints-vec I* $\wedge$
$A_1 *_v x \le b_1 \wedge$
$A_2 *_v x <_v b_2 \wedge$
$x \in$ *Bounded-vec* (*of-int* (*of-nat* (*n* + *1*) $*$ *db n* (*max 1 Bnd*)))
proof −
let *?oi* = *of-int* :: *int* ⇒ *'a*
let *?Bnd* = *?oi Bnd*
define *B* **where** *B* = *?oi* (*db n* (*max 1 Bnd*))
define *A* **where** $A = A_1 @_r A_2$
define *b* **where** $b = b_1 @_v b_2$
define *nr* **where** $nr = nr_1 + nr_2$
have *B0*: $B \ge 0$ **unfolding** *B-def of-int-0-le-iff*
by (*rule is-det-bound-ge-zero*[*OF db*], *auto*)
note *defs* = *A-def b-def nr-def*
from *A1 A2* **have** *A*: $A \in$ *carrier-mat nr n* **unfolding** *defs* **by** *auto*
from *b1 b2* **have** *b*: $b \in$ *carrier-vec nr* **unfolding** *defs* **by** *auto*
from *A1Bnd A2Bnd A1 A2* **have** *ABnd*: $A \in \mathbb{Z}_m \cap$ *Bounded-mat ?Bnd* **unfolding** *defs*

```
    by (auto simp: Ints-mat-elements-mat Bounded-mat-elements-mat elements-mat-append-rows)
  from b1Bnd b2Bnd b1 b2 have bBnd: b ∈ ℤv ∩ Bounded-vec ?Bnd unfolding
defs
    by (auto simp: Ints-vec-vec-set Bounded-vec-vec-set)
  from decomposition-theorem-polyhedra-1[OF A b refl, of db Bnd] ABnd bBnd db
  obtain Y Z where Z: Z ⊆ carrier-vec n
    and finX: finite Z
    and Y: Y ⊆ carrier-vec n
    and finY: finite Y
    and poly: polyhedron A b = convex-hull Y + cone Z
    and ZBnd: Z ⊆ ℤv ∩ Bounded-vec B
    and YBnd: Y ⊆ Bounded-vec B unfolding B-def by blast
  let ?P = {x ∈ carrier-vec n. A1 *v x ≤ b1 ∧ A2 *v x ≤ b2}
  let ?L = ?P ∩ {x. A2 *v x <v b2} ∩ indexed-Ints-vec I
  have polyhedron A b = {x ∈ carrier-vec n. A *v x ≤ b} unfolding polyhedron-def
by auto
  also have ... = ?P unfolding defs
    by (intro Collect-cong conj-cong refl append-rows-le[OF A1 A2 b1])
  finally have poly: ?P = convex-hull Y + cone Z unfolding poly ..
  have x ∈ ?P using x sol-nonstrict less-vec-lesseq-vec[OF sol-strict] by blast
  note sol = this[unfolded poly]
  from set-plus-elim[OF sol] obtain y z where xyz: x = y + z and
    yY: y ∈ convex-hull Y and zZ: z ∈ cone Z by auto
  from convex-hull-carrier[OF Y] yY have y: y ∈ carrier-vec n by auto
  from Caratheodory-theorem[OF Z] zZ
  obtain C where zC: z ∈ finite-cone C and CZ: C ⊆ Z and lin: lin-indpt C
by auto
  from subset-trans[OF CZ Z] lin have card: card C ≤ n
    using dim-is-n li-le-dim(2) by auto
  from finite-subset[OF CZ finX] have finC: finite C .
  from zC[unfolded finite-cone-def nonneg-lincomb-def] finC obtain a
    where za: z = lincomb a C and nonneg: ⋀ u. u ∈ C ⟹ a u ≥ 0 by auto
  from CZ Z have C: C ⊆ carrier-vec n by auto
  have z: z ∈ carrier-vec n using C unfolding za by auto
  have yB: y ∈ Bounded-vec B using yY convex-hull-bound[OF YBnd Y] by auto
  {
    fix D
    assume DC: D ⊆ C
    from finite-subset[OF this finC] have finite D .
    hence ∃ a. y + lincomb a C ∈ ?L ∧ (∀ c ∈ C. a c ≥ 0) ∧ (∀ c ∈ D. a c ≤ 1)
      using DC
    proof (induct D)
      case empty
      show ?case by (intro exI[of - a], fold za xyz, insert sol-strict x xI nonneg ‹x
∈ ?P›, auto)
    next
      case (insert c D)
      then obtain a where sol: y + lincomb a C ∈ ?L
        and a: (∀ c ∈ C. a c ≥ 0) and D: (∀ c ∈ D. a c ≤ 1) by auto
```

from *insert*(*4*) *C* **have** *c*: $c \in$ *carrier-vec n* **and** *cC*: $c \in C$ **by** *auto*
show *?case*
proof (*cases a c* $>$ *1*)
case *False*
thus *?thesis* **by** (*intro exI*[*of - a*], *insert sol a D*, *auto*)
next
case *True*
let *?z* = λ *d*. *lincomb a C* $-$ *d* $\cdot_v$ *c*
let *?x* = λ *d*. *y* + *?z d*
{
fix *d*
have *lin*: *lincomb a* ($C - \{c\}$) $\in$ *carrier-vec n* **using** *C* **by** *auto*
have *id*: *?z d* = *lincomb* (λ *e*. *if e* = *c then* (*a c* $-$ *d*) *else a e*) *C*
unfolding *lincomb-del2*[*OF finC C TrueI cC*]
by (*subst* (*2*) *lincomb-cong*[*OF refl*, *of - - a*], *insert C c lin*, *auto simp*: *diff-smult-distrib-vec*)
{
assume *le*: *d* $\leq$ *a c*
have *?z d* $\in$ *finite-cone C*
proof $-$
have $\forall f \in C$. *0* $\leq$ (λ*e*. *if e* = *c then a c* $-$ *d else a e*) *f* **using** *le a finC* **by** *simp*
then show *?thesis* **unfolding** *id* **using** *le a finC*
by (*simp add*: *C lincomb-in-finite-cone*)
qed
hence *?z d* $\in$ *cone Z* **using** *CZ*
using *finC local.cone-def* **by** *blast*
hence *?x d* $\in$ *?P* **unfolding** *poly*
by (*intro set-plus-intro*[*OF yY*], *auto*)
} note *sol* = *this*
{
fix *w* :: *'a vec*
assume *w*: $w \in$ *carrier-vec n*
have *w* $\cdot$ (*?x d*) = *w* $\cdot$ *y* + *w* $\cdot$ *lincomb a C* $-$ *d* $*$ (*w* $\cdot$ *c*)
by (*subst scalar-prod-add-distrib*[*OF w y*], (*insert C c*, *force*),
subst scalar-prod-minus-distrib[*OF w*], *insert w c C*, *auto*)
} note *scalar* = *this*
note *id sol scalar*
} note *generic* = *this*
let *?fl* = (*of-int* (*floor* (*a c*)) :: *'a*)
define *p* **where** *p* = (*if ?fl* = *a c then a c* $-$ *1 else ?fl*)
have *p-lt-ac*: *p* $<$ *a c* **unfolding** *p-def*
using *floor-less floor-of-int-eq* **by** *auto*
have *p1-ge-ac*: *p* + *1* $\geq$ *a c* **unfolding** *p-def*
using *floor-correct le-less* **by** *auto*
have *p1*: *p* $\geq$ *1* **using** *True* **unfolding** *p-def* **by** *auto*
define *a'* **where** *a'* = (λ*e*. *if e* = *c then a c* $-$ *p else a e*)
have *lin-id*: *lincomb a' C* = *lincomb a C* $-$ *p* $\cdot_v$ *c* **unfolding** *a'-def* **using** *id*

by (*simp add*: *generic*(*1*))
hence *1*: $y + lincomb\ a'\ C \in \{x \in carrier\text{-}vec\ n.\ A_1 *_v x \le b_1 \wedge A_2 *_v x \le b_2\}$
using *p-lt-ac generic*(*2*)[*of p*] **by** *auto*
have *pInt*: $p \in \mathbb{Z}$ **unfolding** *p-def* **using** *sol* **by** *auto*
have $C \subseteq$ *indexed-Ints-vec I* **using** *CZ ZBnd*
using *indexed-Ints-vec-subset* **by** *force*
hence $c \in$ *indexed-Ints-vec I* **using** *cC* **by** *auto*
hence *pvindInts*: $p \cdot_v c \in$ *indexed-Ints-vec I* **unfolding** *indexed-Ints-vec-def* **using** *pInt* **by** *simp*
have *prod*: $A_2 *_v (?x\ b) \in carrier\text{-}vec\ nr_2$ **for** *b* **using** *A2 C c y* **by** *auto*
have *2*: $y + lincomb\ a'\ C \in \{x.\ A_2 *_v x <_v b_2\}$ **unfolding** *lin-id*
proof (*intro less-vecI*[*OF prod b2*] *CollectI*)
fix *i*
assume *i*: $i < nr_2$
from *sol* **have** $A_2 *_v (?x\ 0) <_v b_2$ **using** *y C c* **by** *auto*
from *less-vecD*[*OF this b2 i*]
have *lt*: $row\ A_2\ i \cdot ?x\ 0 < b_2\ \$\ i$ **using** *A2 i* **by** *auto*
from *generic*(*2*)[*of a c*] *i A2*
have *le*: $row\ A_2\ i \cdot ?x\ (a\ c) \le b_2\ \$\ i$
unfolding *less-eq-vec-def* **by** *auto*
from *A2 i* **have** *A2icarr*: $row\ A_2\ i \in carrier\text{-}vec\ n$ **by** *auto*
have $row\ A_2\ i \cdot ?x\ p < b_2\ \$\ i$
proof −
define *lhs* **where** $lhs = row\ A_2\ i \cdot y + row\ A_2\ i \cdot lincomb\ a\ C - b_2\ \$\ i$
define *mult* **where** $mult = row\ A_2\ i \cdot c$
have *le2*: $lhs \le a\ c * mult$ **using** *le* **unfolding** *generic*(*3*)[*OF A2icarr*] *lhs-def mult-def* **by** *auto*
have *lt2*: $lhs < 0 * mult$ **using** *lt* **unfolding** *generic*(*3*)[*OF A2icarr*] *lhs-def* **by** *auto*
from *le2 lt2* **have** $lhs < p * mult$ **using** *p-lt-ac p1 True*
by (*smt dual-order.strict-trans linorder-neqE-linordered-idom mult-less-cancel-right not-less zero-less-one-class.zero-less-one*)
then show *?thesis* **unfolding** *generic*(*3*)[*OF A2icarr*] *lhs-def mult-def* **by** *auto*
qed
thus $(A_2 *_v ?x\ p)\ \$\ i < b_2\ \$\ i$ **using** *i A2* **by** *auto*
qed
have $y + lincomb\ a'\ C = y + lincomb\ a\ C - p \cdot_v c$
by (*subst lin-id*, *insert y C c*, *auto simp*: *add-diff-eq-vec*)
also have $\ldots \in$ *indexed-Ints-vec I* **using** *sol*
by(*intro diff-indexed-Ints-vec*[*OF - - - pvindInts*, *of - n*], *insert c C*, *auto*)
finally have *3*: $y + lincomb\ a'\ C \in$ *indexed-Ints-vec I* **by** *auto*
have *4*: $\forall c \in C.\ 0 \le a'\ c$ **unfolding** *a′-def p-def* **using** *p-lt-ac a* **by** *auto*
have *5*: $\forall c \in insert\ c\ D.\ a'\ c \le 1$ **unfolding** *a′-def* **using** *p1-ge-ac D p-def* **by** *auto*
show *?thesis*
by (*intro exI*[*of - a′*], *intro conjI IntI 1 2 3 4 5*)
qed

qed
}
from *this*[*of C*] **obtain** *a* **where**
sol: *y* + *lincomb a C* ∈ *?L* **and** *bnds*: (∀ *c* ∈ *C*. *a c* ≥ *0*) (∀ *c* ∈ *C*. *a c* ≤ *1*)
by *auto*
show *?thesis*
proof (*intro exI*[*of - y* + *lincomb a C*] *conjI*)
from *ZBnd CZ* **have** *BndC*: *C* ⊆ *Bounded-vec B* **and** *IntC*: $C \subseteq \mathbb{Z}_v$ **by** *auto*
have *lincomb a C* ∈ *Bounded-vec* (*of-nat n* ∗ *B*)
using *lincomb-card-bound*[*OF BndC C B0 - card*] *bnds* **by** *auto*
from *sum-in-Bounded-vecI*[*OF yB this y*] *C*
have *y* + *lincomb a C* ∈ *Bounded-vec* (*B* + *of-nat n* ∗ *B*) **by** *auto*
also have *B* + *of-nat n* ∗ *B* = *of-nat* (*n*+*1*) ∗ *B* **by** (*auto simp*: *field-simps*)
finally show *y* + *lincomb a C* ∈ *Bounded-vec* (*of-int* (*of-nat* (*n* + *1*) ∗ *db n*
(*max 1 Bnd*)))
unfolding *B-def* **by** *auto*
qed (*insert sol*, *auto*)
qed

We get rid of the max-1 operation, by showing that a smaller value of Bnd can only occur in very special cases where the theorem is trivially satisfied.

lemma *small-mixed-integer-solution*: **fixes** A_1 :: *'a mat*
assumes *db*: *is-det-bound db*
and *A1*: A_1 ∈ *carrier-mat* nr_1 *n*
and *A2*: A_2 ∈ *carrier-mat* nr_2 *n*
and *b1*: b_1 ∈ *carrier-vec* nr_1
and *b2*: b_2 ∈ *carrier-vec* nr_2
and *A1Bnd*: $A_1 \in \mathbb{Z}_m$ ∩ *Bounded-mat* (*of-int Bnd*)
and *b1Bnd*: $b_1 \in \mathbb{Z}_v$ ∩ *Bounded-vec* (*of-int Bnd*)
and *A2Bnd*: $A_2 \in \mathbb{Z}_m$ ∩ *Bounded-mat* (*of-int Bnd*)
and *b2Bnd*: $b_2 \in \mathbb{Z}_v$ ∩ *Bounded-vec* (*of-int Bnd*)
and *x*: *x* ∈ *carrier-vec n*
and *xI*: *x* ∈ *indexed-Ints-vec I*
and *sol-nonstrict*: $A_1 *_v x \leq b_1$
and *sol-strict*: $A_2 *_v x <_v b_2$
and *non-degenerate*: $nr_1 \neq 0 \vee nr_2 \neq 0$ ∨ *Bnd* ≥ *0*
shows ∃ *x*.
x ∈ *carrier-vec n* ∧
x ∈ *indexed-Ints-vec I* ∧
$A_1 *_v x \leq b_1$ ∧
$A_2 *_v x <_v b_2$ ∧
x ∈ *Bounded-vec* (*of-int* (*int* (*n*+*1*) ∗ *db n Bnd*))
proof (*cases Bnd* ≥ *1*)
case *True*
hence *max 1 Bnd* = *Bnd* **by** *auto*
with *small-mixed-integer-solution-main*[*OF assms*(*1*−*13*)] *True* **show** *?thesis* **by**
auto
next

case *trivial*: *False*
let *?oi* = *of-int* :: *int* ⇒ *'a*
show *?thesis*
proof (*cases n* = *0*)
case *True*
with *x* **have** *x* ∈ *Bounded-vec b* **for** *b* **unfolding** *Bounded-vec-def* **by** *auto*
with *xI x sol-nonstrict sol-strict* **show** *?thesis* **by** *blast*
next
case *n*: *False*
{
fix *A nr*
assume *A*: *A* ∈ *carrier-mat nr n* **and** *Bnd*: *A* ∈ $\mathbb{Z}_m$ ∩ *Bounded-mat* (*?oi Bnd*)
{
fix *i j*
assume *i* < *nr j* < *n*
with *Bnd A* **have** ∗: *A* $$ (*i*,*j*) ∈ ℤ *abs* (*A* $$ (*i*,*j*)) ≤ *?oi Bnd*
unfolding *Bounded-mat-def Ints-mat-def* **by** *auto*
from *Ints-nonzero-abs-less1*[*OF* ∗(*1*)] ∗(*2*) *trivial*
have *A* $$ (*i*,*j*) = *0*
by (*meson add-le-less-mono int-one-le-iff-zero-less less-add-same-cancel2 of-int-0-less-iff zero-less-abs-iff*)
with ∗(*2*) **have** *Bnd* ≥ *0 A* $$ (*i*,*j*) = *0* **by** *auto*
} **note** *main* = *this*
have *A0*: *A* = 0_m *nr n*
by (*intro eq-matI*, *insert main A*, *auto*)
have *nr* ≠ *0* ⟹ *Bnd* ≥ *0* **using** *main*[*of 0 0*] *n* **by** *auto*
note *A0 this*
} **note** *main* = *this*
from *main*[*OF A1 A1Bnd*] **have** *A1*: A_1 = 0_m nr_1 *n* **and** *nr1*: nr_1 ≠ *0* ⟹ *Bnd* ≥ *0*
by *auto*
from *main*[*OF A2 A2Bnd*] **have** *A2*: A_2 = 0_m nr_2 *n* **and** *nr2*: nr_2 ≠ *0* ⟹ *Bnd* ≥ *0*
by *auto*
let *?x* = 0_v *n*
show *?thesis*
proof (*intro exI*[*of - ?x*] *conjI*)
show A_1 $*_v$ *?x* ≤ b_1 **using** *sol-nonstrict x* **unfolding** *A1* **by** *auto*
show A_2 $*_v$ *?x* $<_v$ b_2 **using** *sol-strict x* **unfolding** *A2* **by** *auto*
show *?x* ∈ *carrier-vec n* **by** *auto*
show *?x* ∈ *indexed-Ints-vec I* **unfolding** *indexed-Ints-vec-def* **by** *auto*
from *nr1 nr2 non-degenerate* **have** *Bnd*: *Bnd* ≥ *0* **by** *auto*
from *is-det-bound-ge-zero*[*OF db Bnd*] **have** *db n Bnd* ≥ *0* **.**
hence *?oi* (*of-nat* (*n* + *1*) ∗ *db n Bnd*) ≥ *0* **by** *simp*
thus *?x* ∈ *Bounded-vec* (*?oi* (*of-nat* (*n* + *1*) ∗ *db n Bnd*)) **by** (*auto simp*: *Bounded-vec-def*)
qed
qed

qed

lemmas *small-mixed-integer-solution-hadamard* =
small-mixed-integer-solution[*OF det-bound-hadamard*, *unfolded det-bound-hadamard-def of-int-mult of-int-of-nat-eq*]

lemma *Bounded-vec-of-int*: **assumes** $v \in$ *Bounded-vec bnd*
shows (*map-vec of-int v* :: *'a vec*) $\in \mathbb{Z}_v \cap$ *Bounded-vec* (*of-int bnd*)
using *assms*
apply (*simp add*: *Ints-vec-vec-set Bounded-vec-vec-set Ints-def*)
apply (*intro conjI*, *force*)
apply (*clarsimp*)
subgoal for *x* **apply** (*elim ballE*[*of - - x*], *auto*)
by (*metis of-int-abs of-int-le-iff*)
done

lemma *Bounded-mat-of-int*: **assumes** $A \in$ *Bounded-mat bnd*
shows (*map-mat of-int A* :: *'a mat*) $\in \mathbb{Z}_m \cap$ *Bounded-mat* (*of-int bnd*)
using *assms*
apply (*simp add*: *Ints-mat-elements-mat Bounded-mat-elements-mat Ints-def*)
apply (*intro conjI*, *force*)
apply (*clarsimp*)
subgoal for *x* **apply** (*elim ballE*[*of - - x*], *auto*)
by (*metis of-int-abs of-int-le-iff*)
done

lemma *small-mixed-integer-solution-int-mat*: **fixes** *x* :: *'a vec*
assumes *db*: *is-det-bound db*
and *A1*: $A_1 \in$ *carrier-mat* nr_1 *n*
and *A2*: $A_2 \in$ *carrier-mat* nr_2 *n*
and *b1*: $b_1 \in$ *carrier-vec* nr_1
and *b2*: $b_2 \in$ *carrier-vec* nr_2
and *A1Bnd*: $A_1 \in$ *Bounded-mat Bnd*
and *b1Bnd*: $b_1 \in$ *Bounded-vec Bnd*
and *A2Bnd*: $A_2 \in$ *Bounded-mat Bnd*
and *b2Bnd*: $b_2 \in$ *Bounded-vec Bnd*
and *x*: $x \in$ *carrier-vec n*
and *xI*: $x \in$ *indexed-Ints-vec I*
and *sol-nonstrict*: *map-mat of-int* $A_1 *_v x \leq$ *map-vec of-int* b_1
and *sol-strict*: *map-mat of-int* $A_2 *_v x <_v$ *map-vec of-int* b_2
and *non-degenerate*: $nr_1 \neq 0 \vee nr_2 \neq 0 \vee Bnd \geq 0$
shows $\exists$ *x* :: *'a vec*.
$x \in$ *carrier-vec n* $\wedge$
$x \in$ *indexed-Ints-vec I* $\wedge$
map-mat of-int $A_1 *_v x \leq$ *map-vec of-int* $b_1 \wedge$
map-mat of-int $A_2 *_v x <_v$ *map-vec of-int* $b_2 \wedge$
$x \in$ *Bounded-vec* (*of-int* (*of-nat* $(n+1) *$ *db n Bnd*))
proof −
let *?oi* = *of-int* :: *int* $\Rightarrow$ *'a*

let *?A1* = *map-mat ?oi* A_1
let *?A2* = *map-mat ?oi* A_2
let *?b1* = *map-vec ?oi* b_1
let *?b2* = *map-vec ?oi* b_2
let *?Bnd* = *?oi Bnd*
from *A1* **have** *A1'*: *?A1* ∈ *carrier-mat* nr_1 *n* **by** *auto*
from *A2* **have** *A2'*: *?A2* ∈ *carrier-mat* nr_2 *n* **by** *auto*
from *b1* **have** *b1'*: *?b1* ∈ *carrier-vec* nr_1 **by** *auto*
from *b2* **have** *b2'*: *?b2* ∈ *carrier-vec* nr_2 **by** *auto*
from *small-mixed-integer-solution*[*OF db A1' A2' b1' b2'*
Bounded-mat-of-int[*OF A1Bnd*] *Bounded-vec-of-int*[*OF b1Bnd*]
Bounded-mat-of-int[*OF A2Bnd*] *Bounded-vec-of-int*[*OF b2Bnd*]
x xI sol-nonstrict sol-strict non-degenerate]
show *?thesis* **.**
qed

lemmas *small-mixed-integer-solution-int-mat-hadamard* =
small-mixed-integer-solution-int-mat[*OF det-bound-hadamard*, *unfolded det-bound-hadamard-def of-int-mult of-int-of-nat-eq*]

end

lemma *of-int-hom-le*: (*of-int-hom.vec-hom v* :: *'a* :: *linordered-field vec*) ≤ *of-int-hom.vec-hom w* ⟷ *v* ≤ *w*
unfolding *less-eq-vec-def* **by** *auto*

lemma *of-int-hom-less*: (*of-int-hom.vec-hom v* :: *'a* :: *linordered-field vec*) $<_v$ *of-int-hom.vec-hom w* ⟷ *v* $<_v$ *w*
unfolding *less-vec-def* **by** *auto*

lemma *Ints-vec-to-int-vec*: **assumes** *v* ∈ $\mathbb{Z}_v$
shows ∃ *w*. *v* = *map-vec of-int w*
proof −
have ∀ *i*. ∃ *x*. *i* < *dim-vec v* ⟶ *v* $ *i* = *of-int x*
using *assms* **unfolding** *Ints-vec-def Ints-def* **by** *auto*
from *choice*[*OF this*] **obtain** *x* **where** ⋀ *i*. *i* < *dim-vec v* ⟹ *v* $ *i* = *of-int* (*x i*)
by *auto*
thus *?thesis*
by (*intro exI*[*of - vec* (*dim-vec v*) *x*], *auto*)
qed

lemma *small-integer-solution*: **fixes** A_1 :: *int mat*
assumes *db*: *is-det-bound db*
and *A1*: A_1 ∈ *carrier-mat* nr_1 *n*
and *A2*: A_2 ∈ *carrier-mat* nr_2 *n*
and *b1*: b_1 ∈ *carrier-vec* nr_1
and *b2*: b_2 ∈ *carrier-vec* nr_2
and *A1Bnd*: A_1 ∈ *Bounded-mat Bnd*

and *b1Bnd*: $b_1 \in$ *Bounded-vec Bnd*
and *A2Bnd*: $A_2 \in$ *Bounded-mat Bnd*
and *b2Bnd*: $b_2 \in$ *Bounded-vec Bnd*
and *x*: $x \in$ *carrier-vec n*
and *sol-nonstrict*: $A_1 *_v x \leq b_1$
and *sol-strict*: $A_2 *_v x <_v b_2$
and *non-degenerate*: $nr_1 \neq 0 \vee nr_2 \neq 0 \vee Bnd \geq 0$
shows $\exists$ *x*.
$x \in$ *carrier-vec n* $\wedge$
$A_1 *_v x \leq b_1 \wedge$
$A_2 *_v x <_v b_2 \wedge$
$x \in$ *Bounded-vec* (*of-nat* (*n+1*) $*$ *db n Bnd*)
proof −
let *?oi* = *rat-of-int*
let *?x* = *map-vec ?oi x*
let *?oiM* = *map-mat ?oi*
let *?oiv* = *map-vec ?oi*
from *x* **have** *xx*: *?x* $\in$ *carrier-vec n* **by** *auto*
have *Int*: *?x* $\in$ *indexed-Ints-vec UNIV* **unfolding** *indexed-Ints-vec-def Ints-def* **by** *auto*
interpret *gram-schmidt-floor n TYPE*(*rat*) **.**
from
small-mixed-integer-solution-int-mat[*OF db A1 A2 b1 b2 A1Bnd b1Bnd A2Bnd b2Bnd xx Int*
- - *non-degenerate*,
folded of-int-hom.mult-mat-vec-hom[*OF A1 x*] *of-int-hom.mult-mat-vec-hom*[*OF A2 x*],
unfolded of-int-hom-less of-int-hom-le, *OF sol-nonstrict sol-strict*, *folded indexed-Ints-vec-UNIV*]
obtain *x* **where**
x: $x \in$ *carrier-vec n* **and**
xI: $x \in \mathbb{Z}_v$ **and**
le: *?oiM* $A_1 *_v x \leq$ *?oiv* b_1 **and**
less: *?oiM* $A_2 *_v x <_v$ *?oiv* b_2 **and**
Bnd: $x \in$ *Bounded-vec* (*?oi* (*int* (*n* + *1*) $*$ *db n Bnd*))
by *blast*
from *Ints-vec-to-int-vec*[*OF xI*] **obtain** *xI* **where** *xI*: *x* = *?oiv xI* **by** *auto*
from *x*[*unfolded xI*] **have** *x*: *xI* $\in$ *carrier-vec n* **by** *auto*
from *le*[*unfolded xI*, *folded of-int-hom.mult-mat-vec-hom*[*OF A1 x*], *unfolded of-int-hom-le*]
have *le*: $A_1 *_v xI \leq b_1$ **.**
from *less*[*unfolded xI*, *folded of-int-hom.mult-mat-vec-hom*[*OF A2 x*], *unfolded of-int-hom-less*]
have *less*: $A_2 *_v xI <_v b_2$ **.**
show *?thesis*
proof (*intro exI*[*of - xI*] *conjI x le less*)
show *xI* $\in$ *Bounded-vec* (*int* (*n* + *1*) $*$ *db n Bnd*)
unfolding *Bounded-vec-def*
proof *clarsimp*

```
      fix i
      assume i: i < dim-vec xI
      with Bnd[unfolded Bounded-vec-def]
      have |x $ i| ≤ ?oi (int (n + 1) * db n Bnd) by (auto simp: xI)
      also have |x $ i| = ?oi (|xI $ i|) unfolding xI using i by simp
      finally show |xI $ i| ≤ (1 + int n) * db n Bnd unfolding of-int-le-iff by
auto
    qed
  qed
qed

corollary small-integer-solution-nonstrict: fixes A :: int mat
  assumes db: is-det-bound db
    and A: A ∈ carrier-mat nr n
    and b: b ∈ carrier-vec nr
    and ABnd: A ∈ Bounded-mat Bnd
    and bBnd: b ∈ Bounded-vec Bnd
    and x: x ∈ carrier-vec n
    and sol: A *_v x ≤ b
    and non-degenerate: nr ≠ 0 ∨ Bnd ≥ 0
  shows ∃ y.
  y ∈ carrier-vec n ∧
  A *_v y ≤ b ∧
  y ∈ Bounded-vec (of-nat (n+1) * db n Bnd)
proof −
  let ?A2 = 0_m 0 n :: int mat
  let ?b2 = 0_v 0 :: int vec
  from non-degenerate have degen: nr ≠ 0 ∨ (0 :: nat) ≠ 0 ∨ Bnd ≥ 0 by auto
  have ∃ y. y ∈ carrier-vec n ∧ A *_v y ≤ b ∧ ?A2 *_v y <_v ?b2
  ∧ y ∈ Bounded-vec (of-nat (n+1) * db n Bnd)
    apply (rule small-integer-solution[OF db A - b - ABnd bBnd - - x sol - degen])
    by (auto simp: Bounded-mat-def Bounded-vec-def less-vec-def)
  thus ?thesis by blast
qed

lemmas small-integer-solution-nonstrict-hadamard =
 small-integer-solution-nonstrict[OF det-bound-hadamard, unfolded det-bound-hadamard-def]


end
```

16 Integer Hull

We define the integer hull of a polyhedron, i.e., the convex hull of all integer solutions. Moreover, we prove the result of Meyer that the integer hull of a polyhedron defined by an integer matrix is again a polyhedron, and give bounds for a corresponding decomposition theorem.

theory *Integer-Hull*

imports
Decomposition-Theorem
Mixed-Integer-Solutions
begin

context *gram-schmidt*
begin
definition *integer-hull P* = *convex-hull* (*P* ∩ $\mathbb{Z}_v$)

lemma *integer-hull-mono*: *P* ⊆ *Q* ⟹ *integer-hull P* ⊆ *integer-hull Q*
unfolding *integer-hull-def*
by (*intro convex-hull-mono*, *auto*)

end

lemma *abs-neg-floor*: |*of-int b*| ≤ *Bnd* ⟹ − (*floor Bnd*) ≤ *b*
using *abs-le-D2 floor-mono* **by** *fastforce*

lemma *abs-pos-floor*: |*of-int b*| ≤ *Bnd* ⟹ *b* ≤ *floor Bnd*
using *abs-le-D1 le-floor-iff* **by** *auto*

context *gram-schmidt-floor*
begin

lemma *integer-hull-integer-cone*: **assumes** *C*: *C* ⊆ *carrier-vec n*
and *CI*: *C* ⊆ $\mathbb{Z}_v$
shows *integer-hull* (*cone C*) = *cone C*
proof
have *cone C* ∩ $\mathbb{Z}_v$ ⊆ *cone C* **by** *blast*
thus *integer-hull* (*cone C*) ⊆ *cone C*
using *cone-cone*[*OF C*] *convex-cone*[*OF C*] *convex-hull-mono*
unfolding *integer-hull-def convex-def* **by** *metis*
{
fix *x*
assume *x* ∈ *cone C*
then obtain *D* **where** *finD*: *finite D* **and** *DC*: *D* ⊆ *C* **and** *x*: *x* ∈ *finite-cone D*
unfolding *cone-def* **by** *auto*
from *DC C CI* **have** *D*: *D* ⊆ *carrier-vec n* **and** *DI*: *D* ⊆ $\mathbb{Z}_v$ **by** *auto*
from *D x finD* **have** *x* ∈ *finite-cone* (*D* ∪ {*0*$_v$ *n*}) **using** *finite-cone-mono*[*of D* ∪ {*0*$_v$ *n*} *D*] **by** *auto*
then obtain *l* **where** *x*: *lincomb l* (*D* ∪ {*0*$_v$ *n*}) = *x*
and *l*: *l* ‘ (*D* ∪ {*0*$_v$ *n*}) ⊆ {*t*. *t* ≥ *0*}
using *finD* **unfolding** *finite-cone-def nonneg-lincomb-def* **by** *auto*
define *L* **where** *L* = *sum l* (*D* ∪ {*0*$_v$ *n*})
define *L-sup* :: *'a* **where** *L-sup* = *of-int* (*floor L* + *1*)
have *L-sup* ≥ *L* **using** *floor-correct*[*of L*] **unfolding** *L-sup-def* **by** *linarith*
have *L* ≥ *0* **unfolding** *L-def* **using** *sum-nonneg*[*of - l*] *l* **by** *blast*
hence *L-sup* ≥ *1* **unfolding** *L-sup-def* **by** *simp*

hence *L-sup* > *0* **by** *fastforce*

let *?f* = λ *y*. *if y* = 0_v *n then L-sup* − *L else 0*
have *lincomb ?f* {0_v *n*} = 0_v *n*
using *already-in-span*[*of* {} 0_v *n*] *lincomb-in-span local.span-empty*
by *auto*
moreover have *lincomb ?f* (*D* − {0_v *n*}) = 0_v *n*
by(*rule lincomb-zero*, *insert D*, *auto*)
ultimately have *lincomb ?f* (*D* ∪ {0_v *n*}) = 0_v *n*
using *lincomb-vec-diff-add*[*of D* ∪ {0_v *n*} {0_v *n*}] *D finD* **by** *simp*
hence *lcomb-f*: *lincomb* (λ *y*. *l y* + *?f y*) (*D* ∪ {0_v *n*}) = *x*
using *lincomb-sum*[*of D* ∪ {0_v *n*} *l ?f*] *finD D x* **by** *simp*
have *sum ?f* (*D* ∪ {0_v *n*}) = *L-sup* − *L*
by (*simp add*: *sum.subset-diff*[*of* {0_v *n*} *D* ∪ {0_v *n*} *?f*] *finD*)
hence *sum* (λ *y*. *l y* + *?f y*) (*D* ∪ {0_v *n*}) = *L-sup*
using *l L-def* **by** *auto*
moreover have (λ *y*. *l y* + *?f y*) ‘ (*D* ∪ {0_v *n*}) ⊆ {*t*. *t* ≥ *0*}
using ‹*L* ≤ *L-sup*› *l* **by** *force*
ultimately obtain *l′* **where** *x*: *lincomb l′* (*D* ∪ {0_v *n*}) = *x*
and *l′*: *l′* ‘ (*D* ∪ {0_v *n*}) ⊆ {*t*. *t* ≥ *0*}
and *sum-l′*: *sum l′* (*D* ∪ {0_v *n*}) = *L-sup*
using *lcomb-f* **by** *blast*

let *?D′* = {*L-sup* $\cdot_v$ *v* | *v*. *v* ∈ *D* ∪ {0_v *n*}}
have *Did*: *?D′* = (λ *v*. *L-sup* $\cdot_v$ *v*) ‘ (*D* ∪ {0_v *n*}) **by** *force*
define *l′′* **where** *l′′* = (λ *y*. *l′* ((*1* / *L-sup*) $\cdot_v$ *y*) / *L-sup*)
obtain *lD* **where** *dist*: *distinct lD* **and** *lD*: *D* ∪ {0_v *n*} = *set lD*
using *finite-distinct-list*[*of D* ∪ {0_v *n*}] *finD* **by** *auto*
let *?lD′* = *map* (($\cdot_v$) *L-sup*) *lD*
have *dist′*: *distinct ?lD′*
using *distinct-smult-nonneg*[*OF - dist*] ‹*L-sup* > *0*› **by** *fastforce*

have *x′*: *lincomb l′′ ?D′* = *x* **unfolding** *x*[*symmetric*] *l′′-def*
unfolding *lincomb-def Did*
proof (*subst finsum-reindex*)
from ‹*L-sup* > *0*› *smult-vec-nonneg-eq*[*of L-sup*] **show** *inj-on* (($\cdot_v$) *L-sup*) (*D* ∪ {0_v *n*})
by (*auto simp*: *inj-on-def*)
show (λ*v*. *l′* (*1* / *L-sup* $\cdot_v$ *v*) / *L-sup* $\cdot_v$ *v*) ∈ ($\cdot_v$) *L-sup* ‘ (*D* ∪ {0_v *n*}) → *carrier-vec n*
using *D* **by** *auto*
from ‹*L-sup* > *0*› **have** *L-sup* ≠ *0* **by** *auto*
then show ($\bigoplus_V$*x*∈*D* ∪ {0_v *n*}. *l′* (*1* / *L-sup* $\cdot_v$ (*L-sup* $\cdot_v$ *x*)) / *L-sup* $\cdot_v$ (*L-sup* $\cdot_v$ *x*)) =
($\bigoplus_V$*v*∈*D* ∪ {0_v *n*}. *l′ v* $\cdot_v$ *v*)
by (*intro finsum-cong*, *insert D*, *auto simp*: *smult-smult-assoc*)
qed
have *D* ∪ {0_v *n*} ⊆ *cone C* **using** *set-in-cone*[*OF C*] *DC zero-in-cone* **by** *blast*
hence *D′*: *?D′* ⊆ *cone C* **using** *cone-smult*[*of L-sup*, *OF - C*] ‹*L-sup* > *0*› **by**

auto
have $D \cup \{0_v\ n\} \subseteq \mathbb{Z}_v$ **unfolding** *zero-vec-def* **using** *DI Ints-vec-def* **by** *auto*
moreover have *L-sup* $\in \mathbb{Z}$ **unfolding** *L-sup-def* **by** *auto*
ultimately have $D'I$: $?D' \subseteq \mathbb{Z}_v$ **unfolding** *Ints-vec-def* **by** *force*

have $1 = sum\ l'\ (D \cup \{0_v\ n\}) * (1\ /\ L\text{-}sup)$ **using** *sum-l'* ‹*L-sup* > *0*› **by** *auto*
also have $sum\ l'\ (D \cup \{0_v\ n\}) = sum\text{-}list\ (map\ l'\ lD)$
using *sum.distinct-set-conv-list*[*OF dist*] *lD* **by** *auto*
also have $map\ l'\ lD = map\ (l' \circ ((\cdot_v)\ (1\ /\ L\text{-}sup)))\ ?lD'$
using *smult-smult-assoc*[*of 1 / L-sup L-sup*] ‹*L-sup* > *0*›
by (*simp add*: *comp-assoc*)
also have $l' \circ ((\cdot_v)\ (1\ /\ L\text{-}sup)) = (\lambda\ x.\ l'\ ((1\ /\ L\text{-}sup) \cdot_v x))$ **by** (*rule comp-def*)
also have $sum\text{-}list\ (map\ \ldots\ ?lD') * (1\ /\ L\text{-}sup) =$
$sum\text{-}list\ (map\ (\lambda y.\ l'\ (1\ /\ L\text{-}sup \cdot_v y) * (1\ /\ L\text{-}sup))\ ?lD')$
using *sum-list-mult-const*[*of - 1 / L-sup ?lD'*] **by** *presburger*
also have $\ldots = sum\text{-}list\ (map\ l''\ ?lD')$
unfolding *l''-def* **using** ‹*L-sup* > *0*› **by** *simp*
also have $\ldots = sum\ l''\ (set\ ?lD')$ **using** *sum.distinct-set-conv-list*[*OF dist'*] **by** *metis*
also have $set\ ?lD' = ?D'$ **using** *lD* **by** *auto*
finally have *sum-l'*: $sum\ l''\ ?D' = 1$ **by** *auto*

moreover have $l''\ `\ ?D' \subseteq \{t.\ t \geq 0\}$
proof
fix y
assume $y \in l''\ `\ ?D'$
then obtain x **where** y: $y = l''\ x$ **and** $x \in ?D'$ **by** *blast*
then obtain v **where** $v \in D \cup \{0_v\ n\}$ **and** x: $x = L\text{-}sup \cdot_v v$ **by** *blast*
hence $0 \leq l'\ v\ /\ L\text{-}sup$ **using** l' ‹*L-sup* > *0*› **by** *fastforce*
also have $\ldots = l''\ x$ **unfolding** *x l''-def*
using *smult-smult-assoc*[*of 1 / L-sup L-sup v*] ‹*L-sup* > *0*› **by** *simp*
finally show $y \in \{t.\ t \geq 0\}$ **using** *y* **by** *blast*
qed
moreover have *finite ?D'* **using** *finD* **by** *simp*

ultimately have $x \in integer\text{-}hull\ (cone\ C)$
unfolding *integer-hull-def convex-hull-def*
using *x' D' D'I convex-lincomb-def*[*of l'' ?D' x*]
nonneg-lincomb-def[*of l'' ?D' x*] **by** *fast*
}
thus $cone\ C \subseteq integer\text{-}hull\ (cone\ C)$ **by** *blast*
qed

theorem *decomposition-theorem-integer-hull-of-polyhedron*:
assumes *db*: *is-det-bound db*
and *A*: $A \in carrier\text{-}mat\ nr\ n$
and *b*: $b \in carrier\text{-}vec\ nr$

and *AI*: $A \in \mathbb{Z}_m$
and *bI*: $b \in \mathbb{Z}_v$
and *P*: *P* = *polyhedron A b*
and *Bnd*: *of-int Bnd* ≥ *Max* (*abs* ‘ (*elements-mat A* ∪ *vec-set b*))
shows ∃ *H C*. *H* ∪ *C* ⊆ *carrier-vec n* ∩ $\mathbb{Z}_v$
∧ *H* ⊆ *Bounded-vec* (*of-nat* (*n* + *1*) ∗ *of-int* (*db n* (*max 1 Bnd*)))
∧ *C* ⊆ *Bounded-vec* (*of-int* (*db n* (*max 1 Bnd*)))
∧ *finite* (*H* ∪ *C*)
∧ *integer-hull P* = *convex-hull H* + *cone C*
proof −
define *MBnd* **where** *MBnd* = *Max* (*abs* ‘ (*elements-mat A* ∪ set_v *b*))
define *DBnd* :: *'a* **where** *DBnd* = *of-int* (*db n* (*max 1 Bnd*))
define *nBnd* **where** *nBnd* = *of-nat* (*n*+*1*) ∗ *DBnd*
have *DBnd0*: *DBnd* ≥ *0* **unfolding** *DBnd-def of-int-0-le-iff*
by (*rule is-det-bound-ge-zero*[*OF db*], *auto*)
have *Pn*: *P* ⊆ *carrier-vec n* **unfolding** *P polyhedron-def* **by** *auto*
have *A* ∈ *Bounded-mat MBnd* ∧ *b* ∈ *Bounded-vec MBnd*
unfolding *MBnd-def Bounded-mat-elements-mat Bounded-vec-vec-set*
by (*intro ballI conjI Max-ge finite-imageI imageI finite-UnI*, *auto*)
hence *A* ∈ *Bounded-mat* (*of-int Bnd*) ∧ *b* ∈ *Bounded-vec* (*of-int Bnd*)
using *Bounded-mat-mono*[*OF Bnd*] *Bounded-vec-mono*[*OF Bnd*] **unfolding** *MBnd-def* **by** *auto*
from *decomposition-theorem-polyhedra-1*[*OF A b P*, *of db Bnd*] *db AI bI this*
obtain *QQ Q C* **where** *C*: *C* ⊆ *carrier-vec n* **and** *finC*: *finite C*
and *QQ*: *QQ* ⊆ *carrier-vec n* **and** *finQ*: *finite QQ* **and** *BndQQ*: *QQ* ⊆ *Bounded-vec DBnd*
and *P*: *P* = *Q* + *cone C*
and *Q-def*: *Q* = *convex-hull QQ*
and *CI*: *C* ⊆ $\mathbb{Z}_v$ **and** *BndC*: *C* ⊆ *Bounded-vec DBnd*
by (*auto simp*: *DBnd-def*)
define *Bnd'* **where** *Bnd'* = *of-nat n* ∗ *DBnd*
note *coneC* = *cone-iff-finite-cone*[*OF C finC*]
have *Q*: *Q* ⊆ *carrier-vec n* **unfolding** *Q-def* **using** *convex-hull-carrier*[*OF QQ*]
.
define *B* **where** *B* = {*x*. ∃ *a D*. *nonneg-lincomb a D x* ∧ *D* ⊆ *C* ∧ *lin-indpt D* ∧ (∀ *d* ∈ *D*. *a d* ≤ *1*)}
{
fix *b*
assume *b* ∈ *B*
then obtain *a D* **where** *b*: *b* = *lincomb a D* **and** *DC*: *D* ⊆ *C*
and *linD*: *lin-indpt D* **and** *bnd-a*: ∀ *d* ∈ *D*. *0* ≤ *a d* ∧ *a d* ≤ *1*
by (*force simp*: *B-def nonneg-lincomb-def*)
from *DC C* **have** *D*: *D* ⊆ *carrier-vec n* **by** *auto*
from *DC finC* **have** *finD*: *finite D* **by** (*metis finite-subset*)
from *D linD finD* **have** *cardD*: *card D* ≤ *n* **using** *dim-is-n li-le-dim*(*2*) **by** *auto*
from *BndC DC* **have** *BndD*: *D* ⊆ *Bounded-vec DBnd* **by** *auto*
from *lincomb-card-bound*[*OF this D DBnd0 - cardD*, *of a*, *folded b*] *bnd-a*
have *b* ∈ *Bounded-vec Bnd'* **unfolding** *Bnd'-def* **by** *force*
}

hence *BndB*: $B \subseteq$ *Bounded-vec Bnd'* **..**
from *BndQQ* **have** *BndQ*: $Q \subseteq$ *Bounded-vec DBnd* **unfolding** *Q-def* **using** *QQ* **by** (*metis convex-hull-bound*)
have *B*: $B \subseteq$ *carrier-vec n*
unfolding *B-def nonneg-lincomb-def* **using** *C* **by** *auto*
from *Q B* **have** *QB*: $Q + B \subseteq$ *carrier-vec n* **by** (*auto elim*!: *set-plus-elim*)
from *sum-in-Bounded-vecI*[*of - DBnd - Bnd' n*] *BndQ BndB B Q*
have $Q + B \subseteq$ *Bounded-vec* (*DBnd* + *Bnd'*) **by** (*auto elim*!: *set-plus-elim*)
also have *DBnd* + *Bnd'* = *nBnd* **unfolding** *nBnd-def Bnd'-def* **by** (*simp add*: *algebra-simps*)
finally have *QB-Bnd*: $Q + B \subseteq$ *Bounded-vec nBnd* **by** *blast*
have *finQBZ*: *finite* $((Q + B) \cap \mathbb{Z}_v)$
proof (*rule finite-subset*[*OF subsetI*])
define *ZBnd* **where** *ZBnd* = *floor nBnd*
let *?vecs* = *set* (*map vec-of-list* (*concat-lists* (*map* (λ *i*. *map* (*of-int* :: - $\Rightarrow$ *'a*) [$-$*ZBnd*..*ZBnd*]) [*0*..<*n*])))
have *id*: *?vecs* = *vec-of-list* ‘
{*as*. *length as* = *n* $\wedge$ ($\forall$ *i*<*n*. $\exists$ *b*. *as* ! *i* = *of-int b* $\wedge$ *b* $\in$ {$-$ *ZBnd*..*ZBnd*}})
unfolding *set-map* **by** (*rule image-cong*, *auto*)
show *finite ?vecs* **by** (*rule finite-set*)
fix *x*
assume $x \in (Q + B) \cap \mathbb{Z}_v$
hence *xQB*: $x \in Q + B$ **and** *xI*: $x \in \mathbb{Z}_v$ **by** *auto*
from *xQB QB-Bnd QB* **have** *xBnd*: $x \in$ *Bounded-vec nBnd* **and** *x*: $x \in$ *carrier-vec n* **by** *auto*
have *xid*: *x* = *vec-of-list* (*list-of-vec x*) **by** *auto*
show $x \in$ *?vecs* **unfolding** *id*
proof (*subst xid*, *intro imageI CollectI conjI allI impI*)
show *length* (*list-of-vec x*) = *n* **using** *x* **by** *auto*
fix *i*
assume *i*: $i < n$
have *id*: *list-of-vec x* ! *i* = *x* \$ *i* **using** *i x* **by** *auto*
from *xBnd*[*unfolded Bounded-vec-def*] *i x* **have** *xiBnd*: *abs* (*x* \$ *i*) $\leq$ *nBnd* **by** *auto*
from *xI*[*unfolded Ints-vec-def*] *i x* **have** *x* \$ *i* $\in \mathbb{Z}$ **by** *auto*
then obtain *b* **where** *b*: *x* \$ *i* = *of-int b* **unfolding** *Ints-def* **by** *blast*
show $\exists$ *b*. *list-of-vec x* ! *i* = *of-int b* $\wedge$ *b* $\in$ {$-$ *ZBnd*..*ZBnd*} **unfolding** *id ZBnd-def*
using *xiBnd* **unfolding** *b* **by** (*intro exI*[*of - b*], *auto intro*!: *abs-neg-floor abs-pos-floor*)
qed
qed
have *QBZ*: $(Q + B) \cap \mathbb{Z}_v \subseteq$ *carrier-vec n* **using** *QB* **by** *auto*
from *decomposition-theorem-polyhedra-2*[*OF QBZ finQBZ*, *folded integer-hull-def*, *OF C finC refl*]
obtain *A' b' nr'* **where** *A'*: *A'* $\in$ *carrier-mat nr' n* **and** *b'*: *b'* $\in$ *carrier-vec nr'*
and *IH*: *integer-hull* (*Q* + *B*) + *cone C* = *polyhedron A' b'*
by *auto*
{

```
fix p
assume p ∈ P ∩ ℤ_v
hence pI: p ∈ ℤ_v and p: p ∈ Q + cone C unfolding P by auto
from set-plus-elim[OF p] obtain q c where
  pqc: p = q + c and qQ: q ∈ Q and cC: c ∈ cone C by auto
from qQ Q have q: q ∈ carrier-vec n by auto
from Caratheodory-theorem[OF C] cC
obtain D where cD: c ∈ finite-cone D and DC: D ⊆ C and linD: lin-indpt
D by auto
from DC C have D: D ⊆ carrier-vec n by auto
from DC finC have finD: finite D by (metis finite-subset)
from cD finD
obtain a where nonneg-lincomb a D c unfolding finite-cone-def by auto
hence caD: c = lincomb a D and a0: ⋀ d. d ∈ D ⟹ a d ≥ 0
  unfolding nonneg-lincomb-def by auto
define a1 where a1 = (λ c. a c − of-int (floor (a c)))
define a2 where a2 = (λ c. of-int (floor (a c)) :: 'a)
define d where d = lincomb a2 D
define b where b = lincomb a1 D
have b: b ∈ carrier-vec n and d: d ∈ carrier-vec n unfolding d-def b-def using
D by auto
have bB: b ∈ B unfolding B-def b-def nonneg-lincomb-def
proof (intro CollectI exI[of - a1] exI[of - D] conjI ballI refl subsetI linD)
  show x ∈ a1 ‘ D ⟹ 0 ≤ x for x using a0 unfolding a1-def by auto
  show a1 c ≤ 1 for c unfolding a1-def by linarith
qed (insert DC, auto)
have cbd: c = b + d unfolding b-def d-def caD lincomb-sum[OF finD D,
symmetric]
  by (rule lincomb-cong[OF refl D], auto simp: a1-def a2-def)
have nonneg-lincomb a2 D d unfolding d-def nonneg-lincomb-def
  by (intro allI conjI refl subsetI, insert a0, auto simp: a2-def)
hence dC: d ∈ cone C unfolding cone-def finite-cone-def using finC finD DC
by auto
have p: p = (q + b) + d unfolding pqc cbd using q b d by auto
have dI: d ∈ ℤ_v using CI DC C unfolding d-def indexed-Ints-vec-UNIV
  by (intro lincomb-indexed-Ints-vec, auto simp: a2-def)
from diff-indexed-Ints-vec[of - - - UNIV, folded indexed-Ints-vec-UNIV, OF -
d pI dI, unfolded p]
have q + b + d − d ∈ ℤ_v using q b d by auto
also have q + b + d − d = q + b using q b d by auto
finally have qbI: q + b ∈ ℤ_v by auto
have p ∈ integer-hull (Q + B) + cone C unfolding p integer-hull-def
  by (intro set-plus-intro dC set-mp[OF set-in-convex-hull] IntI qQ bB qbI,
insert Q B,
      auto elim!: set-plus-elim)
}
hence P ∩ ℤ_v ⊆ integer-hull (Q + B) + cone C by auto
hence one-dir: integer-hull P ⊆ integer-hull (Q + B) + cone C unfolding IH
unfolding integer-hull-def using convex-convex-hull[OF polyhedra-are-convex[OF
```

A′ b′ refl]]
convex-hull-mono **by** *blast*
have *integer-hull* (*Q* + *B*) + *cone C* ⊆ *integer-hull P* + *cone C* **unfolding** *P*
proof (*intro set-plus-mono2 subset-refl integer-hull-mono*)
show *B* ⊆ *cone C* **unfolding** *B-def cone-def finite-cone-def* **using** *finite-subset*[*OF - finC*] **by** *auto*
qed
also have ... = *integer-hull P* + *integer-hull* (*cone C*)
using *integer-hull-integer-cone*[*OF C CI*] **by** *simp*
also have ... = *convex-hull* ($P \cap \mathbb{Z}_v$) + *convex-hull* (*cone C* $\cap \mathbb{Z}_v$)
unfolding *integer-hull-def* **by** *simp*
also have ... = *convex-hull* (($P \cap \mathbb{Z}_v$) + (*cone C* $\cap \mathbb{Z}_v$))
by (*rule convex-hull-sum*[*symmetric*], *insert Pn cone-carrier*[*OF C*], *auto*)
also have ... ⊆ *convex-hull* ((*P* + *cone C*) $\cap \mathbb{Z}_v$)
proof (*rule convex-hull-mono*)
show $P \cap \mathbb{Z}_v$ + *cone C* $\cap \mathbb{Z}_v$ ⊆ (*P* + *cone C*) $\cap \mathbb{Z}_v$
using *add-indexed-Ints-vec*[*of - n - UNIV*, *folded indexed-Ints-vec-UNIV*] *cone-carrier*[*OF C*] *Pn*
by (*auto elim*!: *set-plus-elim*)
qed
also have ... = *integer-hull* (*P* + *cone C*) **unfolding** *integer-hull-def* **..**
also have *P* + *cone C* = *P*
proof −
have *CC*: *cone C* ⊆ *carrier-vec n* **using** *C* **by** (*rule cone-carrier*)
have *P* + *cone C* = *Q* + (*cone C* + *cone C*) **unfolding** *P*
by (*rule assoc-add-vecset*[*symmetric*, *OF Q CC CC*])
also have *cone C* + *cone C* = *cone C* **by** (*rule cone-add-cone*[*OF C*])
finally show *?thesis* **unfolding** *P* **.**
qed
finally have *integer-hull* (*Q* + *B*) + *cone C* ⊆ *integer-hull P* **.**
with *one-dir* **have** *id*: *integer-hull P* = *integer-hull* (*Q* + *B*) + *cone C* **by** *auto*
show *?thesis* **unfolding** *id* **unfolding** *integer-hull-def DBnd-def*[*symmetric*] *nBnd-def*[*symmetric*]
proof (*rule exI*[*of -* (*Q* + *B*) $\cap \mathbb{Z}_v$], *intro exI*[*of - C*] *conjI refl BndC*)
from *QB-Bnd* **show** (*Q* + *B*) $\cap \mathbb{Z}_v$ ⊆ *Bounded-vec nBnd* **by** *auto*
show (*Q* + *B*) $\cap \mathbb{Z}_v \cup C$ ⊆ *carrier-vec n* $\cap \mathbb{Z}_v$
using *QB C CI* **by** *auto*
show *finite* ((*Q* + *B*) $\cap \mathbb{Z}_v \cup C$) **using** *finQBZ finC* **by** *auto*
qed
qed

corollary *integer-hull-of-polyhedron*: **assumes** *A*: *A* ∈ *carrier-mat nr n*
and *b*: *b* ∈ *carrier-vec nr*
and *AI*: $A \in \mathbb{Z}_m$
and *bI*: $b \in \mathbb{Z}_v$
and *P*: *P* = *polyhedron A b*
shows ∃ *A′ b′ nr′*. *A′* ∈ *carrier-mat nr′ n* ∧ *b′* ∈ *carrier-vec nr′* ∧
integer-hull P = *polyhedron A′ b′*
proof −

obtain *Bnd* **where** *Bnd*: $Max\ (abs\ `\ (elements\text{-}mat\ A \cup set_v\ b)) \leq of\text{-}int\ Bnd$
by (*meson ex-le-of-int*)
from *decomposition-theorem-integer-hull-of-polyhedron*[*OF det-bound-fact A b AI bI P Bnd*]
obtain *H C*
where *HC*: $H \cup C \subseteq carrier\text{-}vec\ n \cap \mathbb{Z}_v$ *finite* $(H \cup C)$
and *decomp*: *integer-hull P* = *convex-hull H* + *cone C* **by** *auto*
show *?thesis*
by (*rule decomposition-theorem-polyhedra-2*[*OF - - - - decomp*], *insert HC*, *auto*)
qed

corollary *small-integer-solution-nonstrict-via-decomp*: **fixes** *A* :: *'a mat*
assumes *db*: *is-det-bound db*
and *A*: $A \in carrier\text{-}mat\ nr\ n$
and *b*: $b \in carrier\text{-}vec\ nr$
and *AI*: $A \in \mathbb{Z}_m$
and *bI*: $b \in \mathbb{Z}_v$
and *Bnd*: $of\text{-}int\ Bnd \geq Max\ (abs\ `\ (elements\text{-}mat\ A \cup vec\text{-}set\ b))$
and *x*: $x \in carrier\text{-}vec\ n$
and *xI*: $x \in \mathbb{Z}_v$
and *sol*: $A *_v x \leq b$
shows $\exists\ y.$
$y \in carrier\text{-}vec\ n \wedge$
$y \in \mathbb{Z}_v \wedge$
$A *_v y \leq b \wedge$
$y \in Bounded\text{-}vec\ (of\text{-}nat\ (n+1) * of\text{-}int\ (db\ n\ (max\ 1\ Bnd)))$
proof −
from *x sol* **have** $x \in polyhedron\ A\ b$ **unfolding** *polyhedron-def* **by** *auto*
with *xI x* **have** *xsol*: $x \in integer\text{-}hull\ (polyhedron\ A\ b)$ **unfolding** *integer-hull-def*
by (*meson IntI convex-hull-mono in-mono inf-sup-ord*(*1*) *inf-sup-ord*(*2*) *set-in-convex-hull*)
from *decomposition-theorem-integer-hull-of-polyhedron*[*OF db A b AI bI refl Bnd*]
obtain *H C* **where** *HC*: $H \cup C \subseteq carrier\text{-}vec\ n \cap \mathbb{Z}_v$
$H \subseteq Bounded\text{-}vec\ (of\text{-}nat\ (n + 1) * of\text{-}int\ (db\ n\ (max\ 1\ Bnd)))$
finite $(H \cup C)$ **and**
id: *integer-hull* (*polyhedron A b*) = *convex-hull H* + *cone C*
by *auto*
from *xsol*[*unfolded id*] **have** $H \neq \{\}$ **unfolding** *set-plus-def* **by** *auto*
then obtain *h* **where** *hH*: $h \in H$ **by** *auto*
with *set-in-convex-hull* **have** $h \in convex\text{-}hull\ H$ **using** *HC* **by** *auto*
moreover have $0_v\ n \in cone\ C$ **by** (*intro zero-in-cone*)
ultimately have $h + 0_v\ n \in integer\text{-}hull\ (polyhedron\ A\ b)$ **unfolding** *id* **by** *auto*
also have $h + 0_v\ n = h$ **using** *hH HC* **by** *auto*
also have $integer\text{-}hull\ (polyhedron\ A\ b) \subseteq convex\text{-}hull\ (polyhedron\ A\ b)$
unfolding *integer-hull-def* **by** (*rule convex-hull-mono*, *auto*)
also have *convex-hull* (*polyhedron A b*) = *polyhedron A b* **using** *A b*
using *convex-convex-hull polyhedra-are-convex* **by** *blast*
finally have *h*: $h \in carrier\text{-}vec\ n\ A *_v h \leq b$ **unfolding** *polyhedron-def* **by** *auto*
show *?thesis*

by (*intro exI*[*of* - *h*] *conjI h*, *insert HC hH*, *auto*)
qed

lemmas *small-integer-solution-nonstrict-via-decomp-hadamard* =
small-integer-solution-nonstrict-via-decomp[*OF det-bound-hadamard*, *unfolded det-bound-hadamard-def*]

end
end

References

[1] B. Dutertre and L. M. de Moura. A fast linear-arithmetic solver for DPLL(T). In *Proc. Computer Aided Verification*, volume 4144 of *LNCS*, pages 81–94. Springer, 2006. Extended version available as Technical Report, CSL-06-01, SRI International.

[2] C. H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, 1981.

[3] A. Schrijver. *Theory of linear and integer programming.* John Wiley & Sons, 1998.