NP-hardness of CVP and SVP

Katharina Kreuzer

March 19, 2025

Abstract

This article formalizes the NP-hardness proofs of the Closest Vector Problem (CVP) and the Shortest Vector Problem (SVP) in maximum norm as well as the CVP in any p-norm for $p \geq 1$. CVP and SVP are two fundamental problems in lattice theory. Lattices are a discrete, additive subgroup of $\mathbb{R}^n$ and are used for lattice-based cryptography. The CVP asks to find the nearest lattice vector to a target. The SVP asks to find the shortest non-zero lattice vector.

This entry formalizes the basic properties of lattices, the reduction from CVP to Subset Sum in both maximum and p-norm for $1 \leq p < \infty$ and the reduction of SVP to Partition using the Bounded Homogeneous Linear Equations problem (BHLE) as an intermediate step. The formalization uncovered a number of problems with the existing proofs in the literature [7] and [6].

Contents

1 Introduction **3**

2 Reduction Function **3**

3 Basics on Lattices **4**

4 Change of Basis in a Lattice **8**

5 Partition Problem **40**

6 Subset Sum Problem **41**

7 CVP in ℓ_p for $p \geq 1$ **46**

8 Maximum Norm **54**

9 CVP in ℓ_∞ **55**

10 Representation of Integers in Different Number Systems **63**

11 Additional Lemmas **68**

12 Bounded Homogeneous Linear Equation Problem **72**

13 SVP in ℓ_∞ **114**

1 Introduction

Two problems form the very basis for cryptography on lattices, namely the closest vector problem (CVP) and the shortest vector problem (SVP). Given a finite set of basis vectors in $\mathbb{R}^n$, the set of all linear combinations with integer coefficients forms a lattice. In optimization form, SVP asks for the shortest vector in the lattice and CVP asks for the lattice vector closest to some given target vector, both with respect to some given norm. The proof of NP-hardness of these problems in ℓ_∞ goes back to an early technical report by van Emde-Boas [7]. Indeed, for other norms (especially for the Euclidean norm), NP-hardness of the SVP could only be shown using a randomized reduction [1]. For CVP, NP-hardness has been shown in any p-norm for $p \geq 1$. One exemplary proof can be found in the book by Micciancio and Goldwasser [6, Chapter 3, Thm 3.1].

The CVP and SVP are not only used in cryptography, but in various other areas as well. For example, they closely relate to integer programs (see Lenstra's paper [5]) or finding irreducible factors of polynomials as in Lenstra's paper [3]. For example, the LLL-algorithm by Lenstra, Lenstra and Lovász [4] gives a polynomial-time algorithm for lattice basis reduction which solves integer linear programs in fixed dimensions. Using this reduced basis, one can find good approximations to CVP using Babai's algorithm [2] for certain approximation factors. Still, for arbitrary dimensions, the problem remains NP-hard.

theory *Reduction*

imports
  Main
begin

2 Reduction Function

This definition was taken from the developements at https://github.com/wimmers/poly-reductions "Karp21/Reductions.thy". TODO: When this repo comes into the AFP, link to original definition.

definition *is_reduction :: "('a ⇒ 'b) ⇒ 'a set ⇒ 'b set ⇒ bool"* **where**
  "is_reduction f A B ≡ ∀a. a ∈ A ⟷ f a ∈ B "
end
theory *Lattice_int*

imports
  "Jordan_Normal_Form.Matrix"
  "Jordan_Normal_Form.VS_Connect"

```
  "BenOr_Kozen_Reif.More_Matrix"
begin
```

3 Basics on Lattices

Connect the type *vec* to records of rings, commutative rings, fields and modules in order to use properties of modules such as *lin_indpt* (linear independence).

lemma *dim_carrier: "dim_vec z = dim_col A ⟹ A *$_v$ z ∈ carrier_vec (dim_row A)"*
  by *(metis carrier_vec_dim_vec dim_mult_mat_vec)*

Concrete type conversion *int vec* to *real vec*

definition *real_of_int_vec :: "int vec ⇒ real vec"* **where**
  "real_of_int_vec v = map_vec real_of_int v"

definition *real_to_int_vec :: "real vec ⇒ int vec"* **where**
  "real_to_int_vec v = map_vec floor v"

lemma *dim_vec_real_of_int_vec[simp]: "dim_vec (real_of_int_vec v) = dim_vec v"*
  unfolding *real_of_int_vec_def* **by** *auto*

lemma *real_of_int_vec_nth[simp, intro]:*
  "i<dim_vec v ⟹ (real_of_int_vec v) $ i = real_of_int (v$i)"
by *(simp add: real_of_int_vec_def)*

lemma *real_of_int_vec_vec:*
  "real_of_int_vec (vec n f) = vec n (real_of_int ∘ f)"
by *(auto simp add: real_of_int_vec_def)*

Concrete type conversion *int mat* to *real mat*

definition *real_of_int_mat :: "int mat ⇒ real mat"* **where**
  "real_of_int_mat A = map_mat real_of_int A"

definition *real_to_int_mat :: "real mat ⇒ int mat"* **where**
  "real_to_int_mat A = map_mat floor A"

lemma *dim_col_real_of_int_mat[simp]: "dim_col (real_of_int_mat A) = dim_col A"*
  unfolding *real_of_int_mat_def* **by** *auto*

lemma *dim_row_real_of_int_mat[simp]: "dim_row (real_of_int_mat A) = dim_row A"*
  unfolding *real_of_int_mat_def* **by** *auto*

```
lemma real_of_int_mat_nth[simp, intro]:
  "i<dim_row A ⟹ j<dim_col A ⟹ (real_of_int_mat A) $$ (i,j) = real_of_int
(A $$ (i,j))"
by (simp add: real_of_int_mat_def)

lemma real_of_int_mat_mat:
  "real_of_int_mat (mat n m f) = mat n m (real_of_int ∘ f)"
by (auto simp add: real_of_int_mat_def)

lemma real_of_int_mat_cols:
"cols (real_of_int_mat A) = map real_of_int_vec (cols A)"
by (simp add: list_eq_iff_nth_eq real_of_int_mat_def real_of_int_vec_def)

lemma distinct_cols_real_of_int_mat:
 "distinct (cols A) = distinct (cols (real_of_int_mat A))"
by (smt (verit, best) cols_length distinct_conv_nth index_map_mat(3) nth_map

  of_int_hom.vec_hom_inj real_of_int_mat_cols real_of_int_mat_def real_of_int_vec_def)
```

Algebraic lattices are discrete additive subgroups of $\mathbb{R}^n$. Lattices can be represented by a basis, multiple bases can represent the same lattice.

```
type_synonym int_lattice = "int vec set"
```

Linear independence

```
consts is_indep :: "'a ⇒ bool"

overloading
  is_indep_real ≡ "is_indep :: real mat ⇒ bool"
  is_indep_int ≡ "is_indep :: int mat ⇒ bool"
begin
definition is_indep_real :: "real mat ⇒ bool" where
  "is_indep A = (∀z::real vec. (A *v z = 0v (dim_row A) ∧
    dim_vec z = dim_col A) ⟶ z = 0v (dim_vec z))"
definition is_indep_int :: "int mat ⇒ bool" where
  "is_indep A = (∀z::real vec. ((real_of_int_mat A) *v z = 0v (dim_row
A) ∧
    dim_vec z = dim_col A) ⟶ z = 0v (dim_vec z))"
end
```

Definition of lattices

```
definition is_lattice :: "int_lattice ⇒ bool" where
  "is_lattice L ≡ (∃B::(int mat).
    L = {B *v z | z::int vec. dim_vec z = dim_col B}
    ∧ is_indep B)"
```

The lattice generated by the column vectors of a matrix. This matrix does not need to be linearly independent. Make certain that the output is indeed a lattice and not the entire space.

definition *gen_lattice :: "int mat ⇒ int vec set"* **where**
*"gen_lattice A = {A *$_v$ z | z::int vec. dim_vec z = dim_col A}"*

Theorems for lattices. The aim is to have a change of basis theorem for lattice bases. The theorems are mostly taken from the respective theorems for the type *('a,'k::finite) vec* and adapted to the type *'a vec*.

lemma *finsum_vec_carrier:*
assumes *"f ' A ⊆ carrier_vec nr" "finite A"*
shows *"finsum_vec TYPE('a::ring) nr f A ∈ carrier_vec nr"*
using *assms* **unfolding** *finsum_vec_def*
by *(metis (no_types, lifting) Pi_I' comm_monoid.finprod_closed comm_monoid_vec*

image_subset_iff monoid_vec_simps(2))

lemma *dim_vec_finsum_vec:*
assumes *"f ' A ⊆ carrier_vec nr" "finite A"*
shows *"dim_vec (finsum_vec TYPE('a::ring) nr f A) = nr"*
using *assms finsum_vec_carrier* **by** *(metis carrier_vecD)*

Matrix-Vector-Multiplication as an element in the column span.

lemma *mat_mult_as_col_span:*
assumes *"(A :: 'a :: comm_ring mat) ∈ carrier_mat nr nc"*
and *"(z :: 'a vec) ∈ carrier_vec nc"*
shows *"A *$_v$ z = finsum_vec TYPE('a) nr (λi. z$i ·$_v$ col A i) {0..<nc}"*
(is *"?left = ?right")*
proof -
have **: "finsum_vec TYPE('a) nr (λi. z$i ·$_v$ col A i) {0..<nc} $ j = (A *$_v$ z) $ j"*
if *"j<nr"* **for** *j*
proof -
have *"finsum_vec TYPE('a) nr (λi. z$i ·$_v$ col A i) {0..<nc} $ j = sum (λi. (z$i ·$_v$ col A i) $ j) {0..<nc}"*
by *(subst index_finsum_vec, use assms that* **in** *‹auto›)*
also have *"... = sum (λi. z$i * (col A i $ j)) {0..<nc}"*
using *assms(1) that* **by** *force*
also have *"... = sum (λi. z$i * A $$ (j,i)) {0..<nc}"* **using** *assms that* **by** *auto*
also have *"... = sum (λi. A $$ (j,i) * z$i) {0..<nc}"*
by *(meson mult.commute)*
also have *"... = sum (λi. row A j $ i * z $ i) {0..<nc}"*
using *assms that* **by** *(auto simp add: index_row(1)[symmetric] simp del: index_row(1))*
also have *"... = (A *$_v$ z) $ j"*
unfolding *mult_mat_vec_def scalar_prod_def* **using** *assms that* **by** *auto*
finally show *?thesis* **by** *blast*
qed
then show *?thesis*
proof *(subst eq_vecI[of ?left ?right], goal_cases)*

```
    case (1 i)
    have dim_nr: "dim_vec (A *v z) = nr"
      using assms(1) dim_mult_mat_vec by blast
    show ?case using assms 1 unfolding dim_nr by auto
  next
    case 2
    then show ?case using assms by (subst dim_vec_finsum_vec, auto)
  qed auto
qed
```

A matrix is identical to the matrix generated by its indices.

```
lemma mat_index:
"B = mat (dim_row B) (dim_col B) (λ(i,j). B $$ (i,j))"
by auto
```

Lemmas about the colums of a matrix.

```
lemma col_unit_vec:
assumes "i<dim_col (B:: 'a :: {monoid_mult, semiring_0, zero_neq_one}
mat)"
shows "B *v (unit_vec (dim_col B) i) = col B i" (is "?left = ?right")
proof -
  have "B $$ (ia, i) = (∑ ib = 0..<dim_col B. B $$ (ia, ib) * (if ib
= i then 1 else 0))"
    if "ia < dim_row B" for ia
  proof -
    have "(∑ ib = 0..<dim_col B. B $$ (ia, ib) * (if ib = i then 1 else
0)) =
      (∑ ib = 0..<dim_col B. (if ib = i then B $$ (ia, ib) else 0))"
      by (smt (verit, best) mult.right_neutral mult_zero_right sum.cong)
    also have "... = B $$ (ia, i) +
       (∑ ib∈ {0..<dim_col B}-{i}. (if ib = i then B $$ (ia, ib) else
0))"
      by (simp add: assms)
    also have "... =  B $$ (ia, i)" by simp
    finally show ?thesis by auto
  qed
  then show ?thesis using assms by (subst eq_vecI[of ?left ?right])

    (auto simp add: unit_vec_def scalar_prod_def)
qed

lemma is_indep_not_null:
assumes "is_indep (B:: real mat)" "i<dim_col B"
shows "col B i ≠ 0v (dim_row B)"
proof (rule ccontr)
  assume "¬ (col B i ≠ 0v (dim_row B))"
  then have "col B i = 0v (dim_row B)" by auto
  then have "B *v (unit_vec (dim_col B) i) = 0v (dim_row B)"
  using col_unit_vec assms(2) by metis
```

```
  moreover have "dim_vec (unit_vec (dim_col B) i) = dim_col B" by auto
  moreover have "(unit_vec (dim_col B) i) ≠ 0_v (dim_col B)"
    by (simp add: assms(2))
  ultimately have "¬ (is_indep B)" unfolding is_indep_real_def by auto
  then show False using assms by auto
qed

lemma col_in_gen_lattice:
assumes "i<dim_col B"
shows "col B i ∈ gen_lattice B"
unfolding gen_lattice_def proof (safe)
  have "col B i = B *_v (unit_vec (dim_col B) i)" using col_unit_vec[OF
assms] by auto
  moreover have "dim_vec (unit_vec (dim_col B) i) = dim_col B" by auto
  ultimately show "∃z. col B i = B *_v z ∧ dim_vec z = dim_col B" by
auto
qed
```

4 Change of Basis in a Lattice

Definition of the column span.

```
definition span :: "('a :: semiring_0) mat ⇒ 'a vec set"  where
"span B = {B *_v z | z:: 'a vec. dim_vec z = dim_col B}"
```

Definition of the column dimension of a matrix

```
definition dim :: "int mat ⇒ nat" where
"dim B = (if ∃b. is_indep (real_of_int_mat b) ∧ span (real_of_int_mat
b) = span (real_of_int_mat B)
    then dim_col (SOME b. is_indep (real_of_int_mat b) ∧
    span (real_of_int_mat b) = span (real_of_int_mat B)) else 0)"
```

Abbreviation of the set of columns

```
definition set_cols[simp]: "set_cols A = set (cols A)"
```

For the change of basis, we need to be able to delete and insert a column vector in the column span.

```
definition insert_col:
"insert_col A c = mat_of_cols (dim_row A) (c # cols A)"

definition delete_col:
"delete_col A c = mat_of_cols (dim_row A) (filter (λ x. ¬ (x = c)) (cols
A))"

lemma set_cols_col:
"set_cols A = col A ` {0..<dim_col A}"
by (simp add: cols_def)
```

```
lemma set_cols_subset_col:
assumes "set_cols A ⊆ set_cols B"
  "i<dim_col A"
obtains j where "col A i = col B j"
proof -
  have "col A i ∈ set_cols B" using assms
    by (metis cols_length cols_nth nth_mem set_cols subsetD)
  then have "∃j. col A i = col B j ∧ j<dim_col B"
    using set_cols_col by fastforce
  then show ?thesis using that by blast
qed
```

Monotonicity of the span.

```
lemma span_mono:
assumes "set_cols (A ::'a::comm_ring mat) ⊆ set_cols B" "dim_row A =
dim_row B"
    "distinct (cols A)"
shows "span A ⊆ span B"
using assms proof (induction "card (set_cols B - set_cols A)" arbitrary:
A B rule: less_induct)
  case less
  then show ?case unfolding span_def
  proof (safe, goal_cases)
    case (1 _ z)
    define nr where "nr = dim_row B"
    define nc where "nc = dim_col B"
    have 2: "B ∈ carrier_mat nr nc" unfolding nr_def nc_def by auto
    have 3: "set (cols A) ⊆ set (cols B)" using less.prems(1) by auto
    have 4: "z ∈ carrier_vec (length (cols A))" using 1(5)
    by (metis carrier_vec_dim_vec cols_length)
    have 5: "mat_of_cols nr (cols A) = A"
    by (metis less.prems(2) mat_of_cols_cols nr_def)
    obtain zb where "A *_v z = B *_v zb" "dim_vec zb = dim_col B"
      using helper3[of B nr nc "cols A" z, OF 2 less.prems(3) 3 4] un-
folding nc_def 5 by auto
    then show ?case by auto
  qed
qed
```

Monotonicity of linear independence

```
lemma is_indep_mono:
assumes "set_cols B ⊆ set_cols A" "is_indep A" "distinct (cols B)" "dim_row
A = dim_row B"
shows "is_indep (B::real mat)"
unfolding is_indep_real_def
proof (safe, goal_cases)
  case (1 z)
  define nr where "nr = dim_row A"
  define nc where "nc = dim_col A"
```

```
  have r: "nr = dim_row B" unfolding nr_def  using assms(4) by auto
  have A': "A ∈ carrier_mat nr nc" unfolding nr_def nc_def by auto
  define ss where "ss = cols (B)"
  have d: "distinct (cols B)" using assms(3)
  by (simp add: distinct_cols_real_of_int_mat)
  have sub: "set (cols B) ⊆ set (cols A)"
  using assms(1) real_of_int_mat_cols by auto
  from distinct_list_subset_nths[of "cols B" "cols A", OF d sub]
  obtain ids where ids: "distinct ids" "set ids ⊆ {..<length (cols A)}"
    and ss: "ss = map ((!) (cols A)) ids"
    using assms(1) set_cols ss_def by blast
  let ?ls = " map ((!) (cols A)) (filter (λi. i ∉ set ids) [0..<length
(cols A)])"
  from subindex_permutation2[OF ids] obtain f where
    f: "f permutes {..<length (cols A)}"
    "cols A = permute_list f (?ls @ ss)" using ss by blast
  have f': "f permutes {..<nc}" using f(1) unfolding nc_def by simp
  have *: "⋀x. x ∈ set ?ls ⟹ dim_vec x = nr" using nr_def by auto

  let ?cs1 = "(list_of_vec (0_v (length ?ls) @_v z))"
  have z: "z ∈ carrier_vec (length ss)" unfolding ss_def using 1
  using carrier_dim_vec by auto
  have "0_v nr = B *_v z" using 1 unfolding r by auto
  also have "... = mat_of_cols nr ss *_v z" unfolding ss_def r by simp
  also have "... = mat_of_cols nr (?ls @ ss) *_v vec_of_list (?cs1)"
    using helper2[OF z] * by (metis vec_list)
  also have "... = mat_of_cols nr (permute_list f (?ls @ ss)) *_v vec_of_list
(permute_list f ?cs1)"
    by (auto intro!: mat_of_cols_mult_mat_vec_permute_list[symmetric])
       (metis cols_length f(1) f(2) length_append length_map length_permute_list,
        use ss_def z in <force>)
  also have "... =  A *_v vec_of_list (permute_list f ?cs1)" using f(2)

    by (simp add: nr_def)
  finally have "A *_v vec_of_list (permute_list f ?cs1) = 0_v nr"
  by presburger
  then have zero: "vec_of_list (permute_list f ?cs1) = 0_v nc"
  by (smt (verit, best) assms(2) carrier_vecD cols_length dim_vec_of_list
f(2)
    index_append_vec(2) index_map_mat(2) index_map_mat(3) index_zero_vec(2)
is_indep_real_def
    length_append length_list_of_vec length_permute_list nc_def nr_def
real_of_int_mat_def z)
  then have *:  "permute_list f ?cs1 ! i = 0" if "i<nc" for i
    by (metis index_zero_vec(1) that vec_of_list_index)
  then have "?cs1 ! (f i) = 0" if "i<nc" for i
    by (metis cols_length dim_vec_of_list f(1) index_zero_vec(2) length_permute_list
nc_def
    permute_list_nth that zero)
```

```
  then have "(λi. ?cs1 ! (f i)) ‘ {..<nc} ⊆ {0}" by blast
  then have "(λi. ?cs1 ! i) ‘ (f ‘{..<nc}) ⊆ {0}" by blast
  then have "(λi. ?cs1 ! i) ‘ {..<nc} ⊆ {0}" using permutes_imp_bij[OF
f']
  by (metis cols_length f' length_map nc_def permutes_image)
  then have *: "?cs1 ! i = 0" if "i<nc" for i using that by blast
  then have "z $ i = 0" if "i<dim_vec z" for i using *[of "length ?ls
+ i"]
  by (metis (no_types, lifting) add_diff_cancel_left' add_less_cancel_left
dim_vec_of_list
    index_append_vec(2) index_zero_vec(2) length_list_of_vec length_permute_list

    list_of_vec_append not_add_less1 nth_append nth_list_of_vec that zero)
  then show "z = 0_v (dim_vec z)"
  by (metis eq_vecI index_zero_vec(1) index_zero_vec(2))
qed
```

Linear independent vectors are distinct.

```
lemma is_indep_distinct:
assumes "is_indep (A::real mat)"
shows "distinct (cols A)"
proof (rule ccontr)
  define nc where "nc = dim_col A"
  define nr where "nr = dim_row A"
  assume "¬ distinct (cols A)"
  then obtain i j where ij: "i ≠ j" "i<nc" "j<nc" "col A i = col A j"
    unfolding nc_def by (metis cols_length cols_nth distinct_conv_nth)
  have "A *_v (unit_vec nc i - unit_vec nc j) = 0_v nr"
    by (smt (verit, ccfv_threshold) ij carrier_mat_triv col_dim col_map_mat
col_unit_vec
      index_map_mat(2) index_map_mat(3) minus_cancel_vec mult_minus_distrib_mat_vec
nc_def
      nr_def real_of_int_mat_def unit_vec_carrier)
  moreover have "dim_vec (unit_vec nc i - unit_vec nc j) = nc" by simp
  moreover have "(unit_vec nc i - unit_vec nc j) ≠ 0_v nc" using ij
  by (smt (z3) assms calculation(1) calculation(2) index_minus_vec(1)
index_unit_vec(1)
    index_unit_vec(3) index_zero_vec(1) is_indep_real_def nc_def nr_def)

  ultimately have "¬ is_indep A" unfolding is_indep_real_def using nc_def
nr_def
  by (smt (verit, ccfv_threshold) carrier_matI col_dim col_unit_vec minus_cancel_vec

    mult_minus_distrib_mat_vec unit_vec_carrier)
  then show False using assms by auto
qed
```

Column vectors are in the column span.

```
lemma span_base:
```

```
assumes "(a :: 'a :: {monoid_mult,semiring_0,zero_neq_one} vec) ∈ set_cols
S"
shows "a ∈ span S"
proof -
  obtain i where "i<dim_col S" "a = col S i" using assms
  by (metis atLeastLessThan_iff imageE set_cols_col)
  then have "a = S *v (unit_vec (dim_col S) i)"
    using col_unit_vec[OF <i<dim_col S>, symmetric] by auto
  then show ?thesis unfolding span_def by auto
qed
```

Representing vectors in the column span of a linear independent matrix. The representation function returns the coefficients needed to generate the vector as an element in the column span.

Respresentation function and its lemmas were adapted from "HOL/Modules.thy"

```
definition representation :: "real mat ⇒ real vec ⇒ real vec ⇒ real"
  where "representation B v =
    (if is_indep B ∧ v ∈ (span B) then
      SOME f. (∀ v. f v ≠ 0 ⟶ v ∈ set_cols B) ∧
        B *v (vec (dim_col B) (λi. f (col B i))) = v
    else (λb. 0))"
```

If the column vectors form a basis of the column span, the representation function is unique.

```
lemma unique_representation:
  assumes basis: "is_indep (basis::real mat)"
    and in_basis: "⋀v. f v ≠ 0 ⟹ v ∈ set_cols basis" "⋀v. g v ≠
0 ⟹ v ∈ set_cols basis"
    and eq: "basis *v (vec (dim_col basis) (λi. f (col basis i))) =
           basis *v (vec (dim_col basis) (λi. g (col basis i)))"
  shows "f = g"
proof (rule ext, rule ccontr)
  fix v assume ne: "f v ≠ g v"
  then have in_col: "v ∈ set_cols basis" using in_basis by metis
  have "¬ is_indep basis"
    unfolding is_indep_real_def
  proof (subst not_all, subst not_imp)
    let ?x = "vec (dim_col basis) (λi. (f (col basis i)- g (col basis
i)))"
    have  "∃v∈set_cols basis. f v - g v ≠ 0"
      using ne in_col right_minus_eq by blast
    then have ex: "∃i<dim_col basis. f (col basis i) - g (col basis i)
≠ 0"
      by (metis atLeastLessThan_iff imageE set_cols_col)
    have "basis *v (?x) =
          basis *v (vec (dim_col basis) (λi. f (col basis i))) -
          basis *v (vec (dim_col basis) (λi. g (col basis i)))"
```

```
    proof -
      have *: "?x = vec (dim_col basis) (λi. f (col basis i)) -
        vec (dim_col basis) (λi. g (col basis i))" unfolding minus_vec_def
by auto
      show ?thesis unfolding *
      by (meson carrier_mat_triv mult_minus_distrib_mat_vec vec_carrier)
    qed
    then have "basis *v (?x) = 0v (dim_row basis)" using eq by auto
    moreover have "dim_vec ?x = dim_col basis"
    using dim_vec by blast
    moreover have "?x ≠ 0v (dim_col basis)" using ex
    by (metis (no_types, lifting) index_vec index_zero_vec(1))
    ultimately show "∃x. (basis *v x = 0v (dim_row basis) ∧
         dim_vec x = dim_col basis) ∧ x ≠ 0v (dim_vec x)"
         using real_of_int_vec_def by auto
  qed
  with basis show False by auto
qed
```

More properties on the representation function.

```
lemma
  shows representation_ne_zero: "⋀b. representation basis v b ≠ 0 ⟹
b ∈ set_cols basis"
    and sum_nonzero_representation_eq:
      "is_indep basis ⟹ v ∈ span basis ⟹
      basis *v (vec (dim_col basis) (λi. representation basis v (col basis
i))) = v"
proof -
  { assume basis: "is_indep basis" and v: "v ∈ span basis"
    define p where "p f ⟷
      (∀v. f v ≠ 0 ⟶ v ∈ set_cols basis) ∧
      basis *v (vec (dim_col basis) (λi. f (col basis i))) = v" for f
    obtain z where z: "basis *v z = v" "dim_vec z = dim_col basis"
      using ‹v ∈ span basis› unfolding span_def by auto
    define r where "r = (λv. if (∃i<dim_col basis. v = col basis i)
      then z$(SOME i. i<dim_col basis ∧ v = col basis i) else 0)"
    then have "r (col basis i) = z$i" if "i<dim_col basis" for i
    proof -
      have "z $ (SOME j. j<dim_col basis ∧ col basis ia = col basis j)
= z $ i"
        if "i < dim_col basis"
          "distinct (cols basis)"
          "ia < dim_col basis"
          "col basis i = col basis ia"
        for ia
      proof -
        have "i = ia" using that
        by (metis cols_length cols_nth distinct_conv_nth)
        obtain j where j: "j = (SOME j. j<dim_col basis ∧ col basis i
```

```
= col basis j)" by blast
        then have "j<dim_col basis" "col basis i = col basis j"
        by (smt (verit, best) ‹i = ia› someI_ex that(3))
           (metis (mono_tags, lifting) j that(1) verit_sko_ex')
        then have "i = j" using that
        by (metis cols_length cols_nth distinct_conv_nth)
        then show ?thesis unfolding j(1)
        using that(4) by auto
      qed
      then show ?thesis
      unfolding r_def using that is_indep_distinct[OF basis] by auto
    qed
    then have "z = (vec (dim_col basis) (λi. r (col basis i)))"
       using z(2) by auto
    then have *:
      "basis *v (vec (dim_col basis) (λi. r (col basis i))) = v"
      using z by auto
    define f where "f b = (if b ∈ set_cols basis then r b else 0)" for
b
    have "p f"
      using * unfolding p_def f_def
      by (smt (verit, best) cols_length cols_nth dim_vec eq_vecI index_vec
nth_mem set_cols)
    have *: "representation basis v = Eps p"
      by (simp add: p_def[abs_def] representation_def basis v)
    from someI[of p f, OF ‹p f›] have "p (representation basis v)"
      unfolding * . }
  note * = this

  show "representation basis v b ≠ 0 ⟹ b ∈ set_cols basis" for b
    using * by (cases "is_indep basis ∧ v ∈ span basis") (auto simp:
representation_def)

  show "is_indep basis ⟹ v ∈ span basis ⟹
    basis *v (vec (dim_col basis) (λi. representation basis v (col basis
i))) = v"
    using * by auto
qed


lemma representation_eqI:
  assumes basis: "is_indep basis" and b: "v ∈ span basis"
    and ne_zero: "⋀b. f b ≠ 0 ⟹ b ∈ set_cols basis"
    and eq: "basis *v (vec (dim_col basis) (λi. f (col basis i))) = v"
  shows "representation basis v = f"
  by (rule unique_representation[OF basis])
     (auto simp: representation_ne_zero
      sum_nonzero_representation_eq[OF basis b] ne_zero eq simp del: set_cols)
```

Representation function when extending the column space.

```
lemma representation_extend:
  assumes basis: "is_indep basis" and v: "v ∈ span basis'"
    and basis': "set_cols basis' ⊆ set_cols basis" and d: "dim_row basis
= dim_row basis'"
    and dc: "distinct (cols basis')"
  shows "representation basis v = representation basis' v"
proof (rule representation_eqI[OF basis])
  show v': "v ∈ span basis" using span_mono[OF basis' d[symmetric] dc]
v by auto
  have *: "is_indep basis'" using is_indep_mono[OF basis' basis dc d]
by (auto)
  show "representation basis' v b ≠ 0 ⟹ b ∈ set_cols basis" for b
    using representation_ne_zero basis' by auto
  have dim_row: "dim_row basis' = dim_row basis"
    by (metis "*" Lattice_int.sum_nonzero_representation_eq basis dim_mult_mat_vec
v v')
  show "basis *v vec (dim_col basis) (λi. representation basis' v (col
basis i)) = v "
  proof -
    have not_in': "representation basis' v b = 0" if "b ∉ set_cols basis'"
for b
    using Lattice_int.representation_ne_zero that by blast
    let ?vec = "vec (dim_row basis) (λi. ∑ia = 0..<dim_vec
        (vec (dim_col basis) (λi. Lattice_int.representation basis' v
(col basis i))).
         row basis i $ ia * vec (dim_col basis)
            (λi. Lattice_int.representation basis' v (col basis i)) $
ia)"
    have dim_v: "dim_vec v = dim_row basis"
    using Lattice_int.span_def dim_row v by force
    have "v= ?vec"
    proof (subst eq_vecI[of v ?vec], goal_cases)
      case (1 i)
      obtain ids where ids: "distinct ids" "set ids ⊆ {..<length (cols
basis)}"
        and ss: "cols basis' = map ((!) (cols basis)) ids"
      using distinct_list_subset_nths[OF dc, of "cols basis"] basis' set_cols
by blast
      define ls where "ls = map ((!) (cols basis))
        (filter (λi. i ∉ set ids) [0..<length (cols basis)])"
      from subindex_permutation2[OF ids] obtain p where
        p: "p permutes {..<length (cols basis)}"
        "cols basis = permute_list p (ls @ (cols basis'))" unfolding ls_def
using ss by metis
      then have p_i: "col basis i = (ls @ (cols basis')) ! (p i)" if "i<length
(cols basis)" for i
        by (metis cols_length cols_nth length_permute_list permute_list_nth
that)
      have len_ls: "length ls ≤ length (cols basis)"
```

```
      by (metis length_append length_permute_list linorder_not_less not_add_less1
p(2))
      have in_ls: "col basis j = ls ! (p j)" if "j ∈ p -‘ {0..<length
ls}" for j
        using p_i
        by (metis atLeastLessThan_upt len_ls lessThan_atLeast0 lessThan_iff
lessThan_subset_iff
          nth_append p(1) permutes_in_image subset_code(1) that vimageE)
      have not_in_ls: "col basis j = (cols basis') ! (p j - length ls)"

        if "j ∈ p -‘ {length ls..<length (cols basis)}" for j
      proof -
        have "(ls @ (cols basis')) ! (p j) = (cols basis') ! (p j - length
ls)" using that
        by (meson atLeastLessThan_iff leD nth_append vimage_eq)
        then show ?thesis using p_i[of j] that
        by (metis (mono_tags, lifting) atLeastLessThan_iff lessThan_iff
p(1) permutes_def vimageE)
      qed
      have distinct: "distinct (cols basis)" using is_indep_distinct[OF
basis] .

      have ls_in_basis: "set ls ⊆ set_cols basis" unfolding set_cols
ls_def
        by (auto, metis cols_length cols_nth nth_mem)
      have ls_not_in_basis': "(set ls) ∩ (set_cols basis') = {}"
      proof (subst disjoint_iff, intro allI, intro impI)
        fix x assume "x ∈ set ls"
        then obtain j where j: "col basis j = x" using ls_in_basis
        by (metis imageE set_cols_col subsetD)
        then have "j ∉ set ids"using ‹x∈set ls› unfolding ls_def
        by (smt (verit, del_insts) atLeastLessThan_iff atLeast_upt cols_length
cols_nth distinct
          distinct_conv_nth ids(2) imageE list.set_map mem_Collect_eq
set_filter set_upt subsetD)
        then show "x ∉ set_cols basis'" unfolding set_cols ss j[symmetric]

        by (smt (verit, ccfv_threshold) ‹x ∈ set ls› atLeast_upt distinct
distinct_conv_nth ids(2)
          in_set_conv_nth j length_map lessThan_iff ls_def mem_Collect_eq
nth_map nth_mem
          set_filter subsetD)
      qed

      have rep: "representation basis' v = (SOME f. (∀ v. f v ≠ 0 ⟶
v ∈ set_cols basis') ∧
            basis' *v vec (dim_col basis') (λi. f (col basis' i)) = v)
"
        unfolding representation_def using ‹is_indep basis'› ‹v∈span
```

```
basis'› by auto
      then obtain f where some_f: "f = (SOME f. (∀v. f v ≠ 0 ⟶ v ∈
set_cols basis') ∧
         basis' *v vec (dim_col basis') (λi. f (col basis' i)) = v)" by
blast
      then have f: "∀v. f v ≠ 0 ⟶ v ∈ set_cols basis'"
        "basis' *v vec (dim_col basis') (λi. f (col basis' i)) = v"
        apply (metis not_in' rep)
        by (metis "*" Lattice_int.sum_nonzero_representation_eq rep some_f
v)
      then have rep_f: "representation basis' v = f" using rep some_f
by auto
      have not_in_f: "f b = 0" if "b ∉ set_cols basis' " for b using not_in'
        using f(1) that by blast
      have f_ls_0: "f (ls ! j) = 0" if "j<length ls" for j
        apply (subst not_in_f) using ls_not_in_basis' nth_mem that by
auto

      have "?vec $ i = vec (dim_vec v) (λi. ∑ia = 0..<dim_col basis.
            row basis i $ ia * vec (dim_col basis) (λj. f (col basis
j)) $ ia) $ i"
        using dim_v[symmetric] rep_f by auto
      also have "... = (∑ia = 0..<dim_col basis.
            row basis i $ ia * vec (dim_col basis) (λj. f (col basis
j)) $ ia)"
        using ‹i<dim_vec v› by auto
      also have "... = (∑ia = 0..<length (cols basis). basis $$ (i, ia)
* (f (col basis ia)))"
        using "1" dim_v by force
      also have "... = (∑ia ∈ p -‘ {0..<length (cols basis)}.
        (col basis ia $ i) * (f (col basis ia)))"
        using "1" dim_v p(1) by (simp add: atLeast0LessThan permutes_vimage)
      also have "... = (∑ia ∈ p -‘ {0..<length ls} ∪ p -‘ {length ls..<length
(cols basis)}.
        (col basis ia $ i) * (f (col basis ia)))"
        by (metis (no_types, lifting) bot_nat_0.extremum ivl_disj_un_two(3)
len_ls vimage_Un)
      also have "... = (∑ia ∈ p -‘ {0..< length ls}. (col basis ia $
i) * (f (col basis ia))) +
        (∑ia ∈ p -‘ {length ls..<length (cols basis)}. (col basis ia
$ i) * (f (col basis ia)))"
      proof (subst sum.union_disjoint, goal_cases)
        case 1 then show ?case
        by (metis (no_types, lifting) atLeast_upt len_ls lessThan_subset_iff
p(1)
          permutes_vimage set_upt subset_eq_atLeast0_lessThan_finite vimage_mono)
      next
        case 2 then show ?case
        by (metis (no_types, lifting) atLeast0LessThan diff_is_0_eq' diff_le_self
```

```
        finite_nat_iff_bounded ivl_subset length_map length_upt map_nth
p(1) permutes_vimage
        vimage_mono)
    qed auto
    also have "... = (∑ ia ∈ {0..<length (cols basis')}.
        (col basis' (ia) $ i) * (f (col basis' (ia))))"
    proof -
      have "(∑ ia ∈ p -‘ {0..< length ls}. (col basis ia $ i) * (f
(col basis ia))) =
          (∑ ia ∈ p -‘ {0..< length ls}. (ls ! (p ia) $ i) * (f (ls
! (p ia))))"
        using in_ls by force
      moreover have "... = (∑ ia ∈ {0..< length ls}. (ls ! ia $ i)
* (f (ls ! ia)))"
      proof -
        have inj: "inj_on p (p -‘ {0..<length ls})"
          by (metis inj_onI p(1) permutes_def)
        have "p ‘ p -‘ {0..<length ls} = {0..<length ls}"
        by (metis p(1) permutes_def surj_def surj_image_vimage_eq)
        then show ?thesis using sum.reindex[OF inj,of "(λj. ls ! j
$ i * f (ls ! j))", symmetric]
          unfolding comp_def by auto
      qed
      moreover have "... = 0" by (simp add: f_ls_0)
      ultimately have "(∑ ia ∈ p -‘ {0..< length ls}. (col basis ia
$ i) * (f (col basis ia)))
        = 0" by auto

      moreover have "(∑ ia ∈ p -‘ {length ls..<length (cols basis)}.
(col basis ia $ i) *
        (f (col basis ia))) =
        (∑ ia ∈ p -‘ {length ls..<length (cols basis)}. (col basis'
(p ia -length ls) $ i) *
        (f (col basis' (p ia -length ls))))"
      proof -
        have "(col basis ia $ i) * (f (col basis ia)) = (col basis'
(p ia -length ls) $ i) *
        (f (col basis' (p ia -length ls)))" if "ia ∈ p -‘ {length
ls..<length (cols basis)}"
        for ia using not_in_ls[OF that]
        by (metis add_diff_cancel_left' atLeastLessThan_iff cols_length
cols_nth diff_less_mono
        length_append length_permute_list p(2) that vimageD)
        then show ?thesis by simp
      qed
      moreover have "... = (∑ ia ∈ {length ls..<length (cols basis)}.
(col basis' (ia -length ls) $ i) *
        (f (col basis' (ia -length ls))))"
```

```
        proof -
          have inj: "inj_on p (p -‘ {length ls..<length (cols basis)})"

            by (metis inj_onI p(1) permutes_def)
          have "p ‘ p -‘ {length ls..<length (cols basis)} = {length ls..<length
(cols basis)}"
          by (metis p(1) permutes_def surj_def surj_image_vimage_eq)
          then show ?thesis using sum.reindex[OF inj,of "(λj. col basis'
(j - length ls) $ i *
            f (col basis' (j - length ls)))", symmetric]
             unfolding comp_def by auto
        qed
        moreover have "... = (∑ ia ∈ {length ls..<length ls + length
(cols basis')}.
            (col basis' (ia -length ls) $ i) * (f (col basis' (ia -length
ls))))"
          by (simp add: p(2))
        moreover have "... = (∑ ia ∈ {0..<length (cols basis')}.
            (col basis' (ia) $ i) * (f (col basis' (ia))))"
        proof -
          have inj: "inj_on (λj. j-length ls) {length ls..<length ls +
length (cols basis')}"
            using inj_on_diff_nat by fastforce
          have "x ∈ (λx. x - length ls) ‘ {length ls..<length ls + dim_col
basis'}"
            if "x < dim_col basis'" for x using that
            by (metis add_diff_cancel_left' atLeastLessThan_iff image_eqI
linorder_not_less
              nat_add_left_cancel_less not_add_less1)
          then have "(λj. j - length ls) ‘ {length ls..<length ls + length
(cols basis')} =
            {0..<length (cols basis')}"
            by auto
          then show ?thesis using sum.reindex[OF inj,
            of "(λia. col basis' ia $ i * f (col basis' ia))", symmetric]
             unfolding comp_def by auto
        qed
        ultimately show ?thesis by auto
      qed

      also have "... = (∑ ia ∈ {0..<length (cols basis')}.
          (basis' $$ (i, ia)) * (f (col basis' (ia))))"
          using "1" dim_row dim_v by auto
      also have "... = (∑ ia ∈ {0..<length (cols basis')}.
          (row basis' i $ ia) * (f (col basis' (ia))))"
          using "1" dim_row dim_v by force
      also have "... = (∑ ia ∈ {0..<length (cols basis')}.
          (row basis' i $ ia) * vec (dim_col basis') (λj. f (col basis'
j)) $ ia)" by force
```

```
    also have "... = (row basis' i) · vec (dim_col basis') (λj. f (col
basis' j))"
    by (simp add: scalar_prod_def)
    also have "... = vec (dim_vec v) (λi. (row basis' i) ·
      vec (dim_col basis') (λj. f (col basis' j))) $ i" by (simp add:
"1")
    also have "... = (basis' *v vec (dim_col basis') (λj. f (col basis'
j))) $ i"
      unfolding mult_mat_vec_def using d dim_v by presburger
    also have "... = v $ i" using f(2) by blast
    finally have "?vec $ i = v $ i" by blast
    then show ?case by auto
  next
    case (2)
    then show ?case using dim_v by auto
  qed (auto)
  then show ?thesis unfolding mult_mat_vec_def scalar_prod_def by
auto
 qed
qed
```

The representation of the i-th column vector is the i-th unit vector.

```
lemma representation_basis:
 assumes basis: "is_indep basis" and b: "b ∈ set_cols basis"
 shows "representation basis b = (λv. if v = b then 1 else 0)"
proof (rule unique_representation[OF basis])
 show "representation basis b v ≠ 0 ⟹ v ∈ set_cols  basis" for v
   using representation_ne_zero .
 show "(if v = b then 1 else 0) ≠ 0 ⟹ v ∈ set_cols basis" for v
   using b by (cases "v = b") (auto simp: b)
 have *: "{v. (if v = b then 1 else 0::int) ≠ 0} = {b}"
   by auto
 obtain j where j: "col basis j = b" "j<dim_col basis"
 by (metis atLeastLessThan_iff b imageE set_cols_col)
 have dis: "distinct (cols basis)" using basis is_indep_distinct by
simp
 have unit: "vec (dim_col basis) (λi. if col basis i = b then 1 else
0) =
   unit_vec (dim_col basis) j" (is "?l = ?r")
 proof (subst eq_vecI[of ?l ?r], goal_cases)
   case (1 i)
   then show ?case using dis j
   by (smt (verit, best) cols_length cols_nth dim_vec distinct_conv_nth
index_unit_vec(1)
     index_vec)
 qed  auto
 show "basis *v vec (dim_col basis) (λi. Lattice_int.representation
basis b (col basis i)) =
   basis *v vec (dim_col basis) (λi. if col basis i = b then 1 else 0)"
```

```
    using * sum_nonzero_representation_eq[OF basis span_base[OF b]] un-
folding unit
    by (subst col_unit_vec, simp add: j, unfold j(1), simp)
qed
```

The spanning and independent set of vectors is a basis.

```
lemma spanning_subset_independent:
assumes BA: "set_cols B ⊆ set_cols A" and iA: "is_indep (A::real mat)"

  and AsB:"set_cols A ⊆ span B"
  and d: "distinct (cols B)" and dr: "dim_row A = dim_row B"
shows "set_cols A = set_cols B"
proof (intro antisym[OF _ BA] subsetI)
  have iB: "is_indep B" using is_indep_mono[OF BA iA d dr] .
  fix v assume "v ∈ set_cols A"
  with AsB have "v ∈ span B" by auto
  let ?RB = "representation B v" and ?RA = "representation A v"
  have "?RB v = 1"
    unfolding representation_extend[OF iA ‹v ∈ span B› BA dr d, symmetric]

      representation_basis[OF iA ‹v ∈ set_cols A›] by simp
  then show "v ∈ set_cols B"
    using representation_ne_zero[of B v v] by auto
qed


lemma nth_lin_combo:
assumes "i < dim_row A" "dim_col A = dim_vec b"
shows "(A *_v b) $ i = (∑ j=0..<dim_col A. A $$ (i,j) * b $ j)"
using assms unfolding mult_mat_vec_def scalar_prod_def by auto
```

Insert and delete with the span.

```
lemma dim_col_insert_col:
  "dim_col (insert_col S a) = dim_col S + 1"
by (simp add: insert_col)

lemma dim_col_delete_col:
assumes "a ∈ set_cols S" "distinct (cols S)" "dim_vec a = dim_row S"

shows "dim_col (delete_col S a) = dim_col S - 1"
proof -
  have *: "{x. x ≠ a} ∩ set (cols S) = set (cols S) - {a}" by fastforce
  have "length (filter (λx. x ≠ a) (cols S)) = length (cols S) - 1"
    unfolding distinct_length_filter[OF ‹distinct (cols S)›] *
    by (metis assms(1) assms(2) card_Diff_singleton distinct_card set_cols)
  then have "length (cols (delete_col S a)) = length (cols S) - 1"
    unfolding delete_col by auto
  then show ?thesis by (metis cols_length)
qed
```

```
lemma dim_row_delete_col:
  "dim_row (delete_col T b) = dim_row T"
by (simp add: delete_col)

lemma span_insert1:
assumes "dim_vec (a :: 'a ::comm_ring vec) = dim_row S"
shows "span (insert_col S a) = {x. ∃k y. x = y + k ·v a ∧ y ∈ span S
∧ dim_vec x = dim_row S}"
unfolding span_def proof (safe, goal_cases)
  case (1 x z)
  obtain za zs where z: "z = vCons za zs"
  by (metis "1" add_Suc_right dim_vec insert_col list.size(4) mat_of_cols_carrier(3)

    nat.simps(3) vec_cases)
  have "dim_vec z = dim_col S + 1" using 1 by (simp add: insert_col)
  then have *: "dim_vec zs = dim_col S" using z by auto
  have "mat_of_cols (dim_row S) (a # cols S) *v vCons za zs = S *v zs
+ za ·v a"
  proof (subst mat_of_cols_cons_mat_vec, goal_cases one two three)
    case one
    then show ?case using * by (metis carrier_vec_dim_vec cols_length)
  next
    case three
    then show ?case
    by (simp add: assms comm_add_vec mult_mat_vec_def smult_vec_def)
  qed (use assms in <auto>)
  moreover have "S *v zs ∈ span S" unfolding span_def using * by auto
  moreover have "dim_vec (mat_of_cols (dim_row S) (a # cols S) *v vCons
za zs) =
        dim_row S" by auto
  ultimately show ?case unfolding z insert_col using "*" by blast
next
  case (2 x k y z)
  have "S *v z + k ·v a = mat_of_cols (dim_row S) (cols S) *v z + k ·v
a" by simp
  moreover have "mat_of_cols (dim_row S) (cols S) *v z + k ·v a =
    mat_of_cols (dim_row S) (a # cols S) *v (vCons k z)"
  proof (subst mat_of_cols_cons_mat_vec, goal_cases one two three)
    case one
    then show ?case using 2
    by (metis carrier_vec_dim_vec cols_length)
  next
    case three
    then show ?case
    by (simp add: assms comm_add_vec mult_mat_vec_def smult_vec_def)
  qed (use assms in <auto>)
  moreover have "dim_vec (vCons k z) = dim_col (insert_col S a)"
    using 2 by (simp add: insert_col)
```

```
  ultimately show ?case unfolding insert_col by auto
qed


lemma span_insert2:
assumes "dim_vec a = dim_row S"
shows "span (insert_col S (a :: 'a ::comm_ring vec)) =
    {x. ∃k. (x - k ·v a) ∈ span S ∧ dim_vec x = dim_row S}"
proof -
  have "span (insert_col S a) = {x. ∃k y. x = y + k ·v a ∧ y ∈ span S
∧ dim_vec x = dim_row S}"
    using span_insert1[OF assms] by auto
  moreover have "{x. ∃k y. x = y + k ·v a ∧ y ∈ span S  ∧ dim_vec x
= dim_row S} =
      {x. ∃k. x - k ·v a ∈ Lattice_int.span S ∧ dim_vec x = dim_row S}"
  proof (safe, goal_cases)
    case (1 x k y)
    then have i: "y + ((k ·v a) - (k ·v a)) ∈ Lattice_int.span S"
    by (smt (verit, ccfv_SIG) Lattice_int.span_def assms dim_carrier mem_Collect_eq

      minus_cancel_vec right_zero_vec smult_vec_def vec_carrier)
    have rew: "y + ((k ·v a) - (k ·v a)) = (y + (k ·v a)) - (k ·v a) "
      using assoc_add_vec[of y _ "k ·v a" "- k ·v a" ] by auto
    show ?case using i unfolding rew using assms by auto
  next
    case (2 x k)
    have "x = x + (k ·v a - k ·v a)" using "2"(2) assms by auto
    then have "x = (x - k ·v a) + k ·v a"
      using assoc_add_vec[of x _ "- k ·v a" "k ·v a" ]
      using "2"(2) assms by auto
    moreover have "(x - k ·v a) ∈ Lattice_int.span S"  using "2"(1) by
blast
    ultimately show ?case using 2 by auto
  qed
  ultimately show ?thesis by auto
qed

lemma insert_delete_span:
assumes "(a :: 'a ::comm_ring vec) ∈ set_cols A" "distinct (cols A)"
"dim_vec a = dim_row A"
shows "span (insert_col (delete_col A a) a) = span (A)"
unfolding span_def
proof (safe, goal_cases)
  case (1 x z)
  define nr where "nr = dim_row A"
  define nc where "nc = dim_col A"
  obtain c where c: "mat_of_cols nr (a # cols (delete_col A a)) *v z
= A *v c"
          "dim_vec c = nc"
```

```
  proof (subst helper3[where A = A and nr = nr and nc = nc and ss =
"a # cols (delete_col A a)"
    and v = z], goal_cases)
    case 2
    then show ?case using assms(2) unfolding delete_col
    by (metis (mono_tags, lifting) cols_dim cols_mat_of_cols distinct.simps(2)
distinct_filter
    dual_order.trans filter_is_subset mem_Collect_eq set_filter)
  next
    case 3
    then show ?case using assms(1)
    by (metis (mono_tags, lifting) cols_dim cols_mat_of_cols delete_col
filter_is_subset in_mono
      set_ConsD set_cols subsetI)
  next
    case 4
    then show ?case using 1
    by (metis carrier_vecI insert_col mat_of_cols_carrier(3))
  qed (auto simp add: nc_def nr_def)
  then show ?case unfolding nr_def nc_def
    by (metis delete_col insert_col mat_of_cols_carrier(2) mat_of_cols_cols
nc_def nr_def)
next
  case (2 x z)
  define nr where "nr = dim_row A"
  define nc where "nc = dim_col A"
  have gr0: "dim_col A > 0" using assms(1)
  by (metis atLeastLessThan_iff bot_nat_0.not_eq_extremum cols_def imageE
less_nat_zero_code
    list.set_map set_cols set_upt)
  obtain c where c: "mat_of_cols nr (cols A) *_v z = insert_col (delete_col
A a) a *_v c"
          "dim_vec c = nc"
  proof (subst helper3[where A = "insert_col (delete_col A a) a" and
nr = nr and nc = nc
    and ss = "cols A"  and v = z], goal_cases)
    case 1
    have "dim_row (insert_col (delete_col A a) a) = dim_row A"
    by (simp add: delete_col insert_col)
    moreover have "dim_col (insert_col (delete_col A a) a) = dim_col
A"
      unfolding dim_col_insert_col dim_col_delete_col[OF assms] using
gr0 by simp
    ultimately show ?case unfolding nr_def nc_def by auto
  next
    case 3
    show ?case using assms(1) unfolding insert_col delete_col
    proof (subst cols_mat_of_cols, goal_cases)
      case 1
```

```
      have "a ∈ set (cols A) ⟹ a ∈ carrier_vec (dim_row A)"
        using assms(3) carrier_vecI by blast
      moreover have "x ∈ set (cols (delete_col A a)) ⟹ x ∈ carrier_vec
(dim_row A)" for x
         by (metis cols_dim delete_col mat_of_cols_carrier(2) subsetD)
      ultimately show ?case
      by (metis "1" cols_dim insert_subset list.set(2) mat_of_cols_carrier(2)
nr_def set_cols)

    next
      case 2
      then show ?case
      by (smt (verit, best) cols_dim cols_mat_of_cols filter_set insert_iff
list.set(2)
        member_filter subsetI subset_trans)
    qed
  next
    case 4
    then show ?case using 2 by (metis carrier_vec_dim_vec cols_length)
  qed (auto simp add: nc_def nr_def assms)
  then show ?case unfolding nr_def nc_def
  by (metis Suc_eq_plus1 Suc_pred' assms(1) assms(2) assms(3) dim_col_delete_col

    dim_col_insert_col gr0 mat_of_cols_cols)
qed
```

Deleting a generating element.

```
lemma span_breakdown:
  assumes bS: "(b :: 'a :: comm_ring vec) ∈ set_cols S"
    and aS: "a ∈ span S"
    and b: "dim_vec b = dim_row S"
    and d: "distinct (cols S)"
  shows "∃k. a - k ·v b ∈ span (delete_col S b)"
proof -
  have r: "dim_vec b = dim_row (delete_col S b)"
  by (simp add: b delete_col)
  have s: "span (insert_col (delete_col S b) b) = span S"
    by (subst insert_delete_span[OF bS d b], auto)
  have "span S = {x. ∃k. x - k ·v b ∈ Lattice_int.span (delete_col S
b) ∧
            dim_vec x = dim_row (delete_col S b)}"
    using span_insert2 [of b "delete_col S b"] r unfolding s by auto
  then show ?thesis using assms by auto
qed

lemma uminus_scalar_mult:
  "(-k) ·v a = - (k ·v (a:: 'a ::comm_ring_1 vec))"
unfolding smult_vec_def
by (subst eq_vecI[of "vec (dim_vec a) (λi. - k * a $ i)" "- vec (dim_vec
```

```
a) (λi. k * a $ i)"])
   auto


lemma span_scale: "(x :: 'a ::field vec) ∈ span S ⟹ c ·v x ∈ span
S"
unfolding span_def
proof safe
  fix z assume *: "dim_vec z = dim_col S" "x = S *v z"
  show "∃za. c ·v (S *v z) = S *v za ∧ dim_vec za = dim_col S"
  proof (intro exI[of _ "c ·v z"], safe, goal_cases)
    case 1
    show ?case by (metis "*"(1) carrier_mat_triv carrier_vec_dim_vec
mult_mat_vec)
  next
    case 2
    show ?case by (simp add: "*"(1))
  qed
qed


lemma span_diff:
assumes "(x :: 'a::ring vec) ∈ span S" "y ∈ span S"
    "x ∈ carrier_vec (dim_row S)" "y ∈ carrier_vec (dim_row S)"
shows "x - y ∈ span S"
using assms unfolding span_def
proof (safe, goal_cases)
  case (1 zx zy)
  then have "zx ∈ carrier_vec (dim_col S)" "zy ∈ carrier_vec (dim_col
S)"
    by (metis carrier_vec_dim_vec)+
  then have "S *v zx - S *v zy = S *v (zx - zy)"
   by (subst mult_minus_distrib_mat_vec[where A = S and v = zx and w
= zy, symmetric], auto)
  moreover have "dim_vec (zx-zy) = dim_col S" using 1 by auto
  ultimately show ?case by auto
qed
```

Adding a generating element.

```
lemma in_span_insert:
  assumes a: "(a :: 'a ::field vec) ∈ span (insert_col S b)"
    and na: "a ∉ span S"
    and br: "dim_vec b = dim_row S"
    and bnotS: "b ∉ set_cols S"
    and d: "distinct (cols S)"
  shows "b ∈ span (insert_col S a)"
using assms
proof -
  define nr where "nr = dim_row S"
```

```
  have carrier_a: "a ∈ carrier_vec nr"
    using a br carrier_dim_vec nr_def span_insert2 by fastforce
  have carrier_b: "b ∈ carrier_vec nr"
    by (simp add: br carrier_vecI nr_def)
  have bi: "b ∈ set_cols (insert_col S b)"
    by (simp add: br carrier_dim_vec insert_col)
  have br': "dim_vec b = dim_row (insert_col S b)" using br
    by (simp add: insert_col)
  have di: "distinct (cols (insert_col S b))" using d bnotS
    by (metis br carrier_vec_dim_vec cols_dim cols_mat_of_cols distinct.simps(2)
insert_col
    insert_subset list.simps(15) set_cols)
  from span_breakdown[of b "insert_col S b" a, OF bi a br' di]
  obtain k where k: "a - k ·v b ∈ span (delete_col S b)"
  by (smt (verit, best) assms(1) bnotS br delete_col filter_True mat_of_cols_cols

      mem_Collect_eq set_cols span_insert2)
  have "k ≠ 0"
  proof
    assume "k = 0"
    have s: "set_cols (delete_col S b) ⊆ set_cols S"
    by (metis (no_types, lifting) cols_dim cols_mat_of_cols delete_col
dual_order.trans
      filter_is_subset set_cols)
    have r: "dim_row (delete_col S b) = dim_row S"  by (simp add: delete_col)
    have dc: "distinct (cols (delete_col S b))"
    by (metis cols_dim cols_mat_of_cols d delete_col distinct_filter dual_order.trans

      filter_is_subset)
    have "a = a - k ·v b" using ‹k = 0›
      using a br carrier_dim_vec span_insert2 by fastforce
    then have "a ∈ Lattice_int.span (delete_col S b)" using k by auto
    then have "a ∈ span S" using span_mono[of "delete_col S b" S, OF
s r dc] by auto
    with na show False by blast
  qed
  then have eq: "b = (1/k) ·v a - (1/k) ·v (a - k ·v b)"
  proof -
    have carrier_kb: "- k ·v b ∈ carrier_vec nr" using carrier_b by simp
    have "1 / k ·v (- k ·v b) = -b" by (subst smult_smult_assoc, use ‹k≠0›
in ‹auto›)
    then have "(1/k) ·v (a + (- k ·v b)) = ((1/k) ·v a - b)"
    by (subst smult_add_distrib_vec[where a = "1/k" and v = a and w
= "-k·v b",
      OF carrier_a carrier_kb], auto)
    then have *: "(1/k) ·v (a - k ·v b) = ((1/k) ·v a - b)" unfolding uminus_scalar_mult
      by (metis carrier_a carrier_b minus_add_uminus_vec smult_carrier_vec)
    have **: "(1/k) ·v a - (1/k) ·v (a - k ·v b) = (1/k) ·v a - (1/k) ·v
a + b" unfolding *
```

```
    using carrier_a carrier_b by auto
  then show ?thesis unfolding ** using carrier_a carrier_b by auto
 qed
 from k have k2: "(1/k) ·_v (a - k ·_v b) ∈ span (delete_col S b)"
  by (rule span_scale)
 have sub: "span (delete_col S b) ⊆ span (insert_col S a)"
 proof (subst span_mono, goal_cases)
  case 1
  then show ?case
  by (smt (verit, best) carrier_a cols_dim cols_mat_of_cols delete_col
dual_order.refl
   dual_order.trans filter_is_subset insert_col insert_subset list.set(2)
nr_def set_cols)
 next
  case 2
  then show ?case by (simp add: delete_col insert_col)
 next
  case 3
  then show ?case by (metis cols_dim cols_mat_of_cols d delete_col
distinct_filter
    dual_order.trans filter_is_subset set_cols)
 qed auto
 then have "(1/k) ·_v (a - k ·_v b) ∈ span (insert_col S a)" using k2 sub
by blast
 moreover have "1 / k ·_v a ∈ span (insert_col S a)"
 by (metis Lattice_int.span_base Lattice_int.span_scale carrier_a cols_dim
cols_mat_of_cols
   insertCI insert_col insert_subsetI list.set(2) nr_def set_cols)
 ultimately show ?thesis by (subst eq)
   (metis Lattice_int.span_diff br carrier_a carrier_vecD carrier_vec_dim_vec

    index_minus_vec(2) index_smult_vec(2) insert_col mat_of_cols_carrier(2)
nr_def)
qed



lemma delete_not_in_set_cols:
assumes "b ∉ set_cols T"
shows "delete_col T b = T"
unfolding delete_col using assms
by (metis (mono_tags, lifting) filter_True mat_of_cols_cols set_cols)

lemma delete_col_span:
assumes "(a :: 'a ::comm_ring vec) ∈ span (delete_col T b)" "distinct
(cols T)"
shows "a ∈ span T"
using assms  using span_mono[of "delete_col T b" T]
by (metis (no_types, lifting) cols_dim cols_mat_of_cols delete_col distinct_filter
```

```
  dual_order.trans filter_is_subset in_mono mat_of_cols_carrier(2) set_cols)
```

Changing a generating element.

```
lemma in_span_delete_field:
assumes aT:"(a :: 'a :: field vec) ∈ span T" and aTb: "a ∉ span (delete_col
T b)"
  and dr: "dim_vec b = dim_row T" and d:"distinct (cols T)"
shows "b ∈ span (insert_col (delete_col T b) a)"
proof (subst in_span_insert[of a "delete_col T b" b], goal_cases)
  case 1
  then show ?case proof (cases "b∈set_cols T")
    case True
    show ?thesis using aT insert_delete_span[OF True d dr] by (simp)
  next
    case False
    then show ?thesis by (metis assms(1) assms(2) delete_not_in_set_cols)
  qed
  case 4
  then show ?case
  by (smt (verit) cols_dim cols_mat_of_cols delete_col dual_order.trans
filter_is_subset
    filter_set member_filter set_cols)
next
  case 5
  then show ?case
  by (metis cols_dim cols_mat_of_cols d delete_col distinct_filter dual_order.trans

    filter_is_subset)
qed (use assms in ‹auto simp add: dim_row_delete_col›)
```

Simplifications on multiplication.

```
lemma mat_of_cols_mult_vCons:
assumes "dim_vec (x :: 'a ::comm_ring vec) = dim_row S" "dim_vec zs =
dim_col S"
shows "mat_of_cols (dim_row S) (x # cols S) *v vCons z0 zs = z0 ·v x +
S *v zs"
using assms carrier_vec_dim_vec[of zs]
unfolding assms(2) by (subst mat_of_cols_cons_mat_vec, auto)

lemma mult_mat_vec_ring:
  assumes m: "(A::'a::comm_ring mat) ∈ carrier_mat nr nc" and v: "v
∈ carrier_vec nc"
  shows "A *v (k ·v v) = k ·v (A *v v)" (is "?l = ?r")
proof
  have nr: "dim_vec ?l = nr" using m v by auto
  also have "... = dim_vec ?r" using m v by auto
  finally show "dim_vec ?l = dim_vec ?r".
  show "⋀i. i < dim_vec ?r ⟹ ?l $ i = ?r $ i"
```

```
  proof -
    fix i assume "i < dim_vec ?r"
    hence i: "i < dim_row A" using nr m by auto
    hence i2: "i < dim_vec (A *_v v)" using m by auto
    show "?l $ i = ?r $ i"
    apply (subst (1) mult_mat_vec_def)
    apply (subst (2) smult_vec_def)
    unfolding index_vec[OF i] index_vec[OF i2]
    unfolding mult_mat_vec_def smult_vec_def
    unfolding scalar_prod_def index_vec[OF i]
    by (simp add: mult.left_commute sum_distrib_left)
  qed
qed
```

Adding a span element to the generating set does not change the span.

```
lemma span_redundant:
assumes "(x :: 'a ::comm_ring vec) ∈ span S"
shows "span (insert_col S x) = span S"
proof
  have "∃za. insert_col S x *_v z = S *_v za ∧ dim_vec za = dim_col S"

    if "dim_vec z = dim_col (insert_col S x)" for z
  proof -
    have dim_z: "dim_vec z = dim_col S + 1" using that by (simp add:
insert_col)
    obtain za where za_def: "z = vCons (z$0) za"
    by (metis <dim_vec z = dim_col (insert_col S x)> add_Suc_right dim_vec
insert_col list.size(4)
      mat_of_cols_carrier(3) nat.simps(3) vec_cases vec_index_vCons_0)
    obtain zx where zx_def: "x = S *_v zx" "dim_col S = dim_vec zx"
      using assms unfolding span_def by auto
    have "dim_vec x = dim_row S" using zx_def(1) by auto
    moreover have "dim_vec za = dim_col S"
    by (metis Suc_eq_plus1_left add.commute add_diff_cancel_left' dim_vec_vCons
dim_z za_def)
    ultimately have "insert_col S x *_v z = z$0 ·_v x + S *_v za"
      unfolding insert_col by (subst za_def, subst mat_of_cols_mult_vCons,
auto)
    also have "... = z$0 ·_v (S *_v zx) + S *_v za" unfolding zx_def by auto
    also have "... = S *_v (z $ 0 ·_v zx) + S *_v za"
      by (subst mult_mat_vec_ring[symmetric]) (auto simp add: zx_def(2))
    also have "... = S *_v (z$0 ·_v zx + za)"
      by (metis <dim_vec za = dim_col S> carrier_matI carrier_vec_dim_vec
index_smult_vec(2)
      mult_add_distrib_mat_vec zx_def(2))
    finally have "insert_col S x *_v z = S *_v (z$0 ·_v zx + za)" by blast
    moreover have "dim_vec (z$0 ·_v zx + za) = dim_col S"
    by (metis Suc_eq_plus1_left add.commute add_diff_cancel_left' dim_vec_vCons
dim_z
```

```
      index_add_vec(2) za_def)
    ultimately show ?thesis by blast
  qed
  then show "span (insert_col S x) ⊆ span S" unfolding span_def by auto
next
  have "∃za. S *v z = insert_col S x *v za ∧
              dim_vec za = dim_col (insert_col S x)" if "dim_vec z = dim_col
S" for z
  proof -
    define za where za_def: "za = vCons 0 z"
    have "dim_vec za = dim_col S + 1" using that za_def by force
    then have "dim_vec za = dim_col (insert_col S x)"
    by (simp add: insert_col)
    moreover have "S *v z = insert_col S x *v za"
    proof -
      have "dim_vec x = dim_row S"
      using Lattice_int.span_def assms by force
      moreover have "dim_vec z = dim_col S"
      using that by blast
      moreover have "S *v z = 0 ·v x + S *v z"
      using calculation(1) by auto
      ultimately show ?thesis unfolding insert_col za_def
        by (subst mat_of_cols_mult_vCons)
    qed
    ultimately show ?thesis by blast
  qed
  then show "span S ⊆ span (insert_col S x)" unfolding span_def by auto
qed
```

Transitivity of inserting elements.

```
lemma span_trans:
assumes "(x :: 'a ::comm_ring vec) ∈ span S" "y ∈ span (insert_col S
x)"
shows "y ∈ span S"
using Lattice_int.span_redundant assms by blast
```

More on inserting, deleting and the set of columns.

```
lemma delet_col_not_in_set_cols:
assumes "dim_vec b = dim_row T"
shows "b ∉ set_cols (delete_col T b)"
unfolding delete_col
by (metis (mono_tags, lifting) cols_dim cols_mat_of_cols dual_order.trans
filter_is_subset
filter_set member_filter set_cols)

lemma dim_col_distinct:
assumes "distinct (cols S)"
shows "card (set_cols S) = dim_col S"
by (simp add: assms distinct_card)
```

```
lemma set_cols_mono:
assumes "set_cols S ⊆ set_cols T" "distinct (cols S)" "distinct (cols
T)"
shows "dim_col S ≤ dim_col T"
using assms
by (metis List.finite_set card_mono dim_col_distinct set_cols)

lemma set_cols_insert_col:
assumes "dim_vec b = dim_row U"
shows "set_cols (insert_col U b) = set_cols U ∪ {b}"
by (metis (no_types, opaque_lifting) Un_commute Un_insert_right assms
carrier_vec_dim_vec
  cols_dim cols_mat_of_cols insert_col insert_subset list.set(2) set_cols
sup_bot.left_neutral)


lemma set_cols_real_of_int_mat:
  "set_cols (real_of_int_mat S) = real_of_int_vec ‘ (set_cols S)"
unfolding set_cols using real_of_int_mat_cols by auto

lemma real_of_int_mat_span:
"real_of_int_vec ‘ (span S) ⊆ span (real_of_int_mat S)"
by (smt (verit, ccfv_SIG) Lattice_int.span_def carrier_mat_triv carrier_vec_dim_vec

  image_subset_iff index_map_mat(3) index_map_vec(2) mem_Collect_eq of_int_hom.mult_mat_vec

  real_of_int_mat_def real_of_int_vec_def)

lemma real_of_int_vec_ex:
assumes "x ∈ real_of_int_vec ‘ A"
shows "∃y. x = real_of_int_vec y"
using assms by blast

lemma real_of_int_vec_obtain:
assumes "x ∈ real_of_int_vec ‘ A"
obtains y where "x = real_of_int_vec y"
using assms by blast

lemma real_of_int_vec_inj:
"inj real_of_int_vec"
unfolding real_of_int_vec_def
by (simp add: injI of_int_hom.vec_hom_inj)

lemma real_of_int_mat_mat_of_cols:
assumes "∀i<length cs. cs ! i ∈ carrier_vec nr"
shows "real_of_int_mat (mat_of_cols nr cs) = mat_of_cols nr (map real_of_int_vec
cs)"
proof-
```

```
  have "real_of_int (cs ! j $ i) = map real_of_int_vec cs ! j $ i" if
"j<length cs" "i<nr"
    for i j unfolding nth_map[OF that(1)] proof (subst real_of_int_vec_nth)

      have "dim_vec (cs ! j) = nr" using assms that(1)
      using carrier_vecD by blast
      then show "i < dim_vec (cs ! j)" using that(2) by auto
    qed (auto)
  then show ?thesis  unfolding mat_of_cols_def by (subst real_of_int_mat_mat,
    unfold o_def case_prod_beta, subst cong_mat, auto)
qed


lemma real_of_int_mat_delete_col:
"delete_col (real_of_int_mat T) (real_of_int_vec b) = real_of_int_mat
(delete_col T b)"
unfolding delete_col
proof (subst real_of_int_mat_mat_of_cols, goal_cases)
  case 1 show ?case by (meson cols_dim filter_is_subset nth_mem subset_iff)
next
  case 2
  have "(real_of_int_vec x ≠ real_of_int_vec b) = (x ≠ b)" for x
    by (simp add: inj_eq real_of_int_vec_inj)
  then have *:"filter (λx. x ≠ real_of_int_vec b) (cols (real_of_int_mat
T)) =
    map real_of_int_vec (filter (λx. x ≠ b) (cols T))"
  unfolding real_of_int_mat_cols filter_map comp_def by auto
  show ?case using cong1[OF *] by auto
qed
```

Lemma to exchange a subset of columns from one basis to another. Pay attention that this only works over the reals, not over the integers since we need to do divisions. The `exchange_lemma` was adapted from "HOL/Vector_Spaces.thy". Note that the connection "Jordan_Normal_Form/VS_Connect" between the type `('a,'k::finite) vec` and `'a vec` does not include the theorem `exchange_lemma`. due to some problems in lifting/transfer.

```
lemma exchange_lemma:
  assumes i: "is_indep (S::real mat)"
    and sp: "set_cols S ⊆ span T"
    and d: "distinct (cols T)"
    and dr: "dim_row S = dim_row T"
  shows "∃t'. dim_col t' = dim_col T ∧ set_cols S ⊆ set_cols t' ∧
              set_cols t' ⊆ set_cols S ∪ set_cols T ∧
              distinct (cols t')"

using i sp d dr
proof (induct "card (set_cols T - set_cols S)" arbitrary: S T rule: less_induct)
  case less
  define nr where "nr = dim_row S"
```

```
  have "nr = dim_row T" using less.prems(4) nr_def by blast
  have "nr = dim_row S"  by (simp add: nr_def)
  have "nr = dim_row T"  by (simp add: ‹nr = dim_row T›)
  have sp': "set_cols (S) ⊆ span (T)" using less.prems(2)
    unfolding set_cols_real_of_int_mat using real_of_int_mat_span by
(auto)
  have d': "distinct (cols (T))"
    using distinct_cols_real_of_int_mat less.prems(3) by blast
  have dr': "dim_row (S) = dim_row (T)" by (simp add: less.prems(4))
  have ft: "finite (set_cols T)" by auto
  note S = ‹is_indep S›
  let ?P = "λt'. dim_col t' = dim_col T ∧ set_cols S ⊆ set_cols t' ∧

              set_cols t' ⊆ set_cols S ∪ set_cols T ∧ distinct (cols
t')"
  show ?case
  proof (cases "set_cols S ⊆ set_cols T  ∨ set_cols T ⊆ set_cols S")
    case True
    then show ?thesis
    proof
      assume "set_cols S ⊆ set_cols T"
      then have "set_cols S ⊆ set_cols T"
      using   set_cols_real_of_int_mat by auto
      then show ?thesis using d' by blast
    next
      assume s: "set_cols T ⊆ set_cols S"
      then show ?thesis
      using spanning_subset_independent[OF s S _ less.prems(3) less.prems(4)]
      using less.prems by (metis d' inf_sup_ord(3))
    qed
  next
    case False
    then have st: "¬ set_cols S ⊆ set_cols T" "¬ set_cols T ⊆ set_cols
S"
      by auto
    from st(2) obtain b where b: "b ∈ set_cols T" "b ∉ set_cols S"
      by blast
    have "nr = dim_vec b"
      by (metis ‹nr = dim_row T› b(1) carrier_vecD cols_dim set_cols
subset_eq)
    from b have "set_cols T - {b} - (set_cols S) ⊂ set_cols T - set_cols
S"
      by blast
    then have cardlt: "card (set_cols T - {b} - set_cols S) < card (set_cols
T - set_cols S)"
      using ft by (auto intro: psubset_card_mono)
    from b ft have ct0: "card (set_cols T) ≠ 0"
      by auto
    let ?T_b = "delete_col T b"
```

```
    have "nr = dim_row ?T_b" by (simp add: ‹nr = dim_row T› dim_row_delete_col)
    from ft have ftb: "finite (set_cols ?T_b)"
      by auto
    have dim_col_T_gr_0: "dim_col T > 0" using b(1)
    by (metis atLeastLessThan_iff bot_nat_0.not_eq_extremum cols_def imageE
list.set_map
      set_cols set_upt zero_order(3))
    have dim_col_T_b: "dim_col ?T_b = dim_col T - 1"
    proof -
      have "length (cols T) > 0" using dim_col_T_gr_0 unfolding cols_length
by auto
      have "length (filter (λx. x = b) (cols T)) = 1"
      proof -
        have "length (filter (λx. x = b) (cols T)) =
          card {i. i < dim_col T ∧ col T i = b}"
          unfolding length_filter_conv_card using cols_nth by (metis
cols_length)
        then show ?thesis using d' ‹b ∈ set_cols T›
        by (smt (verit, del_insts) Collect_cong List.finite_set One_nat_def
card.empty
          card.insert distinct_card distinct_filter equals0D list.set(1)
set_cols set_filter
          singleton_conv)
      qed
      then have "length (filter (λx. x ≠ b) (cols T)) = length (cols
T) - 1"
         using sum_length_filter_compl by (metis add_diff_cancel_left')

      then show ?thesis unfolding delete_col by auto
    qed
    have "dim_vec b = dim_row T"
      using b(1) carrier_dim_vec cols_dim set_cols by blast
    have set_cols_T_b: "set_cols ?T_b = set_cols T - {b}"
    proof -
      have "set_cols ?T_b ⊆ set_cols T"
      by (smt (verit) cols_dim cols_mat_of_cols delete_col filter_is_subset
order_trans set_cols)
      moreover have "b ∉ set_cols ?T_b" using ‹dim_vec b = dim_row T›
        by (subst delet_col_not_in_set_cols, auto)
      ultimately show ?thesis
      by (smt (verit, del_insts) Diff_subset cols_dim cols_mat_of_cols
delete_col dual_order.trans
      set_cols set_minus_filter_out)
    qed
    show ?thesis

    proof (cases "set_cols S ⊆ span ?T_b")
      case True
```

```
      from cardlt have cardlt': "card (set_cols ?T_b - set_cols S) <

        card (set_cols T - set_cols S)"
        using set_cols_T_b by presburger
      have d_delete: "distinct (cols (delete_col T b))" using d
      by (metis b(1) card_Diff_singleton card_distinct cols_length d'
dim_col_T_b dim_col_distinct
        set_cols set_cols_T_b)
      have dim_row_STb: "dim_row S = dim_row (delete_col T b)"
        by (simp add: dim_row_delete_col less.prems(4))
      obtain U where U: "dim_col U = dim_col ?T_b" "set_cols S ⊆ set_cols
U"
        "set_cols U ⊆ set_cols S ∪ set_cols ?T_b" "distinct (cols U)"

        using less(1)[OF cardlt' S True d_delete dim_row_STb]  by auto
      have "nr = dim_row U" using U(2) unfolding ‹nr = dim_row S›
        by (smt (verit, best)  carrier_vecD cols_dim image_subset_iff
set_cols
          set_cols_real_of_int_mat st(1) subsetD subsetI)
      let ?w = "insert_col U b"
      have dim_col_w: "dim_col ?w = dim_col U + 1" unfolding insert_col
by simp
      have "dim_vec b = dim_row U" unfolding ‹nr = dim_vec b›[symmetric]

        using ‹nr = dim_row U› .
      then have set_cols_w: "set_cols ?w = set_cols U ∪ {b}"
        by (subst set_cols_insert_col, auto)
      have th0: "set_cols S ⊆ set_cols ?w"
        using U(2) set_cols_w by auto
      have th1: "set_cols ?w ⊆ set_cols S ∪ set_cols T"
        using U b set_cols_w set_cols_T_b by auto
      have bu: "b ∉ set_cols U"
        using b U
        by (metis DiffD2 Un_iff insertCI set_cols_T_b sup.orderE)
      from U(1) have "dim_col U = dim_col T - 1"
      using dim_col_T_b by presburger
      then have th2: "dim_col (?w) = dim_col T" unfolding dim_col_w
      using dim_col_T_gr_0 by linarith
      have th3: "distinct (cols ?w)"
      proof -
        have "b ∉ set_cols U" using bu by blast
        then show ?thesis
        by (metis List.finite_set Suc_pred' U(4) ‹dim_col U = dim_col
T - 1› card_distinct
          card_insert_disjoint cols_length dim_col_T_gr_0 dim_col_distinct
insert_union set_cols
          set_cols_w th2)
      qed
      from th0 th1 th2 th3 have th: "?P ?w" by blast
```

```
    from th show ?thesis by blast
  next
    case False
    then obtain a where a: "a ∈ set_cols S" "a ∉ span ?T_b" by blast
    have ab: "a ≠ b" using a b by blast
    have at: "a ∉ set_cols T" using a ab span_base[of a "?T_b"]
      by (metis DiffI empty_iff insert_iff set_cols_T_b)
    let ?insert_a =  "insert_col ?T_b a"
    have set_cols_insert: "set_cols ?insert_a = {a} ∪ set_cols ?T_b"

      by (smt (verit, ccfv_SIG) Lattice_int.span_def Un_commute a(1)
delete_col dim_mult_mat_vec
        less.prems(2) mat_of_cols_carrier(2) mem_Collect_eq set_cols_insert_col
subsetD)
    have dim_col_insert_a: "dim_col (?insert_a) = dim_col T"
      by (metis One_nat_def Suc_eq_plus1_left Suc_pred' add.commute
cols_length dim_col_T_b
        dim_col_T_gr_0 insert_col list.size(4) mat_of_cols_carrier(3))
    have mlt: "card (set_cols ?insert_a - set_cols S) < card (set_cols
T - set_cols S)"
      using cardlt a b by (metis insert_Diff1 insert_is_Un set_cols_T_b
set_cols_insert)
    have sp': "set_cols S ⊆ span (?insert_a)"
    proof
      fix x
      assume xs: "x ∈ set_cols S"
      let ?insert_b =  "mat_of_cols (dim_row T) (b # cols ?insert_a)"
      have T: "set_cols T ⊆ set_cols ?insert_b"
        using b
        by (smt (verit, ccfv_SIG) cols_dim cols_mat_of_cols delete_col
dual_order.refl
          insert_Diff insert_col insert_is_Un insert_subset list.set(2)
mat_of_cols_carrier(2)
          set_cols set_cols_T_b set_cols_insert)
      have bs: "b ∈ span (?insert_a)"
        using in_span_delete_field
        by (metis <dim_vec b = dim_row T> a(1) a(2) d' in_mono sp')
      from xs sp have "x ∈ span T" using less.prems(2) by blast
      with span_mono[OF T] have x: "x ∈ span (?insert_b)"
      using less.prems(3) by auto
      from span_trans show "x ∈ span (?insert_a)"
      by (metis bs delete_col insert_col mat_of_cols_carrier(2) x)
    qed
    from less(1)[OF mlt S sp'] obtain U where U:
      "dim_col U = dim_col (insert_col (?T_b) a)"
      "set_cols S ⊆ set_cols U" "set_cols U ⊆ set_cols S ∪ set_cols
(insert_col (?T_b) a)"
      "distinct (cols U)"
      by (metis Lattice_int.span_base <dim_vec b = dim_row T> a(2)
```

```
b(1) card_distinct
          card_insert_disjoint cols_length delet_col_not_in_set_cols delete_col
dim_col_distinct
          dim_col_insert_a ftb insert_Diff insert_col insert_is_Un less.prems(3)
less.prems(4)
          mat_of_cols_carrier(2) set_cols set_cols_T_b set_cols_insert)
      from U a b ft at ct0 have "?P U"
      using dim_col_insert_a set_cols_T_b set_cols_insert by auto
      then show ?thesis by blast
    qed
  qed
qed
```

A linearly independent set has smaller or equal cardinality than the span it lies in.

```
lemma independent_span_bound:
  assumes i: "is_indep (S :: int mat)" and d: "distinct (cols T)"
    and sp: "set_cols S ⊆ span T"
    and dr: "dim_row S = dim_row T"
  shows "dim_col S ≤ dim_col T"
proof -
  have i': "is_indep (real_of_int_mat S)" using i unfolding is_indep_real_def
is_indep_int_def
    by auto
  have sp': "set_cols (real_of_int_mat S) ⊆ Lattice_int.span (real_of_int_mat
T)"
    using sp by (metis image_mono real_of_int_mat_span set_cols_real_of_int_mat
subset_trans)
  have d': "distinct (cols (real_of_int_mat T))" using d
    by (simp add: distinct_cols_real_of_int_mat)
  have dr': "dim_row (real_of_int_mat S) = dim_row (real_of_int_mat T)"
using dr
    by (simp add: distinct_cols_real_of_int_mat)
  obtain t' where t: "dim_col t' = dim_col (real_of_int_mat T)"
       "set_cols (real_of_int_mat S) ⊆ set_cols t'"
       "set_cols t' ⊆ set_cols (real_of_int_mat S) ∪ set_cols (real_of_int_mat
T)"
       "distinct (cols t')"
  using exchange_lemma[OF i' sp' d' dr'] by blast
  have dS: "distinct (cols (real_of_int_mat S))" using is_indep_distinct[OF
i']
    using distinct_cols_real_of_int_mat by blast
  have "dim_col S = dim_col (real_of_int_mat S)" by simp
  moreover have "dim_col (real_of_int_mat S) ≤ dim_col t'"
    using set_cols_mono[OF t(2) dS t(4)] by auto
  moreover have "dim_col t' = dim_col T" unfolding t(1) by simp
  ultimately show ?thesis  by auto
qed
```

When two bases B and B' generate the same lattice, both have the same length because the change of basis theorem allows us to convert one basis in the other.

```
lemma gen_lattice_in_span:
assumes "gen_lattice B = gen_lattice B'"
shows "set_cols B ⊆ Lattice_int.span B'"
unfolding gen_lattice_def
by (smt (verit, ccfv_SIG) Lattice_int.span_base Lattice_int.span_def assms
gen_lattice_def subsetI)


lemma basis_exchange:
  assumes gen_eq: "gen_lattice B = gen_lattice B'"
    and "is_indep B" and "is_indep B'"
  shows "dim_col B = dim_col B'"
proof -
  have i:"is_indep (real_of_int_mat B)" "is_indep (real_of_int_mat B')"
    using assms unfolding is_indep_real_def is_indep_int_def by auto
  have dr: "dim_row B = dim_row B'"
  by (smt (z3) dim_mult_mat_vec gen_eq gen_lattice_def index_zero_vec(2)
mem_Collect_eq)
  have "dim_col B ≤ dim_col B'"
    using assms(2) is_indep_distinct[OF i(2)] gen_lattice_in_span[OF assms(1)]
dr
     Lattice_int.independent_span_bound distinct_cols_real_of_int_mat
by presburger
  moreover have "dim_col B' ≤ dim_col B"
    using assms(3) is_indep_distinct[OF i(1)] gen_lattice_in_span[OF assms(1)[symmetric]]
dr
     Lattice_int.independent_span_bound distinct_cols_real_of_int_mat
by presburger
  ultimately show ?thesis by linarith
qed
```

Basis matrix of a lattice

```
definition basis_of :: "int vec set ⇒ int mat" where
  "basis_of L = (SOME B. L = gen_lattice B ∧ is_indep B)"

definition dim_lattice :: "int_lattice ⇒ nat" where
  "dim_lattice L = (THE x. x = dim_col (basis_of L))"

lemma dim_lattice_gen_lattice:
  assumes "is_indep B"
  shows "dim_lattice (gen_lattice B) = dim_col B"
unfolding dim_lattice_def using assms basis_exchange[of "basis_of (gen_lattice
B)" B]
  by (auto simp add: basis_of_def)
     (metis (mono_tags, lifting) tfl_some)
```

A lattice generated by a linearly independent matrix is indeed a lattice.

lemma *is_lattice_gen_lattice:*
  assumes *"is_indep A"*
  shows *"is_lattice (gen_lattice A)"*
unfolding *is_lattice_def gen_lattice_def* **using** *assms* **by** *auto*

end
theory *Partition*

imports
  Main
  Reduction
begin

5 Partition Problem

The Partition Problem is a widely known NP-hard problem. TODO: Reduction proof to SAT

definition *is_partition :: "int list ⇒ nat set ⇒ bool"* **where**
  "is_partition a I = (I ⊆ {0..<length a} ∧ (∑i∈I. a ! i) = (∑i∈({0..<length a}-I). a ! i))"

definition *partition_problem :: "(int list) set "* **where**
  "partition_problem = {a. ∃I. I ⊆ {0..<length a} ∧
        (∑i∈I. a ! i) = (∑i∈({0..<length a}-I). a ! i)}"

definition *partition_problem_nonzero :: "(int list) set "* **where**
  "partition_problem_nonzero = {a. ∃I. I ⊆ {0..<length a} ∧ length a > 0 ∧
        (∑i∈I. a ! i) = (∑i∈({0..<length a}-I). a ! i)}"

end
theory *Subset_Sum*

imports
  Main
  Partition
  "Jordan_Normal_Form.Matrix"
begin

6 Subset Sum Problem

The Subset Sum Problem is a common NP-hard Problem.

definition *subset_sum :: "((int vec) * int) set"* **where**
"subset_sum ≡ {(as,s). (∃xs::int vec.
(∀i<dim_vec xs. xs$i ∈ {0,1}) ∧ xs · as = s ∧ dim_vec xs = dim_vec as)}"

definition *subset_sum_nonzero :: "((int vec) * int) set"* **where**
"subset_sum_nonzero ≡ {(as,s). (∃xs::int vec.
(∀i<dim_vec xs. xs$i ∈ {0,1}) ∧ xs · as = s ∧ dim_vec xs = dim_vec as) ∧ dim_vec as ≠ 0}"

definition *subset_sum_list :: "((int list) * int) set"* **where**
"subset_sum_list ≡ {(as,s). (∃xs::int list.
*(∀i<length xs. xs!i ∈ {0,1}) ∧ (∑i<length as. as!i * xs!i) = s ∧ length xs = length as)}"*

Reduction Subset Sum to Partition.

definition *reduce_subset_sum_partition:: "(int list) ⇒ ((int list) * int)"* **where**
"reduce_subset_sum_partition ≡
(λ a. (a, (if sum_list a mod 2 = 0 then sum_list a div 2 else (∑i<length a. |a!i|) + 1)))"

Well-definedness of reduction function

lemma *sum_list_map_2:*
assumes *"length a = length xs"*
shows *"(sum_list (map2 (*) a xs)) = (∑i<length a. a!i * xs!i)"*
using *assms* **proof** *(induct rule: list_induct2)*
case *(Cons x xs y ys)*
have *"(∑i<length ys. (x # xs) ! i * (y # ys) ! i) + (x # xs) ! length ys * (y # ys) ! length ys =*
*(∑i≤length ys. (x # xs) ! i * (y # ys) ! i)"*
by *(metis (no_types, lifting) lessThan_Suc_atMost sum.lessThan_Suc)*
also have *"... = (x # xs) ! 0 * (y # ys) ! 0 +*
*(∑i<length ys. (x # xs) ! (i+1) * (y # ys) ! (i+1))"*
using *sum.atMost_shift[of "(λi. (x # xs) ! i * (y # ys) ! i)" "length ys"]* **by** *auto*
also have *"... = x * y + sum_list (map2 (*) xs ys)"* **using** *Cons* **by** *auto*
finally have *"x * y + sum_list (map2 (*) xs ys) = (∑i<length ys. (x # xs) ! i * (y # ys) ! i) +*
*(x # xs) ! length ys * (y # ys) ! length ys"* **by** *presburger*
then show *?case* **using** *Cons* **by** *auto*
qed *auto*

```
lemma ex_01_list:
  assumes "I⊆{0..<n}"
  shows "∃xs::int list. (∀i∈I. xs ! i = 1) ∧ (∀i∈{0..<n}-I. xs ! i
= 0) ∧ length xs = n"
using assms proof (induct n arbitrary: I)
  case (Suc n)
  define I' where "I' = I ∩ {0..<n}"
  then have "I'⊆ {0..<n}" by auto
  obtain xs'::"int list" where xs'_def:
    "(∀i∈I'. xs' ! i = 1) ∧ (∀i∈{0..<n} - I'. xs' ! i = 0) ∧ length
xs' = n"
    using Suc(1)[OF ‹I'⊆{0..<n}›] by blast
  define xs where "xs = xs' @ [if n ∈ I then 1 else 0]"
  then have eq: "xs ! i = xs' ! i" if "i<n" for i using that
    by (simp add: nth_append xs'_def)+
  have 1: "(∀i∈I. xs ! i = 1)" unfolding xs_def
  proof (split if_split, safe, goal_cases)
    case (1 i)
    then have "¬ i < n ⟹ [1] ! (i - n) = 1"
    by (metis Suc.prems atLeast0LessThan atMost_iff cancel_comm_monoid_add_class.diff_cance

      lessThan_Suc_atMost linorder_neqE_nat not_le nth_Cons_0 subset_eq)
    with 1 show ?case by (subst nth_append)
     (use eq in ‹auto simp add: xs'_def I'_def Suc xs_def›)
  next
    case (2 i)
    then have "i < n"
      by (metis Suc.prems atLeastLessThan_iff less_antisym subset_eq)
    with 2 show ?case by (subst nth_append)
      (use eq in ‹auto simp add: xs'_def I'_def Suc xs_def›)
  qed
  moreover have 2: "(∀i∈{0..<Suc n} - I. xs ! i = 0)"unfolding xs_def

  proof (split if_split, safe, goal_cases)
    case (1 i)
    then have "i<n" by (metis atLeastLessThan_iff less_antisym)
    with 1 show ?case by (subst nth_append) (auto simp add: xs'_def
I'_def Suc xs_def)
  next
    case (2 i)
    then show ?case by (subst nth_append) (auto simp add: xs'_def I'_def
Suc xs_def)
  qed
  moreover have "length xs = Suc n" by (auto simp add: xs'_def I'_def
Suc xs_def)
  ultimately have "(∀i∈I. xs ! i = 1) ∧ (∀i∈{0..<Suc n} - I. xs ! i
= 0) ∧ length xs = Suc n"
    by auto
  then show ?case by (subst exI, auto)
```

```
qed auto

lemma sum_list_even:
  assumes "a ∈ partition_problem"
  shows "sum_list a mod 2 = 0"
using assms unfolding partition_problem_def
proof (safe, goal_cases)
  case (1 I)
  then have "sum ((!) a) {0..<length a} = sum ((!) a) I + sum ((!) a)
({0..<length a} - I)"
    by (metis finite_atLeastLessThan sum.subset_diff)
  then have "sum ((!) a) {0..<length a} = 2 * sum ((!) a) I" using 1
by auto
  then show ?case by (simp add: sum_list_sum_nth)
qed

lemma well_defined_reduction_subset_sum:
  assumes "a ∈ partition_problem"
  shows "reduce_subset_sum_partition a ∈ subset_sum_list"
using assms unfolding partition_problem_def reduce_subset_sum_partition_def
subset_sum_list_def Let_def
proof (safe, goal_cases)
  case (1 I)
  have sum_even: "sum_list a mod 2 = 0" using sum_list_even[OF assms]
by auto
  then have if_rewrite:
    "(if sum_list a mod 2 = 0 then sum_list a div 2 else (∑ i<length a.
|a!i|) + 1) =
    sum_list a div 2" by auto
  have split_set: "{0..<length a} = I ∪ ({0..<length a}-I)"
    using 1(1) by (subst Un_Diff_cancel, auto)
  have disjoint: "I ∩ ({0..<length a}-I) = {}" by auto
  have "∃xs::int list. (∀i∈I. xs ! i = 1) ∧ (∀i∈{0..<length a}-I. xs
! i = 0) ∧
    length xs = length a" using ex_01_list[OF ‹I ⊆ {0..<length a}›] by
auto
  then obtain xs::"int list" where
    xs_prop: "∀i∈I. xs ! i = 1"
    "∀i∈{0..<length a}-I. xs ! i = 0"
    "length xs = length a" by blast
  then have eq: "(∑ i<length a. a!i * xs!i) = (∑ i∈I. a!i)" using split_set
1(1)
    by (metis (no_types, lifting) finite_lessThan lessThan_atLeast0 mult_cancel_left2

      mult_cancel_right2 sum.cong sum.mono_neutral_right)
  have "sum_list a = (∑ i∈I. a!i) + (∑ i∈{0..<length a}-I. a!i)" us-
ing 1(1)
    by (smt (verit) finite_atLeastLessThan sum.subset_diff sum_list_sum_nth)
  also have "... = 2 * (∑ i∈I. a!i)" using 1(2)[symmetric] by auto
```

```
  also have "... = 2 * (sum_list (map2 (*) a xs))" using eq xs_prop by
(subst sum_list_map_2, auto)
  finally have "sum_list (map2 (*) a xs) = (sum_list a) div 2" by simp
  then show ?case using xs_prop sum_list_map_2 ‹2 * sum ((!) a) I = 2
* sum_list (map2 (*) a xs)› eq
    by (subst exI[of _ xs], auto simp add: if_rewrite)
qed
```

NP-hardness of reduction function

```
lemma NP_hardness_reduction_subset_sum:
  assumes "reduce_subset_sum_partition a ∈ subset_sum_list"
  shows "a ∈ partition_problem"
using assms unfolding partition_problem_def reduce_subset_sum_partition_def
subset_sum_list_def Let_def
proof (safe, goal_cases)
  case (1 xs)
  have "(sum_list a) mod 2 = 0" using 1
  proof -
    have "(∑ i<length a. a ! i * xs ! i) ≤ (∑ i<length a. |a!i|)" us-
ing 1 by (subst sum_mono, auto)
    then have "(∑ i<length a. a ! i * xs ! i) ≠ (∑ i<length a. |a!i|)
+ 1" by auto
    then show ?thesis using 1 by presburger
  qed
  obtain I where I_prop: "I = {i. i< length xs ∧ xs!i = 1}" by blast
  then have split_set: "{0..<length a} = I ∪ ({0..<length a}-I)"
    using I_prop 1 by (subst Un_Diff_cancel, auto)
  have "I⊆{0..<length a}" using I_prop 1 by auto
  moreover have "sum ((!) a) I = sum ((!) a) ({0..<length a} - I)"
  proof -
    have eq: "(∑ i∈I. a!i) = (∑ i<length a. a!i * xs!i)" using split_set
I_prop 1
      by (smt (verit, ccfv_threshold) Diff_iff ‹I ⊆ {0..<length a}› atLeast0LessThan

      atLeastLessThan_iff empty_iff finite_atLeastLessThan insert_iff
mem_Collect_eq
      mult_cancel_left2 mult_cancel_right2 sum.cong sum.mono_neutral_right)
    have "(∑ i∈{0..<length a}-I. a!i) = sum ((!) a) {0..<length a} -
(∑ i∈I. a!i)" using split_set
      by (meson ‹I ⊆ {0..<length a}› finite_atLeastLessThan sum_diff)
    also have "... = sum_list a - sum_list a div 2"
    by (metis "1"(2) ‹sum_list a mod 2 = 0› eq sum_list_sum_nth)
    also have "... = sum_list a div 2" using ‹(sum_list a) mod 2 = 0›
by auto
    finally have "(∑ i∈{0..<length a}-I. a!i) = (∑ i∈I. a!i)"
      using "1"(2) ‹sum_list a mod 2 = 0› eq by presburger
    then show ?thesis by auto
  qed
  ultimately show ?case by (subst exI, auto)
```

qed

The Subset Sum on lists is NP-hard.

```
lemma "is_reduction reduce_subset_sum_partition partition_problem subset_sum_list"
unfolding is_reduction_def
proof (safe, goal_cases)
  case (1 a)
  then show ?case using well_defined_reduction_subset_sum by auto
next
  case (2 a)
  then show ?case using NP_hardness_reduction_subset_sum by auto
qed
```

Subset Sum on vectors is the same as on lists

```
lemma subset_sum_vec_list:
"(as,s) ∈ subset_sum ⟷ (list_of_vec as,s) ∈ subset_sum_list"
proof (safe, goal_cases)
  case 1
  then have "∃xs. (∀i<dim_vec xs. xs $ i ∈ {0, 1}) ∧ xs · as = s ∧
dim_vec xs = dim_vec as"
    unfolding subset_sum_def by auto
  then have " ∃xs. (∀i<length xs. xs ! i ∈ {0, 1}) ∧
             (∑i<length (list_of_vec as). (list_of_vec as) ! i * xs !
i) = s ∧
              length xs = length (list_of_vec as)"
  unfolding scalar_prod_def
  by (metis (no_types, lifting) atLeast0LessThan length_list_of_vec mult.commute

    sum.cong vec_list vec_of_list_index)
  then show ?case unfolding subset_sum_list_def by auto
next
  case 2
  then have " ∃xs. (∀i<length xs. xs ! i ∈ {0, 1}) ∧
             (∑i<length (list_of_vec as). (list_of_vec as) ! i * xs !
i) = s ∧
              length xs = length (list_of_vec as)"
  unfolding subset_sum_list_def by auto
  then have "∃xs. (∀i<dim_vec xs. xs $ i ∈ {0, 1}) ∧ xs · as = s ∧
dim_vec xs = dim_vec as"
  unfolding scalar_prod_def
  by (metis (no_types, lifting) dim_vec_of_list lessThan_atLeast0 mult.commute

    sum.ivl_cong vec_list vec_of_list_index)
  then show ?case unfolding subset_sum_def by auto
qed

end
theory CVP_p
```

```
imports
  Main
  Reduction
  Lattice_int
  Subset_Sum
begin
```

7 CVP in ℓ_p for $p \geq 1$

This file provides the reduction proof of Subset Sum to CVP in ℓ_p. Proof can be easily instantiated for any $p \geq 1$ using the locale variables.

```
definition pth_p_norm_vec :: "nat ⇒ ('a::{abs, power, comm_monoid_add})
vec ⇒ 'a" where
  "pth_p_norm_vec p v = (∑ i<dim_vec v. |v$i|^p)"


locale fixed_p =
fixes p::nat
assumes p_def: "p≥1"
begin

definition "p_norm_vec ≡ pth_p_norm_vec p"
```

The CVP in ℓ_p

```
definition is_closest_vec :: "int_lattice ⇒ int vec ⇒ int vec ⇒ bool"
where
  "is_closest_vec L b v ≡ (is_lattice L) ∧
    (∀ x∈L. p_norm_vec (x - b) ≥ p_norm_vec (v - b) ∧ v∈L)"
```

The decision problem associated with solving CVP exactly.

```
definition gap_cvp :: "(int_lattice × int vec × real) set" where
  "gap_cvp ≡ {(L, b, r). (is_lattice L) ∧ (∃ v∈L. of_int (p_norm_vec
(v - b)) ≤ r^p)}"
```

Reduction function for Subset Sum to CVP in euclidean norm

```
definition gen_basis_p :: "int vec ⇒ int mat" where
  "gen_basis_p as = mat (dim_vec as + 1) (dim_vec as) (λ (i, j). if i
= 0 then as$j
    else (if i = j + 1 then 2 else 0))"


definition gen_t_p :: "int vec ⇒ int ⇒ int vec" where
  "gen_t_p as s = vec (dim_vec as + 1) ((λ i. 1)(0:= s))"


definition reduce_cvp_subset_sum_p ::
  "((int vec) * int) ⇒ (int_lattice * (int vec) * real)" where
```

```
  "reduce_cvp_subset_sum_p ≡ (λ (as,s).
    (gen_lattice (gen_basis_p as), gen_t_p as s, root p (dim_vec as)))"
```

Lemmas for Proof

```
lemma vec_lambda_eq[intro]: "(∀ i<n. a i = b i) ⟶ vec n a = vec n b"
by auto

lemma eq_fun_applic: assumes "x = y" shows "f x = f y"
using assms by auto


lemma sum_if_zero:
  assumes "finite A" "i∈A"
  shows "(∑ j∈A. (if i = j then a j else 0)) = a i"
proof -
  have "(∑ x∈A. if x = i then a x else 0) =
  (if i = i then a i else 0) + (∑ x∈A - {i}. if x = i then a x else 0)"
  using sum.remove[OF assms, of "(λx. if x = i then a x else 0)"] by auto
  then show ?thesis by (simp add: assms(1))
qed


lemma set_compr_elem:
  assumes "finite A" "a∈A"
  shows "{f i | i. i∈A} = {f a} ∪ {f i | i. i∈A-{a}}"
by (safe, use assms in <auto>)


lemma Bx_rewrite:
  assumes x_dim: "dim_vec as = dim_vec x"
  shows "(gen_basis_p as) *_v x =
    vec (dim_vec as + 1) (λ i. if i = 0 then (x · as)
    else (2 * x$(i-1)))"
    (is "?init_vec = ?goal_vec")
proof -
  define n::nat where n_def: "n = dim_vec as"
  have "vec n (λj. (as $ j)) ·  x = (x · as)"
    unfolding n_def scalar_prod_def using x_dim by (simp add: mult.commute)
  moreover have "vec (dim_vec as) (λj. if i = Suc j then 2 else 0) ·
x =
        2 * x $ (i - Suc 0)" if "i < Suc (dim_vec as)" "0 < i" for i
  proof -
    have "(∑ ia = 0..<dim_vec x.
      vec (dim_vec as) (λj.  (if i = (Suc j) then 2 else 0)) $ ia * x
$ ia) =
      (∑ ia<n.  (if i = ia+1 then 2 * (x $ ia) else 0))"
      by (intro sum.cong, auto simp add: n_def x_dim)
    also have "... = (∑ ib∈{1..<n+1}.
        (if i = ib then 2 * (x $ (ib-1)) else 0))"
```

```
    proof -
      have eq: "(λib.  (if i = ib then 2 * x $ (ib - 1) else 0)) ∘ (+)
1
          = (λia.  (if i = ia + 1 then 2 * x $ ia else 0))"
      by auto
      then show ?thesis
        by (subst sum.atLeastLessThan_shift_0[
            of "(λib.  (if i = ib then 2 * x $ (ib - 1) else 0))" 1 "n+1"])
          (subst eq, use lessThan_atLeast0 in <auto>)
    qed
    also have "... = 2 *  (x $ (i-1))"
    proof -
      have finite: "finite {1..<n+1}" by auto
      have is_in: "i ∈ {1..<n+1}" using that by (auto simp add: n_def)
      show ?thesis
      by (subst sum_if_zero[OF finite is_in, of "(λk. 2 * (x $ (k-1)))"],
auto)
    qed
    finally show ?thesis unfolding scalar_prod_def by auto
  qed
  ultimately show ?thesis
    unfolding gen_basis_p_def reduce_cvp_subset_sum_p_def gen_t_p_def
    by (intro eq_vecI, auto simp add: n_def)
qed


lemma Bx_s_rewrite:
  assumes x_dim: "dim_vec as = dim_vec x"
  shows "(gen_basis_p as) *v x - (gen_t_p as s) =
    vec (dim_vec as + 1) (λ i. if i = 0 then  (x · as - s) else  (2 *
x$(i-1) - 1))"
    (is "?init_vec = ?goal_vec")
unfolding gen_t_p_def by (subst  Bx_rewrite[OF assms], auto)


lemma p_norm_vec_Bx_s:
  assumes x_dim: "dim_vec as = dim_vec x"
  shows "p_norm_vec ((gen_basis_p as) *v x - (gen_t_p as s)) =
     |x · as - s|^p + (∑ i=1..<dim_vec as +1. |2*x$(i-1)-1|^p)"
proof -
  let ?init_vec = "(gen_basis_p as) *v x - (gen_t_p as s)"
  let ?goal_vec = "vec (dim_vec as + 1) (λ i. if i = 0 then (x · as - s)
       else  (2 * x$(i-1) - 1))"
  have "p_norm_vec ?init_vec = p_norm_vec ?goal_vec" using Bx_s_rewrite[OF
x_dim] by auto
  also have "... = (∑ i∈{0..<dim_vec as+1}. |?goal_vec$i|^p)"
    unfolding p_norm_vec_def pth_p_norm_vec_def by (metis atLeast0LessThan
dim_vec)
  also have "... = |x · as - s|^p + (∑ i∈{1..<dim_vec as+1}. |2*x$(i-1)-1|^p)"
```

```
  proof -
    have subs: "{0}⊆{0..<dim_vec as+1}" by auto
    have "{0..<dim_vec as +1} = {0} ∪ {1..<dim_vec as +1}" by auto
    then show ?thesis by (subst sum.subset_diff[OF subs],auto)
  qed
  finally show ?thesis by blast
qed
```

gen_basis_p actually generates a basis which spans the *int_lattice* (by definition) and is linearly independent.

```
lemma is_indep_gen_basis_p:
  "is_indep (gen_basis_p as)"
unfolding is_indep_int_def
proof (safe, goal_cases)
case (1 z)
  let ?n = "dim_vec as"
  have z_dim: "dim_vec z = ?n" using 1(2) unfolding gen_basis_p_def by
auto
  have dim_row: "dim_row (gen_basis_p as) = ?n + 1" unfolding gen_basis_p_def
by auto
  have eq: "(real_of_int_mat (gen_basis_p as)) *v z = vec (?n + 1) (λ
i. if i = 0
    then z · (real_of_int_vec as) else (2 * z$(i-1)))"
  (is "(real_of_int_mat (gen_basis_p as)) *v z = ?goal_vec")
  proof -
    have scal_prod_com: "z · real_of_int_vec as = real_of_int_vec as ·
z"
      using comm_scalar_prod[of "real_of_int_vec as" ?n z] z_dim
      by (metis carrier_dim_vec index_map_vec(2) real_of_int_vec_def)
    have *: "row (of_int_hom.mat_hom (mat (?n+1) (?n) (λx.
      (case x of (i, j) ⇒ if i = 0 then as $ j
                          else if i = j + 1 then 2 else 0)))) i =
      (if i=0 then real_of_int_vec as else vec ?n (λj. if i = j + 1 then
2 else 0))"
    (is "?row = ?vec")
    if "i<?n+1" for i
    using that by (auto simp add: real_of_int_vec_def)
    moreover have "vec (dim_vec as) (λj. if i = Suc j then 2 else 0)
· z =
      2 * z $ (i - Suc 0)" if "i < Suc (dim_vec as)" "0<i" for i
    proof -
      have plus_2: "(i-1 = j) = (i = j+1)" for j using 1 that by auto
      have finite: "finite {0..<?n}" and elem: "i-1 ∈ {0..<?n}" using
that 1 by auto
      have vec: "vec (dim_vec as) (λj. if i = j+1 then 2 else 0) $ ia
=
        (if i = ia+1 then 2 else 0)" if "ia<?n" for ia
        using index_vec that by blast
      then have "(∑ ia = 0..<dim_vec z.
```

```
        vec (dim_vec as) (λj. if i = Suc j then 2 else 0) $ ia * z $ ia)
=
        (∑ia = 0..<dim_vec as. (if i = ia+1 then 2 else 0) * z $ ia)"
        using z_dim by auto
      also have "... = (∑ia = 0..<dim_vec as. (if i = ia+1 then 2 * z
$ ia else 0))"
        proof -
          have "(∀n. (0 = (if i = n + 1 then 2 else 0) * z $ n ∨ n +
1 = i) ∧
            (2 * z $ n = (if i = n + 1 then 2 else 0) * z $ n ∨ n + 1
≠ i)) ∨
            (∑n = 0..<dim_vec as. (if i = n + 1 then 2 else 0) * z $
n) =
            (∑n = 0..<dim_vec as. if i = n + 1 then 2 * z $ n else 0)"
by simp
          then show ?thesis by (metis (no_types))
        qed
      also have "... = 2*z$(i- Suc 0)"
        using plus_2 by (smt (verit, ccfv_SIG) One_nat_def sum_if_zero[OF
finite elem,
          of "(λj. 2*z$j)"] sum.cong)
      finally show ?thesis unfolding scalar_prod_def by blast
    qed
    ultimately have row_prod: "?row i · z =
      (if i = 0 then (real_of_int_vec as) · z else 2 * z $ (i - 1))"
    if "i<?n+1" for i
    using that by (subst *[OF that], auto)
    have "?row i · z = (if i = 0 then (z · real_of_int_vec as) else 2
* z $ (i - 1))"
    if "i<?n+1" for i using that row_prod that by (subst scal_prod_com)
auto
    then show ?thesis
      unfolding real_of_int_mat_def gen_basis_p_def mult_mat_vec_def by
auto
  qed
  have "... = 0_v (?n + 1)" using 1(1) dim_row by (subst eq[symmetric],
auto)
  then have "2 * z$(i-1) = 0" if "0<i" and "i<?n +1" for i
    using that by (smt (verit, best) cancel_comm_monoid_add_class.diff_cancel

      empty_iff index_vec index_zero_vec(1) insert_iff not_less_zero zero_less_diff)
  then have "z$i = 0" if "i<?n" for i using that by force
  then show ?case using 1 z_dim unfolding gen_basis_p_def by auto
qed
```

Well-definedness of the reduction function.

```
lemma well_defined_reduction:
  assumes "(as, s) ∈ subset_sum"
  shows "reduce_cvp_subset_sum_p (as, s) ∈ gap_cvp"
```

```
proof -
  obtain x where
    x_dim: "dim_vec x = dim_vec as" and
    x_binary: "∀ i<dim_vec x. x $ i ∈ {0, 1}" and
    x_lin_combo: "x · as = s"
    using assms unfolding subset_sum_def by blast
  define L where L_def: "L = fst (reduce_cvp_subset_sum_p (as, s))"
  define b where b_def: "b = fst (snd (reduce_cvp_subset_sum_p (as, s)))"
  define r where r_def: "r = snd (snd (reduce_cvp_subset_sum_p (as, s)))"

  define B where "B = gen_basis_p as"
  define n where n_def: "n = dim_vec as"
  have "r = root p n" unfolding n_def
    by (simp add: r_def reduce_cvp_subset_sum_p_def)
  then have r_alt: "r^p = n" using p_def by auto
  have init_eq_goal: "B *_v x - b =
    vec (n+1) (λ i. if i = 0 then (x · as - s) else (2 * x$(i-1) - 1))"
    (is "?init_vec = ?goal_vec")
  unfolding B_def b_def reduce_cvp_subset_sum_p_def
  by (auto simp add: Bx_s_rewrite[OF x_dim[symmetric]] n_def)
  have "p_norm_vec (B *_v x - b) =
    |x · as - s|^p + (∑ i=1..<n+1. |2*x$(i-1)-1|^p)"
  unfolding B_def b_def reduce_cvp_subset_sum_p_def
  by (auto simp add: p_norm_vec_Bx_s[OF x_dim[symmetric]] n_def)
  also have  "... ≤ r^p" (is "?left ≤ r^p")
  proof -
    have elem: "x$(i-1)∈{0,1}" if "0<i ∧ i<n+1" for i
      using that x_binary x_dim n_def
      by (smt (verit) add_diff_cancel_right' diff_diff_left diff_less_mono2

      less_add_same_cancel2 less_imp_add_positive less_one linorder_neqE_nat

      nat_1_add_1 not_add_less2)
    then have eq_1:  "|2*x$(i-1)-1|^p = 1" if "0<i ∧ i<n+1" for i
      using elem[OF that] by auto
    have eq_0: "|x · as - s| ^ p = 0" using x_lin_combo p_def by auto
    have "?left = real_of_int ((∑ i = 1..<n + 1. 1))" using eq_0 eq_1
by auto
    then have "?left ≤ n" by auto
    then show ?thesis using r_alt by linarith
  qed
  finally have "p_norm_vec (?init_vec) ≤ r^p" by blast
  moreover have "B *_v x∈L"
  proof -
    have "dim_vec x = dim_col (gen_basis_p as)" unfolding gen_basis_p_def
using x_dim by auto
    then show ?thesis
      unfolding L_def reduce_cvp_subset_sum_p_def gen_lattice_def B_def
by auto
```

```
  qed
  ultimately have witness: "∃v∈L. p_norm_vec (v - b) ≤ r^p" by auto
  have is_indep: "is_indep  B"
    unfolding B_def using is_indep_gen_basis_p[of as] by simp
  have L_int_lattice: "is_lattice L" unfolding L_def reduce_cvp_subset_sum_p_def

    using is_lattice_gen_lattice[OF is_indep] unfolding B_def by auto
  show ?thesis unfolding gap_cvp_def using L_int_lattice witness L_def
b_def r_def by force
qed
```

NP-hardness of reduction function.

```
lemma NP_hardness_reduction:
  assumes "reduce_cvp_subset_sum_p (as, s) ∈ gap_cvp"
  shows "(as, s) ∈ subset_sum"
proof -
  define n where "n = dim_vec as"
  define B where "B = gen_basis_p as"
  define L where "L = gen_lattice B"
  define b where "b = gen_t_p as s"
  define r where "r = root p n"
  have rp_eq_n: "r^p = n" unfolding r_def using p_def by (simp)
  have ex_v: "∃v∈L. p_norm_vec (v - b) ≤ r^p" and is_int_lattice: "is_lattice
L"
    using assms unfolding gap_cvp_def reduce_cvp_subset_sum_p_def L_def
B_def b_def r_def n_def
    by auto
  then obtain v where v_in_L:"v∈L" and ineq:"p_norm_vec (v - b) ≤ r^p"
by blast
  have "∃zs::int vec. v = B *_v zs ∧ dim_vec zs = dim_vec as"
  using v_in_L unfolding L_def gen_lattice_def B_def gen_basis_p_def by
auto
  then obtain zs::"int vec" where v_def: "v = B *_v  zs"
    and zs_dim: "dim_vec zs = dim_vec as" by blast
  have init_eq_goal: "v - b =
    vec (n+1) (λ i. if i = 0 then (zs · as - s) else (2 * zs$(i-1) - 1))"
    (is "?init_vec = ?goal_vec")
  unfolding v_def B_def b_def using Bx_s_rewrite[OF zs_dim[symmetric]]
n_def by simp
  have p_norm_vec_ineq: "p_norm_vec (v-b) = |zs · as - s|^p + (∑ i=1..<n+1.
|2*zs$(i-1)-1|^p)"
  unfolding v_def B_def b_def using p_norm_vec_Bx_s[OF zs_dim[symmetric]]
n_def by simp
  define hide_sum where "hide_sum = (∑ i=1..<n+1. |2*zs$(i-1)-1|^p)"
  have sum_le_rp: "|zs · as - s|^p + hide_sum ≤ r^p"
  unfolding hide_sum_def using ineq  by (subst p_norm_vec_ineq[symmetric])
  then have sum_le_rp_int: " (|zs · as - s| ^ p + hide_sum) ≤ int n"
    unfolding rp_eq_n by linarith
  moreover have zs_ge_n: "hide_sum ≥ n"
```

```
  proof -
    have "|2*zs$(i-1)-1|≥1" if "i∈{1..<n+1}" for i using that by presburger
    then have "(∑ i=1..<n+1. |2*zs$(i-1)-1|^p) ≥ (∑ i=1..<n+1. 1^p)"
      by (smt (verit, ccfv_SIG) linordered_semidom_class.power_mono sum_mono)
    then show ?thesis unfolding hide_sum_def by auto
  qed
  ultimately have zs_part: "hide_sum = n"
    unfolding rp_eq_n
    by (smt (verit, ccfv_threshold) abs_triangle_ineq2_sym abs_zero power_abs
zero_less_power)
  then have "(∑ i=1..<n+1. |2*zs$(i-1)-1|^p) = n" unfolding hide_sum_def
by simp
  then have as_part: "|zs · as - s|=0"
    using sum_le_rp_int unfolding hide_sum_def
    by (smt (verit, best) zero_less_power)
  have "∀ i<n. zs $ i ∈ {0, 1}"
  proof -
    have zs_ge_1:"|2*zs$(i-1)-1|≥1" if "i∈{1..<n+1}" for i using that
by presburger
    then have zsp_ge_1:"|2*zs$(i-1)-1|^p≥1" if "i∈{1..<n+1}" for i us-
ing that by auto
    have "|2*zs$(i-1)-1| = 1" if "i∈{1..<n+1}" for i
    proof (subst ccontr, goal_cases)
      case 1
      then have i_gt_1: "|2*zs$(i-1)-1| > 1" using that by presburger
      then have ip_gt_1: "|2*zs$(i-1)-1|^p > 1"
        using p_def by auto
      then have gt_n: "int (n - 1) + |2 * zs $ (i - 1) - 1| ^ p > n" by
auto
      have ineq1:"(∑ j=1..<n+1. |2*zs$(j-1)-1|^p) =
        (∑ j∈{1..<n+1}-{i}. |2*zs$(j-1)-1|^p) + (|2*zs$(i-1)-1|^p)"
        using that by (subst sum.subset_diff[of "{i}"], auto)
      also have ineq2: "... ≥ (∑ j∈{1..<n+1}-{i}. 1) + (|2*zs$(i-1)-1|^p)"

        by (smt (verit, ccfv_SIG) DiffD1 sum_mono zsp_ge_1)
      also have ineq3: "(∑ j∈{1..<n+1}-{i}. 1) = n - 1"
        by (metis Diff_empty add_diff_cancel_right' card_Diff_insert card_atLeastLessThan

          card_eq_sum empty_iff that)
      finally have ineq4: "(∑ i=1..<n+1. |2*zs$(i-1)-1|^p) ≥  n - 1 + (|2*zs$(i-1)-1|^p)"

        using ineq1 ineq2 ineq3 by (smt (z3) of_nat_1 of_nat_sum sum_mono)
      then have "(∑ i=1..<n+1. |2*zs$(i-1)-1|^p) >  n"
        by (subst order.strict_trans2[OF gt_n ineq4], simp)
      then show False using zs_part unfolding hide_sum_def  by auto
    qed auto
    then have "zs$(i-1) = 0 ∨ zs$(i-1) = 1" if "i∈{1..<n+1}" for i
      using that by force
    then have "zs$i = 0 ∨ zs$i = 1" if "i<n" for i using that
```

```
      by (metis One_nat_def Suc_eq_plus1 atLeastLessThan_iff diff_Suc_1
le_eq_less_or_eq
        less_Suc0 not_less_eq)
    then show ?thesis by simp
  qed
  moreover have "zs · as = s" using as_part by auto
  ultimately have "(∀i<dim_vec zs. zs $ i ∈ {0, 1}) ∧ zs · as = s ∧
dim_vec zs = dim_vec as"
     using zs_dim n_def by auto
  then show ?thesis unfolding subset_sum_def gap_cvp_def by auto
qed
```

CVP is NP-hard in p-norm.

```
lemma "is_reduction reduce_cvp_subset_sum_p subset_sum gap_cvp"
unfolding is_reduction_def
proof (safe, goal_cases)
  case (1 as s)
  then show ?case using well_defined_reduction by auto
next
  case (2 as s)
  then show ?case using NP_hardness_reduction by auto
qed

end

end
theory infnorm

imports
  Main
  "Jordan_Normal_Form.Matrix"
  "LLL_Basis_Reduction.Norms"
begin
```

8 Maximum Norm

The ℓ_∞ norm on vectors is exactly the maximum of the absolute value of all entries.

```
lemma linf_norm_vec_Max:
  "‖v‖_∞ = Max (insert 0 {|v$i| | i. i< dim_vec v})"
proof (induct v)
  case (vCons a v)
  have "Missing_Lemmas.max_list (map abs (list_of_vec (vCons a v)) @ [0])
=
    Missing_Lemmas.max_list ((|a|) # (map abs (list_of_vec v) @ [0]))"
by auto
  also have "... = max (|a|) (‖v‖_∞)" unfolding linf_norm_vec_def by (cases
v, auto)
```

```
  also have "... = max (|a|) (Max (insert 0 {|v$i| | i. i< dim_vec v}))"
using vCons by auto
  also have "... = Max (insert (|a|) (insert 0 {|v$i| | i. i< dim_vec v}))"
by auto
  also have "... = Max (insert 0 (insert (|a|) {|v$i| | i. i< dim_vec v}))"
    by (simp add: insert_commute)
  also have "insert (|a|){|v$i| | i. i< dim_vec v} =
    {|(vCons a v)$i| | i. i< dim_vec (vCons a v)}"
  proof (safe, goal_cases)
    case (1 x)
    show ?case by (subst exI[of _ "0"], auto)
  next
    case (2 x i)
    then show ?case by (subst exI[of _ "Suc i"])
      (use vec_index_vCons_Suc[of a v i, symmetric] in <auto>)
  next
    case (3 x i)
    then show ?case by (subst vec_index_vCons) (subst exI[of _ "i-1"],
auto)
  qed
  finally show ?case unfolding linf_norm_vec_def by auto
qed auto


end
theory CVP_vec

imports
  Main
  Reduction
  Lattice_int
  Subset_Sum
  infnorm
begin
```

9 CVP in ℓ_∞

The closest vector problem.

```
definition is_closest_vec :: "int_lattice ⇒ int vec ⇒ int vec ⇒ bool"
where
  "is_closest_vec L b v ≡ (is_lattice L) ∧
    (∀x∈L. linf_norm_vec  (x - b) ≥  linf_norm_vec (v - b) ∧ v∈L)"
```

The decision problem associated with solving CVP exactly.

```
definition gap_cvp :: "(int_lattice × int vec × int) set" where
  "gap_cvp ≡ {(L, b, r). (is_lattice L) ∧ (∃v∈L. linf_norm_vec (v -
b) ≤ r)}"
```

Reduction function for CVP to subset sum

```
definition gen_basis :: "int vec ⇒ int mat" where
  "gen_basis as = mat (dim_vec as + 2) (dim_vec as) (λ (i, j). if i ∈
{0,1} then as$j
    else (if i = j + 2 then 2 else 0))"


definition gen_t :: "int vec ⇒ int ⇒ int vec" where
  "gen_t as s = vec (dim_vec as + 2) ((λ i. 1)(0:= s+1, 1:= s-1))"


definition reduce_cvp_subset_sum ::
  "((int vec) * int) ⇒ (int_lattice * (int vec) * int)" where
  "reduce_cvp_subset_sum ≡ (λ (as,s).
    (gen_lattice (gen_basis as), gen_t as s, (1::int)))"
```

Lemmas for Proof

```
lemma vec_lambda_eq[intro]: "(∀ i<n. a i = b i) ⟶ vec n a = vec n b"
by auto

lemma eq_fun_applic: assumes "x = y" shows "f x = f y"
using assms by auto


lemma sum_if_zero:
  assumes "finite A" "i∈A"
  shows "(∑ j∈A. (if i = j then a j else 0)) = a i"
proof -
  have "(∑ x∈A. if x = i then a x else 0) =
  (if i = i then a i else 0) + (∑ x∈A - {i}. if x = i then a x else 0)"
  using sum.remove[OF assms, of "(λx. if x = i then a x else 0)"] by auto
  then show ?thesis by (simp add: assms(1))
qed

lemma set_compr_elem:
  assumes "finite A" "a∈A"
  shows "{f i | i. i∈A} = {f a} ∪ {f i | i. i∈A-{a}}"
by (safe, use assms in <auto>)

lemma Bx_rewrite:
  assumes x_dim: "dim_vec as = dim_vec x"
  shows "(gen_basis as) *v x =
    vec (dim_vec as + 2) (λ i. if i ∈ {0,1} then (x · as)
    else (2 * x$(i-2)))"
    (is "?init_vec = ?goal_vec")
proof -
  define n::nat where n_def: "n = dim_vec as"
  have "vec n (λj.  (as $ j)) ·  x = (x · as)"
    unfolding n_def scalar_prod_def using x_dim by (simp add: mult.commute)
```

```
  moreover have "vec (dim_vec as) (λj. if i = Suc (Suc j) then 2 else
0) · x = 2 * x $ (i - 2)"
    if "i < Suc (Suc (dim_vec as))" "0 < i" "i ≠ Suc 0" for i
  proof -
    have "(∑ ia = 0..<dim_vec x.
      vec (dim_vec as) (λj.  (if i = Suc (Suc j) then 2 else 0)) $ ia
* x $ ia) =
      (∑ ia<n.  (if i = ia+2 then 2 * (x $ ia) else 0))"
      by (intro sum.cong, auto simp add: n_def x_dim)
    also have "... = (∑ ib∈{2..<n+2}.
        (if i = ib then 2 * (x $ (ib-2)) else 0))"
    proof -
      have eq: "(λib.  (if i = ib then 2 * x $ (ib - 2) else 0)) ∘ (+)
2
          = (λia.  (if i = ia + 2 then 2 * x $ ia else 0))"
      by auto
      then show ?thesis
        by (subst sum.atLeastLessThan_shift_0[
            of "(λib.  (if i = ib then 2 * x $ (ib - 2) else 0))" 2 "n+2"])
          (subst eq, use lessThan_atLeast0 in <auto>)
    qed
    also have "... = 2 *  (x $ (i-2))"
    proof -
      have finite: "finite {2..<n+2}" by auto
      have is_in: "i ∈ {2..<n+2}" using that by (auto simp add: n_def)
      show ?thesis
      by (subst sum_if_zero[OF finite is_in, of "(λk.2 * (x $ (k-2)))"],
auto)
    qed
    finally show ?thesis unfolding scalar_prod_def by auto
  qed
  ultimately show ?thesis
    unfolding gen_basis_def reduce_cvp_subset_sum_def gen_t_def
    by (intro eq_vecI, auto simp add: n_def)
qed


lemma Bx_s_rewrite:
  assumes x_dim: "dim_vec as = dim_vec x"
  shows "(gen_basis as) *v x - (gen_t as s) =
    vec (dim_vec as + 2) (λ i. if i = 0 then  (x · as - s - 1) else (
      if i = 1 then  (x · as - s + 1) else  (2 * x$(i-2) - 1)))"
    (is "?init_vec = ?goal_vec")
unfolding gen_t_def by (subst  Bx_rewrite[OF assms], auto)


lemma linf_norm_vec_Bx_s:
  assumes x_dim: "dim_vec as = dim_vec x"
  shows "linf_norm_vec ((gen_basis as) *v x - (gen_t as s)) =
```

```
    Max (insert 0 ({ |x · as - s - 1|} ∪ { |x · as - s + 1|} ∪
      { |2*x$(i-2)-1| | i. 1<i ∧ i<dim_vec as+2 }))"
proof -
  let ?init_vec = "(gen_basis as) *_v x - (gen_t as s)"
  let ?goal_vec = "vec (dim_vec as + 2) (λ i. if i = 0 then  (x · as -
s - 1) else (
      if i = 1 then  (x · as - s + 1) else  (2 * x$(i-2) - 1)))"
  define n where n_def: "n = dim_vec as"
  have "linf_norm_vec ?init_vec = linf_norm_vec ?goal_vec" using Bx_s_rewrite[OF
x_dim] by auto
  also have "... = Max (insert 0 {|?goal_vec $i| | i. i<n+2})"
    unfolding linf_norm_vec_Max n_def by auto
  also have "... = Max (insert 0 ({ |x · as - s - 1|} ∪
                       { |x · as - s + 1|} ∪
                       { |2*x$(i-2)-1| | i. 1<i ∧ i<n+2}))"
  proof -
    have "{|?goal_vec $i| | i. i<n+2} =
      {|?goal_vec $0|} ∪ {|?goal_vec $1|} ∪ {|?goal_vec $i| | i. 1<i ∧
i<n+2}"
    proof -
      have "{|?goal_vec $i| | i. i∈{0..<n+2}} =
       {|?goal_vec $0|} ∪ {|?goal_vec $i| | i. i∈{1..<n+2}}"
      by (subst set_compr_elem[of "{0..<n+2}" 0 "(λi. |?goal_vec $i|)"],
auto)
      also have "... = {|?goal_vec $0|} ∪ {|?goal_vec $1|} ∪
        {|?goal_vec $i| | i. i∈{2..<n+2}}"
      by (subst set_compr_elem[of _ 1], auto)
      finally show ?thesis by auto
    qed
    also have "... = { |x · as - s - 1|} ∪ { |x · as - s + 1|} ∪
      {|?goal_vec $i| | i. 1<i ∧ i<n+2}" by auto
    also have "{|?goal_vec $i| | i. 1<i ∧ i<n+2} =
      { |2*x$(i-2)-1| | i. 1<i ∧ i<n+2}"
    proof -
      have "|?goal_vec $i| =  |2*x$(i-2)-1|" if "1<i ∧ i<n+2" for i
      using that n_def by force
      then show ?thesis using n_def by force
    qed
    finally have eq: "{|?goal_vec $i| | i. i<n+2} =
      { |x · as - s - 1|} ∪ { |x · as - s + 1|} ∪
      { |2*x$(i-2)-1| | i. 1<i ∧ i<n+2}" by blast
    then show ?thesis by auto
  qed
  finally show ?thesis using n_def by blast
qed
```

gen_basis actually generates a basis which is spans the *int_lattice* (by definition) and is linearly independent.

lemma *is_indep_gen_basis:*

```
  "is_indep (gen_basis as)"
unfolding is_indep_int_def
proof (safe, goal_cases)
case (1 z)
  let ?n = "dim_vec as"
  have z_dim: "dim_vec z = ?n" using 1(2) unfolding gen_basis_def by
auto
  have dim_row: "dim_row (gen_basis as) = ?n + 2" unfolding gen_basis_def
by auto
  have eq: "(real_of_int_mat (gen_basis as)) *v z = vec (?n + 2) (λ i.
if i ∈ {0,1}
    then (z · (real_of_int_vec as)) else (2 * z$(i-2)))"
  (is "(real_of_int_mat (gen_basis as)) *v z = ?goal_vec")
  proof -
    have scal_prod_com: "z · real_of_int_vec as = real_of_int_vec as ·
z"
      using comm_scalar_prod[of "real_of_int_vec as" ?n z] z_dim
      by (metis carrier_dim_vec index_map_vec(2) real_of_int_vec_def)
    have *: "row (of_int_hom.mat_hom (mat (?n+2) (?n) (λx.
      (case x of (i, j) ⇒ if i = 0 ∨ i = Suc 0 then as $ j
                          else if i = j + 2 then 2 else 0)))) i =
      (if i∈{0,1} then real_of_int_vec as else vec ?n (λj. if i = j +
2 then 2 else 0))"
    (is "?row = ?vec")
    if "i<?n+2" for i
    using that by (auto simp add: real_of_int_vec_def)
    moreover have "vec (dim_vec as) (λj. if i = Suc (Suc j) then 2 else
0) · z = 2 * z $ (i - 2)"
      if "i < Suc (Suc (dim_vec as))" "0 < i" "i ≠ Suc 0" for i
    proof (goal_cases)
    case 1
      have plus_2: "(i-2 = j) = (i = j+2)" for j using 1 that by auto
      have finite: "finite {0..<?n}" and elem: "i-2 ∈ {0..<?n}" using
that 1 by auto
      have vec: "vec (dim_vec as) (λj. if i = j+2 then 2 else 0) $ ia
=
        (if i = ia+2 then 2 else 0)" if "ia<?n" for ia
        using index_vec that by blast
      then have "(∑ ia = 0..<dim_vec z.
        vec (dim_vec as) (λj. if i = Suc (Suc j) then 2 else 0) $ ia *
z $ ia) =
        (∑ ia = 0..<dim_vec as. (if i = ia+2 then 2 else 0) * z $ ia)"
        using z_dim by auto
      also have "... = (∑ ia = 0..<dim_vec as. (if i = ia+2 then 2 * z
$ ia else 0))"
        proof -
          have "(∀ n. (0 = (if i = n + 2 then 2 else 0) * z $ n ∨ n +
2 = i) ∧
            (2 * z $ n = (if i = n + 2 then 2 else 0) * z $ n ∨ n + 2
```

```
≠ i)) ∨
            (∑n = 0..<dim_vec as. (if i = n + 2 then 2 else 0) * z $
n) =
            (∑n = 0..<dim_vec as. if i = n + 2 then 2 * z $ n else 0)"
by simp
          then show ?thesis by (metis (no_types))
        qed
      also have "... = 2*z$(i-2)" using sum_if_zero[OF finite elem, of
"(λj. 2*z$j)"]
        using plus_2 by auto
      finally show ?case unfolding scalar_prod_def by blast
    qed
    ultimately have "?row i · z =
      (if i ∈ {0,1} then (real_of_int_vec as) · z else 2 * z $ (i - 2))"
    if "i<?n+2" for i
    using that by (subst *[OF that], auto)
    then have "?row i · z =
      (if i ∈ {0,1} then z · real_of_int_vec as else 2 * z $ (i - 2))"
    if "i<?n+2" for i using that by (subst scal_prod_com)
    then show ?thesis
      unfolding real_of_int_mat_def gen_basis_def mult_mat_vec_def by
auto
  qed
  have "... = 0v (?n + 2)" using 1(1) dim_row by (subst eq[symmetric],
auto)
  then have "2 * z$(i-2) = 0" if "1<i" and "i<?n +2" for i
    using that by (smt (verit, best) cancel_comm_monoid_add_class.diff_cancel

      empty_iff index_vec index_zero_vec(1) insert_iff not_less_zero zero_less_diff)
  then have "z$i = 0" if "i<?n" for i using that by force
  then show ?case using 1 z_dim unfolding gen_basis_def by auto
qed
```

The CVP is NP-hard in ℓ_∞.

```
lemma well_defined_reduction:
  assumes "(as, s) ∈ subset_sum"
  shows "reduce_cvp_subset_sum (as, s) ∈ gap_cvp"
proof -
  obtain x where
    x_dim: "dim_vec x = dim_vec as" and
    x_binary: "∀i<dim_vec x. x $ i ∈ {0, 1}" and
    x_lin_combo: "x · as = s"
    using assms unfolding subset_sum_def by blast
  define L where L_def: "L = fst (reduce_cvp_subset_sum (as, s))"
  define b where b_def: "b = fst (snd (reduce_cvp_subset_sum (as, s)))"
  define r where r_def: "r = snd (snd (reduce_cvp_subset_sum (as, s)))"
  have "r = 1" by (simp add: r_def reduce_cvp_subset_sum_def Pair_inject
prod.exhaust_sel)
```

```
  define B where "B = gen_basis as"
  define n where n_def: "n = dim_vec as"
  have init_eq_goal: "B *v x - b =
    vec (n+2) (λ i. if i = 0 then  (x · as - s - 1) else (
      if i = 1 then  (x · as - s + 1) else  (2 * x$(i-2) - 1)))"
    (is "?init_vec = ?goal_vec")
  unfolding B_def b_def reduce_cvp_subset_sum_def
  by (auto simp add: Bx_s_rewrite[OF x_dim[symmetric]] n_def)
  have "linf_norm_vec (B *v x - b) =
    Max (insert 0 ({ |x · as - s - 1|} ∪ { |x · as - s + 1|} ∪
      { |2*x$(i-2)-1| | i. 1<i ∧ i<n+2 }))"
  unfolding B_def b_def reduce_cvp_subset_sum_def
  by (auto simp add: linf_norm_vec_Bx_s[OF x_dim[symmetric]] n_def)
  also have  "... ≤ r"
  proof -
    have elem: "x$(i-2)∈{0,1}" if "1<i ∧ i<n+2" for i
      using that x_binary x_dim n_def
      by (smt (verit) add_diff_cancel_right' diff_diff_left diff_less_mono2

      less_add_same_cancel2 less_imp_add_positive less_one linorder_neqE_nat

      nat_1_add_1 not_add_less2)
    then have "|2*x$(i-2)-1| = 1" if "1<i ∧ i<n+2" for i
      using elem[OF that] by auto
    then have "{ |2 * x $ (i - 2) - 1| |i. 1 < i ∧ i < n + 2} ⊆ {1}"

      by (safe, auto)
    then show ?thesis using x_lin_combo ‹r=1› by auto
  qed
  finally have "linf_norm_vec (?init_vec) ≤ r" by blast
  moreover have "B *v x∈L"
  proof -
    have "dim_vec x = dim_col (gen_basis as)" unfolding gen_basis_def
using x_dim by auto
    then show ?thesis
      unfolding L_def reduce_cvp_subset_sum_def gen_lattice_def B_def
by auto
  qed
  ultimately have witness: "∃v∈L. linf_norm_vec (v - b) ≤ r" by auto
  have is_indep: "is_indep  B"
    unfolding B_def using is_indep_gen_basis[of as] by simp
  have L_int_lattice: "is_lattice L" unfolding L_def reduce_cvp_subset_sum_def

    using is_lattice_gen_lattice[OF is_indep] unfolding B_def by auto
  show ?thesis unfolding gap_cvp_def using L_int_lattice witness L_def
b_def r_def by force
qed
```

NP-hardness part of reduction.

```
lemma NP_hardness_reduction:
  assumes "reduce_cvp_subset_sum (as, s) ∈ gap_cvp"
  shows "(as, s) ∈ subset_sum"
proof -
  define n where "n = dim_vec as"
  define B where "B = gen_basis as"
  define L where "L = gen_lattice B"
  define b where "b = gen_t as s"
  have ex_v: "∃v∈L. linf_norm_vec (v - b) ≤ 1" and is_int_lattice:
"is_lattice L"
    using assms unfolding gap_cvp_def reduce_cvp_subset_sum_def L_def
B_def b_def by auto
  then obtain v where v_in_L:"v∈L" and ineq:"linf_norm_vec (v - b) ≤
1" by blast
  have "∃zs::int vec. v = B *v zs ∧ dim_vec zs = dim_vec as"
  using v_in_L unfolding L_def gen_lattice_def B_def gen_basis_def by
auto
  then obtain zs::"int vec" where v_def: "v = B *v  zs"
    and zs_dim: "dim_vec zs = dim_vec as" by blast
  have init_eq_goal: "v - b =
    vec (n+2) (λ i. if i = 0 then  (zs · as - s - 1) else (
      if i = 1 then  (zs · as - s + 1) else  (2 * zs$(i-2) - 1)))"
    (is "?init_vec = ?goal_vec")
  unfolding v_def B_def b_def using Bx_s_rewrite[OF zs_dim[symmetric]]
n_def by simp
  have linf_norm_vec_ineq: "linf_norm_vec (v-b) = Max (insert 0 ({ |zs
· as - s - 1|} ∪
    { |zs · as - s + 1|} ∪ { |2*zs$(i-2)-1| | i. 1<i ∧ i<n+2 }))"
  unfolding v_def B_def b_def using linf_norm_vec_Bx_s[OF zs_dim[symmetric]]
n_def by simp
  have Max_le_1: "Max (insert 0 ({ |zs · as - s - 1|} ∪
    { |zs · as - s + 1|} ∪  { |2*zs$(i-2)-1| | i. 1<i ∧ i<n+2 }))≤1"
  using ineq by (subst linf_norm_vec_ineq[symmetric])
  have "|2*zs$(i-2)-1|≤1" if "1<i ∧ i<n+2" for i using Max_le_1 that by
auto
  then have "zs$(i-2) = 0 ∨ zs$(i-2) = 1" if "1<i ∧ i<n+2" for i
    using that by force
  then have "zs$i = 0 ∨ zs$i = 1" if "i<n" for i using that
    by (metis One_nat_def Suc_less_eq add_2_eq_Suc' add_diff_cancel_right'
zero_less_Suc)
  then have "∀i<n. zs $ i ∈ {0, 1}" by simp
  moreover have "zs · as = s" using Max_le_1 by auto
  ultimately have "(∀i<dim_vec zs. zs $ i ∈ {0, 1}) ∧ zs · as = s ∧
dim_vec zs = dim_vec as"
     using zs_dim n_def by auto
  then show ?thesis unfolding subset_sum_def gap_cvp_def by auto
qed
```

The CVP is NP-hard in ℓ_∞.

```
lemma "is_reduction reduce_cvp_subset_sum subset_sum gap_cvp"
unfolding is_reduction_def
proof (safe, goal_cases)
  case (1 as s)
  then show ?case using well_defined_reduction by auto
next
  case (2 as s)
  then show ?case using NP_hardness_reduction by auto
qed
```

```
end
theory Digits_int
  imports
    Complex_Main
begin
```

10 Representation of Integers in Different Number Systems

This file is an adaption of *Van_der_Waerden/Digits.thy* for representation of the Integers (instead of the natural numbers) in different number systems. The main difference is the integer input in the function *digit*.

First, we look at some useful lemmas for splitting sums.

```
lemma split_sum_first_elt_less: assumes "n<m"
  shows "(∑ i∈{n..<m}. f i) = f n + (∑ i∈{Suc n ..<m}. f i)"
  using sum.atLeast_Suc_lessThan assms by blast

lemma split_sum_mid_less: assumes "i<(n::nat)"
  shows "(∑ j<n. f j) = (∑ j<i. f j) + (∑ j=i..<n. f j)"
proof -
  have "(∑ j<n. f j) = (∑ j∈{..<i} ∪ {i..<n}. f j)"
    using ‹i < n› by (intro sum.cong) auto
  also have "... = (∑ j<i. f j) + (∑ j=i..<n. f j)"
    by (subst sum.union_disjoint) auto
  finally show "(∑ j<n. f j) = (∑ j<i. f j) + (∑ j=i..<n. f j)" .
qed
```

In order to use representation of numbers in a basis *base* and to calculate the conversion to and from integers, we introduce the following locale.

```
locale digits =
  fixes base :: int
  assumes base_pos: "base > 0"
```

begin

Conversion from basis base to integers: `from_digits n d`

n:	`nat`	length of representation in basis base
d:	`nat ⇒ nat`	function of digits in basis base where `d i` is the i-th digit in basis base
output:	`nat`	natural number corresponding to $d(n-1)\ldots d(0)$ as integer

```
fun from_digits :: "nat ⇒ (nat ⇒ nat) ⇒ nat" where
  "from_digits 0 d = 0"
| "from_digits (Suc n) d = d 0 + nat base * from_digits n (d ∘ Suc)"
```

Alternative definition using sum:

```
lemma from_digits_altdef: "from_digits n d = (∑ i<n. d i * nat base
^ i)"
  by (induction n d rule: from_digits.induct)
     (auto simp add: sum.lessThan_Suc_shift o_def sum_distrib_left
       sum_distrib_right mult_ac simp del: sum.lessThan_Suc)
```

```
lemma int_from_digits:
  "int (from_digits n d) = (∑ i<n. int (d i) * base ^ i)"
unfolding from_digits_altdef using base_pos by auto
```

Digit in basis base of some integer number: `digit x i`

x:	`int`	integer
i:	`nat`	index
output:	`int`	i-th digit of representation in basis base of x

```
fun digit :: "int ⇒ nat ⇒ int" where
  "digit x 0 = |x| mod base"
| "digit x (Suc i) = digit (|x| div base) i"
```

Alternative definition using divisor and modulo:

```
lemma digit_altdef: "digit x i = ( |x| div (base ^ i)) mod base"
proof (induction x i rule: digit.induct)
  case (2 x i)
  show ?case by (subst digit.simps(2), subst 2) (smt (verit, ccfv_SIG)
base_pos
       pos_imp_zdiv_neg_iff power.simps(2) zdiv_zmult2_eq zero_less_power)
qed simp
```

Every digit must be smaller that the base.

```
lemma digit_less_base: "digit x i < base"
  using base_pos by (auto simp: digit_altdef)
```

A representation in basis `base` of length n must be less than `base`^{n}.

```
lemma from_digits_less:
```

```
  assumes "∀i<n. d i < base"
  shows "from_digits n d < base ^ n"
using assms proof (induct n d rule: from_digits.induct)
  case (2 n d)
  have "from_digits n (d ∘ Suc) ≤ base ^ n -1" using 2
    by (simp add: "2.hyps")
  moreover have "d 0 ≤ base -1" using 2 by simp
  ultimately have "d 0 + base * from_digits n (d ∘ Suc) ≤
      base - 1 + base * (base^(n) - 1)"
    by (smt (verit, ccfv_SIG) base_pos mult_less_cancel_left_pos)
  then show "from_digits (Suc n) d < base ^ Suc n"
    using base_pos by (simp add: right_diff_distrib)
qed auto
```

Lemmas for *mod* and *div* in number systems of basis *base*:

```
lemma mod_base:  assumes "⋀i. i<n ⟹ d i < base" "n>0"
  shows "from_digits n d mod base = d 0 "
proof -
  have "(∑ i<n. d i * base ^ i) mod base =
          (∑ i<n. d i * base ^ i mod base) mod base"
  by (subst mod_sum_eq[symmetric]) simp
  then show ?thesis using assms
    sum.lessThan_Suc_shift[of "(λi. d i * base ^ i mod base)" "n-1"]
    int_from_digits by simp
qed

lemma mod_base_i:
  assumes "⋀i. i<n ⟹ (d i ::nat) < base" "n>0" "i<n"
  shows "(∑ j=i..<n. d j * base ^ (j-i)) mod base = d i "
proof -
  have eq: "(∑ j=i..<n. d j * base ^ (j-i)) mod base =
        (∑ j=i..<n. d j * base ^ (j-i) mod base) mod base"
    by (subst mod_sum_eq[symmetric]) simp
  then show ?thesis  using assms
    split_sum_first_elt_less[where  f = "(λj. d j * base ^ (j-i) mod
base)"]
    int_from_digits by auto
qed


lemma div_base_i:
  assumes "⋀i. i<n ⟹ (d i::nat) < base" "n>0" "i<n"
  shows "from_digits n d div (base ^i) = (∑ j=i..<n. d j * base ^ (j-i))"
  unfolding int_from_digits proof -
  have base_exp: "base^(j) =  base^(j-i) * base^i"
    if "j∈{i..<n}" for j
    by (metis Nat.add_diff_assoc2 add_diff_cancel_right' atLeastLessThan_iff

        power_add that)
```

```
  have first:"(∑j<i. d j * base ^ j)< base ^ i"
    using assms from_digits_less[where n="i"]
    unfolding int_from_digits by auto
  have ge_0: "0 ≤ (∑j<i. int (d j) * base ^ j)" using base_pos
  by (metis int_from_digits of_nat_0_le_iff)
  have eq: "(∑j<n. d j * base ^ j) =
          (∑j<i. d j * base ^ j) + (∑j=i..<n. d j * base ^ j)"
    using assms split_sum_mid_less[where f="(λj. d j * base^j)"] by auto
  have split_sum: "(∑j<n. d j * base ^ j) =
      (∑j<i. d j * base ^ j) + base^i * (∑j=i..<n. d j * base ^ (j-i))"
    unfolding eq using base_exp mult.assoc sum_distrib_right
    by (smt (z3) mult.commute sum.cong)
  show "(∑i<n. d i * base ^ i) div base ^ i =
             (∑j = i..<n. d j * base ^ (j - i))"
    unfolding split_sum using base_pos first div_pos_pos_trivial[OF ge_0
first]
    by (subst div_mult_self2, auto)
qed
```

Conversions are inverse to each other.

```
lemma digit_from_digits:
  assumes "⋀j. j<n ⟹ d j < base" "n>0" "i<n"
  shows   "digit (from_digits n d) i = d i"
  using assms proof (cases "i=0")
  case True
  then show ?thesis
  using assms(1) assms(3) mod_base by force
next
  case False
  have "from_digits n d div base^i mod base = d i"
    using assms by (auto simp add: div_base_i mod_base_i)
  then show "digit (from_digits n d) i = d i"
    unfolding digit_altdef by auto
qed

lemma div_distrib: assumes "i<n"
  shows "(a*base^n + b) div base^i mod base = b div base^i mod base"
proof -
  have "base^i dvd (a*base^n)" using assms
    by (simp add: le_imp_power_dvd)
  moreover have "a*base^n div base^i mod base = 0"
    by (metis Suc_leI assms dvd_imp_mod_0 dvd_mult
        dvd_mult_imp_div le_imp_power_dvd power_Suc)
  ultimately show ?thesis
    by (metis add.right_neutral div_mult_mod_eq
        div_plus_div_distrib_dvd_left mod_mult_self3)
qed
```

```
lemma(in digits) digits_eq_0:
  assumes "x = 0"
  shows "digit x i = 0"
by (simp add: assms digit_altdef)
end


lemma split_digits_eq_zero:
  assumes "a + base * b = 0" "|a|<base" "(base::int)>2"
  shows "a = 0 ∧ b=0"
using assms proof (cases "b = 0")
  case True
  then show "a=0 ∧ b=0" using assms by auto
next
  case False
  then have "|b| ≥ 1" by auto
  then have "|a| < |base * b|" using assms(2) assms(3)
    by (subst abs_mult) (smt (verit) mult_le_cancel_left1)
  moreover have "|a| = |base * b|" using assms(1) by auto
  ultimately have False by auto
  then show ?thesis by auto
qed


lemma respresentation_in_basis_eq_zero:
  assumes "(∑i<n. c i * base^i) = 0" "(base::int) > 2" "⋀i. i<n ⟹
|c i| < base" "i<n"
  shows "c i = 0"
using assms proof (induction n arbitrary: i c)
  case 0
  then show ?case by auto
next
  case (Suc n)
  have eq_0: "c 0 + base * (∑i<n. c (i+1) * base ^ i) = 0"
    using Suc(2) unfolding sum.lessThan_Suc_shift power_Suc sum.cong[of
"{..<n}" "{..<n}"
      "(λi. c (Suc i) * (base * base ^ i))" "(λi. base * (c (n + 1) *
base ^ n))"]
    by (subst sum_distrib_left)(metis (no_types, lifting) Suc_eq_plus1
mult.left_commute
      mult_cancel_left1 power_0 sum.cong)
  have "c 0 = 0 " and right: "(∑i<n. c (i+1) * base ^ i) = 0"
    using split_digits_eq_zero[OF eq_0 Suc(4)[of 0] Suc(3)] by auto
  have lt_n: "c (i + 1) = 0" if "i<n" for i using Suc(4)
    by (subst Suc(1)[OF right <2<base> _ <i<n>], auto)
  then show ?case
  proof (cases "i=0")
    case False
    then show ?thesis using Suc(5) lt_n less_Suc_eq_0_disj by auto
```

```
  qed (use <c 0 = 0> in <auto>)
qed

end
theory Additional_Lemmas

imports
  Main
  infnorm
  Partition
  Lattice_int
  Digits_int
begin
```

11 Additional Lemmas

Lemma about length and concat.

```
lemma length_concat_map_5:
  "length (concat (map (λi. [f1 i, f2 i, f3 i, f4 i, f5 i]) xs)) = length
xs * 5"
by (induct xs, auto)
```

Lemma about splitting the index of sums.

```
lemma sum_split_idx_prod:
  "(∑ i=0..<k*l::nat. f i) = (∑ i=0..<k. (∑ j=0..<l. f (i*l+j)))"
proof -
  have set_rew: "{0..<k*l} = (λ(i,j). i*l+j) ‘ ({0..<k} × {0..<l})"
  proof (safe, goal_cases)
    case (1 x)
    have "x = (λ(i,j). i*l+j) (x div l, x mod l)" by auto
    moreover have "(x div l, x mod l)∈{0..<k} × {0..<l}" using 1 less_mult_imp_div_less
      by (metis le_less_trans lessThan_atLeast0 lessThan_iff mem_Sigma_iff
        mod_less_divisor mult_zero_right neq0_conv zero_le)
    ultimately show ?case by blast
  next
    case (2 x i j)
    then show ?case
    by (auto, metis less_nat_zero_code linorder_neqE_nat mod_lemma mult.commute
nat_mod_lem)
  qed
  have inj: "inj_on (λ(i, y). i * l + y) ({0..<k} × {0..<l})"
    unfolding inj_on_def by (auto)
      (metis add.commute add_right_imp_eq linorder_neqE_nat mod_mult_self2
mult.commute
        mult_cancel_right nat_mod_lem not_less_zero,
       metis add.commute le0 le_less_trans mod_mult_self2 mult.commute
nat_mod_lem)
```

```
  have "(∑ i∈{0..<k*l}. f i) = (∑ (i,j)∈{0..<k}×{0..<l}. f (i*l+j))"

    unfolding set_rew using inj
    by (subst sum.reindex[of "(λ(i, j). i * l + j)" "({0..<k} × {0..<l})"
f])
       (auto simp add: prod.case_distrib)
  also have "... = (∑ i∈{0..<k}. (∑ j∈{0..<l}. f (i*l+j)))"
    using sum.cartesian_product[of "(λi j. f (i*l+j))" "{0..<l}" "{0..<k}",
symmetric]
    by auto
  finally show ?thesis by auto
qed
```

Helping lemma to split sums.

```
lemma lt_M:
  assumes "M > (∑ i<(n::nat). |b i|::int)"
          "∀ i<n. |x i| ≤ 1"
  shows "|(∑ i<n. x i * b i)| < M"
proof -
  have "|(∑ i<(n::nat). x i * b i)::int| ≤ (∑ i<n. |x i * b i|)" using
sum_abs by auto
  moreover have "... = (∑ i<n. |x i| * |b i|)" using abs_mult by metis
  moreover have "... ≤ (∑ i<n. |b i|)" using assms
    by (smt (verit, best) lessThan_iff mult_cancel_right2 sum_mono zero_less_mult_iff)
  moreover have "... = (∑ i<n. |b i|)" using sum_distrib_left by metis
  ultimately have "|(∑ i<n. x i * b i)| ≤ (∑ i<n. |b i|)" by linarith
  then show ?thesis using assms by auto
qed
```

```
lemma split_sum:
  "(∑ i<(n::nat). x i * (a i + M * b i)::int) = (∑ i<n. x i * a i) +
M * (∑ i<n. x i * b i)"
proof -
  have "(∑ i<(n::nat). x i * (a i + M * b i)) = (∑ i<n. x i * a i) +
(∑ i<n. M * x i * b i)"
    by (simp add: distrib_left mult.commute mult.left_commute)
  also have "... = (∑ i<n. x i * a i) + M * (∑ i<n. x i * b i)"
    using sum_distrib_left[symmetric, where r=M and f="(λi. x i*b i)"
and A = "{0..<n}"]
    by (metis (no_types, lifting) lessThan_atLeast0 mult.assoc sum.cong)
  finally show ?thesis by blast
qed
```

```
lemma split_eq_system:
  assumes "M > (∑ i<n::nat. |a i|::int)"
```

```
          "∀i<n. |x i| ≤ 1"
          "(∑i<n. x i * (a i + M * b i)) = 0"
  shows   "(∑i<n. x i * a i) = 0 ∧ (∑i<n. x i * b i) = 0"
using assms proof (safe, goal_cases)
  case 1
  then show ?case
  proof (cases "(∑i<n. x i * b i) = 0")
    case True
    then show ?thesis using assms(3) split_sum[of x a M b n] by auto
  next
    case False
    then have "|(∑i<n. x i * a i)| < M * |(∑i<n. x i * b i)|"
      using lt_M[OF assms(1) assms(2)] False
      by (smt (verit, best) mult_less_cancel_left2)
    moreover have "|(∑i<n. x i * a i)| = M * |(∑i<n. x i * b i)|"
      using assms(3) split_sum[of x a M b n] calculation by linarith
    ultimately have False by linarith
    then show ?thesis by auto
  qed
next
  case 2
  then show ?case
  proof (cases "(∑i<n. x i * b i) = 0")
    case True
    then show ?thesis using split_sum 2 using lt_M[OF assms(1) assms(2)]
       by auto
  next
    case False
    then have "|(∑i<n. x i * a i)| < M * |(∑i<n. x i * b i)|"
      using lt_M[OF assms(1) assms(2)] False
      by (smt (verit, best) mult_less_cancel_left2)
    moreover have "|(∑i<n. x i * a i)| = M * |(∑i<n. x i * b i)|"
      using split_sum[of x a M b n] assms calculation by linarith
    ultimately have False by linarith
    then show ?thesis by auto
  qed
qed
```

Lemmas about splitting into 4 or 5 cases.

Split into 4 modulo classes

```
lemma lt_4_split: "(i::nat) < 4 ⟶ i = 0 ∨ i = 1 ∨ i = 2 ∨ i = 3"
by auto

lemma mod_exhaust_less_4_int: "(i::int) mod 4 = 0 ∨ i mod 4 = 1 ∨ i
mod 4 = 2 ∨ i mod 4 = 3"
using MacLaurin.mod_exhaust_less_4 by auto

lemma mod_4_choices:
  assumes "i mod 4 = 0 ⟶ P i"
```

```
          "i mod 4 = 1 ⟶ P i"
          "i mod 4 = 2 ⟶ P i"
          "i mod 4 = 3 ⟶ P i"
  shows "P (i::nat)"
using assms mod_exhaust_less_4 by auto

lemma mod_4_if_split:
  assumes "i mod 4 = 0 ⟶ P = P0 i"
          "i mod 4 = 1 ⟶ P = P1 i"
          "i mod 4 = 2 ⟶ P = P2 i"
          "i mod 4 = 3 ⟶ P = P3 i"
  shows "P = (if i mod 4 = 0 then P0 i else
                (if i mod 4 = 1 then P1 i else
                (if i mod 4 = 2 then P2 i else P3 (i::nat))))" (is "?P i")
using mod_exhaust_less_4  by (auto simp add: assms)
```

Split into 5 modulo classes

```
lemma lt_5_split: "(i::nat) < 5 ⟶ i = 0 ∨ i = 1 ∨ i = 2 ∨ i = 3 ∨
i = 4"
by auto

lemma mod_exhaust_less_5_int:
  "(i::int) mod 5 = 0 ∨ i mod 5 = 1 ∨ i mod 5 = 2 ∨ i mod 5 = 3 ∨ i
mod 5 = 4"
using lt_5_split by linarith

lemma mod_exhaust_less_5:
  "(i::nat) mod 5 = 0 ∨ i mod 5 = 1 ∨ i mod 5 = 2 ∨ i mod 5 = 3 ∨ i
mod 5 = 4"
using lt_5_split by linarith

lemma mod_5_choices:
  assumes "i mod 5 = 0 ⟶ P i"
          "i mod 5 = 1 ⟶ P i"
          "i mod 5 = 2 ⟶ P i"
          "i mod 5 = 3 ⟶ P i"
          "i mod 5 = 4 ⟶ P i"
  shows "P (i::nat)"
using assms mod_exhaust_less_5 by auto

lemma mod_5_if_split:
  assumes "i mod 5 = 0 ⟶ P = P0 i"
          "i mod 5 = 1 ⟶ P = P1 i"
          "i mod 5 = 2 ⟶ P = P2 i"
          "i mod 5 = 3 ⟶ P = P3 i"
          "i mod 5 = 4 ⟶ P = P4 i"
  shows "P = (if i mod 5 = 0 then P0 i else
                (if i mod 5 = 1 then P1 i else
                (if i mod 5 = 2 then P2 i else
```

```
              (if i mod 5 = 3 then P3 i else
                                  P4 (i::nat)))))" (is "?P i")
using mod_exhaust_less_5  by (auto simp add: assms)
```

Representation of natural number in interval using lower bound.

```
lemma split_lower_plus_diff:
  assumes "s ∈ {n..<m::nat}"
  obtains j where "s = n+j" and "j<m-n"
using assms
by (metis atLeastLessThan_iff diff_diff_left le_Suc_ex zero_less_diff)


end
theory BHLE

imports
  Main
  Additional_Lemmas
begin
```

12 Bounded Homogeneous Linear Equation Problem

```
definition bhle :: "(int vec * int) set" where
  "bhle ≡ {(a,k). ∃x. a · x = 0 ∧ dim_vec x = dim_vec a ∧ dim_vec a
> 0 ∧
      x ≠ 0_v (dim_vec x) ∧ ‖x‖_∞ ≤ k}"
```

Reduction of bounded homogeneous linear equation to partition problem

For the reduction function, one defines five-tuples for every element in a. The last tuple plays an important role. It consists only of four elements in order to add constraints to the other tuples. These values are formed in a way such that they form all constraints needed for the reductions. Note, that some indices have been changed with respect to [7] to enable better indexing in the vectors/lists.

```
definition b1 :: "nat ⇒ int ⇒ int ⇒ int" where
  "b1 i M a = a + M * (5^(4*i-4) + 5^(4*i-3) + 5^(4*i-1))"

definition b2 :: "nat ⇒ int ⇒ int" where
  "b2 i M = M * (5^(4*i-3) + 5^(4*i))"

definition b2_last :: "nat ⇒ int ⇒ int" where
  "b2_last i M = M * (5^(4*i-3) + 1)"

definition b3 :: "nat ⇒ int ⇒ int" where
  "b3 i M =  M * (5^(4*i-4) + 5^(4*i-2))"
```

```
definition b4 :: "nat ⇒ int ⇒ int ⇒ int" where
  "b4 i M a = a + M * (5^(4*i-2) + 5^(4*i-1) + 5^(4*i))"

definition b4_last :: "nat ⇒ int ⇒ int ⇒ int" where
  "b4_last i M a = a + M * (5^(4*i-2) + 5^(4*i-1) + 1)"

definition b5 :: "nat ⇒ int ⇒ int" where
  "b5 i M = M * (5^(4*i-1))"
```

Change order of indices in [7] such that $b3$ is in last place and can be omitted in the last entry. This ensures that the weight of the solution is 1 or -1, essential for the proof of NP-hardnes.

```
definition b_list :: "int list ⇒ nat ⇒ int ⇒ int list" where
  "b_list as i M = [b1 (i+1) M (as!i), b2 (i+1) M, b4 (i+1) M (as!i),
b5 (i+1) M, b3 (i+1) M]"

definition b_list_last :: "int list ⇒ nat ⇒ int ⇒ int list" where
  "b_list_last as n M = [b1 n M (last as), b2_last n M, b4_last n M (last
as), b5 n M]"

definition gen_bhle :: "int list ⇒ int vec" where
"gen_bhle as = (let M = 2*(∑ i<length as. |as!i|)+1; n = length as in
  vec_of_list (concat
  (map (λi. b_list as i M) [0..<n-1])
  @ (if n>0 then b_list_last as n M else [])))"
```

The reduction function.

```
definition reduce_bhle_partition:: "(int list) ⇒ ((int vec) * int)" where
  "reduce_bhle_partition ≡ (λ a. (gen_bhle a, 1))"
```

Lemmas for proof

```
lemma dim_vec_gen_bhle:
  assumes "as≠[]"
  shows "dim_vec (gen_bhle as) = 5 * (length as) - 1"
using assms
proof (induct as rule: list_nonempty_induct)
  case (single x)
  then show ?case unfolding gen_bhle_def Let_def b_list_def b_list_last_def
by auto
next
  case (cons x xs)
  define M where "M = (2 * (∑ i<length (x # xs). |(x # xs) ! i|) + 1)"
  then show ?case using cons unfolding gen_bhle_def b_list_def b_list_last_def

    Let_def M_def[symmetric]
    by (subst dim_vec_of_list)+
       (use length_concat_map_5[of
```

```
      "(λi. b1 (i + 1) M ((x#xs) ! i))"
      "(λi. b2 (i + 1) M             )"
      "(λi. b4 (i + 1) M ((x#xs) ! i))"
      "(λi. b5 (i + 1) M             )"
      "(λi. b3 (i + 1) M             )"] in ‹simp›)
qed

lemma dim_vec_gen_bhle_empty:
  "dim_vec (gen_bhle []) = 0"
unfolding gen_bhle_def by auto
```

Lemmas about length

```
lemma length_b_list:
  "length (b_list a i M) = 5" unfolding b_list_def by auto

lemma length_b_list_last:
  "length (b_list_last a n M) = 4" unfolding b_list_last_def by auto

lemma length_concat_map_b_list:
  "length (concat (map (λi. b_list as i M) [0..<k])) = 5 * k"
by (subst length_concat)(simp add: comp_def length_b_list sum_list_triv)
```

Last values of *gen_bhle*

```
lemma gen_bhle_last0:
  assumes "length as > 0"
  shows "(gen_bhle as) $ ((length as -1) * 5) =
    b1 (length as) (2*(∑ i<length as. |as!i|)+1) (last as)"
unfolding gen_bhle_def Let_def
proof (subst vec_of_list_append, subst index_append_vec(1), goal_cases)
  case 1
  then show ?case using assms
    by (subst dim_vec_of_list)+ (split if_splits,
        subst length_b_list_last, subst length_concat_map_b_list, auto)

next
  case 2
  then show ?case using assms
  by (subst dim_vec_of_list, subst length_concat_map_b_list, subst vec_index_vec_of_list)+

     (auto split: if_splits simp add: b_list_last_def)
qed


lemma gen_bhle_last1:
  assumes "length as > 0"
  shows "(gen_bhle as) $ ((length as -1) * 5 + 1) =
    b2_last (length as) (2*(∑ i<length as. |as!i|)+1)"
unfolding gen_bhle_def Let_def
proof (subst vec_of_list_append, subst index_append_vec(1),
```

```
  goal_cases)
  case 1
  then show ?case using assms
    by (subst dim_vec_of_list)+ (split if_split,
      subst length_b_list_last, subst length_concat_map_b_list, auto)

next
  case 2
  then show ?case using assms
  by (subst dim_vec_of_list, subst length_concat_map_b_list, subst vec_index_vec_of_list)+

     (auto split: if_splits simp add: b_list_last_def)
qed
```

The third entry of the last tuple is omitted, thus we skip one lemma

```
lemma gen_bhle_last3:
  assumes "length as > 0"
  shows "(gen_bhle as) $ ((length as -1) * 5 + 2) =
    b4_last (length as) (2*(∑ i<length as. |as!i|)+1) (last as)"
unfolding gen_bhle_def Let_def
proof (subst vec_of_list_append, subst index_append_vec(1),
  goal_cases)
  case 1
  then show ?case using assms
    by (subst dim_vec_of_list)+ (split if_split,
      subst length_b_list_last, subst length_concat_map_b_list, auto)

next
  case 2
  then show ?case using assms
  by (subst dim_vec_of_list, subst length_concat_map_b_list, subst vec_index_vec_of_list)+

     (auto split: if_splits simp add: b_list_last_def)
qed

lemma gen_bhle_last4:
  assumes "length as > 0"
  shows "(gen_bhle as) $ ((length as-1) * 5 + 3) =
    b5 (length as) (2*(∑ i<length as. |as!i|)+1)"
unfolding gen_bhle_def Let_def
proof (subst vec_of_list_append, subst index_append_vec(1),
  goal_cases)
  case 1
  then show ?case using assms
    by (subst dim_vec_of_list)+ (split if_split,
      subst length_b_list_last, subst length_concat_map_b_list, auto)

next
  case 2
```

```
  then show ?case using assms
  by (subst dim_vec_of_list, subst length_concat_map_b_list, subst vec_index_vec_of_list)+

     (auto split: if_splits simp add: b_list_last_def)
qed
```

Up to last values of *gen_bhle*

```
lemma b_list_nth:
  assumes "i<length as-1" "j<5"
  shows "concat (map (λi. b_list as i M) [0..<length as - 1]) ! (i *
5 + j) =
      b_list as i M ! j"
proof -
  have "map (λi. b_list as i M) [0..<length as - 1] =
        map (λi. b_list as i M) [0..<i] @
        map (λi. b_list as i M) [i..<length as - 1]"
    using assms
    by (metis append_self_conv2 less_zeroE linorder_neqE_nat map_append
upt.simps(1) upt_append)
  then have "concat (map (λi. b_list as i M) [0..<length as - 1]) =
        concat (map (λi. b_list as i M) [0..<i]) @
        concat (map (λi. b_list as i M) [i..<length as - 1])"
    by (subst concat_append[of "map (λi. b_list as i M) [0..<i]"
      "map (λi. b_list as i M) [i..<length as -1]", symmetric], auto)
  moreover have "concat (map (λi. b_list as i M) [i..<length as - 1])
=
    (b_list as i M @ concat (map (λi. b_list as i M) [i+1..<length as
- 1]))"
    using assms upt_conv_Cons by fastforce
  ultimately have concat_unfold: "concat (map (λi. b_list as i M) [0..<length
as - 1]) =
        concat (map (λi. b_list as i M) [0..<i]) @
        (b_list as i M @ concat (map (λi. b_list as i M) [i+1..<length
as - 1]))"
    by auto
  have "concat (map (λi. b_list as i M) [0..<length as - 1]) ! (i * 5
+ j) =
    (b_list as i M @ concat (map (λi. b_list as i M) [i+1..<length as
- 1])) ! j"
    unfolding concat_unfold
    by (subst nth_append_length_plus[of "concat (map (λi. b_list as i
M) [0..<i])"
      "b_list as i M @ concat (map (λi. b_list as i M) [i + 1..<length
as - 1])" j, symmetric])
       (subst length_concat_map_b_list, simp add: mult.commute)
  moreover have "(b_list as i M @ concat (map (λi. b_list as i M) [i+1..<length
as - 1])) ! j =
    b_list as i M ! j" using assms length_b_list
    by (subst nth_append[of "b_list as i M" "concat (map (λi. b_list as
```

```
i M)
      [i+1..<length as - 1])" j], auto)
  ultimately show ?thesis by auto
qed


lemma b_list_nth0:
  assumes "i<length as-1"
  shows "concat (map (λi. b_list as i M) [0..<length as - 1]) ! (i *
5) =
      b_list as i M ! 0"
using b_list_nth[OF assms, of 0] by auto

lemma gen_bhle_0:
  assumes "i<length as-1"
  shows "(gen_bhle as) $ (i * 5) =
    b1 (i+1) (2*(∑ i<length as. |as!i|)+1) (as!i)"
unfolding gen_bhle_def Let_def
proof (subst vec_of_list_append, subst index_append_vec(1), goal_cases)
  case 1
  then show ?case using assms
    by (subst dim_vec_of_list)+ (split if_split,
      subst length_b_list_last, subst length_concat_map_b_list, auto)

next
  case 2
  define M where "M = (2 * (∑ i<length as. |as ! i|) + 1)"
  then show ?case unfolding M_def[symmetric] using assms
  by (subst dim_vec_of_list, subst length_concat_map_b_list, subst vec_index_vec_of_list)+

     (subst b_list_nth0[OF assms, of M], auto split: if_splits, simp add:
b_list_def)
qed

lemma gen_bhle_1:
  assumes "i<length as-1"
  shows "(gen_bhle as) $ (i * 5 + 1) =
    b2 (i+1) (2*(∑ i<length as. |as!i|)+1)"
unfolding gen_bhle_def Let_def
proof (subst vec_of_list_append, subst index_append_vec(1), goal_cases)
  case 1
  then show ?case using assms
    by (subst dim_vec_of_list)+ (split if_split,
      subst length_b_list_last, subst length_concat_map_b_list, auto)

next
  case 2
  define M where "M = (2 * (∑ i<length as. |as ! i|) + 1)"
  then show ?case unfolding M_def[symmetric] using assms
```

```
  by (subst dim_vec_of_list, subst length_concat_map_b_list, subst vec_index_vec_of_list)+

     (subst b_list_nth[OF assms, of 1 M], auto split: if_splits, simp
add: b_list_def)
qed

lemma gen_bhle_4:
  assumes "i<length as-1"
  shows "(gen_bhle as) $ (i * 5 + 4) =
    b3 (i+1) (2*(∑ i<length as. |as!i|)+1)"
unfolding gen_bhle_def Let_def
proof (subst vec_of_list_append, subst index_append_vec(1), goal_cases)
  case 1
  then show ?case using assms
    by (subst dim_vec_of_list)+ (split if_split,
      subst length_b_list_last, subst length_concat_map_b_list, auto)

next
  case 2
  define M where "M = (2 * (∑ i<length as. |as ! i|) + 1)"
  then show ?case unfolding M_def[symmetric] using assms
  by (subst dim_vec_of_list, subst length_concat_map_b_list, subst vec_index_vec_of_list)+

     (subst b_list_nth[OF assms, of 4 M], auto split: if_splits, simp
add: b_list_def)
qed

lemma gen_bhle_2:
  assumes "i<length as-1"
  shows "(gen_bhle as) $ (i * 5 + 2) =
    b4 (i+1) (2*(∑ i<length as. |as!i|)+1) (as!i)"
unfolding gen_bhle_def Let_def
proof (subst vec_of_list_append, subst index_append_vec(1), goal_cases)
  case 1
  then show ?case using assms
    by (subst dim_vec_of_list)+ (split if_split,
      subst length_b_list_last, subst length_concat_map_b_list, auto)

next
  case 2
  define M where "M = (2 * (∑ i<length as. |as ! i|) + 1)"
  then show ?case unfolding M_def[symmetric] using assms
  by (subst dim_vec_of_list, subst length_concat_map_b_list, subst vec_index_vec_of_list)+

     (subst b_list_nth[OF assms, of 2 M], auto split: if_splits, simp
add: b_list_def)
qed

lemma gen_bhle_3:
```

```
  assumes "i<length as-1"
  shows "(gen_bhle as) $ (i * 5 + 3) =
    b5 (i+1) (2*(∑ i<length as. |as!i|)+1)"
unfolding gen_bhle_def Let_def
proof (subst vec_of_list_append, subst index_append_vec(1), goal_cases)
  case 1
  then show ?case using assms
    by (subst dim_vec_of_list)+ (split if_split,
      subst length_b_list_last, subst length_concat_map_b_list, auto)

next
  case 2
  define M where "M = (2 * (∑ i<length as. |as ! i|) + 1)"
  then show ?case unfolding M_def[symmetric] using assms
  by (subst dim_vec_of_list, subst length_concat_map_b_list, subst vec_index_vec_of_list)+

     (subst b_list_nth[OF assms, of 3 M], auto split: if_splits, simp
add: b_list_def)
qed
```

Well-definedness of reduction function

```
lemma well_defined_reduction_subset_sum:
  assumes "a ∈ partition_problem_nonzero"
  shows "reduce_bhle_partition a ∈ bhle"
using assms unfolding partition_problem_nonzero_def reduce_bhle_partition_def
bhle_def
proof (safe, goal_cases)
  case (1 I)
```

Given a subset I, we must construct a vector x such that the properties of `bhle` hold.

```
  have "finite I" using 1 by (meson subset_eq_atLeast0_lessThan_finite)
  have "length a > 0" using ‹a≠[]› by auto
  define n where "n = length a"
  define minus_x::"int list" where "minus_x = [0,0,-1,1,1]"
  define plus_x::"int list" where "plus_x = [1,-1,0,-1,0]"
  define plus_x_last::"int list" where "plus_x_last = [1,-1,0,-1]"
  define plus_minus::"int list" where "plus_minus = (if n-1∈I then plus_x
else minus_x)"
  define minus_plus::"int list" where "minus_plus = (if n-1∈I then minus_x
else plus_x)"
  define x::"int vec" where
    "x = vec_of_list(concat (map (λi. if i∈I then plus_minus else minus_plus)
[0..<n-1])
       @ plus_x_last)"
  have length_plus_minus: "length plus_minus = 5"
    unfolding plus_minus_def plus_x_def minus_x_def by auto
  have length_minus_plus: "length minus_plus = 5"
    unfolding minus_plus_def plus_x_def minus_x_def by auto
```

```
  have length_concat_map: "length (concat (map (λi. if i ∈ I then plus_minus
else minus_plus)
    [0..<b])) = b*5" for b
    using length_plus_minus length_minus_plus by (induct b, auto)
  have dimx_eq_5dima:"dim_vec x = length a * 5 - 1"
  unfolding x_def dim_vec_of_list length_append length_concat_map plus_x_last_def

    using ‹length a > 0› unfolding n_def[symmetric] by auto
  have "0 < dim_vec x" unfolding dimx_eq_5dima using ‹length a > 0› by
linarith
  define M where "M = 2*(∑ i<length a. |a!i|)+1"
```

Some conditional lemmas for the proof.

```
  have x_nth:
    "x $ (i*5+j) = (if i∈I then plus_minus ! j else minus_plus ! j)" if
"i<n-1" "j<5" for i j
  proof -
    have lt: "i * 5 + j < length (concat (map (λi. if i ∈ I then plus_minus
else minus_plus)
      [0..<n - 1]))"
      using that length_concat_map by auto
    have len_rew: "i*5 = length (concat (map (λi. if i ∈ I then plus_minus
else minus_plus)
      [0..<i]))"
    proof -
      have if_rew: "(λi. if i∈I then plus_minus else minus_plus) =
        (λi. [(if i∈I then plus_minus!0 else minus_plus!0),
              (if i∈I then plus_minus!1 else minus_plus!1),
              (if i∈I then plus_minus!2 else minus_plus!2),
              (if i∈I then plus_minus!3 else minus_plus!3),
              (if i∈I then plus_minus!4 else minus_plus!4)])"
       unfolding plus_minus_def minus_plus_def plus_x_def minus_x_def
by auto
      then show ?thesis
      unfolding if_rew length_concat_map_5[of
        "(λi. if i∈I then plus_minus!0 else minus_plus!0)"
        "(λi. if i∈I then plus_minus!1 else minus_plus!1)"
        "(λi. if i∈I then plus_minus!2 else minus_plus!2)"
        "(λi. if i∈I then plus_minus!3 else minus_plus!3)"
        "(λi. if i∈I then plus_minus!4 else minus_plus!4)"
        "[0..<i]"] by auto
    qed
    have map_rew: "map f [0..<n-1] = map f [0..<i] @ map f [i..<n-1]"

      for f ::"nat ⇒ int list"
      using that(1) by (metis append_Nil map_append not_gr_zero upt_0
upt_append)
    have "concat (map (λi. if i ∈ I then plus_minus else minus_plus)
[0..<n-1]) ! (i * 5 + j) =
```

```
          concat (map (λi. if i ∈ I then plus_minus else minus_plus)
[i..<n-1]) ! j"
     by (subst map_rew, subst concat_append, subst len_rew)
        (subst nth_append_length_plus[of
          "concat (map (λi. if i ∈ I then plus_minus else minus_plus)
[0..<i])"], simp)
    also have "... = (if i ∈ I then plus_minus!j else minus_plus!j)"
    proof -
      have concat_rewr: "concat (map f [i..<n-1])=
       (f i) @ (concat (map f [i+1..<n-1]))" for f::"nat ⇒ int list"
       using that(1) upt_conv_Cons by force
      have length_if: "length (if i ∈ I then plus_minus else minus_plus)
= 5"
        using length_plus_minus length_minus_plus by auto
      show ?thesis unfolding concat_rewr nth_append length_if using ‹j<5›
by auto
    qed
    finally show ?thesis unfolding x_def by (subst vec_index_vec_of_list)

      (subst nth_append, use lt in ‹auto›)
  qed

  have x_nth0:
    "x $ (i*5) = (if i∈I then plus_minus ! 0 else minus_plus ! 0)" if
"i<n-1" for i
    using that by (subst x_nth[of i 0,symmetric], auto)

  have x_nth_last:
    "x $ ((length a -1)*5+j) = plus_x_last ! j"
    if "j<4" for j
  using that unfolding  x_def vec_of_list_index using nth_append_length_plus[of

    "concat (map (λi. if i ∈ I then plus_minus else minus_plus) [0..<n
- 1])"
    "plus_x_last" j] unfolding length_concat_map n_def
  by auto

  have x_nth0_last:
    "x $ ((length a-1) *5) = plus_x_last ! 0"
  by (subst x_nth_last[of 0,symmetric], auto)

  have gen_bhle_in_I_plus:
    "(∑j=0..<5. (gen_bhle a) $ (i*5+j) * x $ (i*5+j)) =
     (b1 (i+1) M (a!i)) - (b2 (i+1) M) - (b5 (i+1) M)" if "i∈I-{length
a-1}" "n-1∈I" for i
  proof -
    have "(∑j=0..<5. (gen_bhle a) $ (i*5+j) * x $ (i*5+j)) =
            (gen_bhle a) $ (i*5) * x $ (i*5) +
            (gen_bhle a) $ (i*5+1) * x $ (i*5+1) +
```

```
          (gen_bhle a) $ (i*5+2) * x $ (i*5+2) +
          (gen_bhle a) $ (i*5+3) * x $ (i*5+3) +
          (gen_bhle a) $ (i*5+4) * x $ (i*5+4)"
      by (simp add: eval_nat_numeral)
    also have "... = (b1 (i+1) M (a!i)) - (b2 (i+1) M) - (b5 (i+1) M)"

    using that 1 n_def ‹length a > 0›
    apply (subst gen_bhle_0[of i a], fastforce)
    apply (subst gen_bhle_1[of i a], fastforce)
    apply (subst gen_bhle_2[of i a], fastforce)
    apply (subst gen_bhle_3[of i a], fastforce)
    apply (subst gen_bhle_4[of i a], fastforce)
    apply (subst x_nth[of i], fastforce, fastforce)+
    apply (subst x_nth0[of i], fastforce)
    apply (unfold M_def plus_minus_def minus_plus_def plus_x_def minus_x_def)
    apply (simp add: eval_nat_numeral)
    done
    finally show ?thesis by auto
  qed

  have gen_bhle_in_I_minus:
    "(∑j=0..<5. (gen_bhle a) $ (i*5+j) * x $ (i*5+j)) =
     (b3 (i+1) M) - (b4 (i+1) M (a!i)) + (b5 (i+1) M)" if "i∈I-{length
a-1}" "n-1∉I" for i
  proof -
    have "(∑j=0..<5. (gen_bhle a) $ (i*5+j) * x $ (i*5+j)) =
          (gen_bhle a) $ (i*5) * x $ (i*5) +
          (gen_bhle a) $ (i*5+1) * x $ (i*5+1) +
          (gen_bhle a) $ (i*5+2) * x $ (i*5+2) +
          (gen_bhle a) $ (i*5+3) * x $ (i*5+3) +
          (gen_bhle a) $ (i*5+4) * x $ (i*5+4)"
      by (simp add: eval_nat_numeral)
    also have "... = (b3 (i+1) M) - (b4 (i+1) M (a!i)) + (b5 (i+1) M)"

    using that 1 n_def ‹length a > 0›
    apply (subst gen_bhle_0[of i a], fastforce)
    apply (subst gen_bhle_1[of i a], fastforce)
    apply (subst gen_bhle_2[of i a], fastforce)
    apply (subst gen_bhle_3[of i a], fastforce)
    apply (subst gen_bhle_4[of i a], fastforce)
    apply (subst x_nth[of i], fastforce, fastforce)+
    apply (subst x_nth0[of i], fastforce)
    apply (unfold M_def plus_minus_def minus_x_def)
    apply (simp add: eval_nat_numeral)
    done
    finally show ?thesis by auto
  qed

  have gen_bhle_not_in_I_plus:
```

```
    "(∑j=0..<5. (gen_bhle a) $ (i*5+j) * x $ (i*5+j)) =
     (b3 (i+1) M) - (b4 (i+1) M (a!i)) + (b5 (i+1) M)" if "i∈{0..<n}-I-{n-1}"
"n-1∈I" for i
  proof -
    have "(∑j=0..<5. (gen_bhle a) $ (i*5+j) * x $ (i*5+j)) =
            (gen_bhle a) $ (i*5) * x $ (i*5) +
            (gen_bhle a) $ (i*5+1) * x $ (i*5+1) +
            (gen_bhle a) $ (i*5+2) * x $ (i*5+2) +
            (gen_bhle a) $ (i*5+3) * x $ (i*5+3) +
            (gen_bhle a) $ (i*5+4) * x $ (i*5+4)"
      by (simp add: eval_nat_numeral)
    also have "... = (b3 (i+1) M) - (b4 (i+1) M (a!i)) + (b5 (i+1) M)"

    using that 1 n_def ‹length a > 0›
    apply (subst gen_bhle_0[of i a], fastforce)
    apply (subst gen_bhle_1[of i a], fastforce)
    apply (subst gen_bhle_2[of i a], fastforce)
    apply (subst gen_bhle_3[of i a], fastforce)
    apply (subst gen_bhle_4[of i a], fastforce)
    apply (subst x_nth[of i], fastforce, fastforce)+
    apply (subst x_nth0[of i], fastforce)
    apply (unfold M_def minus_plus_def minus_x_def)
    apply (simp add: eval_nat_numeral)
    done
    finally show ?thesis by auto
  qed

  have gen_bhle_not_in_I_minus:
    "(∑j=0..<5. (gen_bhle a) $ (i*5+j) * x $ (i*5+j)) =
     (b1 (i+1) M (a!i)) - (b2 (i+1) M) - (b5 (i+1) M)" if "i∈{0..<n}-I-{n-1}"
"n-1∉I" for i
  proof -
    have "(∑j=0..<5. (gen_bhle a) $ (i*5+j) * x $ (i*5+j)) =
            (gen_bhle a) $ (i*5) * x $ (i*5) +
            (gen_bhle a) $ (i*5+1) * x $ (i*5+1) +
            (gen_bhle a) $ (i*5+2) * x $ (i*5+2) +
            (gen_bhle a) $ (i*5+3) * x $ (i*5+3) +
            (gen_bhle a) $ (i*5+4) * x $ (i*5+4)"
      by (simp add: eval_nat_numeral)
    also have "... = (b1 (i+1) M (a!i)) - (b2 (i+1) M) - (b5 (i+1) M)"

    using that 1 n_def ‹length a > 0›
    apply (subst gen_bhle_0[of i a], fastforce)
    apply (subst gen_bhle_1[of i a], fastforce)
    apply (subst gen_bhle_2[of i a], fastforce)
    apply (subst gen_bhle_3[of i a], fastforce)
    apply (subst gen_bhle_4[of i a], fastforce)
    apply (subst x_nth[of i], fastforce, fastforce)+
    apply (subst x_nth0[of i], fastforce)
```

```
    apply (unfold M_def minus_plus_def plus_x_def)
    apply (simp add: eval_nat_numeral)
    done
    finally show ?thesis by auto
  qed

  have gen_bhle_last:
    "(∑ j=0..<4. (gen_bhle a) $ ((n-1)*5+j) * x $ ((n-1)*5+j)) =
     (b1 n M (a!(n-1))) - (b2_last n M) - (b5 n M)"
  proof -
    have "(∑ j=0..<4. (gen_bhle a) $ ((n-1)*5+j) * x $ ((n-1)*5+j)) =
            (gen_bhle a) $ ((n-1)*5) * x $ ((n-1)*5) +
            (gen_bhle a) $ ((n-1)*5+1) * x $ ((n-1)*5+1) +
            (gen_bhle a) $ ((n-1)*5+2) * x $ ((n-1)*5+2) +
            (gen_bhle a) $ ((n-1)*5+3) * x $ ((n-1)*5+3)"
      by (simp add: eval_nat_numeral)
    also have "... = (b1 n M (a!(n-1))) - (b2_last n M) - (b5 n M)"
    using 1 n_def <length a > 0> unfolding n_def
    apply (subst gen_bhle_last0[of a, OF <length a > 0>])
    apply (subst gen_bhle_last1[of a, OF <length a > 0>])
    apply (subst gen_bhle_last3[of a, OF <length a > 0>])
    apply (subst gen_bhle_last4[of a, OF <length a > 0>])
    apply (subst x_nth_last, simp)+
    apply (subst x_nth0_last, simp add: n_def)
    apply (unfold M_def plus_x_last_def)
    apply (auto simp add: eval_nat_numeral last_conv_nth)
    done
    finally show ?thesis by auto
  qed
```

The actual proof.

```
  have "(gen_bhle a) · x = 0"
  proof -
    define f where "f = (λi. (∑ j = 0..<5. gen_bhle a $ (i*5+j) * x $
(i*5+j)))"
    have "(gen_bhle a) · x = (∑ i<n*5 -1. (gen_bhle a) $ i * x $ i) "
      unfolding M_def n_def gen_bhle_def scalar_prod_def dimx_eq_5dima

      using lessThan_atLeast0 by auto
    also have "... = (∑ i<(n-1)*5. (gen_bhle a) $ i * x $ i) +
      (∑ i = (n-1)*5..<(n-1)*5 +4. (gen_bhle a) $ i * x $ i)"
    proof (subst split_sum_mid_less[of "(n-1)*5" "n*5-1"], goal_cases)
      case 1
      then show ?case unfolding n_def using <0 < length a> by linarith
    next
      case 2
      have "n * 5 - 1 = (n-1) * 5 + 4" unfolding n_def using <0 < length
a> by linarith
      then show ?case by auto
```

```
    qed
    also have "... = (∑ i = 0..<n-1. f i) +
      (∑ j=0..<4. gen_bhle a $ ((n-1)*5+j) * x $ ((n-1)*5+j))"
    proof -
      have *: "(+) ((n - 1) * 5) ‘ {0..<4} = {(n-1)*5..<(n-1)*5+4}" by
auto
      have "(∑ i = (n - 1) * 5..<(n - 1) * 5 + 4. gen_bhle a $ i * x
$ i) =
        (∑ j = 0..<4. gen_bhle a $ ((n - 1) * 5 + j) * x $ ((n - 1) *
5 + j))"
      using sum.reindex[of "(λj. (n-1)*5+j)" "{0..<4}" "(λi. gen_bhle
a $ i * x $ i)"]
      unfolding comp_def * by auto
      then show ?thesis unfolding f_def lessThan_atLeast0
      by (subst sum_split_idx_prod[of "(λi. (gen_bhle a) $ i * x $ i)"
"n-1" 5], auto)
    qed
    also have "... = (∑ i∈I-{n-1}. f i) + (∑ i∈{0..<n}-I-{n-1}. f i)
+
      (∑ j=0..<4. gen_bhle a $ ((n-1)*5+j) * x $ ((n-1)*5+j))"
    proof -
      have "I - {n - 1} ∪ (({0..<n} - I) - {n - 1}) = {0..<n-1}"
        using "1"(1) n_def by auto
      then show ?thesis
        by (subst sum.union_disjoint[of "I - {n - 1}" "{0..<n} - I - {n
- 1}", symmetric])
           (auto simp add: ‹finite I›)
    qed
    also have "... = M * (∑ i∈{0..<n-1}. 5^(4*(i+1)-4) - 5^(4*(i+1)))
       + M * 5^(4*n-4) - M"
    proof (cases "n-1∈I")
      case True
      have "sum f (I - {n - 1}) + sum f ({0..<n} - I - {n - 1}) +
        (∑ j = 0..<4. gen_bhle a $ ((n - 1) * 5 + j) * x $ ((n - 1) *
5 + j)) =
        (∑ i∈I-{n-1}. (b1 (i+1) M (a!i)) - (b2 (i+1) M) - (b5 (i+1) M))

        + (∑ i∈{0..<n}-I-{n-1}. (b3 (i+1) M)  - (b4 (i+1) M (a!i)) +
(b5 (i+1) M))
        + (b1 n M (a!(n-1))) - (b2_last n M) - (b5 n M)"
      proof -
        have "(∑ i∈I-{n-1}. f i) =
              (∑ i∈I-{n-1}. (b1 (i+1) M (a!i)) - (b2 (i+1) M) - (b5 (i+1)
M)) "
        unfolding f_def using gen_bhle_in_I_plus[OF _ True] by (simp add:
n_def)
        moreover have "(∑ i∈{0..<n}-I-{n-1}. f i) =
              (∑ i∈{0..<n}-I-{n-1}. (b3 (i+1) M)  - (b4 (i+1) M (a!i))
+ (b5 (i+1) M)) "
```

```
        unfolding f_def using gen_bhle_not_in_I_plus[OF _ True] by (meson
sum.cong)
        ultimately show ?thesis unfolding f_def using gen_bhle_last by
auto
      qed
      also have "... = (∑ i∈I-{n-1}.  (a!i) + (M * 5^(4*(i+1)-4) - M *
5^(4*(i+1))))
        + (∑ i∈{0..<n}-I-{n-1}. - (a!i) + (M * 5^(4*(i+1)-4) - M * 5^(4*(i+1))))

        + (a!(n-1)) + M * 5^(4*n-4) - M*1"
      proof -
        have "b1 (i + 1) M (a ! i) - b2 (i + 1) M - b5 (i + 1) M =
           (a!i) + (M * 5^(4*(i+1)-4) - M * 5^(4*(i+1)))" if "i∈I-{n-1}"
for i
        unfolding b1_def b2_def b5_def
        by (smt (verit, best) add_uminus_conv_diff right_diff_distrib)
        moreover have "b3 (i + 1) M - b4 (i + 1) M (a ! i) + b5 (i +
1) M =
          - (a!i) + (M * 5^(4*(i+1)-4) - M * 5^(4*(i+1)))" if "i∈{0..<n}
- I - {n - 1}" for i
        unfolding b3_def b4_def b5_def
        by (smt (verit, best) add_uminus_conv_diff right_diff_distrib)
        moreover have "b1 n M (a ! (n - 1)) - b2_last n M - b5 n M =
          (a!(n-1)) + M * 5^(4*n-4) - M"
        unfolding b1_def b2_last_def b5_def by (simp add: distrib_left)
        moreover have "- b4_last n M (a ! (n - 1)) + b5 n M =
          -(a!(n-1)) - M * 5^(4*n-2) - M"
        unfolding b4_last_def b5_def by (simp add: distrib_left)
        ultimately show ?thesis by auto
      qed
      also have "... = (∑ i∈I-{n-1}.  (a!i)) + (∑ i∈{0..<n}-I-{n-1}.
- (a!i))
        + M * (∑ i∈{0..<n-1}. 5^(4*(i+1)-4) - 5^(4*(i+1)))
        + (a!(n-1)) + M * 5^(4*n-4) - M"
      proof -
        have sets1: "{0..<n - 1} ∩ I = I - {n -1}" using "1"(1) n_def
by auto
        have sets2: "{0..<n - 1} - I = {0..<n}- I - {n-1}" using "1"(1)
n_def by auto
        have subs: "(∑ i∈I-{n-1}. M * 5^(4*i+4*1-4) - M * 5^(4*i+4*1))
+
              (∑ i∈{0..<n}-I-{n-1}. M * 5^(4*i+4*1-4) - M * 5^(4*i+4*1))
=
              (∑ i = 0..<n-1. M * 5^(4*i+4*1-4) - M * 5^(4*i+4*1))"
        using sum.Int_Diff[of "{0..<n-1}" "(λi. M * 5^(4*i+4*1-4) - M
* 5^(4*i+4*1))" "I"]
          ‹finite I› unfolding sets1 sets2  by auto
        show ?thesis
          apply (subst distrib_left)+
```

```
        apply (subst sum.distrib)+
        apply (subst sum_distrib_left)
        apply (subst right_diff_distrib)+
        apply (subst subs[symmetric])
        apply auto
        done
    qed
    also have "... = (∑i∈I. (a!i)) + (∑i∈{0..<n}-I. - (a!i))
      + M * (∑i∈{0..<n-1}. 5^(4*(i+1)-4) - 5^(4*(i+1)))
      + M * 5^(4*n-4) - M"
    proof -
      have *: "a ! (n-1) = sum ((!) a) (I ∩ {n-1})" using True by auto
      have "sum ((!) a) (I - {n-1}) + a ! (n-1) = sum ((!) a) I"
      by (subst sum.Int_Diff[of "I" _ "{n-1}"], unfold *, auto simp
add: ‹finite I›)
      then show ?thesis using True by auto
    qed
    also have "... = M * (∑i∈{0..<n-1}. 5^(4*(i+1)-4) - 5^(4*(i+1)))
      + M * 5^(4*n-4) - M "
      unfolding n_def using 1(2) by (subst sum_negf, auto)
    finally show ?thesis by auto

  next
  case False
    have "sum f (I - {n - 1}) + sum f ({0..<n} - I - {n - 1}) +
      (∑j = 0..<4. gen_bhle a $ ((n - 1) * 5 + j) * x $ ((n - 1) *
5 + j)) =
      (∑i∈{0..<n}-I-{n-1}. (b1 (i+1) M (a!i)) - (b2 (i+1) M) - (b5
(i+1) M))
      + (∑i∈I-{n-1}. (b3 (i+1) M)  - (b4 (i+1) M (a!i)) + (b5 (i+1)
M))
      + (b1 n M (a!(n-1))) - (b2_last n M) - (b5 n M)"
    proof -
      have "(∑i∈{0..<n}-I-{n-1}. f i) =
            (∑i∈{0..<n}-I-{n-1}. (b1 (i+1) M (a!i)) - (b2 (i+1) M)
- (b5 (i+1) M)) "
      unfolding f_def using gen_bhle_not_in_I_minus[OF _ False] by (simp
add: n_def)
      moreover have "(∑i∈I-{n-1}. f i) =
            (∑i∈I-{n-1}. (b3 (i+1) M)  - (b4 (i+1) M (a!i)) + (b5
(i+1) M)) "
      unfolding f_def using gen_bhle_in_I_minus[OF _ False] by (simp
add: n_def)
      ultimately show ?thesis unfolding f_def using gen_bhle_last by
auto
    qed
    also have "... = (∑i∈{0..<n}-I-{n-1}.  (a!i) + (M * 5^(4*(i+1)-4)
- M * 5^(4*(i+1))))
      + (∑i∈I-{n-1}. - (a!i) + (M * 5^(4*(i+1)-4) - M * 5^(4*(i+1))))
```

```
        + (a!(n-1)) + M * 5^(4*n-4) - M*1"
      proof -
        have "b1 (i + 1) M (a ! i) - b2 (i + 1) M - b5 (i + 1) M =
           (a!i) + (M * 5^(4*(i+1)-4) - M * 5^(4*(i+1)))" if "i∈{0..<n}
- I - {n - 1}" for i
        unfolding b1_def b2_def b5_def
        by (smt (verit, best) add_uminus_conv_diff right_diff_distrib)
        moreover have "b3 (i + 1) M - b4 (i + 1) M (a ! i) + b5 (i +
1) M =
          - (a!i) + (M * 5^(4*(i+1)-4) - M * 5^(4*(i+1)))" if "i∈I-{n-1}"
for i
        unfolding b3_def b4_def b5_def
        by (smt (verit, best) add_uminus_conv_diff right_diff_distrib)
        moreover have "b1 n M (a ! (n - 1)) - b2_last n M - b5 n M =
          (a!(n-1)) + M * 5^(4*n-4) - M"
        unfolding b1_def b2_last_def b5_def by (simp add: distrib_left)
        moreover have "- b4_last n M (a ! (n - 1)) + b5 n M =
          -(a!(n-1)) - M * 5^(4*n-2) - M"
        unfolding b4_last_def b5_def by (simp add: distrib_left)
        ultimately show ?thesis by auto
      qed
      also have "... = (∑ i∈{0..<n}-I-{n-1}.  (a!i)) + (∑ i∈I-{n-1}.
- (a!i))
        + M * (∑ i∈{0..<n-1}. 5^(4*(i+1)-4) - 5^(4*(i+1)))
        + (a!(n-1)) + M * 5^(4*n-4) - M"
      proof -
        have sets1: "{0..<n - 1} ∩ I = I - {n -1}" using "1"(1) n_def
by auto
        have sets2: "{0..<n - 1} - I = {0..<n}- I - {n-1}" using "1"(1)
n_def by auto
        have subs: "(∑ i∈{0..<n}-I-{n-1}. M * 5^(4*i+4*1-4) - M * 5^(4*i+4*1))
+
              (∑ i∈I-{n-1}. M * 5^(4*i+4*1-4) - M * 5^(4*i+4*1)) =
              (∑ i = 0..<n-1. M * 5^(4*i+4*1-4) - M * 5^(4*i+4*1))"
        using sum.Int_Diff[of "{0..<n-1}" "(λi. M * 5^(4*i+4*1-4) - M
* 5^(4*i+4*1))" "I"]
          ‹finite I› unfolding sets1 sets2  by auto
        show ?thesis
          apply (subst distrib_left)+
          apply (subst sum.distrib)+
          apply (subst sum_distrib_left)
          apply (subst right_diff_distrib)+
          apply (subst subs[symmetric])
          apply auto
          done
      qed
      also have "... = (∑ i∈{0..<n}-I. (a!i)) + (∑ i∈I. - (a!i))
        + M * (∑ i∈{0..<n-1}. 5^(4*(i+1)-4) - 5^(4*(i+1)))
```

```
        + M * 5^(4*n-4) - M"
      proof -
        have *: "({0..<n}-I) ∩ {n-1} = {n-1}" using False n_def ‹length
a > 0› by auto
        then have **: "a ! (n-1) = sum ((!) a) ({0..<n}-I) ∩ {n-1})"
using False by auto
        have "sum ((!) a) ({0..<n}-I) - {n-1}) + a ! (n-1) = sum ((!)
a) ({0..<n}-I)"
        by (subst sum.Int_Diff[of "{0..<n}-I" _ "{n-1}"], unfold * **,
auto simp add: ‹finite I›)
        then show ?thesis using False by auto
      qed
      also have "... = M * (∑i∈{0..<n-1}. 5^(4*(i+1)-4) - 5^(4*(i+1)))
        + M * 5^(4*n-4) - M "
        unfolding n_def using 1(2) by (subst sum_negf, auto)
      finally show ?thesis by auto
    qed
      also have "... = M * ((∑i∈{1..<n}. 5^(4*i-4) - 5^(4*i)) + 5^(4*n-4)
- 1)"
      proof -
        have sums: "(∑i = Suc 0..<Suc (n - 1). 5 ^ (4 * i - 4) - 5 ^
(4 * i)) =
                    sum ((λi. 5 ^ (4*(i+1) - 4) - 5 ^ (4*(i+1)))) {0..<n
- 1}"
        using sum.atLeast_Suc_lessThan_Suc_shift[of "(λi. 5^(4*i-4) -
5^(4*i))" 0 "n-1"]
        unfolding comp_def by auto
        show ?thesis
        by (subst distrib_left[symmetric], subst right_diff_distrib, subst
mult_1_right)
           (subst sums[symmetric], use ‹0 < length a› n_def in ‹force›)
      qed
      also have "... = M * (((∑i∈{1..<n}. 5^(4*i-4)) + 5^(4*n-4)) - ((∑i∈{1..<n}.
5^(4*i)) + 1))"
        using sum.distrib[of "(λi. 5^(4*i-4))" "(λi. (-1) * 5^(4*i))"
"{1..<n}"]
        by (simp add: sum_subtractf)
      also have "... = M * ((∑i∈{1..<n+1}. 5^(4*i-4)) - (∑i∈{0..<n}.
5^(4*i)))"
        using sum.atLeastLessThan_Suc[of 1 n "(λi. 5^(4*i-4))"]
              sum.atLeast_Suc_lessThan[of 0 n "(λi. 5^(4*i))"]
        by (smt (verit, best) One_nat_def Suc_eq_plus1 Suc_leI ‹0 < length
a› mult_eq_0_iff
          n_def power_0)
      also have "... = M * ((∑i∈{0..<n}. 5^(4*i)) - (∑i∈{0..<n}. 5^(4*i)))"
        using sum.atLeast_Suc_lessThan_Suc_shift[of "(λi. 5^(4*i-4))"
0 n] by auto
      also have "... = 0" by auto
    finally show ?thesis by blast
```

```
  qed

  moreover have "dim_vec x = dim_vec (gen_bhle a)"
    using dim_vec_gen_bhle[OF ‹a≠[]›] dimx_eq_5dima by simp
  moreover have "x ≠ 0_v (dim_vec x)"
  proof (rule ccontr)
    assume "¬ x ≠ 0_v (dim_vec x)"
    then have "x = 0_v (dim_vec x)" by auto
    have "dim_vec x > 3" using ‹0 < length a› dimx_eq_5dima by linarith
    have "(n - Suc 0) * 5 + 3 < dim_vec x"
      unfolding dimx_eq_5dima n_def using ‹length a > 0› by linarith
    then have "x $ ((n-1)*5 + 3) = 0" using ‹dim_vec x > 3›
      by (subst ‹x=0_v (dim_vec x)›, subst index_zero_vec(1)[of "(n-1)*5+3"],
auto)
    moreover have "x $ ((n-1)*5+3) ≠ 0"
    proof -
      have "¬ ((n - 1) * 5 + 3 < (n - 1) * 5)" by auto
      then show ?thesis unfolding x_def vec_of_list_index nth_append
length_concat_map
      plus_x_last_def by auto
    qed
    ultimately show False by auto
  qed
  moreover have "‖x‖_∞ ≤ 1"
  proof -
    let ?x_list = "(concat (map (λi. if i ∈ I then plus_minus
      else minus_plus) [0..<n-1]))"
    have set: "set (?x_list) ⊆ {-1,0,1::int}"
      using ‹length a > 0› 1 unfolding n_def plus_minus_def minus_plus_def

      plus_x_def minus_x_def
      by (subst set_concat, subst set_map)(auto simp add: atLeast0LessThan)
    have "?x_list ! i ∈ {-1,0,1::int}" if "i< length ?x_list" for i
      using nth_mem[OF that] set by auto
    then have *:"?x_list ! i ∈ {-1,0,1::int}" if "i< (n - 1) * 5" for
i using that
      unfolding length_concat_map by auto
    have **: "plus_x_last ! (i - (n - 1) * 5)∈{-1,0,1::int}"
      if "¬ (i<(n-1)*5)" "i<dim_vec x" for i
    proof -
      have "(i - (n - 1) * 5)<4" using that ‹length a > 0›
        unfolding dimx_eq_5dima n_def by auto
      then show ?thesis unfolding plus_x_last_def
      by (smt (z3) add.assoc add_diff_cancel_right' diff_is_0_eq insertCI
less_Suc_eq not_le
      not_less_iff_gr_or_eq nth_Cons' numeral_1_eq_Suc_0 numeral_Bit0
plus_1_eq_Suc)
    qed
    have "x$i∈{-1,0,1::int}" if "i<dim_vec x" for i
```

```
    using that * ** unfolding x_def vec_of_list_index nth_append length_concat_map

      plus_x_last_def by auto
  then have "|x$i|≤1" if "i<dim_vec x" for i using that by fastforce
  then show ?thesis unfolding linf_norm_vec_Max
    by (subst Max_le_iff, auto simp add: exI[of "(λi. dim_vec x > i)"
0] ‹dim_vec x > 0›)
 qed
 ultimately show ?case by (subst exI[of _ x], auto)
qed
```

NP-hardness of reduction function

```
lemma NP_hardness_reduction_subset_sum:
 assumes "reduce_bhle_partition a ∈ bhle"
 shows "a ∈ partition_problem_nonzero"
using assms unfolding reduce_bhle_partition_def bhle_def partition_problem_nonzero_def
proof (safe, goal_cases)
 case (1 x)
```

Given a vector x from `bhle` find the subset I such that it has the same sum as its complement.

```
 have "length a > 0" using 1(3) dim_vec_gen_bhle dim_vec_gen_bhle_empty
by auto
 define I where "I = {i∈{0..<length a}. x $ (5*i)≠0}"
 define n where "n = length a"
 then have "n > 0" using ‹length a>0› by auto
 have dim_vec_x_5n: "dim_vec x = 5 * n - 1" unfolding n_def using 1
 by (metis dim_vec_gen_bhle dim_vec_gen_bhle_empty less_not_refl2)
 have "I⊆{0..<length a}" using 1 I_def by auto
```

This is the trickiest part in the proof. One first has to generate equations from x which form conditions on the entries of x. To do so, we consider the formula `gen_bhle a · x = 0` and rewrite it in basis 5. Every digit then gives us an arithmetic expression in entries of x equals to zero. From these equations, we can deduce the wanted properties.

```
 moreover have "sum ((!) a) I = sum ((!) a) ({0..<length a} - I)"
 proof -
  define M where "M = 2 * (∑ i<length a. |a ! i|) + 1"
```

Rewriting the first formula in a huge sum.

```
  define a0 where "a0 = (λi. if i mod (5::nat) ∈ {0,2} then a!(i div
5) else 0)"
  define a1 where "a1 = (λi. if i mod (5::nat) ∈ {0,4} then 1 else
0::int)"
  define a1_last where "a1_last = (λi. if i mod (5::nat) ∈ {0} then
1 else 0::int)"
  define a2 where "a2 = (λi. if i mod (5::nat) ∈ {0,1} then 1 else
0::int)"
```

```
    define a3 where "a3 = (λi. if i mod (5::nat) ∈ {4,2} then 1 else
0::int)"
    define a3_last where "a3_last = (λi. if i mod (5::nat) ∈ {2} then
1 else 0::int)"
    define a4 where "a4 = (λi. if i mod (5::nat) ∈ {0,2,3} then 1 else
0::int)"
    define a5 where "a5 = (λi. if i mod (5::nat) ∈ {1,2} then
                          (if i div 5 < n-1 then 5^(4*(i div 5 +1)) else
1) else 0::int)"

    define a0_rest' where "a0_rest' =
          (λi. a1 i * 5 ^ (4 * (i div 5)) + a2 i * 5 ^ (4 * (i div 5)
+ 1) +
          a3 i * 5 ^ (4 * (i div 5) + 2) + a4 i * 5 ^ (4 * (i div 5) +
3) + a5 i)"
    define a0_last where "a0_last = (λi.
          a1_last i * 5 ^ (4 * (i div 5)) + a2 i * 5 ^ (4 * (i div 5)
+ 1) +
          a3_last i * 5 ^ (4 * (i div 5) + 2) + a4 i * 5 ^ (4 * (i div
5) + 3) +
          a5 i)"

    define a0_rest where "a0_rest = (λi. if i div 5 < n-1 then a0_rest'
i else a0_last i)"

    let ?P0 = "(λi. b1 (i div 5 + 1) M (a!(i div 5)))"
    let ?P1 = "(λi. (if i div 5 < n-1 then b2 (i div 5 + 1) M else b2_last
(i div 5 + 1) M))"
    let ?P4 = "(λi. (if i div 5 < n-1 then b3 (i div 5 + 1) M else 0))"
    let ?P2 = "(λi. (if i div 5 < n-1 then b4 (i div 5 + 1) M (a!(i div
5))
                      else b4_last (i div 5 + 1) M (a!(i div 5))))"
    let ?P3 = "(λi. b5 (i div 5 + 1) M)"

    have cases_a0: "(a0 i + M * (a0_rest i)) =
      (if i mod 5 = 0 then ?P0 i else
      (if i mod 5 = 1 then ?P1 i else
      (if i mod 5 = 2 then ?P2 i else
      (if i mod 5 = 3 then ?P3 i else ?P4 i))))"
    if "i<5*n" for i
    proof (cases "i div 5 < n-1")
    case True
    then show ?thesis
    by (subst mod_5_if_split[of i "a0 i + M * (a0_rest i)" ?P0 ?P1 ?P2
?P3 ?P4])
       (auto simp add: a0_def a0_rest_def a0_rest'_def
         a1_def a2_def a3_def a4_def a5_def b1_def b2_def b3_def b4_def
b5_def)
    next
```

```
    case False
    then have "i div 5 = n-1" using that by auto
    then show ?thesis
      by (subst mod_5_if_split[of i "a0 i + M * (a0_rest i)" ?P0 ?P1 ?P2
?P3 ?P4])
         (auto simp add: False a0_def a0_rest_def a0_last_def
         a1_def a1_last_def a2_def a3_def a3_last_def a4_def a5_def
         b1_def b2_def b2_last_def b3_def b4_def b4_last_def b5_def)
    qed
```

Splitting of the first part of the sum containing the a_i.

```
    have gen_bhle_nth: "gen_bhle a $ i = a0 i + M * (a0_rest i)"
      if "i<dim_vec (gen_bhle a)" for i
    proof -
      have dim_gen_bhle: "dim_vec (gen_bhle a) = 5 * n-1"
        unfolding n_def using <length a >0> dim_vec_gen_bhle by auto
      have gen_bhle_subst: "gen_bhle a $ i = (concat
         (map (λi. b_list a i M) [0..<n - 1]) @ b_list_last a n M ) !
i"
      (is "gen_bhle a $ i = (?concat_map @ ?last)! i")
        unfolding gen_bhle_def Let_def unfolding M_def[symmetric] n_def
vec_index_vec_of_list
        using <length a > 0> by auto
      have len_concat_map: "length ?concat_map = 5 * (n-1)"using length_concat_map_b_list
.
      show ?thesis
      proof (cases "i div 5 < n-1")
        case True
        then have "i<5*(n-1)" by auto
        then have j_th: "(?concat_map @ ?last)! i = ?concat_map ! i"

          by (subst nth_append, subst len_concat_map, auto)
        have "concat (map (λi. b_list a i M) [0..<n - 1]) ! i =
             concat (map (λi. b_list a i M) [0..<n - 1]) ! (i mod 5 +
5*(i div 5))"
          using mod_mult_div_eq[of i 5,symmetric] by auto
        also have "... = (concat (map (λi. b_list a i M) [0..<i div 5])
@
          concat (map (λi. b_list a i M) [i div 5..<n - 1]) ) ! (i mod
5 + 5*(i div 5))"
          by (smt (z3) True append_self_conv2 concat_append less_zeroE
linorder_neqE_nat
            map_append upt.simps(1) upt_append)
        also have "... = concat (map (λi. b_list a i M) [i div 5..<n -
1]) ! (i mod 5)"
          using nth_append_length_plus[of "concat (map (λi. b_list a i
M) [0..<i div 5])"
            "concat (map (λi. b_list a i M) [i div 5..<n - 1])" "i mod
5"]
```

```
          length_concat_map_b_list[of a M "i div 5"]
        by auto
      also have "... = (b_list a (i div 5) M @
        concat (map (λi. b_list a i M) [i div 5 +1..<n - 1])) ! (i mod
5)"
        using True upt_conv_Cons by auto
      also have "... = b_list a (i div 5) M  ! (i mod 5)"
        unfolding nth_append b_list_def by auto
      finally have i_th: "?concat_map ! i = b_list a (i div 5) M ! (i
mod 5)"
        by auto
      show ?thesis
      apply(subst gen_bhle_subst, subst j_th, subst i_th, subst cases_a0)

        subgoal apply (use that dim_vec_gen_bhle n_def <length a >
0> in <auto>) done
        subgoal apply (subst mod_5_if_split[of i "b_list a (i div 5)
M ! (i mod 5)"
          ?P0 ?P1 ?P2 ?P3 ?P4])
          apply (use True in <auto simp add: b_list_def that>) done
      done
    next
      case False
      then have "i ≥ 5*(n-1)" by auto
      then obtain j where "i = 5*(n-1) + j" and "j<4"
        using <i<dim_vec (gen_bhle a)> <length a > 0>
        unfolding dim_gen_bhle n_def
        by (subst split_lower_plus_diff[of i "5*(length a-1)" "5*(length
a)"], auto)
      have j_th: "(?concat_map @ ?last)! i = ?last ! j"
        using <i<dim_vec (gen_bhle a)> nth_append_length_plus[of ?concat_map
?last j]
        unfolding len_concat_map by (auto simp add: <i = 5*(n-1) + j>
<j<4>)
      have "j = i mod 5" using <i = 5*(n-1) + j> <j<4> by auto
      have "n = i div 5 + 1" using <i = 5*(n-1) + j> <j<4> <length
a > 0> n_def by auto
      then have "i div 5 = n-1" by auto
      have "last a = a ! (i div 5)" unfolding <i div 5 = n-1>
        by (subst last_conv_nth[of a]) (use <length a > 0> n_def in
<auto>)
      have *: "i mod 5 = 4 ⟹ [] ! 0 = 0" by (use <j < 4> <j = i mod
5> in <presburger>)
      show ?thesis
        apply(subst gen_bhle_subst, subst j_th, subst cases_a0)
          subgoal apply (use that dim_vec_gen_bhle n_def <length a
> 0> in <auto>) done
          subgoal apply (subst mod_5_if_split[of i "b_list_last a n
M ! j" ?P0 ?P1 ?P2 ?P3 ?P4])
```

```
          apply (auto simp add: b_list_last_def that <j = i mod 5>
<n = i div 5 + 1>
          <last a = a ! (i div 5)> *)
          done
        done
    qed
  qed


  have "gen_bhle a · x = (∑ i<5*n-1. x $ i *  (a0 i + M * a0_rest i))"
    using 1(1) gen_bhle_nth  unfolding scalar_prod_def Let_def dim_vec_x_5n
1(2)[symmetric]
    by (smt (verit, best) lessThan_atLeast0 lessThan_iff mult.commute
sum.cong)

  then have sum_gen_bhle: "(∑ i<5 * n-1. x $ i * (a0 i + M * a0_rest
i)) = 0"
    using 1(1) by simp
```

The first equation containing the a_i

```
  have eq_0: "(∑ i<n. (x $ (i*5) + x $ (i*5+2)) * a!i) = 0" and
       eq_0': "(∑ i<5*n-1. x$i * (a0_rest i)) = 0"
  proof -
    have *: "(∑ i<5*n-1. |a0 i|) < M"
    proof -
      have "5 * n - 1 = Suc (5 * n - 2)" using <n > 0> by auto
      have "5 * n - 2 = Suc (5 * n - 3)" using <n > 0> by auto
      have "5 * n - 3 = Suc (5 * n - 4)" using <n > 0> by auto
      have "5 * n - 4 = Suc (5 * n - 5)" using <n > 0> by auto
      have "(∑ i<5*n-1. |a0 i|) = (∑ i<5*(n-1). |a0 i|) + (∑ j<4.|a0
((n-1)*5+j)|)"
      proof -
        have "5*n-5 = 5*(n-1)" by auto
        have "(∑ i<5*n-1. |a0 i|) =
          (∑ i<5*n-5. |a0 i|) + |a0 (5*(n-1))| + |a0 (5*(n-1)+1)| +
          |a0 (5*(n-1)+2)| + |a0 (5*(n-1)+3)|"
          unfolding <5 * n - 1 = Suc (5 * n - 2)> sum.lessThan_Suc[of
"(λi.|a0 i|)" "5 * n - 2"]
          unfolding <5 * n - 2 = Suc (5 * n - 3)> sum.lessThan_Suc[of
"(λi.|a0 i|)" "5 * n - 3"]
          unfolding <5 * n - 3 = Suc (5 * n - 4)> sum.lessThan_Suc[of
"(λi.|a0 i|)" "5 * n - 4"]
          unfolding <5 * n - 4 = Suc (5 * n - 5)> sum.lessThan_Suc[of
"(λi.|a0 i|)" "5 * n - 5"]
          unfolding <5*n-5 = 5*(n-1)>
          by (auto simp add: Suc3_eq_add_3 add.commute)
          moreover have "|a0 (5*(n-1))| + |a0 (5*(n-1)+1)| + |a0 (5*(n-1)+2)|
+ |a0 (5*(n-1)+3)| =
            (∑ j<4.|a0 ((n-1)*5+j)|)" by (simp add:eval_nat_numeral)
```

```
        ultimately show ?thesis using <5*n-5 = 5*(n-1)> by auto
      qed
      also have "... = (∑ i<n-1. (∑ j<5. |a0 (i*5+j)|)) + (∑ j<4.|a0
((n-1)*5+j)|)"
        using sum_split_idx_prod[of "(λi. |a0 i|)" "n-1" 5]
        by (simp add: lessThan_atLeast0 mult.commute)
      also have "... = (∑ i<n. (∑ j<5::nat. |if j ∈ {0, 2} then a !
i else 0|))"
      proof -
        have "(5::nat) = Suc 4" by auto
        have "(∑ i<n-1. (∑ j<5::nat. |a0 (i*5+j)|)) =
          (∑ i<n-1. (∑ j<5::nat. |if j ∈ {0, 2} then a ! i else 0|))"
          by(rule sum.cong[OF refl], rule sum.cong[OF refl],
            auto simp add: a0_def div_mult_self3[of 5])
        moreover have "(∑ j<4::nat.|a0 ((n-1)*5+j)|) =
          (∑ j<4::nat.|if j ∈ {0, 2} then a ! (n-1) else 0|)"
          by(rule sum.cong[OF refl],
            auto simp add: a0_def div_mult_self3[of 5])
        moreover have "(∑ j<4::nat.|if j ∈ {0, 2} then a ! (n-1) else
0|) =
          (∑ j<5::nat.|if j ∈ {0, 2} then a ! (n-1) else 0|)"
          unfolding <5=Suc 4>
          using sum.lessThan_Suc[of "(λj.|if j ∈ {0, 2} then a ! (n-1)
else 0|)" "4::nat"]
          by auto
        ultimately have *: "(∑ i<n-1. (∑ j<5. |a0 (i*5+j)|)) + (∑ j<4.|a0
((n-1)*5+j)|) =
          (∑ i<n-1. (∑ j<5::nat. |if j ∈ {0, 2} then a ! i else 0|))
+
          (∑ j<5::nat. |if j ∈ {0, 2} then a ! (n-1) else 0|)" by auto
        show ?thesis by (subst *, subst sum.lessThan_Suc[symmetric],
auto simp add: <n>0>)
      qed
      also have "... = (∑ i<n. 2 * |a ! i|)" by (auto simp add: eval_nat_numeral)
      also have "... = 2 * (∑ i<n. |a ! i|)"
        by (simp add: sum_distrib_left)
      finally show ?thesis unfolding M_def n_def by linarith
    qed
    have **: "∀ i<5*n-1. |x $ i| ≤ 1" using 1(5)
      by (metis dim_vec_x_5n order_trans vec_index_le_linf_norm)
    have "(∑ i<5*n-1. x $ i * a0 i) = 0 ∧ (∑ i<5*n-1. x$i * (a0_rest
i)) = 0"
      using split_eq_system[OF * ** sum_gen_bhle] by auto
    moreover have "(∑ i<5*n-1. x $ i * a0 i) = (∑ i<n. (x $ (i*5)
+ x $ (i*5+2)) * a!i)"
    proof -
      let ?g = "(λj. x $ j * (if j mod 5 ∈ {0, 2} then a ! (j div 5)
else 0))"
      let ?h = "(λi. (x $ (i * 5) + x $ (i * 5 + 2)) * a ! i)"
```

```
    have "(∑ j = i*5..<i*5+5. ?g j) = ?h i" if "i<n-1" for i
    proof -
      have div_rule: "(i * 5 + xa) div 5 = i" if "xa <5" for xa us-
ing that by auto
      have "(∑ j = i*5..<i*5+5. ?g j) = sum ?g ((+) (i * 5) ‘ {0..<5})"
        by (simp add: add.commute)
      also have "... = (∑ j<5. x$(i*5 + j) * (if j ∈ {0, 2} then a!i
else 0))"
        using mod_mult_self3[of i 5] div_rule
        by (subst sum.reindex[of "(λj. i*5+j)" "{0..<5}"], simp, unfold
comp_def)
          (metis (no_types, lifting) One_nat_def lessThan_atLeast0
lessThan_iff
          nat_mod_lem not_less_eq not_numeral_less_one sum.cong)
      also have "... = x$(i*5 + 0) * a!i + x$(i*5 + 2) * a!i"
        by (auto simp add: eval_nat_numeral split: if_splits)
      finally show ?thesis by (simp add: distrib_left mult.commute)
    qed
    then have *: "(∑ i<(n-1)*5. x $ i * a0 i) = (∑ i<n-1. ?h i)"
      unfolding a0_def by (subst sum.nat_group[symmetric], auto)
    have **: "(∑ j = 5*(n-1)..<5*(n-1)+4. x $ j * a0 j) = ?h (n-1)"
    proof -
      have div_rule: "((n-1) * 5 + xa) div 5 = (n-1)" if "xa <4" for
xa using that by auto
      have "(∑ j = (n-1)*5..<(n-1)*5+4. ?g j) = sum ?g ((+) ((n-1)
* 5) ‘ {0..<4})"
        by (simp add: add.commute)
      also have "... = (∑ j<4. x$((n-1)*5 + j) * (if j ∈ {0, 2} then
a!(n-1) else 0))"
      proof -
        have *: "(∑ xa = 0..<4.?g ((n - 1) * 5 + xa)) =
        (∑ j = 0..<4. x $ ((n - 1) * 5 + j) * (if j ∈ {0, 2} then
a ! (n - 1) else 0))"
        (is "(∑ xa = 0..<4. ?g' xa) = (∑ j = 0..<4. ?h' j)")
          by (rule sum.cong) auto
        show ?thesis
        using mod_mult_self3[of "n-1" 4] div_rule
        by (subst sum.reindex[of "(λj. (n-1)*5+j)" "{0..<4}"], simp,

          unfold comp_def lessThan_atLeast0) (use * in ‹auto›)
      qed
      also have "... = x$((n-1)*5 + 0) * a!(n-1) + x$((n-1)*5 + 2)
* a!(n-1)"
        by (auto simp add: eval_nat_numeral split: if_splits)
      finally show ?thesis unfolding a0_def by (simp add: distrib_left
mult.commute)
    qed
    have "5 * (n - 1) < 5 * n - 1" using ‹n> 0› by auto
    then have ***: "(∑ i<5*n-1. x $ i * a0 i) = (∑ i<5*(n-1). x $
```

```
i * a0 i) +
          (∑ i=5*(n-1)..<5*n-1. x $ i * a0 i)"
        by (subst split_sum_mid_less[of "5*(n-1)" "5*n-1"], auto)
        have ****: "5 * n - 1 = 5 * (n - 1) + 4" using <n> 0> by auto
        show ?thesis
          by (unfold ***, subst mult.commute[of 5 "n-1"], unfold * ****
**)
             (subst sum.lessThan_Suc[of ?h "n-1", symmetric], auto simp
add: <n>0>)
      qed
      ultimately show "(∑ i<n. (x $ (i*5) + x $ (i*5+2)) * a!i) = 0"
and
                      "(∑ i<5*n-1. x$i * (a0_rest i)) = 0" by auto
    qed

    let ?eq_0'_left = "(∑ i<5*n-1. x$i * (a0_rest i))"
    interpret digits 5 by (simp add: digits_def)
    have digit_a0_rest: "digit ?eq_0'_left k = 0" for k
      using eq_0' by (simp add: eq_0' digit_altdef)
```

Define the digits in basis 5.

```
    define d1 where "d1 = (λi. x$(i*5) + (if i<n-1 then x$(i*5+4) else
0))"
    define d2 where "d2 = (λi. x$(i*5) + x$(i*5+1))"
    define d3 where "d3 = (λi. (if i<n-1 then x$(i*5+4) else 0) + x$(i*5+2))"
    define d4 where "d4 = (λi. x$(i*5) + x$(i*5+2) + x$(i*5+3))"
    define d5 where "d5 = (λi. x$(5*i+1) + x$(5*i+2))"

    define d where "d = (λk.
      (if k mod 4 = 0 then
          (if k = 0 then d5 (n-1) + d1 0 else (d1 (k div 4) + d5 (k div
4 -1))) else
      (if k mod 4 = 1 then d2 (k div 4) else
      (if k mod 4 = 2 then d3 (k div 4) else d4 (k div 4)))))"
```

Rewrite the sum in basis 5.

```
    have rewrite_digits: "(∑ i<5*n-1. x$i * (a0_rest i)) = (∑ k<4*n.
d k * 5^k)"
    proof -
      define f1::"nat ⇒ nat ⇒ int" where "f1 = (λi j.
            ((if i<n-1 then (if j ∈ {0, 4} then 1 else 0) else (if j∈{0}
then 1 else 0))
               * 5 ^ (4 * i) +
            (if j ∈ {0, 1} then 1 else 0) * 5 ^ (4 * i + 1) +
            (if i<n-1 then (if j ∈ {4, 2} then 1 else 0) else (if j∈{2}
then 1 else 0))
              * 5 ^ (4 * i + 2) +
            (if j ∈ {0, 2, 3} then 1 else 0) * 5 ^ (4 * i + 3) +
            (if j ∈ {1, 2} then if i < n - 1 then 5 ^ (4 * (i + 1)) else
```

```
1 else 0)))"
      have "f1 (n-1) 4 = 0" unfolding f1_def by auto
      define f2 where "f2 = (λi.
          x $ (i * 5) * (5 ^ (4 * i) + 5 ^ (4 * i + 1) + 5 ^ (4 * i +
3)) +
          x $ (i * 5 + 1) * (5 ^ (4 * i + 1) + (if i < n - 1 then 5 ^
(4 * (i + 1)) else 1)) +
          x $ (i * 5 + 4) * (if i<n-1 then 5 ^ (4 * i) + 5 ^ (4 * i +
2) else 0) +
          x $ (i * 5 + 2) * (5 ^ (4 * i + 2) + 5 ^ (4 * i + 3) +
            (if i < n - 1 then 5 ^ (4 * (i + 1)) else 1)) +
          x $ (i * 5 + 3) * (5 ^ (4 * i + 3)))"
      define f3 where "f3 = (λi.
          d1 i * (5 ^ (4 * i)) + d5 i * (if i < n - 1 then 5 ^ (4 * (i
+ 1)) else 1) +
          d2 i * 5 ^ (4 * i + 1) + d3 i * 5 ^ (4 * i + 2) + d4 i * 5 ^
(4 * i + 3))"
      have f2_f3: "f2 i = f3 i" if "i<n" for i
      proof (cases "i<n-1")
        case True
        then have "f2 i = x $ (i * 5) * 5 ^ (4 * i) + x $ (i * 5) * 5
^ (4 * i + 1) +
          x $ (i * 5) * 5 ^ (4 * i + 3) +
          x $ (i * 5 + 1) * 5 ^ (4 * i + 1) + x $ (i * 5 + 1) * 5 ^ (4
* (i + 1)) +
          x $ (i * 5 + 4) * 5 ^ (4 * i) + x $ (i * 5 + 4) * 5 ^ (4 * i
+ 2) +
          x $ (i * 5 + 2) * 5 ^ (4 * i + 2) + x $ (i * 5 + 2) * 5 ^ (4
* i + 3) +
          x $ (i * 5 + 2) * 5 ^ (4 * (i + 1)) + x $ (i * 5 + 3) * 5 ^
(4 * i + 3)"
        unfolding f2_def by (subst ring_distribs)+ simp
        also have "... = f3 i" unfolding f3_def d1_def d2_def d3_def d4_def
d5_def using True
          by (subst distrib_right)+  (simp add: mult.commute[of 5 i])
        ultimately show ?thesis by auto
      next
        case False
        then have "f2 i = x $ (i * 5) * 5 ^ (4 * i) + x $ (i * 5) * 5
^ (4 * i + 1) +
          x $ (i * 5) * 5 ^ (4 * i + 3) +
          x $ (i * 5 + 1) * 5 ^ (4 * i + 1) + x $ (i * 5 + 1) +
          x $ (i * 5 + 2) * 5 ^ (4 * i + 2) + x $ (i * 5 + 2) * 5 ^ (4
* i + 3) +
          x $ (i * 5 + 2) + x $ (i * 5 + 3) * 5 ^ (4 * i + 3)"
        unfolding f2_def by (subst ring_distribs)+ simp
      also have "... = f3 i" unfolding f3_def d1_def d2_def d3_def d4_def
d5_def using False
        by(simp add: algebra_simps)
```

```
      ultimately show ?thesis by auto
    qed
    define x_pad where "x_pad = (λi. if i<5*n-1 then x$i else 1)"
    have pad: "(∑ i<5*n-1. x$i * (a0_rest i)) = (∑ i<5*n. x_pad i *
(a0_rest i))"
    proof -
      have "Suc (5 * n - 1) = 5*n" using ‹n>0› by auto
      have "(∑ i<5 * n - 1. x_pad i * a0_rest i) = (∑ i<5 * n - 1.
x$i * a0_rest i)"
        by(rule sum.cong)(auto simp: x_pad_def)
      moreover have "x_pad (5 * n - 1) * a0_rest (5 * n - 1) = 0"
      proof -
        have "¬((5 * n - 1) div 5 < n - 1)" using ‹n>0› by auto
        moreover have "(5 * n - 1) mod 5 = 4"
        proof -
          have "5 * n - 1 = 5*(n-1)+4" using ‹n>0› by auto
          show ?thesis unfolding ‹5 * n - 1 = 5*(n-1)+4› by auto
        qed
        ultimately show ?thesis
          unfolding a0_rest_def a0_last_def a1_last_def a2_def a3_last_def
a4_def a5_def
          by auto
      qed
      ultimately show ?thesis
        using sum.lessThan_Suc[of "(λi. x_pad i * (a0_rest i))" "5 *
n - 1"]
        unfolding ‹Suc (5 * n - 1) = 5*n› by auto
    qed
    have *: "(∑ i<5*n. x_pad i * (a0_rest i)) =
      (∑ i<n.(∑ j<5. x_pad (i*5+j) * (a0_rest (i*5+j))))"
      (is "... =  (∑ i<n.(∑ j<5. ?f0 i j))")
      using sum_split_idx_prod[of "(λi. x_pad i * a0_rest i)" n 5]
      unfolding mult.commute[of n 5] using atLeast0LessThan by auto
    also have  "... = (∑ i<n. ∑ j<5. x_pad (i * 5 + j) * f1 i j)"
    proof (subst sum.cong[of _ _ "(λi. (∑ j<(5::nat). ?f0 i j))"
          "(λi. (∑ j<(5::nat).  x_pad (i * 5 + j) * f1 i j))", symmetric],
simp, goal_cases)
      case (1 i)
      have **: "a0_rest (i * 5 + j) = f1 i j"
        if "j<5" for j using that lt_5_split[of j] 1
      unfolding f1_def a0_rest_def a0_rest'_def a0_last_def
          a1_def a1_last_def a2_def a3_def a3_last_def a4_def a5_def
        by auto
      show ?case
        by (rule sum.cong) (use ** 1 in auto)
    next
      case 2
      then show ?case using * by auto
    qed
```

```
    also have "... = (∑ i<n. f2 i)"
    proof (rule sum.cong[OF refl], goal_cases)
      case (1 i)
      show ?case
      proof (cases "i<n-1")
        case True
        then show ?thesis unfolding f1_def x_pad_def f2_def
        by (auto simp add: eval_nat_numeral)
      next
        case False
        then have "i = n-1" using 1 by auto
        then have "(∑ j<5. x_pad (i * 5 + j) * f1 i j) =
          x_pad ((n-1) * 5 + 0) * f1 (n-1) 0 +
          x_pad ((n-1) * 5 + 1) * f1 (n-1) 1 +
          x_pad ((n-1) * 5 + 2) * f1 (n-1) 2 +
          x_pad ((n-1) * 5 + 3) * f1 (n-1) 3 +
          x_pad ((n-1) * 5 + 4) * f1 (n-1) 4"
        by (simp add: eval_nat_numeral)
        also have "...  =
          x_pad ((n-1) * 5 + 0) * f1 (n-1) 0 +
          x_pad ((n-1) * 5 + 1) * f1 (n-1) 1 +
          x_pad ((n-1) * 5 + 2) * f1 (n-1) 2 +
          x_pad ((n-1) * 5 + 3) * f1 (n-1) 3"
        using ‹f1 (n-1) 4 = 0› by auto
        also have "... = f2 i"
        proof -
          have "x_pad ((n-1) * 5 + 0) * f1 (n-1) 0 =
            x $ ((n-1)*5) * (5^(4 * (n - 1)) + 5 ^ (4 * (n - 1) + 1)
+ 5 ^ (4 * (n - 1) + 3))"
          unfolding f1_def x_pad_def using ‹n>0› by auto
          moreover have "x_pad ((n-1) * 5 + 1) * f1 (n-1) 1 =
            x $ ((n-1)*5 + 1) * (5^(4 * (n - 1) + 1) + 1)"
          unfolding f1_def x_pad_def using ‹n>0› by auto
          moreover have "x_pad ((n-1) * 5 + 2) * f1 (n-1) 2 =
            x $ ((n-1)*5 + 2) * (5^(4 * (n - 1) + 2) + 5 ^ (4 * (n -
1) + 3) + 1)"
          unfolding f1_def x_pad_def using ‹n>0› by auto
          moreover have "x_pad ((n-1) * 5 + 3) * f1 (n-1) 3 =
            x $ ((n-1)*5 + 3) * 5^(4 * (n - 1) + 3)"
          unfolding f1_def x_pad_def using ‹n>0› by auto
          ultimately show ?thesis unfolding f2_def using ‹i=n-1› by
auto
        qed
        finally show ?thesis by auto
      qed
    qed
    also have "... = (∑ i<n-1. f2 i) + f2 (n-1)"
      by (subst sum.lessThan_Suc[of f2 "n-1", symmetric])
      (use Suc_diff_1 ‹0 < length a› n_def in ‹presburger›)
```

```
      also have "... = (∑i<n-1. f3 i) + f3 (n-1)"
        by (subst sum.cong[of "{..<n-1}" "{..<n-1}" f2 f3], auto simp
add: f2_f3)
           (use f2_f3[of "n-1"] ‹length a > 0› n_def in ‹auto›)
      also have "... = (∑i<n-1.  d1 i * (5 ^ (4 * i)) + d5 i *  5 ^ (4
* (i + 1)) +
          d2 i * 5 ^ (4 * i + 1) + d3 i * 5 ^ (4 * i + 2) + d4 i * 5 ^
(4 * i + 3) ) +
          d1 (n-1) * (5 ^ (4 * (n-1))) + d5 (n-1) + d2 (n-1) * 5 ^ (4
* (n-1) + 1) +
          d3 (n-1) * 5 ^ (4 * (n-1) + 2) + d4 (n-1) * 5 ^ (4 * (n-1) +
3)"
        unfolding f3_def by auto
      also have "... = (∑i<n-1.  d1 i * (5 ^ (4 * i))) + d1 (n-1) * (5
^ (4 * (n-1))) +
                       (∑i<n-1. d2 i * 5 ^ (4 * i + 1)) + d2 (n-1) * 5
^ (4 * (n-1) + 1) +
                       (∑i<n-1. d3 i * 5 ^ (4 * i + 2)) + d3 (n-1) * 5
^ (4 * (n-1) + 2) +
                       (∑i<n-1. d4 i * 5 ^ (4 * i + 3)) + d4 (n-1) * 5
^ (4 * (n-1) + 3) +
                       (∑i<n-1. d5 i *  5 ^ (4 * (i + 1))) + d5 (n-1)"
      by auto
      also have "... = (∑i<n. d1 i * (5 ^ (4 * i))) +
                       (∑i<n. d2 i * 5 ^ (4 * i + 1)) +
                       (∑i<n. d3 i * 5 ^ (4 * i + 2)) +
                       (∑i<n. d4 i * 5 ^ (4 * i + 3)) +
                       (∑i<n-1. d5 i *  5 ^ (4 * (i + 1))) +
                        d5 (n-1)" (is "... = ?f4")
        using sum.lessThan_Suc[of "(λi. d1 i * 5 ^ (4 * i))" "n-1"]
        using sum.lessThan_Suc[of "(λi. d2 i * 5 ^ (4 * i + 1))" "n-1"]
        using sum.lessThan_Suc[of "(λi. d3 i * 5 ^ (4 * i + 2))" "n-1"]
        using sum.lessThan_Suc[of "(λi. d4 i * 5 ^ (4 * i + 3))" "n-1"]
        using ‹0 < length a› n_def by force
      finally have "(∑i<5*n. x_pad i * (a0_rest i)) = ?f4" using * by
auto
      then have "(∑i<5*n-1. x$i * (a0_rest i)) = ?f4" using pad by
auto
      moreover have "(∑i<n. d1 i * (5 ^ (4 * i))) = d1 0 + (∑i<n-1.
d1 (i + 1) * (5 ^ (4 * (i+1))))"
        using sum.lessThan_Suc_shift[of "(λi. d1 i * 5 ^ (4 * i))" "n-1"]

        using ‹0 < length a› n_def by force
      moreover have "(∑i<n-1. d1 (i + 1) * (5 ^ (4 * (i+1)))) +
        (∑i<n-1. d5 i *  5 ^ (4 * (i + 1))) =
        (∑i<n-1. (d1 (i+1) + d5 i) *  5 ^ (4 * (i + 1)))"
        unfolding sum.distrib[of "(λi. d1 (i + 1) * 5 ^ (4 * (i + 1)))"

          "(λi. d5 i * 5 ^ (4 * (i + 1)))" "{..<n-1}", symmetric]
```

```
        distrib_right by simp
    moreover have "(∑i<n-1. (d1 (i+1) + d5 i) *  5 ^ (4 * (i + 1)))
=
      (∑i∈{1..<n}. (d1 i + d5 (i-1)) *  5 ^ (4 * i))"
    proof -
      have "bij_betw (λi. i - 1) {1..<n} {..<n - 1}" unfolding bij_betw_def

        by (auto simp add: inj_on_def)
           (metis One_nat_def Suc_diff_1 Suc_leI Suc_mono <0 < length
a> add_diff_cancel_left'
            atLeastLessThan_iff image_eqI n_def plus_1_eq_Suc zero_less_Suc)
      then show ?thesis
      by (subst sum.reindex_bij_betw[of "(λi. i-1)" "{1..<n}" "{..<n-1}"

        "(λi. (d1 (i + 1) + d5 i) * 5 ^ (4 * (i + 1)))", symmetric])
auto
    qed
    moreover have "(∑i∈{1..<n}. (d1 i + d5 (i-1)) *  5 ^ (4 * i))
+ d1 0 + d5 (n-1) =
      (∑i<n. (if i = 0 then d5 (n-1) + d1 0 else (d1 i + d5 (i-1)))
* 5 ^ (4 * i))"
       using sum.atLeast_Suc_lessThan[OF <0<length a>,
        of "(λi. (if i = 0 then d5 (n - 1) + d1 0 else d1 i + d5 (i
- 1)) * 5 ^ (4 * i))"]
        unfolding atLeast0LessThan n_def by (auto)
    ultimately have "(∑i<5*n-1. x$i * (a0_rest i)) = (∑i<n.
      (if i = 0 then d5 (n - 1) + d1 0 else d1 i + d5 (i - 1)) * 5 ^
(4 * i) +
       d2 i * 5 ^ (4 * i + 1) + d3 i * 5 ^ (4 * i + 2) + d4 i * 5 ^
(4 * i + 3))"
      (is "(∑i<5*n-1. x$i * (a0_rest i)) = (∑i<n. ?f5 i)") by auto
    moreover have "... = (∑i<n. (∑j<4. d (i*4+j) * 5^(i*4+j)))"
    proof (rule sum.cong, goal_cases)
      case (2 i)
      have d_rew: "(∑j<4. d (i * 4 + j) * 5 ^ (i * 4 + j)) =
        d (i * 4) * 5 ^ (i * 4) + d (i * 4 + 1) * 5 ^ (i * 4 + 1) +
        d (i * 4 + 2) * 5 ^ (i * 4 + 2) + d (i * 4 + 3) * 5 ^ (i * 4
+ 3)"
        by (simp add: eval_nat_numeral)
      have d1_rew: "d (i * 4) = (if i = 0 then d5 (n - 1) + d1 0 else
d1 i + d5 (i - 1))"
        unfolding d_def by auto
      have d2_rew: "d (i*4+1) = d2 i" unfolding d_def
        by (metis add.commute add.right_neutral div_mult_self3 mod_Suc
mod_div_trivial
        mod_mult_self2_is_0 one_eq_numeral_iff plus_1_eq_Suc semiring_norm(85)
zero_neq_numeral)
      have d3_rew: "d (i*4+2) = d3 i" unfolding d_def
        by (metis add.commute add_2_eq_Suc' add_cancel_right_left div_less
```

```
div_mult_self1
          less_Suc_eq mod_mult_self1 nat_mod_lem numeral_Bit0 one_add_one
zero_neq_numeral)
        have d4_rew: "d (i*4+3) = d4 i" unfolding d_def by auto
        show ?case by (subst d_rew, subst d1_rew, subst d2_rew, subst
d3_rew, subst d4_rew)
           (auto simp add: mult.commute)
      qed auto
      moreover have "... = (∑k<4*n. d k * 5^k)"
        using sum_split_idx_prod[of "(λk. d k * 5^k)" n 4, symmetric]
        by (simp add: lessThan_atLeast0 mult.commute)
      ultimately show ?thesis by auto
    qed
```

Some helping lemmas to get equations.

```
    have xi_le_1: "|x$i|≤1" if "i< dim_vec x" for i
      using 1(5) that unfolding linf_norm_vec_Max by auto
    have xs_le_2: "|x$i + x$j|≤2" if "i< dim_vec x" "j< dim_vec x" for
i j
    proof -
      have "|x$i + x$j| ≤ |x$i| + |x$j|"
      by (auto simp add: abs_triangle_ineq)
      then show ?thesis using xi_le_1[OF that(1)] xi_le_1[OF that(2)]
by auto
    qed

    have if_xi_le_1: "|(if i < n - 1 then x $ (i * 5 + 4) else 0)| ≤ 1"

      if "i< n" for i
      using that xi_le_1[of "i*5+4"] unfolding dim_vec_x_5n by auto

    have prec_0: "i * 5 < dim_vec x" if "i<n" for i
      using that unfolding dim_vec_x_5n  by auto
    have prec_i: "i * 5 + j < dim_vec x" if "i<n" "j<4" for i j
      using that unfolding dim_vec_x_5n  by auto

    have abs_d1: "|d1 i|≤2" if "i<n" for i unfolding d1_def
      using xi_le_1[OF prec_0[OF that]] if_xi_le_1[OF that] by auto

    have abs_d2: "|d2 i|≤2" if "i<n" for i unfolding d2_def
      using xi_le_1[OF prec_0[OF that]] xi_le_1[OF prec_i[OF that, where
j=1]] by auto

    have abs_d3: "|d3 i|≤2" if "i<n" for i unfolding d3_def
      using xi_le_1[OF prec_i[OF that, where j=2]] if_xi_le_1[OF that]
by auto

    have abs_d4: "|d4 i|≤3" if "i<n" for i unfolding d4_def
      using xi_le_1[OF prec_0[OF that]]
```

```
        xs_le_2[OF prec_i[OF that, where j=2] prec_i[OF that, where j=3]]
by auto

    have abs_d5: "|d5 i|≤2" if "i<n" for i unfolding d5_def
      using xi_le_1[OF prec_i[OF that, where j=2]]
        xi_le_1[OF prec_i[OF that, where j=1]] by (simp add: mult.commute)

    have " |d5 (n - Suc 0) + d1 0| < 5"
      using abs_d5[of "n-1"] abs_d1[of 0] ‹0 < length a› n_def by fastforce
    moreover have "|d1 (i div 4) + d5 (i div 4 - Suc 0)| < 5"
      if "0 < i" and "i<4*n" and "i mod 4 = 0" for i
    using that abs_d1[of "i div 4"] abs_d5[of "i div 4 - 1"] ‹0 < length
a› n_def by fastforce
    moreover have "|d2 (i div 4)| < 5" if "i<4*n" and "i mod 4 = 1" for
i
    using that abs_d2[of "i div 4"] ‹0 < length a› n_def by fastforce
    moreover have "|d3 (i div 4)| < 5" if "i<4*n" and "i mod 4 = 2" for
i
    using that abs_d3[of "i div 4"] ‹0 < length a› n_def by fastforce
    moreover have "|d4 (i div 4)| < 5" if "i<4*n" and "i mod 4 = 3" for
i
    using that abs_d4[of "i div 4"] ‹0 < length a› n_def by fastforce
    ultimately have d_lt_5: "|d i| < 5" if "i < 4 * n" for i
      using that by (subst mod_4_choices[of i], unfold d_def, auto)

    have sum_zero: "(∑k<4*n. d k * 5^k) = 0" using eq_0' rewrite_digits
by auto

    then have d_eq_0: "d k = 0" if "k<4*n" for k
      using respresentation_in_basis_eq_zero[OF sum_zero _ _ that] d_lt_5
by auto
```

These are the main equations.

```
    have eq_1: "x$(i*5) + (if i <n-1 then x$(i*5+4) else 0) + x$((i-1)*5+1)
+ x$((i-1)*5+2) = 0"
       if "i∈{1..<n}" for i
      using that d_eq_0[of "4*i"] unfolding d_def d1_def d5_def by (auto
simp add: mult.commute)

    have eq_2: "x$0 + (if 0<n-1 then x $ 4 else 0) + x$((n-1)*5+1) + x$((n-1)*5+2)
= 0"
      using d_eq_0 [of 0] ‹0 < n› unfolding d_def d1_def d5_def
      by (cases ‹n = 1›) (simp_all add: ac_simps)

    have eq_3: "x$(i*5) + x$(i*5+1) = 0" if "i∈{0..<n}" for i
      using that d_eq_0[of "4*i+1"] unfolding d_def d2_def
      by (auto, metis add.right_neutral div_less div_mult_self1 mult.commute
one_less_numeral_iff
       plus_1_eq_Suc semiring_norm(76) zero_neq_numeral)
```

```
    have eq_4: "(if i<n-1 then x$(i*5+4) else 0) + x$(i*5+2) = 0" if "i∈{0..<n}"
for i
    proof -
      have **: "(4 * i + 2) div 4 = i" by auto
      have *: "(if (4 * i + 2) div 4 < n - 1 then x $ ((4 * i + 2) div
4 * 5 + 2) else 0) +
        x $ ((4 * i + 2) div 4 * 5 + 3) = (if i<n-1 then x$(i*5+2) else
0) + x$(i*5+3)"
        unfolding ** by auto
      show ?thesis using that d_eq_0[of "4*i+2"] unfolding d_def d3_def
*
      by (smt (verit, ccfv_threshold) <0 < length a> add.right_neutral
add_self_div_2
        add_self_mod_2 atLeastLessThan_iff bits_one_mod_two_eq_one div_mult2_eq

        div_mult_self4 mod_mult2_eq mod_mult_self3 mult.commute mult_2
mult_is_0 n_def
        nat_1_add_1 nat_mod_lem neq0_conv numeral_Bit0 one_div_two_eq_zero)
    qed

    have eq_5: "x$(i*5) + x$(i*5+2) + x$(i*5+3) = 0" if "i∈{0..<n}" for
i
      using that d_eq_0[of "4*i+3"] unfolding d_def d4_def by auto

    have eq_3': "x$(i*5) = - x$(i*5+1)" if "i∈{0..<n}" for i
      using eq_3[OF that] by auto

    have eq_4': "x$(i*5+4) = - x$(i*5+2)" if "i∈{0..<n-1}" for i
      using that eq_4[of i] by force


    have eq_1': "x$(i*5) + (if i<n-1 then x$(i*5+4) else 0) =
      x$((i-1)*5) + x$((i-1)*5+4)" if "i∈{1..<n}" for i
    proof -
      have *: "i - 1 ∈ {0..<n}" using that by auto
      have **: "i-1 ∈{0..<n-1}" using that by auto
      then show ?thesis using eq_1[OF that]
      by (subst eq_3'[OF *], subst eq_4'[OF **], auto)
    qed
```

This defines the weight of the solution, since $x_{i,0} + x_{i,4}$ does not depend on the index i. We take $x_{n-1,0} + x_{n-1,4}$, since we omitted the last element (thus $x_{n-1,4} = 0$) to ensure that the weight has absolute value at most 1.

```
    define w where "w = x$((n-1)*5)"

    have w_eq_02: "w = x$(i*5) + (if i<n-1 then x$(i*5+4) else 0)" if
"i∈{0..<n}" for i
    proof -
```

```
      have "i≤n-1" using that by auto
      then show ?thesis
      proof (induct rule: Nat.inc_induct)
        case (step m)
        then show ?case unfolding w_def using eq_1'[of "Suc m"] by auto
      qed (unfold w_def, auto)
    qed

    have "|w| ≤ 1"  using xi_le_1[of "(n-1)*5"] ‹n > 0›
      unfolding w_def dim_vec_x_5n by auto
```

Rule out the all zero solution.

```
    moreover have "w≠0"
    proof (rule ccontr)
      assume "¬ w ≠ 0"
      then have "w = 0" by blast
      then have "x$((n-1)*5) = 0" unfolding w_def by auto
      have zero_eq_min2: "x$(i*5) = - (if i<n-1 then x$(i*5+4) else 0)"
if "i∈{0..<n}" for i
        using w_eq_02[OF that] ‹w=0› by auto
      have two0_plus_4: "2 * x$(i*5) + x$(i*5+3) = 0" if "i ∈ {0..<n-1}"
for i
        using that eq_5[of i] eq_4'[OF that] zero_eq_min2 by auto

      have p_zero: "x$(i*5) = 0" if "i<n"  for i
        using that
      proof (cases "i=n-1")
        case True
        then show ?thesis using that ‹x$((n-1)*5) = 0› by auto
      next
        case False
        then have "i ∈ {0..<n - 1}" using that by auto
        show ?thesis
        proof (rule ccontr)
          assume "x $ (i * 5) ≠ 0"
          then have "|2 * x $ (i * 5) + x $ (i * 5 + 3)|≥1"
            using xi_le_1[OF prec_0[where i=i]] xi_le_1[OF prec_i[where
i=i and j=3]]
            by (auto simp add: ‹i<n›)
          then show False using two0_plus_4[OF ‹i ∈ {0..<n - 1}›] by
auto
        qed
      qed
      have "x$j = 0" if "j<5*n" "j mod 5 = 0" for j
      proof -
        from ‹j mod 5 = 0› ‹j < 5 * n› obtain i where "i*5 = j" "i<n"
          by auto
        then show ?thesis unfolding ‹i*5 = j›[symmetric] using p_zero[OF
‹i<n›] by auto
```

```
    qed

    moreover have p_one: "x$(i*5+1) = 0" if "i<n"  for i
      using p_zero[OF that] eq_3' that by auto
    then have "x$j = 0" if "j<5*n" "j mod 5 = 1" for j
    proof -
      obtain i where "i*5+1 = j" "i<n" using <j<5*n> <j mod 5 = 1>

      by (metis add.commute less_mult_imp_div_less mod_mult_div_eq mult.commute)
      then show ?thesis unfolding <i*5+1 = j>[symmetric] using p_one[OF
<i<n>] by auto
    qed

    moreover have p_four: "x$(i*5+4) = 0" if "i<n-1" for i
      using w_eq_02[of i] that p_zero unfolding <w=0> by auto
    then have "x$j = 0" if "j<5*n-1" "j mod 5 = 4" for j
    proof -
      obtain i where "i*5+4 = j" "i<n-1" using <j<5*n-1> <j mod 5 =
4>
      by (metis add.assoc add.commute less_Suc_eq less_diff_conv less_mult_imp_div_less

        mod_mult_div_eq mult.commute mult_Suc_right not_less_eq numeral_nat(3)
plus_1_eq_Suc)
      then show ?thesis unfolding <i*5+4 = j>[symmetric] using p_four
by auto
    qed

    moreover have p_two: "x$(i*5+2) = 0" if "i<n"  for i
    proof (cases "i<n-1")
      case True
      then show ?thesis using p_four[OF <i<n-1>] eq_4' by auto
    next
      case False
      then have "i=n-1" using that by auto
      show ?thesis using eq_4[of "n-1"] unfolding <i=n-1> using <n>0>
by auto
    qed
    then have "x$j = 0" if "j<5*n" "j mod 5 = 2" for j
    proof -
      obtain i where "i*5+2 = j" "i<n" using <j<5*n> <j mod 5 = 2>

      by (metis add.commute less_mult_imp_div_less mod_mult_div_eq mult.commute)
      then show ?thesis unfolding <i*5+2 = j>[symmetric] using p_two[OF
<i<n>] by auto
    qed

    moreover have p_three: "x$(i*5+3) = 0" if "i<n"  for i
      using eq_5[of i] that p_two[OF that] p_zero[OF that] by auto
    then have "x$j = 0" if "j<5*n" "j mod 5 = 3" for j
```

```
      proof -
        obtain i where "i*5+3 = j" "i<n" using ‹j<5*n› ‹j mod 5 = 3›

        by (metis add.commute less_mult_imp_div_less mod_mult_div_eq mult.commute)
        then show ?thesis unfolding ‹i*5+3 = j›[symmetric] using p_three[OF
‹i<n›] by auto
      qed
      ultimately have "x$j = 0" if "j<5*n-1" for j
      using mod_5_choices[of j "(λj. x $ j = 0)"] that by auto
      then have "x = 0v (dim_vec x)" unfolding dim_vec_x_5n[symmetric]
by auto
      then show False using 1(4) by auto
    qed
```

Then we can deduce the wanted property for both cases $w = 1$ and $w = -1$.
The only differences between the two cases is the switch of signs.

```
    ultimately have "w=1 ∨ w = -1" by auto
    then consider (pos) "w=1" | (neg) "w=-1" by blast
    then show ?thesis
    proof cases
      case pos
      have 01:"(x$(i*5) = 0 ∧ x$(i*5+4) = 1) ∨ (x$(i*5) = 1 ∧ x$(i*5+4)
= 0)" if "i<n-1" for i
      proof -
        have "i * 5 < dim_vec x" using that ‹n>0› unfolding dim_vec_x_5n
by auto
        then have "x$(i*5) ∈ {-1,0,1}" using xi_le_1[of "i*5"]
          by auto
        then consider (minus) "x$(i*5) = -1" | (zero) "x$(i*5) = 0" |
(plus) "x$(i*5) = 1"
          by blast
        then show ?thesis
        proof (cases)
          case minus
          then have "2 = x $ (i * 5 + 4)" using w_eq_02[of i] that ‹w=1›
by auto
          then have False using xi_le_1[of "i*5+4"] that unfolding dim_vec_x_5n
            by linarith
          then show ?thesis by auto
        next
          case zero
          then have "x $ (i * 5 + 4) = 1" using w_eq_02[of i] that un-
folding ‹w=1› by auto
          then show ?thesis using zero by auto
        next
          case plus
          then have "x $ (i * 5 + 4) = 0" using w_eq_02[of i] that un-
folding ‹w=1› by auto
          then show ?thesis using plus by auto
```

```
    qed
  qed

  have "n-1∈I" using w_def unfolding ‹w=1› I_def n_def[symmetric]

    by (auto simp add: ‹n>0› mult.commute)

  have I_0: "x$(i*5) = 1" if "i∈I" for i
  proof (cases "i<n-1")
    case True
    then show ?thesis using 01[OF True] that unfolding I_def n_def[symmetric]

    by (simp add: mult.commute)
  next
    case False
    then have "i=n-1" using that unfolding I_def n_def[symmetric]
by auto
    then show ?thesis using w_def ‹w=1› by auto
  qed
  have I_2: "x$(i*5+2) = 0" if "i∈I" for i
  proof (cases "i<n-1")
    case True
    then have "x $ (i * 5 + 4) = 0" using 01[OF True] that unfold-
ing I_def n_def[symmetric]
    by (simp add: mult.commute)
    then show ?thesis using eq_4[of i] that True unfolding I_def
n_def[symmetric] by auto
  next
    case False
    then have "i=n-1" using that unfolding I_def n_def[symmetric]
by auto
    then have "x$(i*5+1) = -1" using I_0[OF that] eq_3'[of i] by
(simp add: ‹0 < n›)
    moreover have "1 + x $ (i * 5 + 1) + x $ (i * 5 + 2) = 0"
      using eq_2 w_eq_02[of 0] unfolding ‹i=n-1›
      by (metis ‹0 < n› add_cancel_right_left atLeast0LessThan
        lessThan_iff mult_zero_left pos)
    ultimately show ?thesis by auto
  qed
  have notI_0: "x$(i*5) = 0" if "i∈{0..<n} - I" for i
  proof (cases "i<n-1")
    case True
    then show ?thesis using 01[OF True] that unfolding I_def n_def[symmetric]

    by (simp add: mult.commute)
  next
    case False
    then have "i=n-1" using that unfolding I_def n_def[symmetric]
by auto
```

```
      then show ?thesis using <n-1∈I> that by auto
    qed
    have notI_2: "x$(i*5+2) = -1" if "i∈{0..<n} - I" for i
    proof (cases "i<n-1")
      case True
      then have "x $ (i * 5 + 4) = 1" using 01[OF True] that unfold-
ing I_def n_def[symmetric]
      by (simp add: mult.commute)
      then show ?thesis using eq_4[of i] that True unfolding I_def
n_def[symmetric] by auto
    next
      case False
      then have "i=n-1" using that unfolding I_def n_def[symmetric]
by auto
      then show ?thesis using <n-1∈I> that by auto
    qed

    have "0 = (∑ i∈I. (x $ (i * 5) + x $ (i * 5 + 2)) * a ! i) +
      (∑ i∈{0..<n} - I. (x $ (i * 5) + x $ (i * 5 + 2)) * a ! i)"
      unfolding eq_0[symmetric]
      by (subst sum.subset_diff[of I]) (auto simp add: I_def n_def lessThan_atLeast0)
    moreover have "(x $ (i * 5) + x $ (i * 5 + 2)) = 1" if "i∈I" for
i
      using I_0 I_2 that by auto
    moreover have "(x $ (i * 5) + x $ (i * 5 + 2)) = -1" if "i∈{0..<n}
- I" for i
      using notI_0 notI_2 that by auto
    ultimately have "0 = (∑ i∈I. a ! i) - (∑ i∈{0..<n} - I. a ! i)"

      by (auto simp add: sum_negf)
    then show ?thesis unfolding n_def by auto
  next
    case neg
    have 01:"(x$(i*5) = 0 ∧ x$(i*5+4) = -1) ∨ (x$(i*5) = -1 ∧ x$(i*5+4)
= 0)" if "i<n-1" for i
    proof -
      have "i * 5 < dim_vec x" using that <n>0> unfolding dim_vec_x_5n
by auto
      then have "x$(i*5) ∈ {-1,0,1}" using xi_le_1[of "i*5"]
        by auto
      then consider (minus) "x$(i*5) = -1" | (zero) "x$(i*5) = 0" |
(plus) "x$(i*5) = 1"
        by blast
      then show ?thesis
      proof (cases)
        case plus
        then have "-2 = x $ (i * 5 + 4)" using w_eq_02[of i] that <w=-1>
by force
        then have False using xi_le_1[of "i*5+4"] that unfolding dim_vec_x_5n
```

```
          by linarith
        then show ?thesis by auto
      next
        case zero
        then have "x $ (i * 5 + 4) = -1" using w_eq_02[of i] that un-
folding <w=-1> by auto
        then show ?thesis using zero by auto
      next
        case minus
        then have "x $ (i * 5 + 4) = 0" using w_eq_02[of i] that un-
folding <w=-1> by auto
        then show ?thesis using minus by auto
      qed
    qed

    have "n-1∈I" using w_def unfolding <w=-1> I_def n_def[symmetric]

      by (auto simp add: <n>0> mult.commute)

    have I_0: "x$(i*5) = -1" if "i∈I" for i
    proof (cases "i<n-1")
      case True
      then show ?thesis using 01[OF True] that unfolding I_def n_def[symmetric]

      by (simp add: mult.commute)
    next
      case False
      then have "i=n-1" using that unfolding I_def n_def[symmetric]
by auto
      then show ?thesis using w_def <w=-1> by auto
    qed
    have I_2: "x$(i*5+2) = 0" if "i∈I" for i
    proof (cases "i<n-1")
      case True
      then have "x $ (i * 5 + 4) = 0" using 01[OF True] that unfold-
ing I_def n_def[symmetric]
      by (simp add: mult.commute)
      then show ?thesis using eq_4[of i] that True unfolding I_def
n_def[symmetric] by auto
    next
      case False
      then have "i=n-1" using that unfolding I_def n_def[symmetric]
by auto
      then have "x$(i*5+1) = 1" using I_0[OF that] eq_3'[of i] by (simp
add: <0 < n>)
      moreover have "-1 + x $ (i * 5 + 1) + x $ (i * 5 + 2) = 0"
        using eq_2 w_eq_02[of 0] unfolding <i=n-1>
        by (metis <0 < n> add_cancel_right_left lessThan_atLeast0 lessThan_iff
```

```
          mult_zero_left neg)
        ultimately show ?thesis by auto
      qed
      have notI_0: "x$(i*5) = 0" if "i∈{0..<n} - I" for i
      proof (cases "i<n-1")
        case True
        then show ?thesis using 01[OF True] that unfolding I_def n_def[symmetric]

        by (simp add: mult.commute)
      next
        case False
        then have "i=n-1" using that unfolding I_def n_def[symmetric]
by auto
        then show ?thesis using ‹n-1∈I› that by auto
      qed
      have notI_2: "x$(i*5+2) = 1" if "i∈{0..<n} - I" for i
      proof (cases "i<n-1")
        case True
        then have "x $ (i * 5 + 4) = -1" using 01[OF True] that unfold-
ing I_def n_def[symmetric]
        by (simp add: mult.commute)
        then show ?thesis using eq_4[of i] that True unfolding I_def
n_def[symmetric] by auto
      next
        case False
        then have "i=n-1" using that unfolding I_def n_def[symmetric]
by auto
        then show ?thesis using ‹n-1∈I› that by auto
      qed

      have "0 = (∑i∈I. (x $ (i * 5) + x $ (i * 5 + 2)) * a ! i) +
        (∑i∈{0..<n} - I. (x $ (i * 5) + x $ (i * 5 + 2)) * a ! i)"
        unfolding eq_0[symmetric]
        by (subst sum.subset_diff[of I]) (auto simp add: I_def n_def lessThan_atLeast0)
      moreover have "(x $ (i * 5) + x $ (i * 5 + 2)) = -1" if "i∈I" for
i
        using I_0 I_2 that by auto
      moreover have "(x $ (i * 5) + x $ (i * 5 + 2)) = 1" if "i∈{0..<n}
- I" for i
        using notI_0 notI_2 that by auto
      ultimately have "0 = (∑i∈I. a ! i) - (∑i∈{0..<n} - I. a ! i)"

        by (auto simp add: sum_negf)
      then show ?thesis unfolding n_def by auto
    qed
  qed
  ultimately show ?case using ‹length a > 0› by auto
qed
```

The Gap-SVP is NP-hard.

```
lemma "is_reduction reduce_bhle_partition partition_problem_nonzero bhle"
unfolding is_reduction_def
proof (safe, goal_cases)
  case (1 a)
  then show ?case using well_defined_reduction_subset_sum by auto
next
  case (2 a)
  then show ?case using NP_hardness_reduction_subset_sum by auto
qed

end
theory SVP_vec

imports
  BHLE
begin
```

13 SVP in ℓ_∞

The reduction of SVP to BHLE in l_∞ norm

The shortest vector problem.

```
definition is_shortest_vec :: "int_lattice  ⇒ int vec ⇒ bool" where
  "is_shortest_vec L v ≡ (is_lattice L) ∧ (∀x∈L. ‖x‖∞ ≥ ‖v‖∞ ∧ v∈L)
"
```

The decision problem associated with solving SVP exactly.

```
definition gap_svp :: "(int_lattice × int) set" where
  "gap_svp ≡ {(L, r). (is_lattice L) ∧ (dim_lattice L ≥ 2) ∧
      (∃v∈L. ‖v‖∞ ≤ r ∧ v ≠ 0v (dim_vec v))}"
```

Generate a lattice to solve SVP for reduction.

Here, the factor K'' from the paper [7] was changed to be $2 \cdot \mathbf{k} \cdot (k+1) \cdot \sum_i |\mathbf{a_i}|$ in order for the proofs to finish.

```
definition gen_svp_basis :: "int vec ⇒ int ⇒ int mat" where
  "gen_svp_basis a k = mat (dim_vec a + 1) (dim_vec a + 1)
    (λ (i, j). if (i < dim_vec a) ∧ (j< dim_vec a) then (if i = j then
1 else 0)
      else (if (i < dim_vec a) ∧ (j ≥ dim_vec a) then 0
      else (if (i ≥ dim_vec a) ∧ (j < dim_vec a) then (k+1) * (a $ j)
      else 2*k*(k+1)* (∑ i ∈ {0 ..< dim_vec a}. |a $ i|) +1 )))"
```

Reduction SVP to bounded homogeneous linear equation problem in ℓ_∞ norm.

```
definition reduce_svp_bhle:: "(int vec × int) ⇒ (int_lattice × int)"
where
```

```
  "reduce_svp_bhle ≡ (λ (a, k). (gen_lattice (gen_svp_basis a k), k))"
```

Lemmas for proof

```
lemma gen_svp_basis_mult:
  assumes "dim_vec z = dim_vec a + 1"
  shows "(gen_svp_basis a k) *_v z = vec (dim_vec a + 1)
         (λi. if i < dim_vec a then z$i else (k+1) * (∑ i ∈ {0 ..< dim_vec
a}. z $ i * a $ i) +
              (2*k*(k+1)* (∑ i ∈ {0 ..< dim_vec a}. |a $ i|) +1) * (z$(dim_vec
a)))"
using assms proof (subst vec_eq_iff, safe, goal_cases)
case 1
  then show ?case using assms unfolding gen_svp_basis_def by auto
next
case (2 i)
  then show ?case proof (cases "i<dim_vec a")
  case True
    have "{0..<dim_vec a} = insert i {0..<dim_vec a}" using True by auto
    then have "(∑ia = 0..<dim_vec a. (if i = ia then 1 else 0) * z $
ia) =
      (∑ia ∈ insert i {0..<dim_vec a}. (if i = ia then 1 else 0) * z
$ ia)" by auto
    also have "... = z$i" by (subst sum.insert_remove, auto)
    finally have "(∑ia = 0..<dim_vec a. (if i = ia then 1 else 0) * z
$ ia) = z $ i"
      by auto
    then show ?thesis unfolding mult_mat_vec_def gen_svp_basis_def scalar_prod_def

      using True assms by auto
  next
  case False
    then have "i = dim_vec a" using 2 by auto
    then show ?thesis unfolding gen_svp_basis_def using assms
      by (auto simp add: scalar_prod_def sum_distrib_left mult.commute
mult.left_commute)
  qed
qed

lemma gen_svp_basis_mult_real:
  assumes "dim_vec z = dim_vec a + 1"
  shows "real_of_int_mat (gen_svp_basis a k) *_v z = vec (dim_vec a + 1)

         (λi. if i < dim_vec a then z$i else (k+1) * (∑ i ∈ {0 ..< dim_vec
a}. z $ i * a $ i) +
              (2*k*(k+1)* (∑ i ∈ {0 ..< dim_vec a}. |a $ i|) +1) * (z$(dim_vec
a)))"
using assms proof (subst vec_eq_iff, safe, goal_cases)
case 1
  then show ?case using assms unfolding gen_svp_basis_def by auto
```

```
next
case (2 i)
  then show ?case proof (cases "i<dim_vec a")
  case True
    have "{0..<dim_vec a} = insert i {0..<dim_vec a}" using True by auto
    then have "(∑ ia = 0..<dim_vec a. (if i = ia then 1 else 0) * z $
ia) =
      (∑ ia ∈ insert i {0..<dim_vec a}. (if i = ia then 1 else 0) * z
$ ia)" by auto
    also have "... = z$i" by (subst sum.insert_remove, auto)
    finally have "(∑ ia = 0..<dim_vec a. (if i = ia then 1 else 0) * z
$ ia) = z $ i"
      by auto
    then have "(∑ ia = 0..<dim_vec a. real_of_int (if i = ia then 1 else
0) * z $ ia) = z $ i"
      by (smt (verit, best) of_int_hom.hom_one of_int_hom.hom_zero sum.cong)
    then show ?thesis unfolding mult_mat_vec_def gen_svp_basis_def scalar_prod_def

      using True assms by auto
  next
  case False
    then have "i = dim_vec a" using 2 by auto
    then show ?thesis unfolding gen_svp_basis_def using assms
      by (auto simp add: scalar_prod_def sum_distrib_left mult.commute
mult.left_commute)
  qed
qed

lemma gen_svp_basis_mult_last:
  assumes "dim_vec z = dim_vec a + 1"
  shows "((gen_svp_basis a k) *v z) $ (dim_vec a) =
         (k+1) * (∑ i ∈ {0 ..< dim_vec a}. z $ i * a $ i) +
              (2*k*(k+1)* (∑ i ∈ {0 ..< dim_vec a}. |a $ i|) +1) * (z$(dim_vec
a))"
using gen_svp_basis_mult[OF assms] by auto
```

The set generated by *gen_svp_basis* is indeed linearly independent.

```
lemma is_indep_gen_svp_basis:
  assumes "k>0"
  shows "is_indep (gen_svp_basis a k)"
unfolding is_indep_int_def
proof (safe, goal_cases)
case (1 z)
  have dim_row_dim_vec: "dim_row (gen_svp_basis a k) = dim_vec z"
    using 1 unfolding gen_svp_basis_def by auto
  then have suc_dim_a_dim_z: "dim_vec z = dim_vec a + 1" unfolding gen_svp_basis_def
by auto
  have alt1: "(real_of_int_mat (gen_svp_basis a k) *v z) $ i = 0" if "i<
dim_vec a +1"for i
```

```
    using 1(1) that unfolding gen_svp_basis_def by auto
  have z_upto_last: "z$i = 0" if "i < dim_vec a" for i
  proof -
    have elem: "(real_of_int_mat (gen_svp_basis a k) *v z) $ i = z $ i"

      using gen_svp_basis_mult_real[OF suc_dim_a_dim_z] that by auto
    show ?thesis by (subst elem[symmetric]) (use alt1[of i] that in ‹auto›)

  qed
  moreover have "z $ (dim_vec a) = 0"
  proof -
    have "0 = (real_of_int_mat (gen_svp_basis a k) *v z) $ (dim_vec a)"
using alt1 by auto
    also have "... = (real_of_int k + 1) * (∑i = 0..<dim_vec a. z $ i
* real_of_int (a $ i)) +
      (2 * real_of_int k * (real_of_int k + 1) * (∑x = 0..<dim_vec a.
real_of_int (|a $ x|)) + 1) *
      z $ dim_vec a"
      using gen_svp_basis_mult_real[OF suc_dim_a_dim_z] by auto
    also have "... =  real_of_int (2 * k * (k + 1) *  (∑x = 0..<dim_vec
a. |a $ x|)  + 1 ) * (z$dim_vec a)"
      using suc_dim_a_dim_z using z_upto_last by auto
    finally have "0 = real_of_int (2 * k * (k + 1) *  (∑x = 0..<dim_vec
a. |a $ x|) + 1 ) *
                      (z$dim_vec a)" by blast
    moreover have "real_of_int (2 * k * (k + 1) *  (∑x = 0..<dim_vec
a. |a $ x|) + 1 ) ≠ 0"
    proof (rule ccontr, goal_cases)
    case 1
      then have eq: "real_of_int (∑x = 0..<dim_vec a. |a $ x|) = - 1
/ real_of_int (k * (k + 1))"
        using assms
        by (smt (verit, best) mult_minus_right of_int_hom.hom_0 pos_zmult_eq_1_iff

         pos_zmult_eq_1_iff_lemma)
      have "k≥1" using assms by auto
      have "real_of_int (∑x = 0..<dim_vec a. |a $ x|) ∈ ℤ" by auto
      moreover have "- 1 / real_of_int (k * (k + 1)) ∉ ℤ" using ‹k≥1›

      by (smt (verit, del_insts) "1" linordered_semiring_strict_class.mult_pos_pos

        mult_minus_right of_int_hom.hom_0 pos_zmult_eq_1_iff)
      ultimately show False using eq by auto
    qed

    ultimately show ?thesis by auto
  qed
  ultimately have "z$i =  0" if "i < dim_vec z" for i using that suc_dim_a_dim_z
    by (cases ‹dim_vec a = i›) simp_all
```

```
  then show ?case
    by auto
qed



lemma insert_set_comprehension:
  "insert (f x) {f i |i. i<(x::nat)} = {f i | i. i<x+1}"
using less_SucE by fastforce

lemma bhle_k_pos:
  assumes "(a,k) ∈ bhle"
  shows "k>0"
using assms unfolding bhle_def
proof (safe, goal_cases)
case (1 v)
  have "∃i<dim_vec v. |v $ i| > 0" using 1 by auto
  then have "‖v‖∞ > 0" unfolding linf_norm_vec_Max using 1 by (subst
Max_gr_iff, auto)
  then show ?thesis using 1 by auto
qed

lemma svp_k_pos:
  assumes "reduce_svp_bhle (a, k) ∈ gap_svp"
  shows "k>0"
proof -
  obtain v where v_in_lattice: "v∈gen_lattice (gen_svp_basis a k)"
    and infnorm_v: "‖v‖∞ ≤ k"
    and v_nonzero:  "v ≠ 0v (dim_vec v)"
    using assms unfolding reduce_svp_bhle_def gap_svp_def by force
  have "∃ i < dim_vec v. |v $ i| > 0" using v_nonzero by auto
  then have "‖v‖∞ > 0" unfolding linf_norm_vec_Max by (subst Max_gr_iff,
auto)
  then show ?thesis using infnorm_v by auto
qed



lemma svp_dim_vec_a:
  assumes "reduce_svp_bhle (a, k) ∈ gap_svp"
  shows "dim_vec a > 0"
proof -
  have "2 ≤ dim_lattice (gen_lattice (gen_svp_basis a k))"
    using assms unfolding reduce_svp_bhle_def gap_svp_def by auto
  then have "2 ≤ dim_col (gen_svp_basis a k)"
    using dim_lattice_gen_lattice[of "gen_svp_basis a k",OF is_indep_gen_svp_basis]
      svp_k_pos[OF assms] by auto
  then show ?thesis unfolding gen_svp_basis_def by auto
qed
```

```
lemma bhle_dim_vec_a:
  assumes "(a, k) ∈ bhle"
  shows "dim_vec a > 0"
using assms unfolding bhle_def by auto


lemma first_lt_second:
  assumes "k>0" and z_le_k:"⋀i. i< dim_vec a ⟹ |z $ i| ≤ k"
  shows "2 * |(k + 1) * (∑ i = 0..<dim_vec a. z $ i * a $ i)|
      < (|2 * k * (k + 1) * (∑ i = 0..<dim_vec a. |a $ i|) + 1|::int)"
proof -
  have take_k1_out: "|(k + 1) * (∑ i = 0..<dim_vec a. z $ i * a $ i)|
=
    (k+1) * |∑ i=0..<dim_vec a. z $ i * a $ i|"
    using ‹k>0› by (smt (verit, best) mult_minus_right mult_nonneg_nonneg)
  have "|∑ i = 0..<dim_vec a. z $ i * a $ i| ≤ (∑ i = 0..<dim_vec a.
|z $ i * a $ i|)"
    by (subst sum_abs[of "(λi. z$i * a$i)" "{0..<dim_vec a}"],simp)
  also have "... = (∑ i=0..<dim_vec a. |z $ i| * |a $ i|)"
    by (meson abs_mult)
  also have "... ≤ (∑ i=0..<dim_vec a.  k * |a $ i|)"
    by (subst sum_mono) (auto simp add: z_le_k mult_right_mono)
  also have "... = k * (∑ i=0..<dim_vec a. |a $ i|)"
  by (metis mult_hom.hom_sum)
  finally have "|(∑ i=0..<dim_vec a. z $ i * a $ i)| ≤ k * (∑ i=0..<dim_vec
a. |a $ i|)"
    by blast
  then show ?thesis using take_k1_out using ‹0 < k› by auto
qed
```

Well-definedness of reduction function

```
lemma well_defined_reduction_svp:
  assumes "(a,k) ∈ bhle"
  shows "reduce_svp_bhle (a,k) ∈ gap_svp"
using assms unfolding reduce_svp_bhle_def gap_svp_def bhle_def
proof (safe, goal_cases)
  case (1 x)
  have "k>0" using assms bhle_k_pos by auto
  then show ?case using is_lattice_gen_lattice is_indep_gen_svp_basis
by auto
next
  case (2 x)
  have "dim_lattice (gen_lattice (gen_svp_basis a k)) = dim_col (gen_svp_basis
a k)"
    using dim_lattice_gen_lattice assms bhle_k_pos is_indep_gen_svp_basis
by presburger
  also have "... =  dim_vec a + 1" unfolding gen_svp_basis_def by auto
  also have "... ≥ 2" using bhle_dim_vec_a[OF assms] by auto
  finally show ?case by auto
```

```
next
  case (3 x)
  let ?x = "vec (dim_vec x + 1) (λi. if i< dim_vec x then x$i else 0)"
  define v where "v = (gen_svp_basis a k) *_v ?x"
  have eigen_v: "v = ?x" unfolding v_def
  apply (subst gen_svp_basis_mult[where a= a and k=k and z=" ?x"], auto
simp add: 3(2) comp_def)
  apply (subst vec_eq_iff, auto simp add: 3(1) scalar_prod_def)
  proof (goal_cases one two)
    case (one i)
    then show ?case by (auto simp add: comp_def 3(2))
  next
    case (two i)
    have "(∑ i = 0..<dim_vec a. (x $ i) * (a $ i)) = 0"
      using 3 unfolding scalar_prod_def
      by (smt (verit) mult.commute of_int_hom.hom_zero of_int_mult of_int_sum
sum.cong)
    then show ?case using 3 by auto
  qed
  have "dim_vec ?x = dim_col (gen_svp_basis a k)"
    using 3(2) unfolding gen_svp_basis_def by auto
  then have "v ∈ gen_lattice (gen_svp_basis a k)"
    unfolding v_def gen_lattice_def by auto
  moreover have "‖v‖_∞ ≤ k"
  proof -
    have "‖v‖_∞ ≤ ‖x‖_∞" unfolding eigen_v linf_norm_vec_Max
    proof (rule Max.boundedI, goal_cases _ _ three)
    case (three b)
      have dim_x_nonzero: "dim_vec x ≠ 0" using 3(3) 3(2) by auto
      then have nonempty: "(∃a∈{|x $ i| |i. i < dim_vec x}. 0 ≤ a)"
        using abs_ge_zero by blast
      have " |x $ i| ≤ Max (insert 0 {|x $ j| |j. j < dim_vec x})"
        if "i < dim_vec x" for i
      using that by (subst Max_ge, auto)
      moreover have "0 ≤ Max (insert 0 {|x $ i| |i. i < dim_vec x})"
using 3 nonempty
        by (subst Max_ge_iff, auto)
      ultimately show ?case using three by auto
    qed auto
    then show ?thesis using 3 by auto
  qed
  moreover have "v ≠ 0_v (dim_vec v)" using 3(3)
  proof (safe)
    assume "0 < dim_vec a"
    assume "v = 0_v (dim_vec v)"
    have fst: "x ≠ 0_v (dim_vec x)" using 3(4) by blast
    have snd: "?x = 0_v (dim_vec v)" using ‹v = 0_v (dim_vec v)›  eigen_v
by auto
    have "x$i = 0" if "i< dim_vec x" for i using snd
```

```
    by (smt (z3) add.commute dim_vec eigen_v index_map_vec(2) index_vec
index_zero_vec(1)
      of_int_eq_iff of_int_hom.hom_zero that
      trans_less_add2)
  then show False using fst by auto
  qed
  ultimately show ?case by blast
qed
```

NP-hardness of reduction function

```
lemma NP_hardness_reduction_svp:
  assumes "reduce_svp_bhle (a,k) ∈ gap_svp"
  shows "(a,k) ∈ bhle"
proof (cases "(∑ i<dim_vec a. a$i) = 0")
  case True
  have a_nonempty: "dim_vec a > 0" using svp_dim_vec_a[OF assms] by auto
  have "k>0" using svp_k_pos[OF assms] by auto
  define x where "x = vec (dim_vec a) (λi. k)"
  have "a · x = 0" unfolding x_def scalar_prod_def
  by (auto simp add: True sum_distrib_right[symmetric])
     (metis True lessThan_atLeast0)
  moreover have "dim_vec x = dim_vec a" unfolding x_def by auto
  moreover have "x ≠ 0_v (dim_vec x)"
  proof -
    have "∃i< dim_vec x. x $ i ≠ 0" unfolding x_def using ‹k>0› a_nonempty
by auto
    then show ?thesis using vec_eq_iff[of "x" "0_v (dim_vec x)"] by auto
  qed
  moreover have "real_of_int (‖x‖∞) ≤ k"
  proof -
    have "vec (dim_vec a) (λi. k) $ j = k" if "j < dim_vec a" for j us-
ing that by auto
    then have "Max {|vec (dim_vec a) (λi. k) $ i| |i. i < dim_vec a} =

                Max {|k| |i. i < dim_vec a}" by metis
    also have "... = Max {k}" using ‹k>0› a_nonempty
      by (smt (verit, best) Collect_cong singleton_conv)
    also have "... = k" by auto
    finally have "Max {|vec (dim_vec a) (λi. k) $ i| |i. i < dim_vec a}
= k" by blast
    then show ?thesis unfolding x_def linf_norm_vec_Max using ‹k>0› by
auto
  qed
  ultimately show ?thesis using assms unfolding reduce_svp_bhle_def gap_svp_def
bhle_def
    by fastforce
next
  case False
  show ?thesis using assms unfolding reduce_svp_bhle_def gap_svp_def
```

```
bhle_def
  proof (safe, goal_cases)
    case (1 v)
    have a_nonempty: "dim_vec a > 0" using svp_dim_vec_a[OF assms] by
auto
    have "k>0"  unfolding linf_norm_vec_Max
    proof -
      have "∃i<dim_vec v. |v $ i| > 0" using 1 by auto
      then have "‖v‖∞ > 0" unfolding linf_norm_vec_Max by (subst Max_gr_iff,
auto)
      then show ?thesis using 1 by auto
    qed
    then have "k≥1" by auto
    obtain z where z_def:
      "v = (gen_svp_basis a k) *v  z"
      "dim_vec z = dim_vec a + 1"
      using 1 unfolding gen_lattice_def gen_svp_basis_def by auto
    have dim_v_dim_a:"dim_vec v = dim_vec a + 1"
      using 1 unfolding gen_lattice_def gen_svp_basis_def by auto
    define x where "x = vec (dim_vec a) (($) z)"
    have v_eq_z_upto_last: "v $ i = z $ i" if "i< dim_vec a" for i
      by (subst z_def) (subst gen_svp_basis_mult; use that z_def in ‹auto›)
    have v_le_k: "|v $ i| ≤ k" if "i< dim_vec a + 1" for i
      using 1 dim_v_dim_a that unfolding linf_norm_vec_Max
      using Max_le_iff[of "{|v $ i| |i. i < dim_vec v}" k]
      by fastforce
    then have z_le_k: "|z $ i| ≤ k" if "i< dim_vec a" for i
      using v_eq_z_upto_last[OF that] that
      by (metis less_add_one less_trans)

    have v_last_zero: "v$(dim_vec a) = 0"
    proof (rule ccontr)
      assume ccontr_assms:"v $ dim_vec a ≠ 0"
      have v_last: "v$(dim_vec a) =
        (k+1) * (∑i = 0..<dim_vec a. z $ i * a $ i) +
        (2*k*(k+1) * (∑x = 0..<dim_vec a. |a $ x|) + 1) * (z $ dim_vec
a) "
        (is "v$(dim_vec a) = ?first + ?second")
        by (auto simp: z_def gen_svp_basis_mult_last)
      then have "?first ≠ 0 ∨ ?second ≠ 0"
        using ccontr_assms by auto
      then consider
        (first) "?first ≠ 0 ∧ ?second = 0" |
        (second) "?second ≠ 0 ∧ ?first = 0" |
        (both) "?first ≠ 0 ∧ ?second ≠ 0"
        by blast
      then show False
      proof (cases)
        case first
```

```
      then have gr_1: "|∑ i = 0..<dim_vec a. z $ i * a $ i| ≥ 1"
        using ‹k>0› by auto
      have "|v$ dim_vec a| = |?first|" using first v_last by auto
      also have "... = (k+1) * |∑ i = 0..<dim_vec a. z $ i * a $ i|"
        using ‹k>0›
        by (smt (z3) mult_le_0_iff mult_minus_right)
      also have "... > k" using first gr_1 ‹k>0›
      by (smt (verit, best) mult_le_cancel_left1)
      finally have "|v$ dim_vec a| > k" by auto
      then show ?thesis using v_le_k[of "dim_vec a"] by auto
    next
      case second
      have "|z $ dim_vec a| ≥ 1" using second by auto
      have sum_a_ge_1:"(∑ x<dim_vec a. |a $ x|) ≥1"
        using False atLeast0LessThan
        by (metis abs_sum_abs int_one_le_iff_zero_less not_less sum_abs
zero_less_abs_iff)
      then have "2*k*(k+1)*(∑ x<dim_vec a. |a $ x|) + 1 > k"
      proof -
        have "2*(k+1)*(∑ x<dim_vec a. |a $ x|)≥1" using ‹k>0›
          by (smt (verit, ccfv_SIG) int_distrib(2) sum_a_ge_1 zmult_zless_mono2)
        then show ?thesis using ‹k>0›
          by (smt (verit, best) pos_mult_pos_ge sum_a_ge_1)
      qed
      moreover have "|?second| ≥ |2*k*(k+1)*(∑ x<dim_vec a. |a $ x|)
+ 1|"
        using ‹|z $ dim_vec a| ≥ 1›
        by (subst abs_mult) (simp add: lessThan_atLeast0 mult_le_cancel_left1)
      ultimately have "|v $ dim_vec a| > k" using second v_last by linarith
      then show ?thesis using v_le_k[of "dim_vec a"] by auto
    next
      case both
      have z_gr_one:"|real_of_int (z$dim_vec a)|≥1" using both by auto
      let ?second' = "2 * k * (k + 1) * (∑ i=0..<dim_vec a. |a $ i|)
+ 1"
      have "(∑ i = 0..<dim_vec a. z $ i * a $ i) ≠ 0" using both by
auto
      then have one: "k < |?first|"
        by (smt (z3) mult_less_cancel_left2 mult_minus_right)
      then have first_nonzero: "|?first| ≠ 0" using ‹k>0› by auto
      have two'':"2 * |?first| < |?second'|"
        using first_lt_second[OF ‹k>0› z_le_k] by auto
      then have two': "|real_of_int ?second'  / real_of_int ?first|
> 2"
      proof -
        have "0<real_of_int |?first|" using first_nonzero by presburger
        have "2 * real_of_int |?first| < real_of_int |?second'|" us-
ing two'' by linarith
        then have "2 <  real_of_int |?second'| / real_of_int |?first|"
```

```
            by (subst pos_less_divide_eq[OF <0<real_of_int |?first|>],
auto)
          also have "... = |real_of_int ?second' / real_of_int ?first|"
by simp
          finally show ?thesis by presburger
        qed
        then have two:"|(real_of_int ?second'  / real_of_int ?first) *

            real_of_int (z$dim_vec a)| > 2"
        proof -
          have "|(real_of_int ?second'  / real_of_int ?first) *
            real_of_int (z$dim_vec a)| = |real_of_int ?second'  / real_of_int
?first| *
            |real_of_int (z$dim_vec a)|" by (subst abs_mult, auto)
          also have ineq: "... ≥ |real_of_int ?second'  / real_of_int
?first|"
            using z_gr_one by (smt (z3) mult_less_cancel_left2)
          finally have "|(real_of_int ?second'  / real_of_int ?first) *

            real_of_int (z$dim_vec a)|≥ |real_of_int ?second'  / real_of_int
?first|" by blast
          then show ?thesis using two' by linarith
        qed
        have "real_of_int |v $ dim_vec a| / real_of_int |?first| ≥ 1"
        proof -
          have "real_of_int |v $ dim_vec a| / real_of_int |?first| =
            |real_of_int (v $ dim_vec a)| / |real_of_int ?first|"
            using of_int_abs[of "v $ dim_vec a"] of_int_abs[of "?first"]
by auto
          also have "... = |real_of_int (v $ dim_vec a) / real_of_int ?first|"
            using abs_divide[symmetric] by auto
          also have "... = |(real_of_int ?first + real_of_int ?second'
* real_of_int (z$dim_vec a)) /
            real_of_int ?first|" using v_last by auto
          also have "... = |real_of_int ?first / real_of_int ?first +
            (real_of_int ?second'  / real_of_int ?first)* real_of_int
(z$dim_vec a)|"
            by (metis (no_types, lifting) add_divide_distrib times_divide_eq_left)
          also have "... = |1 + (real_of_int ?second'  / real_of_int ?first)
*
            real_of_int (z$dim_vec a)|"
            using first_nonzero by (smt (verit, del_insts) of_int_eq_0_iff
one_eq_divide_iff)
          also have "... ≥ 1" using two by linarith
          finally show "real_of_int |v $ dim_vec a| / real_of_int |?first|
≥ 1" by blast
        qed
        then have "real_of_int |v $ dim_vec a| ≥ real_of_int |?first|"
```

```
        by (simp add: le_divide_eq_1)
      then have "|v $ dim_vec a| ≥ |?first|" using of_int_le_iff by
blast
      then have "|v $ dim_vec a| > k" using one by auto
      then show ?thesis using v_le_k[of "dim_vec a"] by auto
    qed
  qed
  have z_last_zero: "z$dim_vec a = 0"
  proof (rule ccontr)
    assume ccontr_assms:"z $ dim_vec a ≠ 0"
    then have "|z $ dim_vec a| ≥ 1" by auto
    have "(k+1) * (∑ i ∈ {0 ..< dim_vec a}. z $ i * a $ i) +
      (2*k*(k+1)* (∑ i ∈ {0 ..< dim_vec a}. |a $ i|) +1) * (z$(dim_vec
a)) = 0"
      (is "?first + ?second * (z$(dim_vec a)) = 0")
      using v_last_zero z_def gen_svp_basis_mult_last by auto
    then have abs_eq: "|?first| = |?second * (z$(dim_vec a))|" by linarith
    moreover have "|?first| < |?second * (z$(dim_vec a))|"
    proof -
      have "|?first| ≤ 2*|?first|" by auto
      then have "|?first| < |?second|" using first_lt_second[OF ‹k>0›
z_le_k, of a] by auto
      moreover have "|?second| ≤ |?second|*|z$dim_vec a|"
        using ‹|z $ dim_vec a| ≥ 1› by (simp add: mult_le_cancel_left1)
      ultimately have "|?first| < |?second|*|z$dim_vec a|" by linarith
      then show ?thesis by linarith
    qed
    ultimately show False by auto
  qed

  have v_real_z: "v = z" using v_eq_z_upto_last v_last_zero z_last_zero
    by (metis Suc_eq_plus1 dim_v_dim_a less_Suc_eq vec_eq_iff z_def(2))

  have "a · x = 0"
  proof -
    have "0 = ((gen_svp_basis a k) *v  z) $ (dim_vec a)"
      using v_last_zero z_def by auto
    also have "... = (k+1) * (∑ i ∈ {0 ..< dim_vec a}. z $ i * a $
i)"
      by (subst gen_svp_basis_mult, auto simp add: z_def z_last_zero)
    finally have zero: "(k+1) * (∑ i ∈ {0 ..< dim_vec a}. z $ i * a
$ i) = 0" by auto
    then show ?thesis unfolding scalar_prod_def using x_def ‹k>0›
    by (smt (verit, ccfv_SIG) atLeastLessThan_iff dim_vec index_vec
mult.commute
       mult_eq_0_iff of_int_hom.hom_0 sum.cong)
  qed
  moreover have "dim_vec x = dim_vec a" unfolding x_def by auto
  moreover have "x ≠ 0v (dim_vec x)"
```

```
    proof -
      have "∃i≤dim_vec a. v$i ≠ 0" using 1 unfolding gen_lattice_def
gen_svp_basis_def by auto
      then obtain i where ‹i ≤ dim_vec a› ‹v $ i ≠ 0› by blast
      then have ‹dim_vec a ≠ i› using v_last_zero by auto
      with ‹i ≤ dim_vec a› have ‹i < dim_vec a› by simp
      with ‹v $ i ≠ 0› have "z $ i ≠ 0" using v_real_z z_def
        by auto
      then have "∃i< dim_vec a. x$i ≠ 0" using x_def ‹i<dim_vec a›
by auto
      then show ?thesis using x_def by auto
    qed
    moreover have "‖x‖∞ ≤ k"
    proof -
      have "|z$i| = |vec (dim_vec a) (($) z) $ i|" if "i < dim_vec a" for
i using that by auto
      then have "Max (insert 0 {|vec (dim_vec a) (($) z) $ i| |i. i <
dim_vec a}) =
            Max (insert 0 {|z $ i| |i. i < dim_vec a})"
        by (smt (z3) Collect_cong Setcompr_eq_image mem_Collect_eq)
      also have "... = Max (insert 0 {|z $ i| |i. i < dim_vec a + 1})"

      proof -
        have "Max (insert 0 {|z $ i| |i. i < dim_vec a}) =
          Max (insert 0 (insert ( |z$dim_vec a|) {|z $ i| |i. i < dim_vec
a}))"
        proof -
          have "Max {|z $ i| |i. i < dim_vec a} ≥ 0"
            using ‹dim_vec a >0› by (subst Max_ge_iff, auto)
          then have "Max {|z $ i| |i. i < dim_vec a} ≥ |z$dim_vec a|"
            using z_last_zero by simp
          then have max_subst: "Max {|z $ i| |i. i < dim_vec a} =
              Max (insert |z $ dim_vec a| {|z $ i| |i. i < dim_vec a})"

            using ‹dim_vec a > 0› by (subst Max_insert)+ (auto)
          then show ?thesis using ‹dim_vec a > 0› by (subst Max_insert)+
(auto)
        qed
        then show ?thesis
          using insert_set_comprehension[of "(λi. |z $ i|)" "dim_vec a"]
by auto
      qed
      finally have "Max (insert 0 {|vec (dim_vec a) (($) z) $ i| |i. i <
dim_vec a}) =
                    Max (insert 0 {|z $ i| |i. i < dim_vec a +1})"
        by blast
      then have "‖x‖∞ = ‖v‖∞" unfolding linf_norm_vec_Max
        using x_def z_def v_real_z  by auto
      then show ?thesis using 1 by auto
```

```
    qed
    ultimately show ?case by force
  qed
qed
```

The SVP is NP-hard in ℓ_∞.

```
lemma "is_reduction reduce_svp_bhle bhle gap_svp"
unfolding is_reduction_def
proof (safe, goal_cases)
  case (1 as s)
  then show ?case using well_defined_reduction_svp by auto
next
  case (2 as s)
  then show ?case using NP_hardness_reduction_svp by auto
qed



end
```

References

[1] M. Ajtai. The shortest vector problem in L2 is NP-hard for randomized reductions (extended abstract). In *Proceedings of the thirtieth annual ACM symposium on Theory of computing - STOC '98*, pages 10–19, Dallas, Texas, United States, 1998. ACM Press.

[2] L. Babai. On Lovász lattice reduction and the nearest lattice point problem. *Combinatorica*, 6:1–13, 1986.

[3] A. K. Lenstra. Lattices and factorization of polynomials. *SIGSAM Bull.*, 15:15–16, 1981.

[4] A. K. Lenstra, H. Lenstra, and L. Lovasz. Factoring polynomials with rational coefficients. *MATH. ANN*, 261:515–534, 1982.

[5] H. W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

[6] D. Micciancio and S. Goldwasser. *Complexity of Lattice Problems*. Springer US, Boston, MA, 2002.

[7] P. van Emde Boas. Another NP-Complete Partition Problem and the Complexity of Computing Short Vectors in a Lattice. tech. report 81-04. Technical report, Mathematisch Instituut, Roetersstraat 15, 1018 WB Amsterdam, The Netherlands, 1981.