Hermite Normal Form

By Jose Divasón and Jesús Aransay

July 7, 2015

Abstract

The Hermite Normal Form is a canonical matrix analogue of Reduced Echelon Form, but involving matrices over more general rings. In this work we formalise an algorithm to compute the Hermite Normal Form of a matrix by means of elementary row operations, taking advantage of the Echelon Form AFP entry. We have proven the correctness of such an algorithm and refined it to immutable arrays. Furthermore, we have also formalised the uniqueness of the Hermite Normal Form of a matrix. Code can be exported and some examples of execution involving \( \mathbb{Z} \)-matrices and \( \mathbb{K}[x] \)-matrices are presented as well.

Contents

1 Hermite Normal Form  2
  1.1 Some previous properties  2
  1.1.1 Rings  2
  1.1.2 Polynomials  2
  1.1.3 Units  3
  1.1.4 Upper triangular matrices  4
  1.1.5 More properties of mod type  6
  1.1.6 Normalisation factor  6
  1.1.7 Div and Mod  6
  1.2 Units, associated and congruent relations  7
  1.3 Associates and residues functions  8
    1.3.1 Concrete instances in Euclidean rings  10
    1.3.2 Concrete case of the integer ring  10
  1.4 Definition of Hermite Normal Form  11
    1.4.1 Echelon form up to row k  12
    1.4.2 Hermite Normal Form up to row k  13
  1.5 Definition of an algorithm to compute the Hermite Normal Form  14
  1.6 Proving the correctness of the algorithm  15
    1.6.1 The proof  15
1.6.2 Proving that the Hermite Normal Form is computed by means of elementary operations ............. 42
1.6.3 The final theorem .............................................. 45
1.7 Proving the uniqueness of the Hermite Normal Form ........ 45
1.8 Examples of execution ............................................ 50

2 Hermite Normal Form refined to immutable arrays 50
2.1 Definition of the algorithm over immutable arrays ........ 50
2.2 Proving the equivalence between definitions of both representations ........................................ 51
2.3 Examples of execution using immutable arrays ............. 53

1 Hermite Normal Form
theory Hermite
  imports
    ../Echelon-Form/Echelon-Form-Inverse
    ../Echelon-Form/Examples-Echelon-Form-Abstract
begin

1.1 Some previous properties
1.1.1 Rings
  subclass (in bezout-ring-div) ring-div
  proof
  qed

  subclass (in euclidean-ring) ring-div
  proof
  qed

1.1.2 Polynomials
  lemma coeff-dvd-poly: [:coeff a (degree a):] dvd (a::'a::{field} poly)
  proof (cases a=0)
    case True
    thus ?thesis unfolding dvd-def by simp
    next
    case False
    show ?thesis
      by (metis (no-types, hide-lams) coeff-pCons-0 degree-mod-less degree-pCons-0
dvd-refl
dvd-trans gcd-poly.simps(2) leading-coeff-0-iff not-less0 poly-gcd-dvd1 poly-gcd-greatest)
  qed

  lemma poly-dvd-antisym2:
    fixes p q :: 'a::{field} poly
assumes dvd1: \( p \text{ dvd } q \) and dvd2: \( q \text{ dvd } p \)
shows \( p \text{ div } \lceil \text{coeff } p (\text{degree } p) \rceil = q \text{ div } \lceil \text{coeff } q (\text{degree } q) \rceil \)
proof (cases \( p = 0 \))
  case True 
  thus ?thesis 
    by (metis dvd1 dvd-0-left-iff)
next 
  case False 
  have \( q \neq 0 \) 
    by (metis False dvd2 dvd-0-left-iff)
  have \( \text{degree } p = \text{degree } q \) 
    using \( p \text{ dvd } q \) \( q \text{ dvd } p \) \( q \neq 0 \) \( p \neq 0 \) 
    by (intro order-antisym dvd-imp-degree-le)
  from \( p \text{ dvd } q \) obtain \( a \) where \( a: q = p \ast a .. \)
  with \( q \neq 0 \) have \( a \neq 0 \) by auto
  with degree \( a \) \( p \neq 0 \) have degree \( a = 0 \)
  by (simp add: degree-mul-eq)
  from \( p \text{ dvd } q \) obtain \( a \) where \( a: q = p \ast a .. \)
  with \( a \neq 0 \) show ?thesis 
    proof (cases \( a \), auto split: if_splits, metis \( a \neq 0 \))
    fix \( aa \) :: \( 'a \)
    assume \( a1: aa \neq 0 \)
    assume \( q = \text{smult } aa \) \( p \)
    have \( [:aa \ast \text{coeff } p (\text{degree } p) :] \text{ dvd } \text{smult } aa \) \( p \)
      using \( a1 \) by (metis (no_types) coeff-dvd-poly coeff-smult degree-smult-eq)
    thus \( p \text{ div } \lceil \text{coeff } p (\text{degree } p) \rceil = \text{smult } aa \) \( p \text{ div } \lceil aa \ast \text{coeff } p (\text{degree } p) \rceil \)
      using \( a1 \) by (simp add: False coeff-dvd-poly dvd-div-div-eq-mult)
    qed 
qed

1.1.3 Units

lemma unit-prod2: \( \text{is-unit } (x \ast y) = (\text{is-unit } x \land \text{is-unit } y) \)
  using unit-prod assms dvd-mult-left dvd-mult-right
unfolding is-unit-def by metis

lemma unit-setprod: 
  assumes \( \text{finite } S \)
  shows \( \text{is-unit } (\text{setprod } (\lambda i. U \$ i \$ i) S) = (\forall i \in S. \text{is-unit } (U \$ i \$ i)) \)
  using assms
proof (induct)
  case empty 
  thus ?case by auto
next 
  case (insert \( a \) \( S \))
  have \( \text{setprod } (\lambda i. U \$ i \$ i) (\text{insert } a \) \( S \) \) \( \text{setprod } (\lambda i. U \$ i \$ i) S \)
    by (simp add: insert.hyps(2))
  thus ?case using unit-prod2 insert.hyps by auto
qed
1.1.4 Upper triangular matrices

lemma is-unit-diagonal:
  fixes U :: 'a::{comm-ring-1, semiring-div} "'n::{finite, wellorder} "'n::{finite, wellorder}
  assumes U: upper-triangular U
  and det-U: is-unit (det U)
  shows \( \forall i. \text{is-unit} \ (U \$_i \$_i) \)
proof –
  have is-unit (setprod (\( \lambda i. \text{U}_i \$_i \)) UNIV)
    using det-U det-upperdiagonal[of U] U
  unfolding upper-triangular-def by auto
  hence \( \forall i \in \text{UNIV}. \text{is-unit} \ (U \$_i \$_i) \) using unit-setprod[of UNIV U] by simp
  thus \(?thesis by simp\)
qed

lemma upper-triangular-mult:
  fixes A :: 'a::{semiring-1} "'n::{mod-type} "'n::{mod-type}
  assumes A: upper-triangular A
  and B: upper-triangular B
  shows upper-triangular (A**B)
proof (unfold upper-triangular-def matrix-matrix-mult-def, vector, auto)
  fix i j::'n
  assume ji: j < i
  show \( \sum k \in \text{UNIV}. \text{A}_i \$_k \$_j = 0 \)
proof (rule setsum.neutral, clarify)
    fix x
    show A \$_i \$_x \$_j = 0
proof (cases x < i)
      case True
      thus \(?thesis by simp\)
    next
      case False
      hence x > j using ji by auto
      thus \(?thesis using A B ji unfolding upper-triangular-def by auto\)
  qed
  qed

lemma upper-triangular-adjugate:
  fixes A::('a::comm-ring-1,'n::{wellorder, finite}) vec, 'n) vec
  assumes A: upper-triangular A
  shows upper-triangular (adjugate A)
proof (auto simp add: cofactor-def upper-triangular-def adjugate-def transpose-def cofactorM-def)
  fix i j::'n assume ji: j < i
  def B \equiv (\( \chi k. \text{if} \ k = j \text{ then} e \ i \text{ else row k A} \))
  have Bji: B \$_j \$_j = 0 using ji unfolding B-def e-def by (simp add: \( \delta\)-def)
  have ut-B: upper-triangular B
    using A unfolding upper-triangular-def B-def e-def
    using ji by (auto simp add: \( \delta\)-def row-def)
have \( \det(\text{minor}_M A \; j \; i) = (\prod_{i \in \text{UNIV}} B \; i \; i) \)

unfolding \(\det-\text{minor}_M\)-row \(B\)-def[\(\text{symmetric}\)]

using \(\det-\text{upper-diagonal}\) using \(ut-B\) unfolding \(\text{upper-triangular-def}\) by \(\text{auto}\)

also have \(... = 0\) by \((\text{rule setprod-zero, } \text{auto intro: } B_{jj})\)

finally show \(\det(\text{minor}_M A \; j \; i) = 0\).

qed

lemma \(\text{upper-triangular-inverse}\):

fixes \(A::(\{\text{semiring-div, comm-ring-1}\}, 'n::(\{\text{wellorder, finite}\}) \; \text{vec}, 'n) \; \text{vec}\)

assumes \(A::\text{upper-triangular} \; A\)

and \(\text{inv-A:: invertible} \; A\)

shows \(\text{upper-triangular} \; (\text{matrix-inv} \; A)\)

using \(\text{upper-triangular-adjugate[OF } A]\)

unfolding \(\text{invertible-imp-matrix-inv[OF inv-A]}\)

unfolding \(\text{scalar-matrix-mult-def upper-triangular-def} \) by \(\text{auto}\)

lemma \(\text{upper-triangular-mult-diagonal}\):

fixes \(A::(\{\text{semiring-1}\}, 'n::(\{\text{wellorder, finite}\}) \; \text{vec}, 'n) \; \text{vec}\)

assumes \(A::\text{upper-triangular} \; A\)

and \(B::\text{upper-triangular} \; B\)

shows \((A**B) \; i \; i = A \; i \; i \; * B \; i \; i\)

proof --

have \(\text{UNIV-rw: } \text{UNIV} = (\text{insert } i \; (\text{UNIV} - \{i\}))\) by \(\text{auto}\)

have \(\text{setsum-0: } (\sum_{k \in \text{UNIV} - \{i\}}. A \; i \; k \; * B \; k \; i) = 0\)

proof \((\text{rule setsum.neutral, rule})\)

fix \(x\) assume \(x::x \in \text{UNIV} - \{i\}\)

show \(A \; i \; x \; * B \; x \; i = 0\)

proof \((\text{cases } x<i)\)

case \(\text{True}\)

thus \(\text{thesis}\) using \(A\) unfolding \(\text{upper-triangular-def}\) by \(\text{auto}\)

next

case \(\text{False}\)

hence \(x>i\) using \(x\) by \(\text{auto}\)

thus \(\text{thesis}\) using \(B\) unfolding \(\text{upper-triangular-def}\) by \(\text{auto}\)

qed

qed

have \((A**B) \; i \; i = (\sum_{k \in \text{UNIV}. A \; i \; k \; * B \; k \; i})\)

unfolding \(\text{matrix-matrix-mult-def by simp}\)

also have \(... = (\sum_{k \in (\text{insert } i \; (\text{UNIV} - \{i\}))}. A \; i \; k \; * B \; k \; i)\)

using \(\text{UNIV-rw by simp}\)

also have \(... = (A \; i \; i \; * B \; i \; i) + (\sum_{k \in \text{UNIV} - \{i\}. A \; i \; k \; * B \; k \; k} \; i)\)

by \((\text{rule setsum.insert, simp-all})\)

finally show \(\text{thesis}\) unfolding \(\text{setsum-0 by simp}\)

qed
1.1.5 More properties of mod type

lemma add-left-neutral:
  fixes a :: 'n :: mod-type
  shows \((a + b = a) = (b = 0)\)
  by (auto, metis add-left-cancel monoid-add-class.add.right-neutral)

lemma from-nat-1: from-nat 1 = 1
  unfolding from-nat-def o-def Abs'-def
  by (metis Rep-1 Rep-mod of-nat-1 one-def)

1.1.6 Normalisation factor

lemma dvd-normalisation-factor:
  assumes \(a \text{ div normalisation-factor } a = b \text{ div normalisation-factor } b\)
  shows \(a \text{ dvd } b\)
  by (metis assms dvd-0 dvd-refl normalisation-factor-dvd-iff)

lemma a-dvd-a-div-normalisation-factor: \(a \text{ dvd } a \text{ div normalisation-factor } a\)
  by (metis dvd-refl normalisation-0-iff normalisation-factor-dvd-iff)

lemma a-div-a-dvd-normalisation-factor: \(a \text{ div normalisation-factor } a \text{ dvd } a\)
  by (metis div-by-0 dvd-refl normalisation-factor-0 normalisation-factor-dvd-iff)

lemma div-normalisation-factor:
  assumes \(xa \text{ div normalisation-factor } xa \neq xb \text{ div normalisation-factor } xb\)
  and \(xa \text{ div normalisation-factor } xa \text{ dvd } xb \text{ div normalisation-factor } xb\)
  shows \(\neg (xb \text{ div normalisation-factor } xb \text{ dvd } xa \text{ div normalisation-factor } xa)\)
  using assms
  by (auto, metis (mono-tags) associated-def associated-iff-normed-eq div-by-0 dvd-eq-mod-eq-0
       mod-by-0 normalisation-factor-0 normalisation-factor-dvd-iff normalisation-factor-dvd-iff)

1.1.7 Div and Mod

lemma dvd-minus-eq-mod:
  fixes c :: 'a :: ring-div
  assumes \(c \neq 0 \text{ and } c \text{ dvd } a - b\)
  shows \(a \text{ mod } c = b \text{ mod } c\)
  using assms dvd-div-mult-self[of c]
  by (metis add.commute diff-add-cancel mod-mult-self1)

lemma eq-mod-dvd-minus:
  fixes c :: 'a :: ring-div
  assumes \(c \neq 0 \text{ and } a \text{ mod } c = b \text{ mod } c\)
  shows \(c \text{ dvd } a - b\)
  using assms
  by (metis (no-types, hide-lams) add.commute diff-0 diff-add-cancel
diff-minus-eq-add dvd-eq-mod-eq-0 mod-0 mod-add-right-eq)

lemma dvd-cong-not-eq-mod:
  fixes c :: 'a :: ring-div
assumes \( xa \mod c \neq xb \) and \( c \mid xa \mod c - xb \) and \( c \neq 0 \)
shows \( xb \mod c \neq xb \)
using assms
by (metis (no-types, lifting) diff-add-cancel dvdE mod-mod-trivial
     semiring-div-class.mod-mult-self4)

lemma \( \text{diff-mod-cong-0} \):
fixes \( c \) :: 'a :: ring-div
assumes \( xa \mod c \neq xb \mod c \) and \( c \mid xa \mod c - xb \mod c \)
s shows \( c = 0 \)
using assms dvd-cong-not-eq-mod mod-mod-trivial by blast

lemma \( \text{cong-diff-mod} \):
fixes \( c \) :: 'a :: ring-div
assumes \( xa \neq xb \) and \( c \mid xa - xb \) and \( xa = xa \mod c \)
s shows \( xb \neq xb \mod c \)
by (metis assms diff-eq-diff-eq diff-numeral-special(12) dvd-0-left dvd-minus-eq-mod)

lemma \( \text{exists-k-mod} \):
fixes \( c \) :: 'a :: ring-div
shows \( \exists k. a \mod c = a + k \cdot c \)
by (metis add.commute diff-add-cancel diff-minus-eq-add
     mod-div-equality2 mult.commute mult-minus-left)

1.2 Units, associated and congruent relations

context \( \text{semiring-1} \)
begin

definition Units = \{ x::'a. (\exists k. 1 = x \cdot k) \}\nend

context \( \text{ring-1} \)
begin

definition cong::'a::'a =>'a => bool
  where cong a c b = (\exists k. (a - c) = b \cdot k)

lemma cong-eq: cong a c b = (b dvd (a - c))
  unfolding ring-1-class.cong-def dvd-def by simp
end

context \( \text{semiring-div} \)
begin

lemma Units-eq: Units = \{ x. is-unit x \} unfolding is-unit-def Units-def dvd-def
..
lemma associated-eq: associated a b = ( ∃ u ∈ Units. a = u * b)
unfolding associated-def Units-def is-unit-def dvd-def
by (auto, metis associated-def associated-iff-div-unit dvdE dvd-triv-left is-unit-def)
  (metis mult-1-right mult-assoc mult-commute, metis mult-commute)
end

context ring-div
begin

definition associated-rel = {(a,b). associated a b}

lemma equiv-associated:
  shows equiv UNIV associated-rel
unfolding associated-rel-def equiv-def refl-on-def sym-def trans-def
by (auto simp add: associated-comm)
(simp add: associated-def, meson associated-def dvd-trans)

definition congruent-rel b = {(a,c). cong a c b}

lemma relf-congruent-rel:
  refl (congruent-rel b)
unfolding refl-on-def congruent-rel-def
unfolding cong-def
by (auto, metis mult-zero-right)

lemma sym-congruent-rel:
  sym (congruent-rel b)
unfolding sym-def congruent-rel-def unfolding cong-def
by (auto, metis add-commute add-minus-cancel diff-conv-add-uminus
  minus-mult-commute mult-left-commute mult-1-left)

lemma trans-congruent-rel:
  trans (congruent-rel b)
unfolding trans-def congruent-rel-def unfolding cong-def
by (auto, metis add-assoc diff-add-cancel
diff-conv-add-uminus distrib-left)

lemma equiv-congruent:
  equiv UNIV (congruent-rel b)
unfolding equiv-def
using refl-congruent-rel sym-congruent-rel trans-congruent-rel by auto
end

1.3 Associates and residues functions

context semiring-div
begin

definition ass-function :: ('a ⇒ 'a) ⇒ bool
  where ass-function f = ((∀ a. associated a (f a)) ∧ pairwise (λa b. ¬ associated a b) (range f))
definition Complete-set-non-associates S
    = (∃f. ass-function f ∧ f'UNIV = S ∧ (pairwise (λa b. ¬ associated a b) S))
end

context ring-1
begin

definition res-function :: ('a ⇒ 'a ⇒ 'a) ⇒ bool
where res-function f = (∀c. (∀a b. cong a b c ←→ f c a = f c b)
∧ pairwise (λa b. ¬ cong a b c) (range (f c))
∧ (∀a. ∃k. f c a = a + k*c))

definition Complete-set-residues g
    = (∃f. res-function f ∧ (∀c. (pairwise (λa b. ¬ cong a b c) (f c'UNIV)) ∧ g c
                      = f c'UNIV))
end

lemma ass-function-Complete-set-non-associates:
    assumes f: ass-function f
    shows Complete-set-non-associates (f'UNIV)
    unfolding Complete-set-non-associates-def ass-function-def
    apply (rule exI[of - f])
    using f unfolding ass-function-def unfolding pairwise-def associated-def by fast

lemma in-Ass-not-associated:
    assumes Ass-S: Complete-set-non-associates S
    and x: x∈S and y: y∈S and x-not-y: x≠y
    shows ¬ associated x y
    using assms unfolding Complete-set-non-associates-def pairwise-def by auto

lemma ass-function-0:
    assumes r: ass-function ass
    shows (ass x = 0) = (x = 0)
    using assms unfolding ass-function-def associated-def pairwise-def
    by (metis dvd-0-left-iff)+

lemma ass-function-0':
    assumes r: ass-function ass
    shows (ass x div x = 0) = (x=0)
    using assms unfolding ass-function-def associated-def pairwise-def
    by (auto, metis dvd-0-left-iff dvd-div-eq-mult mult.commute mult-zero-right)

lemma res-function-Complete-set-residues:
    assumes f: res-function f
shows $\text{Complete-set-residues} \ (\lambda c. (f c) \ \text{UNIV})$

unfolding $\text{Complete-set-residues-def}$

apply (rule exI [of $f$]) using $f$ unfolding $\text{res-function-def}$ by blast

lemma $\text{in-Res-not-congruent}$:

assumes $\text{res-g}$: $\text{Complete-set-residues \ g}$

and $x$: $x \in g \ \text{b and} \ y$: $y \in g \ \text{b and} \ x$-$\not\equiv y$

shows $\neg \text{cong \ x \ y \ b}$

using $\text{assms}$

unfolding $\text{Complete-set-residues-def}$

unfolding $\text{pairwise-def}$

by auto

1.3.1 Concrete instances in Euclidean rings

definition $\text{ass-function-euclidean} (p ::'a::\{euclidean-ring\}) = p \ \text{div normalisation-factor \ p}$

definition $\text{res-function-euclidean} \ b \ (n ::'a::\{euclidean-ring\}) = (\text{if} \ b = 0 \ \text{then} \ n \ \text{else} \ (n \ \text{mod} \ b))$

lemma $\text{ass-function-euclidean}: \text{ass-function \ ass-function-euclidean}$

unfolding $\text{ass-function-def \ image-def \ ass-function-euclidean-def}$

unfolding $\text{associated-def \ pairwise-def}$

by (auto simp add: $\text{div-normalisation-factor \ a-dvd-a-normalisation-factor}$

$a$-$\text{div-a-normalisation-factor}$)

lemma $\text{res-function-euclidean}$:

$\text{res-function} (\text{res-function-euclidean})$

by (auto simp add: $\text{pairwise-def \ res-function-def \ cong-eq \ image-def \ res-function-euclidean-def}$

$dvd-minus-eq-mod$

(auto simp add: $\text{dvd-cong-not-eq-mod \ eq-mod-dvd-minus \ dvd-cong-0 \ cong-diff-mod \ exists-k-mod}$)

1.3.2 Concrete case of the integer ring

definition $\text{ass-function-int} \ (n :: \text{int}) = \text{abs} \ n$

lemma $\text{abs-mod-less}$:

assumes $b$: $b \neq 0$

shows $|xa \ \text{mod} \ b| < |b |:\text{int}|$

by (metis $b$ $\text{abs-of-nonneg \ abs-of-nonpos \ neg-less-iff-less}$

$\text{neg-mod-conj \ not-less \ not-less-iff-gr-or-eq \ pos-mod-conj}$)

lemma $\text{ass-function-int}: \text{ass-function-int} = \text{ass-function-euclidean}$

unfolding $\text{fun-eq-iff \ ass-function-int-def \ ass-function-euclidean-def}$

by (metis $\text{gcd-0 \ gcd-0-int}$)

lemma $\text{ass-function-int-UNIV} (\text{ass-function-int': UNIV}) = \{x. x \geq 0\}$

unfolding $\text{ass-function-int-def \ image-def}$

by (auto, metis $\text{abs-of-nonneg}$)
1.4 Definition of Hermite Normal Form

It is worth noting that there is not a single definition of Hermite Normal Form in the literature. For instance, some authors restrict their definitions to the case of square nonsingular matrices. Other authors just work with integer matrices. Furthermore, given a matrix $A$ its Hermite Normal Form $H$ can be defined to be upper triangular or lower triangular. In addition, the transformation from $A$ to $H$ can be made by means of elementary row operations or elementary column operations. In our case, we will work as general as possible, so our input will be any matrix (including nonsquare ones). The output will be an upper triangular matrix obtained by means of elementary row operations.

Hence, given a complete set of nonassociates and a complete set of residues, $H$ is said to be in Hermite Normal Form if:

1. $H$ is in Echelon Form
2. The first nonzero element of a nonzero row belongs to the complete set of nonassociates
3. Let $h$ be the first nonzero element of a nonzero row. Then each element above $h$ belongs to the corresponding complete set of residues of $h$

A matrix $H$ is the Hermite Normal Form of a matrix $A$ if:

1. There exists an invertible matrix $P$ such that $A = PH$
2. $H$ is in Hermite Normal Form

The Hermite Normal Form is usually applied to integer matrices. As we have already said, there is no one single definition of it, so some authors impose different conditions. In the particular case of integer matrices, leading coefficients (the first nonzero element of a nonzero row) are usually required to be positive, but it is also possible to impose them to be negative since we would only have to multiply by $-1$.

In the case of the elements $h_{ik}$ above a leading coefficient $h_{ij}$, some authors demand $0 \leq h_{ik} < h_{ij}$, other ones impose the conditions $h_{ik} \leq 0$ and $|h_{ik}| < h_{ij}$, and other ones $-\frac{h_{ij}}{2} < h_{ik} \leq \frac{h_{ij}}{2}$. More different options are also possible.

All the possibilities can be represented selecting a complete set of nonassociates and a complete set of residues. The algorithm to compute the Hermite Normal Form will be parameterised by functions which obtain the appropriate leading coefficient and the suitable elements above them. We can execute the algorithm with different functions to get exactly which Hermite Normal Form we want. Once we fix such a complete set of nonassociates
and the corresponding complete set of residues, the Hermite Normal Form is unique.

1.4.1 Echelon form up to row k

We present the definition of echelon form up to a row k (not included).

**Definition**  
\( \text{echelon-form-upt-row } A \ k = (\forall i. \to-nat \ i < k \land \text{is-zero-row } i \ A \rightarrow \neg (\exists j. j > i \land \to-nat \ j < k \land \neg \text{is-zero-row } j \ A)) \land (\forall i. j. i < j \land \to-nat \ j < k \land \neg \text{is-zero-row } i \ A \land \neg \text{is-zero-row } j \ A \rightarrow (\text{LEAST } n. A \ S \ i \ S \ n \neq 0) < (\text{LEAST } n. A \ S \ j \ S \ n \neq 0)) \)

**Lemma**  
\( \text{echelon-form-upt-row-condition1-explicit} \)

* Assumes \( \text{echelon-form-upt-row } A \ k \)  
* And \( \to-nat \ i < k \) and \( \text{is-zero-row } i \ A \)  
* Shows \( \neg (\exists j. j > i \land \to-nat \ j < k \land \neg \text{is-zero-row } j \ A) \)  
* Using \( \text{assms unfolding} \ \text{echelon-form-upt-row-def by blast} \)

**Lemma**  
\( \text{echelon-form-upt-row-condition1-explicit}' \)

* Assumes \( \text{echelon-form-upt-row } A \ k \)  
* And \( \to-nat \ i < k \) and \( \text{is-zero-row } i \ A \) and \( i \leq j \) and \( \to-nat \ j < k \)  
* Shows \( \text{is-zero-row } j \ A \)  
* Proof (cases \( i=j \))  
  * Case True thus ?thesis using \( \text{assms by auto} \)  
  * Next  
    * Case False thus ?thesis using \( \text{assms unfolding} \ \text{echelon-form-upt-row-def by simp} \)  
* Qed

**Lemma**  
\( \text{echelon-form-upt-row-condition1-explicit-neg} \)

* Assumes \( \text{echelon-form-upt-row } A \ k \)  
* And \( i A: \neg \text{is-zero-row } i \ A \) and \( \text{ia-i: ia < i} \)  
* And \( \text{i: to-nat i < k} \)  
* Shows \( \neg \text{is-zero-row } ia \ A \)  
* Proof  
  * Have \( \text{to-nat ia < k} \) by \( (\text{metis ia-i i less-trans to-nat-mono}) \)  
  * Thus ?thesis using \( \text{assms unfolding} \ \text{echelon-form-upt-row-def by blast} \)  
* Qed

**Lemma**  
\( \text{echelon-form-upt-row-condition2-explicit} \)

* Assumes \( \text{echelon-form-upt-row } A \ k \)  
* And \( \text{ia < j and to-nat j < k and \neg is-zero-row ia A and \neg is-zero-row j A} \)  
* Shows \( (\text{LEAST } n. A \ S \ ia \ S \ n \neq 0) < (\text{LEAST } n. A \ S \ j \ S \ n \neq 0) \)  
* Using \( \text{assms unfolding} \ \text{echelon-form-upt-row-def by auto} \)

12
lemma echelon-form-upt-row-intro:
assumes(∀ i. to-nat i < k ∧ is-zero-row i A → (∃ j. i < j ∧ to-nat j < k ∧ ¬ is-zero-row j A))
and (∀ i j. i < j ∧ to-nat j < k ∧ ¬ is-zero-row i A ∧ ¬ is-zero-row j A → (LEAST n. A $ i \cdot n \neq 0) < (LEAST n. A $ j \cdot n \neq 0))
shows echelon-form-upt-row A k
using assms unfolding echelon-form-upt-row-def by simp

by (simp add; to-nat-less-card echelon-form-def echelon-form-upt-row-def ncols-def nrows-def
    echelon-form-upt-k-def is-zero-row-upt-k-def is-zero-row-def)

1.4.2 Hermite Normal Form up to row k

Predicate to check if a matrix is in Hermite Normal form up to row k (not included).

deinition Hermite-upt-row A k associates residues =
  (Complete-set-non-associates associates ∧
   Complete-set-residues residues ∧
   echelon-form-upt-row A k ∧
   (∀ i. to-nat i < k ∧ ¬ is-zero-row i A → A $ i \cdot (LEAST n. A $ i \cdot n \neq 0) ∈ associates) ∧
   (∀ i. to-nat i < k ∧ ¬ is-zero-row i A → (∀ j<i. A $ j \cdot (LEAST n. A $ i \cdot n \neq 0) ∈ residues (A $ i \cdot (LEAST n. A $ i \cdot n \neq 0))))
)

The definition of Hermite Normal Form is now introduced:

deinition Hermite::'a::{bezout-ring-div} set ⇒ ('a ⇒ 'a set) ⇒ (('a, 'b::{mod-type}) vec, 'c::{mod-type}) vec ⇒ bool
where Hermite associates residues A =
  (Complete-set-non-associates associates ∧ (Complete-set-residues residues) ∧
    echelon-form A ∧ (∀ i. ¬ is-zero-row i A → A $ i $ (LEAST n. A $ i $ n \neq 0) ∈ associates) ∧
    (∀ i. ¬ is-zero-row i A → (∀ j<i. A $ j $ (LEAST n. A $ i $ n \neq 0)) ∈ residues (A $ i $ (LEAST n. A $ i $ n \neq 0))))
)

lemma Hermite-Hermite-upt-row: Hermite ass res A = Hermite-upt-row A (nrows A) ass res
by (simp add; Hermite-def Hermite-upt-row-def ncols-def nrows-def is-zero-row-def echelon-form-echelon-form-upt-row)

lemma Hermite-intro:
assumes Complete-set-non-associates associates
and Complete-set-residues residues
and echelon-form A
and (∀ i. ¬ is-zero-row i A → A $ i $ (LEAST n. A $ i $ n ≠ 0) ∈ associates)
and (∀ i. ¬ is-zero-row i A → (∀ j. j < i → A $ j $ (LEAST n. A $ i $ n ≠ 0) ∈ residues (A $ i $ (LEAST n. A $ i $ n ≠ 0))))
shows Hermite associates residues A
using assms unfolding Hermite-def by simp

1.5 Definition of an algorithm to compute the Hermite Normal Form

The algorithm is parameterised by three functions:

- The function that computes de Bézout identity (necessary to compute the echelon form).
- The function that given an element, it returns its representative element in the associated equivalent class, which will be an element in the complete set of nonassociates.
- The function that given two elements a and b, it returns its representative element in the congruent equivalent class of b, which will be an element in the complete set of residues of b.

\[
\text{primrec \ Hermite-reduce-above } :: \ 'a::ring-divRING'\text{-}cols::mod-type'\text{-}rows::mod-type\Rightarrow\text{nat} \\
\Rightarrow'\text{-}rows\Rightarrow'\text{-}cols\Rightarrow ('a\Rightarrow'a\Rightarrow'a) \Rightarrow ('a'\text{-}cols::mod-type'\text{-}rows::mod-type) \\
\text{where Hermite-reduce-above A 0 i j res } = A \\
| \text{Hermite-reduce-above A (Suc n) i j res } = (let i'=((\text{from-nat n})::'rows); \\
Aij = A $ i $ j; \\
Ai'j = A $ i' $ j \\
in \text{Hermite-reduce-above (row-add A i' i ((\text{res Aij (Ai'j)}) - (Ai'j)) div Aij)) n i j res) \\
\]

\[ \text{definition \ Hermite-of-row-i ass res A i } = ( \\
\text{if is-zero-row i A } \\
\text{then A } \\
\text{else } \\
| \text{let j } = (\text{LEAST n. A $ i $ n ≠ 0); Aij = (A $ i $ j); } \\
A' = \text{mult-row A i ((ass Aij) div Aij)} \\
in \text{Hermite-reduce-above A' (to-nat i) i j res) } \\
\]

\[ \text{definition \ Hermite-of-upt-row-i A i ass res } = \text{foldl (Hermite-of-row-i ass res) A (map from-nat [0..<i]) } \\
\]

\[ \text{definition \ Hermite-of A ass res bezout } = (let A' = \text{echelon-form-of A bezout in Hermite-of-upt-row-i A' (nrows A) ass res) } \\
\]
1.6 Proving the correctness of the algorithm

1.6.1 The proof

lemma Hermite-reduce-above-preserves:
assumes \( n : n \leq \text{to-nat} \, a \)
shows \((\text{Hermite-reduce-above} \, A \, n \, i \, j \, \text{res}) \, a \, b = A \, a \, b\)
using \( n \)
proof (induct \( n \) arbitrary: \( A \))
case 0 thus \(?\) case by simp
next
case (Suc \( n \))
thus \(?\) case by (auto simp add: Let-def row-add-def)
\( \text{metis Suc-le-eq from-nat-mono from-nat-to-nat-id less-irrefl to-nat-less-card} \)
\qed

lemma Hermite-reduce-above-works:
assumes \( n : n \leq \text{to-nat} \, i \) and \( a : \text{to-nat} \, a < n \)
shows \((\text{Hermite-reduce-above} \, A \, n \, i \, j \, \text{res}) \, a \, b = \text{row-add} \, A \, a \, i \, (\text{res} \, (A \, i \, j) \, (A \, a \, j) - (A \, a \, j)) / (A \, i \, j) \, a \, b \)
using \( n \, a \)
proof (induct \( n \) arbitrary: \( A \))
case 0 thus \(?\) case by (simp add: row-add-def)
next
case (Suc \( n \))
def \( A' := \text{row-add} \, A \, (\text{from-nat} \, n) \, i \)
\((\text{res} \, (A \, i \, j) \, (A \, \text{from-nat} \, n \, j) - A \, \text{from-nat} \, n \, j) / (A \, i \, j) \)
have \( n : n < \text{nrows} \, A \)
unfolding nrows-def by (metis Suc.prems(1) Suc-le-eq less-trans to-nat-less-card)
show \(?\) case
proof (cases to-nat \( a = n \))
case False
have \( a \text{-less-n: to-nat} \, a < n \) by (metis False Suc.prems(2) less-antisym)
have \( \text{Hermite-reduce-above} \, 15 \)
\( A' \, n \, i \, j \, \text{res} \, A \, a \, b \)
by (simp add: Let-def A'-def)
also have \( \text{...} = \text{row-add} \, A' \, a \, i \, ((\text{res} \, (A' \, i \, j) \, (A' \, a \, j) - A' \, a \, j) / (A' \, i \, j) \, a \, b \)
\( \text{by (rule Suc.hyps[OF \,- a\text{-less-n}], simp add: Suc.prems(1) Suc-leD}) \)
also have \( \text{...} = \text{row-add} \, A \, a \, i \, ((\text{res} \, (A \, i \, j) \, (A \, a \, j) - A \, a \, j) \)
\( A \, i \, j \, \text{res} \, A \, a \, b \)
unfolding row-add-def A'-def
using a-less-n Suc.prems n to-nat-from-nat-id[OF n[unfolded nrows-def]]
by auto
finally show \(?\) thesis .
next
case True
hence \( a \text{-eq-fn-n:} \, a = \text{from-nat} \, n \) by auto
have \( \text{Hermite-reduce-above} \, 15 \)
}\( A \, n \, i \, j \, \text{res} \, A \, a \, b \)
by auto
finally show \(?\) thesis .
\[ A' n i j \text{ res } a \ b \]
\[ \text{by (simp add: Let-def A'-def)} \]
\[ \text{also have } \ldots = A' a b \]
\[ \text{by (rule Hermite-reduce-above-preserves, simp add: True)} \]
\[ \text{finally show } \text{thesis unfolding A'-def a-eq-fn-n } . \]
\text{qed}
\text{qed}

lemma \text{Hermite-of-row-preserves-below:}
\text{assumes } i\leq a\text{ shows (Hermite-of-row-i ass res A i) } a \ b = A a b \]
\text{proof (auto simp add: Hermite-of-row-i-def Let-def)}
\[ \text{let } ?M= (\text{mult-row } A i (\text{ass } (A \ i \ ?n) (\text{LEAST n. } A \ i \ n \neq 0)) \div A \ i \ ?n) \]
\[ \text{let } ?H= \text{Hermite-reduce-above } ?M \ (\text{to-nat i}) \ i \ (\text{LEAST n. } A \ i \ n \neq 0) \text{ res} \]
\[ \text{have } ?H a b = ?M a b \]
\[ \text{by (rule Hermite-reduce-above-preserves)} \]
\[ (\text{metis i-a not-le not-less-iff-gr-or-eq to-nat-mono'}) \]
\[ \text{also have } \ldots = A a b \text{ unfolding mult-row-def using i-a by fastforce} \]
\[ \text{finally show } \text{thesis unfolding A'-def a-eq-fn-n } . \]
\text{qed}

lemma \text{Hermite-of-row-preserves-previous-cols:}
\text{assumes } b\leq (\text{LEAST n. } A \ i \ n \neq 0) \text{ and } \not\text{is-zero-row i A} \text{ shows (Hermite-of-row-i ass res A i) } a \ b = A a b \]
\text{proof (auto simp add: Hermite-of-row-i-def Let-def)}
\[ \text{let } ?n= (\text{LEAST n. } A \ i \ n \neq 0) \]
\[ \text{let } ?M= (\text{mult-row } A i (\text{ass } (A \ i \ ?n) \div A \ i \ ?n)) \]
\[ \text{let } ?H= \text{Hermite-reduce-above } ?M \ (\text{to-nat i}) \ i \ (\text{LEAST n. } A \ i \ n \neq 0) \text{ res} \]
\[ \text{have Aib: } A i b = 0 \text{ by (metis mono-tags b not-less-Least)} \]
\[ \text{show } ?H a b = A a b \]
\text{proof (cases a\geq i)}
\[ \text{case True} \]
\[ \text{have } ?H a b = ?M a b \]
\[ \text{by (rule Hermite-reduce-above-preserves) (metis True to-nat-mono')} \]
\[ \text{also have } \ldots = A a b \text{ using Aib unfolding mult-row-def by auto} \]
\[ \text{finally show } \text{thesis } . \]
\text{next}
\[ \text{let } ?R= \text{row-add } ?M a i (\text{res } (\text{?M } i \ ?n) (\text{?M } a \ ?n) - ?M a \ ?n) \div ?M i \ ?n) \]
\[ \text{case False} \]
\[ \text{hence ia: i>a by simp} \]
\[ \text{have } ?H a b = ?R a b \text{ by (rule Hermite-reduce-above-works, auto simp add: ia to-nat-mono)} \]
\[ \text{also have } \ldots = A a b \text{ using ia Aib unfolding row-add-def mult-row-def by auto} \]
\[ \text{finally show } \text{thesis } . \]
lemma echelon-form-Hermite-of-condition1:
  fixes res ass i A
  defines M ≡ mult-row A i (ass (A $ i $ (LEAST n. A $ i $ n $= 0))) div A
  defines H ≡ Hermite-reduce-above M (to-nat i) (LEAST n. A $ i $ n $= 0) res
  assumes e: echelon-form A
  and a: ass-function ass
  and not-zero-iA: ¬ is-zero-row i A
  and zero-ia-H: is-zero-row ia H
  and ia-j: ia $< j
  shows is-zero-row j H
  proof (cases is-zero-row ia A)
    case True
    have zero-jA: is-zero-row j A by (metis True e echelon-form-condition1 ia-j)
    have ij: i $< j by (metis e echelon-form-condition1 neq-iff not-zero-iA zero-jA)
    show ?thesis
      proof (auto simp add: is-zero-row-def is-zero-row-upt-k-def ncols-def)
        fix a
        have H $ j $ a $= M $ j $ a
          unfolding H
          by (rule Hermite-reduce-above-preserves) (metis dual-order.strict-iff-order ij
          to-nat-mono)
        also have ... $= A $ j $ a unfolding M mult-row-def using ij by auto
        also have ... $= 0 using zero-jA by (simp add: is-zero-row-def is-zero-row-upt-k-def
        ncols-def)
        finally show H $ j $ a $= 0 .
      qed
    qed
    next
    case False note not-zero-ia-A=True
    let ?n=(LEAST n. A $ ia $ n $= 0)
    have A-ia-n: A $ ia $ ?n $= 0
      by (metis (mono-tags, lifting) LeastI is-zero-row-def is-zero-row-upt-k-def not-zero-ia-A)
    show ?thesis
      proof (cases i $\leq$ ia)
        case True
        have H $ ia $ ?n $= M $ ia $ ?n
          unfolding H by (rule Hermite-reduce-above-preserves, simp add: True
          to-nat-mono)
        also have ... $= 0 unfolding M mult-row-def using A-ia-n ass-function-0[OF
        a] by auto
        finally have H $ ia $ ?n $= 0 .
        hence not-zero-ia-H: ¬ is-zero-row ia H
        unfolding is-zero-row-def is-zero-row-upt-k-def ncols-def by auto
        thus ?thesis using zero-ia-H by contradiction
      next
    qed
case False
let \( ?m = \text{LEAST } m. \ A \& ?m \neq 0 \)
let \( ?R = \text{row-add } M \ i \ a \ ((\text{res } (M \& i \& ?m) (M \& i \& ?m) - M \& i \& ?m)) \div M \& i \& ?m) \)

have ia-less-i: \( ia < i \) by (metis False not-less)

have \( nm: \ ?n < ?m \) by (rule echelon-form-condition2-explicit[OF e ia-less-i not-zero-ia-A not-zero-iA])

have \( A \& i \& ?n = 0 \) by (metis (full-types) nm not-less-Least)

have \( H \& i \& ?n \neq 0 \) by (simp add: ia-less-i to-nat-mono)

also have ... = 0 unfolding M mult-row-def by auto

finally have \( H \& i \& ?n \neq 0 \). hence not-zero-ia-H: \( \text{¬ is-zero-row } ia \ H \)
unfolding is-zero-row-def is-zero-row-upt-k-def ncols-def by auto

thus \( ?\text{thesis} \) using zero-ia-H by contradiction

qed

qed

lemma row-zero-A-imp-row-zero-H:
 fixes \( \text{res } \text{ass } i \ A \)
 defines \( M: M \equiv \text{mult-row } A \ i \ (\text{ass } (A \& i \& (\text{LEAST } n. \ A \& i \& n \neq 0))) \div A \& i \& (\text{LEAST } n. \ A \& i \& n \neq 0)) \)
 defines \( H: H \equiv \text{Hermite-reduce-above } M \ (\text{to-nat } i) \ i \ (\text{LEAST } n. \ A \& i \& n \neq 0) \) \res

assumes \( e: \text{echelon-form } A \)
 and not-zero-iA: \( \text{¬ is-zero-row } i \ A \)
 and zero-j-A: \( \text{is-zero-row } j \ A \)
 shows \( \text{is-zero-row } j \ H \)

proof (auto simp add: is-zero-row-def is-zero-row-upt-k-def ncols-def)
 fix \( a \)
 have \( A \& j \& a = 0 \)
     using zero-j-A
     by (simp add: is-zero-row-def is-zero-row-upt-k-def ncols-def)
 show \( H \& j \& a = 0 \)
 proof (cases \( i \leq j \))
   case True
   have \( H \& j \& a = M \& j \& a \)
     unfolding \( H \) by (rule Hermite-reduce-above-preserves, simp add: True to-nat-mono')
   also have ... = 0 unfolding M mult-row-def using True A-ja by auto
   finally show \( ?\text{thesis} \).
 next
 let \( ?n = (\text{LEAST } n. \ A \& i \& n \neq 0) \)
 let \( ?R = \text{row-add } M \ j \ i \ ((\text{res } (M \& i \& ?n) (M \& j \& ?n) - M \& j \& ?n)) \div M \& i \& ?n) \)
 case False
 hence \( ji: j < i \) by simp
have $H \not\subseteq j \not\subseteq a = \not\exists R \not\subseteq j \not\subseteq a$

unfolding $H$ by (rule Hermite-reduce-above-works, auto simp add: ji to-nat-mono)
also have ... = 0
using ji A-ja not-zero-iA e echelon-form-condition1 zero-j-A
unfolding row-add-def M mult-row-def by blast
finally show \textit{thesis}.
qed

lemma Hermite-reduce-above-Least-eq-le:

\begin{align*}
\text{fixes } & \text{res ass i A} \\
\text{defines } & M : M \equiv \text{mult-row A i (ass (A \not\subseteq i \not\subseteq (\text{LEAST} n. \ A \not\subseteq i \not\subseteq n \not= 0)) \ div A} \\
& \not\subseteq i \not\subseteq (\text{LEAST} n. \ A \not\subseteq i \not\subseteq n \not= 0)) \\
\text{defines } & H : H \equiv \text{Hermite-reduce-above M (to-nat i) } \not\subseteq (\text{LEAST} n. \ A \not\subseteq i \not\subseteq n \not= 0) \\
\text{res} \\
\text{assumes } & i-ia : i < ia \\
\text{and not-zero-ia-H} : \neg \text{is-zero-row ia H} \\
\text{shows} & (\text{LEAST} n. \ A \not\subseteq ia \not\subseteq n \not= 0) = (\text{LEAST} n. \ H \not\subseteq ia \not\subseteq n \not= 0) \\
\text{proof} \quad (\text{rule Least-equality}) \\
\text{let } & \not\exists n = (\text{LEAST} n. \ H \not\subseteq ia \not\subseteq n \not= 0) \\
\text{have } & A \not\subseteq ia \not\subseteq ?n = M \not\subseteq ia \not\subseteq ?n \text{ unfolding M mult-row-def using i-ia by auto} \\
\text{also have ... = 0 } \text{ unfolding H} \\
& \text{by (rule Hermite-reduce-above-preserves[symmetric])} \\
& (\text{metis i-ia dual-order.strict-iff-order to-nat-mono'}) \\
\text{also have ... } \not\equiv 0 \text{ by (metis (mono-tags) LeastI is-zero-row-def' not-zero-ia-H)} \\
\text{finally show } A \not\subseteq ia \not\subseteq (\text{LEAST} n. \ H \not\subseteq ia \not\subseteq n \not= 0) \not\equiv 0 .
\end{align*}

next

fix y

assume A-ia-y : A \not\subseteq ia \not\subseteq y \not= 0

have H \not\subseteq ia \not\subseteq y = M \not\subseteq ia \not\subseteq y \text{ unfolding H} \\
& \text{by (rule Hermite-reduce-above-preserves)} \\
& (\text{metis i-ia dual-order.strict-iff-order to-nat-mono'}) \\
\text{also have ... } \not\equiv 0 \text{ unfolding M mult-row-def using i-ia A-ia-y by auto} \\
\text{finally show } (\text{LEAST} n. \ H \not\subseteq ia \not\subseteq n \not= 0) \leq y \text{ by (rule Least-le)}
\)

qed

lemma Hermite-reduce-above-Least-eq:

\begin{align*}
\text{fixes } & \text{res ass i A} \\
\text{defines } & M : M \equiv \text{mult-row A i (ass (A \not\subseteq i \not\subseteq (\text{LEAST} n. \ A \not\subseteq i \not\subseteq n \not= 0)) \ div A} \\
& \not\subseteq i \not\subseteq (\text{LEAST} n. \ A \not\subseteq i \not\subseteq n \not= 0)) \\
\text{defines } & H : H \equiv \text{Hermite-reduce-above M (to-nat i) } \not\subseteq (\text{LEAST} n. \ A \not\subseteq i \not\subseteq n \not= 0) \\
\text{res} \\
\text{assumes } & a : \text{ass-function ass} \\
\text{and not-zero-iA} : \neg \text{is-zero-row i A} \\
\text{shows} & (\text{LEAST} n. \ A \not\subseteq i \not\subseteq n \not= 0) = (\text{LEAST} n. \ H \not\subseteq i \not\subseteq n \not= 0) \\
\text{proof} \quad (\text{rule Least-equality[symmetric])}
\end{align*}
let \(?n=\text{(LEAST } n. \ A \ S i \ S n \neq 0)\)
have \(\text{Ain}: \ A \ S i \ S ?n \neq 0\)
  by (metis (mono-tags, lifting) \text{LeastI is-zero-row-def'} \text{not-zero-iA})
have \(H \ S i \ S ?n = M \ S i \ S ?n\)
  unfolding \(H\)
  by (rule \text{Hermite-reduce-above-preserves}, \text{simp})
also have \(\ldots \neq 0\) unfolding \(M\) mult-row-def by (auto \text{simp add: Ain ass-function-0[}OF \text{a]})
finally show \(H \ S i \ S ?n \neq 0\)
fix \(y\) assume \(H-iy: H \ S i \ S y \neq 0\)
show \(\text{(LEAST } n. \ A \ S i \ S n \neq 0) \leq y\)
proof (rule \text{Least-le}, \text{rule ccontr}, \text{simp})
  assume \(\text{Aiy}: \ A \ S i \ S y = 0\)
  have \(H \ S i \ S y = M \ S i \ S y\)
    unfolding \(H\)
    by (rule \text{Hermite-reduce-above-preserves}, \text{simp})
also have \(\ldots = 0\) unfolding \(Aiy\) by \text{simp}
finally show \(\text{False using } H-iy\) by \text{contradiction}
qed

\text{qed}

\text{lemma } \text{Hermite-reduce-above-Least-eq-ge:}
fixes \(\text{res} \ \text{ass i A}\)
defines \(\text{M}: \ M \equiv \text{mult-row } A \ i \ \text{(ass } (A \ S i \ S (\text{LEAST } n. \ A \ S i \ S n \neq 0)) \ S i \ S (\text{LEAST } n. \ A \ S i \ S n \neq 0))\)
defines \(\text{H}: \ H \equiv \text{Hermite-reduce-above } M \ \text{(to-nat } i) \ \text{(LEAST } n. \ A \ S i \ S n \neq 0)\) res
assumes \(c: \text{echelon-form } A\)
and \(\text{not-zero-iA}: \neg \text{is-zero-row } i A\)
and \(\text{not-zero-ia-A}: \neg \text{is-zero-row } ia A\)
and \(\text{not-zero-ia-H}: \neg \text{is-zero-row } ia H\)
and \(\text{ia-less-i}: \ ia < i\)
shows \(\text{(LEAST } n. \ H \ S ia \ S n \neq 0) = (\text{LEAST } n. \ A \ S ia \ S n \neq 0)\)
proof –
let \(?\text{least-H } = (\text{LEAST } n. \ H \ S ia \ S n \neq 0)\)
let \(?\text{least-A } = (\text{LEAST } n. \ A \ S ia \ S n \neq 0)\)
let \(?n= (\text{LEAST } n. \ A \ S i \ S n \neq 0)\)
let \(?\text{Ain } = A \ S i \ S ?n\)
let \(?R=row-add \ M \ ia \ i ((\text{res } (M \ S i \ S ?n) \ (M \ S ia \ S ?n)) - M \ S ia \ S ?n) \ S div \ M \ S i \ S ?n)\)
have \(\text{A-ia-least-A}: \ A \ S ia \ S ?\text{least-A} \neq 0\)
  by (metis (mono-tags, lifting) \text{LeastI is-zero-row-def'} \text{not-zero-ia-A})
have \(\text{H-ia-least-H}: \ H \ S ia \ S ?\text{least-H} \neq 0\)
  by (metis (mono-tags, lifting) \text{LeastI is-zero-row-def'} \text{not-zero-ia-H})
have \(\text{A-i-least-ia-0}: \ A \ S i \ S (\text{LEAST } n. \ A \ S ia \ S n \neq 0) = 0\)
proof –
have \((\text{LEAST } n. \ A \ \odot \ ia \ \odot n \neq 0) < (\text{LEAST } n. \ A \ \odot i \ \odot n \neq 0)\)
using \(e\) echelon-form-condition1 echelon-form-condition2-explicit
\(ia-less-i\) \(not-zero-i\) \(A\) by blast
thus \(?thesis\) using \(not-less-Least\) by blast
qed
have \(H-ia-least-A: H \ \odot ia \ ?least-A \neq 0\)
proof –
have \(H \ \odot ia \ ?least-A = ?R \ \odot ia \ ?least-A\)
unfolding \(H\)
by \(\text{rule Hermite-reduce-above-works}, \text{simp-all add}: ia-less-i \text{ to-nat-mono}\)
also have \(... \neq 0\) using \(ia-less-i\) unfolding row-add-def \(M\) mult-row-def
by \(\text{auto simp add}: A-i-least-ia-0 A-ia-least-A\)
finally show \(?thesis\).
qed
have \(A \ \odot ia \ ?least-H = 0\)
proof –
have \((\text{LEAST } n. \ A \ \odot ia \ \odot n \neq 0) < (\text{LEAST } n. \ A \ \odot i \ \odot n \neq 0)\)
using \(e\) echelon-form-condition1 echelon-form-condition2-explicit
\(ia-less-i\) \(not-zero-i\) \(A\) by blast
thus \(?thesis\) using \(not-less-Least least-H-le-least-A\)
by \(\text{metis} (\text{mono-tags}) \text{ dual-order} \cdot \text{strict-trans2}\)
qed
have \(A \ \odot ia \ ?least-H \neq 0\)
proof –
have \(ia-not-i: ia \neq i\) using \(ia-less-i\) by simp
have \(?R \ \odot ia \ ?least-H = H \ \odot ia \ ?least-H\)
unfolding \(H\)
by \(\text{rule Hermite-reduce-above-works}[\text{symmetric}], \text{simp-all add}: ia-less-i \text{ to-nat-mono}\)
also have \(... \neq 0\) by \(\text{rule H-ia-least-H}\)
finally have \(R-ia-least-H: ?R \ \odot ia \ ?least-H \neq 0\).
hence \(A \ \odot ia \ ?least-H + (\text{res (ass } (\text{?Ain}) \text{ div } ?Ain \ast ?Ain)\)
\((A \ \odot ia \ (\text{LEAST } n. \ A \ \odot i \ \odot n \neq 0)) - A \ \odot ia \ (\text{LEAST } n. \ A \ \odot i \ \odot n \neq 0))\)
\(\neq 0\)
using \(ia-not-i\) unfolding row-add-def \(M\) mult-row-def by auto
thus \(?thesis\) using \(ia-less-i\) \(A-i-least-H\) unfolding row-add-def \(M\) mult-row-def
by auto
qed
hence \(least-A-le-least-H: ?least-A \leq \text{?least-H}\) by \(\text{metis} (\text{poly-guards-query}\)
\text{Least-le})
show \(?thesis\) using \(least-A-le-least-H\) \(least-H-le-least-A\) by simp
qed

lemma Hermite-reduce-above-Least:
fixes \(res\) \(ass\) \(i\) \(A\)
defines \( M \equiv \text{mult-row } A \ i \ (\text{ass } (A \ \$ \ i \ (\text{LEAST } n. \ A \ \$ \ i \$n \neq 0))) \)
\( \text{div } A \ \$ \ i \ (\text{LEAST } n. \ A \ \$ \ i \$ n \neq 0)) \)
defines \( H \equiv \text{Hermite-reduce-above } M \ (\text{to-nat } i) \ i \ (\text{LEAST } n. \ A \ \$ \ i \$ n \neq 0) \) res
assumes \( e: \text{echelon-form } A \)
and \( a: \text{ass-function ass} \)
and \( \text{not-zero-iA: } \neg \text{is-zero-row } i \ A \)
and \( \text{not-zero-ia-A: } \neg \text{is-zero-row } ia \ A \)
and \( \text{not-zero-ia-H: } \neg \text{is-zero-row } ia \ H \)
shows \((\text{LEAST } n. \ H \ \$ \ ia \$ n \neq 0) = (\text{LEAST } n. \ A \ \$ \ ia \$ n \neq 0)\)
proof (cases \( ia<i \))
case True
show \(?thesis\)
unfolding \( H \ M \)
by (rule Hermite-reduce-above-Least-eq-ge[\text{OF } e \ \text{not-zero-iA } \\text{not-zero-ia-A } - True])
(metis \( H \ M \ \text{not-zero-ia-H} \))
next
case False
hence \( i\le ia \) by simp
show \(?thesis\)
proof (cases \( ia=i \))
case True
show \(?thesis\)
unfolding \( True \ H \ M \)
by (rule Hermite-reduce-above-Least-eq[\text{symmetric, OF } a \ \text{not-zero-iA}])
next
case False
hence \( i\ < \ ia \) using \( i\le ia \) by simp
show \(?thesis\)
unfolding \( H \ M \)
by (rule Hermite-reduce-above-Least-eq-le[\text{symmetric, OF } i\-ia], \text{metis } H \ M \ \text{not-zero-ia-H})
qed
qed

lemma \( \text{echelon-form-Hermite-of-condition2}: \)
fixes \( \text{res } a \ i \ A \)
defines \( M: M \equiv \text{mult-row } A \ i \ (\text{ass } (A \ \$ \ i \ (\text{LEAST } n. \ A \ \$ \ i \$ n \neq 0))) \ \text{div } A \ \$ \ i \ (\text{LEAST } n. \ A \ \$ \ i \$ n \neq 0)) \)
defines \( H: H \equiv \text{Hermite-reduce-above } M \ (\text{to-nat } i) \ i \ (\text{LEAST } n. \ A \ \$ \ i \$ n \neq 0) \) res
assumes \( e: \text{echelon-form } A \)
and \( a: \text{ass-function ass} \)
and \( \text{not-zero-iA: } \neg \text{is-zero-row } i \ A \)
and \( \text{ia-less-j: } ia < j \)
and \( \text{not-zero-ia-H: } \neg \text{is-zero-row } ia \ H \)
and \( \text{not-zero-j-H: } \neg \text{is-zero-row } j \ H \)
shows \((\text{LEAST } n. \ H \# \ ia \# n \neq 0) < (\text{LEAST } n. \ H \# j \# n \neq 0)\)

proof –

let \(?n\) = \((\text{LEAST } n. \ A \# i \# n \neq 0)\)

have \(\text{Ain}: \ A \# i \# \ ?n \neq 0\)
  by (metis (mono-tags) LeastI is-zero-row-def' not-zero-iA)

have \(\text{not-zero-j-A} \vdash \neg \text{is-zero-row} \ j \ A\)

unfolding \(H\ M\) by blast

have \(\text{not-zero-ia-A} \vdash \neg \text{is-zero-row} \ ia \ A\)
  using row-zero-A-impl-zero-H[OF \(\text{not-zero-iA} \) not-zero-ia-H]

unfolding \(H\ M\) by blast

have \(\text{Least-le-A} \vdash (\text{LEAST } n. \ A \# ia \# n \neq 0) < (\text{LEAST } n. \ A \# j \# n \neq 0)\)
  by (rule echelon-form-condition2-explicit[OF \(\text{ia-less-j} \) \(\text{not-zero-ia-A} \) \(\text{not-zero-j-A} \)])

ultimately show \(?\text{thesis}\) using Least-le-A by simp

next

case False

hence \(\text{ia-less-i} \vdash \ia\leq i\) by simp

show \(?\text{thesis}\) proof (cases \(i=\ia\))
  case True thus \(?\text{thesis}\)
    using Hermite-reduce-above-Least-eq[OF \(\text{a not-zero-iA} \) Least-le-A]
    using Hermite-reduce-above-Least-eq-le[OF \(\text{ia-less-j} \)]
    using not-zero-j-H unfolding \(H\ M\) by fastforce

next

case False

hence \(\text{ia-less-ia} \vdash \ia<i\) using \(\text{ia-less-i}\) by simp

have \(\text{Least-M-ia-A-ia} \vdash (\text{LEAST } n. \ H \# ia \# n \neq 0) = (\text{LEAST } n. \ H \# ia \# n \neq 0)\)
  unfolding \(H\ M\)

  by (rule Hermite-reduce-above-Least-eq-ge[OF \(\text{not-zero-iA} \) not-zero-ia-A - ia-less-ia])
  (metis \(H\ M\) not-zero-ia-A)

show \(?\text{thesis}\) proof (cases \(j<i\))
  case True
  have \(\text{Least-M-j} \vdash (\text{LEAST } n. \ H \# j \# n \neq 0) = (\text{LEAST } n. \ A \# j \# n \neq 0)\)

23
unfolding $H M$
\[(\text{metis } H M \text{ not-zero-j-H})\]
thus $\cong$thesis by (simp add: Least-H-ia-A-ia Least-le-A)
next
case False
hence $j$-ge-i: $j \geq i$ by auto
show $\cong$thesis
proof (cases $j = i$)
case True
have $(\text{LEAST } n. H \cdot j \cdot n \neq 0) = (\text{LEAST } n. A \cdot j \cdot n \neq 0)$
unfolding $H M$
using Hermite-reduce-above-Least-eq True a not-zero-iA by fastforce
thus $\cong$thesis by (simp add: Least-H-ia-A-ia Least-le-A)
next
case False
hence $j$-sg-i: $j > i$ using $j$-ge-i by simp
have $(\text{LEAST } n. H \cdot j \cdot n \neq 0) = (\text{LEAST } n. A \cdot j \cdot n \neq 0)$
unfolding $H M$
by (rule Hermite-reduce-above-Least-eq-le symmetric, OF $j$-sg-i)
\[(\text{metis } H M \text{ not-zero-j-H})\]
thus $\cong$thesis by (simp add: Least-H-ia-A-ia Least-le-A)
qed
qed
qed
qed

lemma echelon-form-Hermite-of-row:
assumes a: ass-function ass
and res-function res
and e: echelon-form $A$
shows echelon-form (Hermite-of-row-i ass res $A$ $i$)
proof (rule echelon-form-intro, auto simp add: Hermite-of-row-i-def Let-def)
fix $ia$ $j$
assume is-zero-row $i$ $A$ and is-zero-row $ia$ $A$ and $ia < j$
thus is-zero-row $j$ $A$ using echelon-form-condition1 [OF e] by blast
next
fix $ia$ $j$
assume is-zero-row $i$ $A$ and $ia < j$ and $\neg$ is-zero-row $ia$ $A$ and $\neg$ is-zero-row $j$ $A$
thus $(\text{LEAST } n. A \cdot ia \cdot n \neq 0) < (\text{LEAST } n. A \cdot j \cdot n \neq 0)$
using echelon-form-condition2 [OF e] by blast
next
fix $ia$ $j$
assume $\neg$ is-zero-row $i$ $A$
and is-zero-row $ia$ (Hermite-reduce-above
(mult-row $A$ $i$ (ass $A \cdot i \cdot (\text{LEAST } n. A \cdot i \cdot n \neq 0)$) div $A \cdot i$ $\cdot (\text{LEAST}$
n. \( A \notin i (\text{LEAST } n. A \notin i n \neq 0)\))

(to-nat \(i\)) \(i (\text{LEAST } n. A \notin i n \neq 0)\) \text{res}

\and \text{ia} < j

\text{thus} \(\text{is-zero-row } j\) \(\text{Hermite-reduce-above}\)

\(\text{mult-row } A \in i \text{(ass } (A \notin i n \text{ (LEAST } n. A \notin i n \neq 0)) \text{div } A \notin i n \text{ (LEAST } n. A \notin i n \neq 0)\))\)

(to-nat \(i\)) \(i (\text{LEAST } n. A \notin i n \neq 0)\) \text{res}

\text{using} \text{echelon-form-Hermite-of-condition1}[\text{OF e a}] \text{by blast}

\text{next}

\text{fix} \text{ia} j

\text{let} ?H=(\text{Hermite-reduce-above} \text{(mult-row } A \in i \text{(ass } (A \notin i n \text{ (LEAST } n. A \notin i n \neq 0)) \text{div } A \notin i n \text{ (LEAST } n. A \notin i n \neq 0)\))\)

(to-nat \(i\)) \(i (\text{LEAST } n. A \notin i n \neq 0)\) \text{res}

\text{assume} \neg \text{is-zero-row } i A

\and \text{ia} < j

\and \neg \text{is-zero-row ia } ?H

\and \neg \text{is-zero-row } j ?H

\text{thus} \(\text{LEAST } n. \text{?H} \notin i n \neq 0)\) \(\text{LEAST } n. \text{?H} \notin j n \neq 0)\)

\text{using} \text{echelon-form-Hermite-of-condition2}[\text{OF e a}] \text{by blast}

\text{qed}

\text{lemma} \text{echelon-form-fold-Hermite-of-row-i}:

\text{assumes} e : \text{echelon-form A and a: ass-function ass and r: res-function res}

\text{shows} \text{echelon-form (foldl \text{Hermite-of-row-i ass res} A \text{map from-nat \[0..<k\]})}

\text{proof (induct k)}

\text{case 0}

\text{thus} \text{?case by (simp add: e)}

\text{next}

\text{case (Suc k)}

\text{show} \text{?case by (simp, rule echelon-form-Hermite-of-row-i[OF a r Suc.hyps])}

\text{qed}

\text{ lemma} \text{echelon-form-Hermite-of-upt-row-i}:

\text{assumes} e : \text{echelon-form A and a: ass-function ass and r: res-function res}

\text{shows} \text{echelon-form (Hermite-of-upt-row-i A k ass res)}

\text{unfolding} \text{Hermite-of-upt-row-i-def}

\text{using} \text{echelon-form-fold-Hermite-of-row-i assms by auto}

\text{lemma} \text{echelon-form-Hermite-of}: \text{fixes} A::\{bezout-ring-div\} \text{ cols::\{mod-type\} rows::\{mod-type\}}

\text{assumes a: ass-function ass}

\text{and r: res-function res}

\text{and b: is-bezout-ext bezout}

\text{shows} \text{echelon-form (Hermite-of A ass res bezout)}

\text{unfolding} \text{Hermite-of-def Hermite-of-upt-row-i-def Let-def nrows-def}

\text{by (rule echelon-form-fold-Hermite-of-row-i[OF echelon-form-echelon-form-of[OF}
lemma in-ass-Hermite-of-row:
  assumes a: ass-function ass
  and res-function res
  and not-zero-i-A: ¬ is-zero-row i A
  shows (Hermite-of-row-i ass res A i) $ i $ (LEAST n. (Hermite-of-row-i ass res A i) $ i $ n $\neq 0) $\in$ range ass
unfolding Hermite-of-row-i-def using not-zero-i-A
proof (auto simp add: Let-def)
  let $?M=$(mult-row A i (ass (A $ i $ (LEAST n. A $ i $ n $\neq 0))) div A $ i $ (LEAST n. A $ i $ n $\neq 0)))
  let $?H=Hermite-reduce-above $?M (to-nat i) i (LEAST n. A $ i $ n $\neq 0) res
  let $?Ain=A $ i $ (LEAST n. A $ i $ n $\neq 0)
  have Ain: ?Ain $\neq 0$
    by (metis (mono-tags) LeastI is-zero-row-def' not-zero-i-A)
  have least-eq: (LEAST n. $?H $ i $ n $\neq 0) = (LEAST n. A $ i $ n $\neq 0)
    by (rule Hermite-reduce-above-Least-eq[OF a not-zero-i-A, symmetric])
  have $?H $ i $ (LEAST n. $?H $ i $ n $\neq 0) = $?M $ i $ (LEAST n. $?H $ i $ n $\neq 0)
    by (rule Hermite-reduce-above-preserves, simp)
  also have ... = ass (A $ i $ (LEAST n. A $ i $ n $\neq 0)) div A $ i $ (LEAST n. A $ i $ n $\neq 0))
    unfolding mult-row-def least-eq[unfolded mult-row-def] by simp
  also have ... = ass ?Ain
  proof (rule dvd-div-mult-self)
    show ?Ain dvd ass ?Ain
      using a unfolding ass-function-def associated-def by simp
  qed
  also have ... $\in$ range ass by simp
finally show $?H $ i $ (LEAST n. $?H $ i $ n $\neq 0) $\in$ range ass .
qed

lemma Hermite-of-upt-row-preserves-below:
  assumes i: to-nat a$\geq$k
  shows Hermite-of-upt-row-i A k ass res $ a $ b = A $ a $ b
  using i
proof (induct k)
  case 0
  thus ?case unfolding Hermite-of-upt-row-i-def by auto
next
  case (Suc k)
  show ?case
qed
lemma not-zero-Hermite-reduce-above:
  fixes ass i A
  defines M: \( M \equiv (\text{mult-row } A \ i \ (\text{ass} (A \ i \ (\text{LEAST } n. \ A \ i \ n \neq 0)) \ \text{div} \ A \ i \ (\text{LEAST } n. \ A \ i \ n \neq 0))) \)
  assumes not-zero-a-A: \( \neg \text{is-zero-row } a \ A \)
  and not-zero-i-A: \( \neg \text{is-zero-row } i \ A \)
  and e: \( \text{echelon-form } A \)
  and a: \( \text{ass-function } \text{ass} \)
  and n: \( n \leq \text{to-nat } i \)
  shows \( \neg \text{is-zero-row } a \ (\text{Hermite-reduce-above } M \ n \ i \ (\text{LEAST } n. \ A \ i \ n \neq 0)) \)
proof –
  let ?H = (\text{Hermite-reduce-above } M \ n \ i \ (\text{LEAST } n. \ A \ i \ n \neq 0) \ \text{res})
  let ?n=\text{LEAST } n. \ A \ a \ n \neq 0
  let ?m=\text{LEAST } n. \ A \ i \ n \neq 0
  have Aan: \( A\ a \ ?n \neq 0 \)
    by (metis \text{Hermite-reduce-above-preserve} True)
    also have \( \ldots \neq 0 \) unfolding \( \text{mult-row-def} \) using \( \text{ass-function-0} \[\text{OF } a] \ A\ a \ n \neq 0 \) by \text{auto}
  finally show \( \text{thesis} \) unfolding \text{is-zero-row-def} \text{is-zero-row-upt-k-def} \text{ncols-def} by \text{auto}
next
  let ?R=\text{row-add } M \ a \ i
    ((\text{res } (M \ i \ ?m) \ (M \ a \ ?m) - M \ a \ ?m) \ \text{div} \ M \ i \ ?m)
  case False
  hence a-n: \( \text{to-nat } a \ < \ n \) by \text{simp}
  have ai: \( a \ < \ i \)
    by (metis False dual-order tran nat-less-le nat-less-iff-or-eq nat-less-ord)
    also have \( \ldots \neq 0 \) by \( \text{echelon-form-condition2-explicit} \[\text{OF } e \ a \ \neg \text{zeror-a-A} \ \neg \text{zeror-i-A}] \)
  hence A\ i: \( A \ i \ (\text{LEAST } n. \ A \ a \ n \neq 0) = 0 \)
    by (metis \text{full-types} nat-less-Least)
  have a-not-i: \( \neg a \neq i \) by (metis False)
  have ?H \( a \ ?n = ?R \ a \ ?n \)
    by (\text{rule \text{Hermite-reduce-above-works} [OF } n \ a \ n \] )
  also have \( \ldots \neq 0 \) using \( \neg \text{a-not-i } A\ a \ A\ \text{Ain} \) unfolding \( \text{row-add-def} \) \( \text{M} \) \( \text{mult-row-def} \)
  finally show \( \text{thesis} \) unfolding \( \text{is-zero-row-def} \) \( \text{is-zero-row-upt-k-def} \) \( \text{ncols-def} \) by \text{auto}
qed
qed

27
lemma Least-Hermite-of-row-i:
  assumes i: ¬ is-zero-row i A
  and e: echelon-form A
  and a: ass-function ass
  shows (LEAST n. Hermite-of-row-i ass res A i n \neq 0) = (LEAST n. A i \neq 0)
proof –
  let ?M = mult-row A i (ass (A i (LEAST n. A i \neq 0)) div A i n)
  let ?H = Hermite-reduce-above ?M (to-nat i) (LEAST n. A i \neq 0) res
  have (LEAST n. Hermite-of-row-i ass res A i n \neq 0) = (LEAST n. ?H i n \neq 0)
    using i unfolding Hermite-of-row-i-def unfolding Let-def by auto
    also have ... = (LEAST n. A i n \neq 0)
      by (rule Hermite-reduce-above-Least[OF e a i])
    (rule not-zero-Hermite-reduce-above[OF i i e a], simp)
finally show ?thesis.
qed

lemma Least-Hermite-of-row-i2:
  assumes i: ¬ is-zero-row i A and k: ¬ is-zero-row k A
  and e: echelon-form A
  and a: ass-function ass
  shows (LEAST n. Hermite-of-row-i ass res A k i n \neq 0) = (LEAST n. A i \neq 0)
proof –
  let ?M = mult-row A k (ass (A k (LEAST n. A k \neq 0)) div A k n)
  let ?H = Hermite-reduce-above ?M (to-nat k) (LEAST n. A k n \neq 0) res
  have (LEAST n. Hermite-of-row-i ass res A k i n \neq 0) = (LEAST n. ?H i n \neq 0)
    using k unfolding Hermite-of-row-i-def unfolding Let-def by auto
    also have ... = (LEAST n. A i n \neq 0)
      by (rule Hermite-reduce-above-Least[OF e a k])
    (simp add: a e i k not-zero-Hermite-reduce-above)
finally show ?thesis.
qed

lemma Hermite-of-row-i-works:
  fixes i A ass
  defines n:n ≡ (LEAST n. A i n \neq 0)
  defines M:M ≡ (mult-row A i (ass (A i n) div A i n))
  assumes ai: a \leq i
  and i: ¬ is-zero-row i A
  shows Hermite-of-row-i ass res A i a b =
row-add M a i ((res (M $ i $ n)) (M $ a $ n))  
− M $ a $ n) div M $ i $ n) $ a $ b
proof 
let $?H$=Hermite-reduce-above M (to-nat i) i n res
have Hermite-of-row-i ass res A i $ a $ b = $?H $ a $ b
unfolding Hermite-of-row-i-def Let-def M n
also have ... = row-add M a i ((res (M $ i $ n)) (M $ a $ n)) 
− M $ a $ n) div M $ i $ n) $ a $ b
by (rule Hermite-reduce-above-works, auto simp add: ai to-nat-mono)
finally show $?thesis $.
qed

lemma Hermite-of-row-i-works2:
fixes i A ass
defines n:n ≡ (LEAST n. A $ i $ n ≠ 0)
defines M:M ≡ (mult-row A i (ass (A $ i $ n) div A $ i $ n))
assumes i: ¬ is-zero-row i A
shows Hermite-of-row-i ass res A i $ a $ b
proof 
let $?H$=Hermite-reduce-above M (to-nat i) i n res
have Hermite-of-row-i ass res A i $ i $ b = M $ i $ b
unfolding Hermite-of-row-i-def Let-def M n
also have ... = M $ i $ b by (rule Hermite-reduce-above-preserves, simp)
finally show $?thesis $.
qed

lemma Hermite-of-upt-row-preserves-nonzero-rows-ge:
assumes i: ¬ is-zero-row i A and i2: to-nat i≥k
shows ¬ is-zero-row i (Hermite-of-upt-row-i A k ass res)
proof 
let $?n$=LEAST n. A $ i $ n ≠ 0
have Ain: A $ i $ ?n $ n ≠ 0 by (metis (mono-tags) LeastI i is-zero-row-def')
have Hermite-of-upt-row-i A k ass res $ i $ ?n = A $ i $ ?n
by (rule Hermite-of-upt-row-preserves-below[OF i2])
also have ... $ ≠ 0$ by (metis (mono-tags) LeastI i is-zero-row-def')
finally have Hermite-of-upt-row-i A k ass res $ i $ (LEAST n. A $ i $ n ≠ 0) $ ≠ 0$.
thus $?thesis$ unfolding is-zero-row-def is-zero-row-upt-k-def ncols-def by auto
qed

lemma Hermite-of-upt-row-i-Least-ge:
assumes i: ¬ is-zero-row i A
and i2: to-nat i≥k
shows (LEAST n. Hermite-of-upt-row-i A k ass res $ i $ n ≠ 0) = (LEAST n. Hermite-of-upt-row-i A k ass res $ i $ n ≠ 0) = (LEAST n.
\[ A \neq i \neq n \neq 0 \]

**proof** (rule Least-equality)

let \(?n = \text{LEAST } n. A \neq i \neq n \neq 0\)

have \(A \neq i \neq n \neq 0\) by (metis (mono-tags) LeastI i is-zero-row-def')

have \(\text{Hermite-of-upt-row-i } A \ k \ \text{ass res } A \neq i \neq n \neq 0\) \(\neq 0\)

fix \(y\)

assume \(H\): \(\text{Hermite-of-upt-row-i } A \ k \ \text{ass res } A \neq i \neq n \neq 0\)

show \((\text{LEAST } n. A \neq i \neq n \neq 0) \leq y\)

qed

**lemma** \(\text{Hermite-of-upt-row-i-Least}\):

assumes \(iA\): \(\neg \text{is-zero-row } i \ A\)

and \(e\): \(\text{echelon-form } A\)

and \(a\): \(\text{ass-function ass}\)

and \(r\): \(\text{res-function res}\)

and \(k\): \(k \leq \text{nrows } A\)

shows \((\text{LEAST } n. \text{Hermite-of-upt-row-i } A \ k \ \text{ass res } A \neq i \neq n \neq 0) = (\text{LEAST } n. A \neq i \neq n \neq 0)\)

proof (cases to-nat \(i \geq k\))

\[ \text{case True} \]

thus \(\text{thesis using } \text{Hermite-of-upt-row-i-Least-ge } iA\) by blast

\[ \text{next case False} \]

hence \(i-less-k\): \(\text{to-nat } i < k\) by simp

thus \(\text{thesis using } e \ iA \ k\)

proof (induct \(k\) arbitrary: \(A\))

\[ \text{case 0} \]

thus \(\text{thesis}\)

unfolding \(\text{Hermite-of-upt-row-i-def}\) by simp

next case \((\text{Suc } k)\)

have \(k\): \(k \leq \text{nrows } A\) using Suc.prems unfolding nrows-def by simp

have \(k2\): \(k \leq \text{nrows } A\) using Suc.prems unfolding nrows-def by simp

def \(A' \equiv (\text{foldl } (\text{Hermite-of-row-i } \text{ass res } A \ (\text{map from-nat } [0..<k])))\)

have \(A'\)-def2: \(A' = \text{Hermite-of-upt-row-i } A \ k \ \text{ass res}\)

unfolding \(\text{Hermite-of-upt-row-i-def } A'\)-def ..

have \(e\): \(\text{echelon-form } A'\)

unfolding \(A'\)-def2

by (rule \text{echelon-form-Hermite-of-upt-row-i}[OF - a \ r], auto simp add: Suc.prems)

show \(\text{thesis}\)

proof (cases to-nat \(i = k\))

\[ \text{case True}\]

have \(i-fn-k\): \(\text{from-nat } k = i\) by (metis True from-nat-to-nat-id)

have \(\text{not-zero-i-A' } \equiv \neg \text{is-zero-row } i \ A'\)
unfolding \( A'\text{-def2} \)
by (rule Hermite-of-upt-row-preserves-nonzero-rows-ge, auto simp add: Suc.prems True)
have Hermite-of-upt-row-i A (Suc k) ass res = Hermite-of-row-i ass res A' (from-nat k)
unfolding Hermite-of-upt-row-i-def A'\text{-def} by auto
also have (LEAST n. ... $ i \$ n \neq 0) = (LEAST n. A' $ i \$ n \neq 0)
unfolding i-fn-k by (rule Least-Hermite-of-row-i [OF not-zero-i-A' e a])
also have ... = (LEAST n. A $ i \$ n \neq 0)
unfolding A'\text{-def2} by (rule Hermite-of-upt-row-i-Least-ge, auto simp add: True Suc.prems)
finally show \(?thesis\).
next
case False
hence i-less-k: to-nat i < k using Suc.prems by simp
hence i-less-k2: i < from-nat k
by (metis from-nat-mono from-nat-to-nat-id k2 nrows-def)
show \(?thesis\)
proof (cases is-zero-row (from-nat k) A')
case True
have H: Hermite-of-upt-row-i A (Suc k) ass res = Hermite-of-upt-row-i A k ass res
using True by (simp add: Hermite-of-upt-row-i-def Hermite-of-row-i-def A'-def Let-def )
show \(?thesis\) unfolding H by (rule Suc.hyps[OF i-less-k], auto simp add: Suc.prems k)
next
case False
have not-zero-i-A': ¬ is-zero-row i A'
using e False i-less-k2 echelon-form-condition1 by blast
have Hermite-of-upt-row-i A (Suc k) ass res = Hermite-of-row-i ass res A' (from-nat k)
unfolding Hermite-of-upt-row-i-def A'-def by auto
also have (LEAST n. ... $ i \$ n \neq 0) = (LEAST n. A' $ i \$ n \neq 0)
by (rule Least-Hermite-of-row-i2 [OF not-zero-i-A' False e a])
also have ... = (LEAST n. A $ i \$ n \neq 0)
unfolding A'-def2 by (rule Suc.hyps[OF i-less-k], auto simp add: Suc.prems k)
finally show \(?thesis\).
qed
qed
qed


lemma Hermite-of-upt-row-preserves-nonzero-rows:
assumes i: ¬ is-zero-row i A
and e: echelon-form A

31
and $a$: ass-function ass
and $r$: res-function res
and $k$: $k \leq \text{nrows } A$
shows $\neg \text{is-zero-row } i \,(\text{Hermite-of-upt-row-i } A \, k \, \text{ass })$

proof –
let $?n = \text{LEAST } n$, $A \, ?n \, i \, ?n \neq 0$
have $\text{Ain}: A \, ?n \, i \, ?n \neq 0$ by (metis (mono-tags) LeastI is-zero-row-def')
show $?\text{thesis}$
proof (cases $\text{to-nat } i \geq k$
  case True thus $?\text{thesis}$ using Hermite-of-upt-row-preserves-nonzero-rows-ge $i$
by blast
next
  case False
  hence $i\text{-less-}k$: $\text{to-nat } i < k$ by auto
  thus $?\text{thesis}$ using $i \, k$
proof (induct $k$
  case 0 thus $?\text{case}$ by (metis less-nat-zero-code)
  next
    case $(\text{Suc } k)$
    have $k\text{-nrows}: k \leq \text{nrows } A$ using $\text{Suc.prems}$ unfolding nrows-def by auto
    have $k\text{-nrows2}: k < \text{nrows } A$ using $\text{Suc.prems}$ unfolding nrows-def by auto
    def $A'\equiv (\text{foldl } (\text{Hermite-of-row-i } \text{ass } \text{res}) \, A \,(\text{map from-nat } [0..<k]))$
    have $A'\text{-def2}: A' = \text{Hermite-of-upt-row-i } A \, k \, \text{ass res}$
    unfolding Hermite-of-upt-row-i-def $A'\text{-def } ..$
    have $\text{least-}A'\text{-}A$: $(\text{LEAST } n. \, A' \, ?n \, i \, ?n \neq 0) = (\text{LEAST } n. \, A \, ?n \, i \, ?n \neq 0)$
    unfolding $A'\text{-def2}$
    by (rule Hermite-of-upt-row-i-Least[OF - e a r], auto simp add: $k\text{-nrows}$ $\text{Suc.prems}$)
    have $e$: $\text{echelon-form } A'$
      unfolding $A'\text{-def2}$ by (simp add: a e echelon-form-Hermite-of-upt-row-i $r$)
    show $?\text{case}$
    proof (cases $\text{to-nat } i = k$
      let $?M = \text{mult-row } A' \, i \,(\text{ass } (A' \, ?n \, i \, ?n \neq 0)) \,(\text{div } A' \, ?n \, i \, ?n \neq 0))$
      case True
      hence $fn-k-i$: $\text{from-nat } k = i$ by (metis from-nat-to-nat-id)
      have $not-zero-i-A'\equiv \neg \text{is-zero-row } i \, A'$
      by (unfold $A'\text{-def2}$
        (rule Hermite-of-upt-row-preserves-nonzero-rows-ge, auto simp add: True $\text{Suc.prems}$)
      have $A'\text{-i-}i$: $(A' \, ?n \, i \, ?n \neq 0) \neq 0$
      by (metis (mono-tags) LeastI is-zero-row-def' not-zero-i-$A'$)
    have $\text{Hermite-of-upt-row-i } A \,(\text{Suc } k) \, \text{ass res } ?n = $
    Hermite-of-row-i $\text{ass res } A' \,(\text{from-nat } k) \, ?n$
    unfolding Hermite-of-upt-row-i-def $A'\text{-def } \text{by simp}$
    also have $... = ?M \, ?n$ unfolding $fn-k-i$
    by (rule Hermite-of-row-i-works2[OF not-zero-i-$A'$])
    also have $... \neq 0$

32
using $A'$-i-l unfolding mult-row-def
by (simp add: ass-function-0 ![OF a] least-$A'$-A)

finally show ?thesis unfolding is-zero-row-def is-zero-row-upt-k-def ncols-def
by auto

next
case False
hence i-k: to-nat i < k by (metis Suc.prems(1) less-antisym)
hence i-k2: i < from-nat k using i-k Suc.prems
by (metis from-nat mono from-nat-to-nat-id k-nrows2 nrows-def)

have not-zero-i- $A'$: ¬ is-zero-row i $A'$ unfolding $A'$-def2
by (rule Suc.hyps ![OF i-k Suc.prems(2) k-nrows!])

thus ?thesis
by (metis mono-tags least-$A'$-A not-less-Least)

qed

let $?m$ = (LEAST n. $A'$ $\leq$ i $\leq$ n $\neq$ 0) $\neq$ 0 unfolding least-$A'$-A ![symmetric]
by (metis (mono-tags) LeastI is-zero-row-def' not-zero-i-$A'$)

have Akn: $A'$ $(\leq$ from-nat k $\leq$ i $\leq$ n $\neq$ 0) = 0
proof -
  have (LEAST n. $A'$ $\leq$ i $\leq$ n $\neq$ 0) $<$ (LEAST n. $A'$ $(\leq$ from-nat k $\leq$ n $\neq$ 0)
  $\neq$ 0)
  by (rule echelon-form-condition2-explicit ![OF e i-k2 not-zero-i-$A'$ False])
  thus ?thesis by (metis mono-tags least-$A'$-A not-less-Least)
qed

let $?M$ = mult-row $A'$ $(\leq$ from-nat k $\leq$ i $\leq$ n $\neq$ 0)

have Hermite-of-upt-row-i $A$ (Suc k) ass res $\leq$ i $\leq$ ?n =
  Hermite-of-row-i ass res $A'$ $(\leq$ from-nat k $\leq$ i $\leq$ ?n)
unfolding Hermite-of-upt-row-i-def $A'$-def by simp
also have ...
  = row-add (mult-row $A'$ $(\leq$ from-nat k $\leq$ i $\leq$ ?m)) i (from-nat k)
  (res ![?M $\leq$ from-nat k $\leq$ ?m] ![i $\leq$ ?m] ![?M $\leq$ i $\leq$ ?m])
  div ![?M $\leq$ from-nat k $\leq$ ?m] ![i $\leq$ ?m] ![?M $\leq$ i $\leq$ ?m])
  div ![?M $\leq$ from-nat k $\leq$ ?m] ![i $\leq$ ?m] ![?M $\leq$ i $\leq$ ?m])

by (rule Hermite-of-row-i-works ![OF i-k2 False])
also have ... $\neq$ 0 using i-k2 Akn unfolding row-add-def mult-row-def
by auto

finally show ?thesis unfolding is-zero-row-def is-zero-row-upt-k-def
ncols-def by auto

next
case True
have Hermite-of-upt-row-i $A$ (Suc k) ass res = Hermite-of-upt-row-i $A$ k
ass res
thus ?thesis using not-zero-i-$A'$ unfolding $A'$-def2 by simp
qed

qed
lemma Hermite-of-upt-row-i-in-range:
fixes k ass res
assumes not-zero-i-A: ¬ is-zero-row i A
and e: echelon-form A
and a: ass-function ass
and r: res-function res
and k: to-nat i < k
and k2: k ≤ nrows A
shows Hermite-of-upt-row-i A k ass res $(\text{LEAST } n. A \$_i n \neq 0) \in \text{range ass}
using k not-zero-i-A k2
proof (induct k)
case 0
thus ?case by auto
next
case (Suc k)
  have k: k ≤ nrows A using Suc.prems unfolding nrows-def by simp
  have k2: k < nrows A using Suc.prems unfolding nrows-def by simp
  have A′-def: A′ = Hermite-of-upt-row-i A k ass res unfolding Hermite-of-upt-row-i-def
  have not-zero-A′: ¬ is-zero-row i A′
    using Hermite-of-upt-row-preserves-nonzero-rows[OF not-zero-i-A e a r k]
  unfolding A′-def by simp
  have e-A′: echelon-form A′ by (metis A′-def a e echelon-form-fold-Hermite-of-row-i
r)
  have least-eq: (\text{LEAST } n. (Hermite-of-row-i ass res A′ \$_i i) \$_i n \neq 0) = (\text{LEAST } n. A′ \$_i n \neq 0)
    by (rule Least-Hermite-of-row-i[OF not-zero-A′ e-A′ a])
  have least-eq2: (\text{LEAST } n. A′ \$_i n \neq 0) = (\text{LEAST } n. A \$_i n \neq 0)
    unfolding A′-def2
    by (rule Hermite-of-upt-row-i-Least[OF not-zero-i-A e a r k])
show ?case
proof (cases to-nat i = k)
case True
  have fn-k-i: from-nat k = i by (metis True from-nat-to-nat-id)
  have (Hermite-of-upt-row-i A (Suc k) ass res) \$_i n \neq 0)
   = (Hermite-of-row-i ass res A′ \$_i i) \$_i n \neq 0
      unfolding Hermite-of-upt-row-i-def
    by (simp add: A′-def fn-k-i)
also have ... = (Hermite-of-row-i ass res A′ \$_i i) \$_i n \neq 0)
unfolding least-eq least-eq2 ..
also have ... ∈ range ass by (rule in-ass-Hermite-of-row[OF a r not-zero-A'])
finally show ?thesis.
next
case False
hence i-less-k: to-nat i < k using Suc.prems by auto
hence i-less-k2: i < from-nat k using Suc.prems
by (metis from-nat mono from-nat-to-nat-id k2 nrows-def)
show ?thesis
proof (cases is-zero-row (from-nat k) A')
case True
have Hermite-of-upt-row-i A (Suc k) ass res = Hermite-of-upt-row-i A k ass res
using True by (simp add: Hermite-of-upt-row-i-def Hermite-of-row-i-def A'-def Let-def)
thus ?thesis using Suc.hyps not-zero-i-A k i-less-k by auto
next
case False
have (Hermite-of-upt-row-i A (Suc k) ass res) $ i \#$ (LEAST n. A $ i \# n \neq 0)
= (Hermite-of-row-i ass res A' (from-nat k)) $ i \#$ (LEAST n. A $ i \# n \neq 0)
unfolding Hermite-of-upt-row-i-def A'-def by auto
also have ... = A' $ i \#$ (LEAST n. A $ i \# n \neq 0)
proof (rule Hermite-of-row-preserves-previous-cols[OF - False e-A']
show (LEAST n. A $ i \# n \neq 0) < (LEAST n. A' $ mod-type-class.from-nat k $ n \neq 0)
unfolding least-eq2[ symmetric]
by (rule echelon-form-condition2-explicit[OF e-A' i-less-k2 not-zero-A'
False])
qed
also have ... ∈ range ass
unfolding A'-def using Suc.prems Suc.hyps
unfolding Hermite-of-upt-row-i-def using i-less-k by auto
finally show ?thesis.
qed
qed
qed

lemma Hermite-of-upt-row-preserves-zero-rows-ge:
assumes i: is-zero-row i A
and k: k ≤ nrows A
and ik: to-nat i≥k
shows is-zero-row i (Hermite-of-upt-row-i A k ass res)
proof (unfold is-zero-row-def', clarify)
fix j
have Hermite-of-upt-row-i A k ass res $ i \# j = A $ i \# j

35
by \(\text{metis \ Hermite-of-upt-row-preserves-below \ ik}\)
also have \(\ldots = 0\) using \(i\) unfolding \(\text{is-zero-row-def \ is-zero-row-upt-k-def \ ncols-def}\)
by \(\text{simp}\)
finally show \(\text{Hermite-of-upt-row-i} \ A \ k \ \text{ass} \ \text{res} \ S \ i \ S \ j = 0\).
qed

lemma \(\text{Hermite-of-upt-row-preserves-zero-rows}:\)
fixes \(A::a:\{\text{bezout-ring-div}\} \ \text{`cols::\{mod-type\}} \ \text{`rows::\{mod-type\}}\)
assumes \(i: \text{is-zero-row} \ i \ A\)
and \(e: \text{echelon-form} \ A \ \text{and} \ a: \text{ass-function} \ \text{ass} \ \text{and} \ v: \text{res-function} \ \text{res} \ \text{and} \ k: k\ \leq \ n\text{rows} \ A\)
shows \(\text{is-zero-row} \ i \ (\text{Hermite-of-upt-row-i} \ A \ k \ \text{ass} \ \text{res})\)
proof (cases to-nat \(i \geq k\))
  case True
  show \(\text{?thesis}\) by (rule \(\text{Hermite-of-upt-row-preserves-zero-rows-ge} [OF \ i \ k \ True]\))
next
  case False
  hence \(i-k: \text{to-nat} \ i < k\) by \(\text{simp}\)
  show \(\text{?thesis}\) using \(k \ i-k\)
  proof (induct \(k\))
    case 0
    thus \(\text{?case}\) unfolding \(\text{Hermite-of-upt-row-i-def}\) by (simp add: \(i\))
  next
    case (Suc \(k\))
    have \(k: k \leq \text{nrows} \ A\) using Suc.prems unfolding nrows-def by auto
    have \(k2: k < \text{nrows} \ A\) using Suc.prems unfolding nrows-def by simp
    def \(A':=\text{foldl} (\text{Hermite-of-row-i} \ \text{ass} \ \text{res}) \ A \ (\text{map} \ \text{from-nat} \ [0..<k])\)
    have \(A'-def2: A' = \text{Hermite-of-upt-row-i} \ A \ k \ \text{ass} \ \text{res}\)
      unfolding \(\text{Hermite-of-upt-row-i-def} \ A'-def\)
    show \(\text{?case}\) unfolding \(\text{is-zero-row-def}\)'
    proof (clarify, cases to-nat \(i = k\))
      fix \(j\)
      case True
      have \(fn-k-i: \text{from-nat} \ k = i\) by (metis True from-nat-to-nat-id)
      have \(\text{Hermite-of-upt-row-i} \ (\text{Suc} \ k) \ \text{ass} \ \text{res} = \(\text{Hermite-of-row-i} \ \text{ass} \ \text{res} \ A' \ i)\)
        unfolding \(\text{Hermite-of-upt-row-i-def}\)
      by (simp add: \(A'-def \ fn-k-i)\)
      moreover have \(\text{is-zero-row} \ i \ (\text{Hermite-of-upt-row-i} \ A \ k \ \text{ass} \ \text{res})\)
        by (rule \(\text{Hermite-of-upt-row-preserves-zero-rows-ge} [OF \ i \ k]\), simp add: \(True)\)
      ultimately show \(\text{Hermite-of-upt-row-i} \ (\text{Suc} \ k) \ \text{ass} \ \text{res} \ S \ i \ S \ j = 0\)
        unfolding \(\text{is-zero-row-def} \ A'-def2 \ \text{Hermite-of-row-i-def}\) by auto
    next
    fix \(j\)
    case False
    hence \(i-less-k: \text{to-nat} \ i < k\) using Suc.prems by auto
    hence \(i-less-k2: i < \text{from-nat} \ k\) using Suc.prems

36
by (metis from-nat-mono from-nat-to-nat-id k2 nrows-def)

show (Hermite-of-upt-row-i A (Suc k) ass res) $i \leq j = 0$

proof (cases is-zero-row (from-nat k) A')
  case True
  have is-zero-row i (Hermite-of-upt-row-i A k ass res) by (metis Suc.hyps i-less-k k)
  moreover have Hermite-of-upt-row-i A (Suc k) ass res $i \leq j = \text{Hermite-of-upt-row-i}$
  A k ass res $i \leq j$
  using True by (simp add: Hermite-of-upt-row-i-def Hermite-of-row-i-def A'-def Let-def)
  ultimately show ?thesis unfolding is-zero-row-def by auto
  next
  case False
  have is-zero-row i (Hermite-of-upt-row-i A k ass res) by (metis Suc.hyps i-less-k k)
  moreover have ~ is-zero-row i (Hermite-of-upt-row-i A k ass res)
  using echelon-form-condition1
  by (metis A'-def2 False a r echelon-form-Hermite-of-upt-row-i i-less-k2)
  ultimately show ?thesis by contradiction
  qed

qed

lemma Hermite-of-preserves-zero-rows:
  fixes A::'a::{bezout-ring-div} "'cols::{mod-type} "'rows::{mod-type}
  assumes i: is-zero-row i (echelon-form-of A bezout)
  and a: ass-function ass
  and r: res-function res
  and b: is-bezout-ext bezout
  shows is-zero-row i (Hermite-of A ass res bezout)
  unfolding Hermite-of-def Let-def
  by (rule Hermite-of-upt-row-preserves-zero-rows[OF echelon-form-echelon-form-of(OF b) a r])
  (auto simp add: nrows-def)

lemma Hermite-of-Least:
  fixes A::'a::{bezout-ring-div} "'cols::{mod-type} "'rows::{mod-type}
  assumes i: ~ is-zero-row i (Hermite-of A ass res bezout)
  and a: ass-function ass
  and r: res-function res
  and b: is-bezout-ext bezout
  shows (LEAST n. Hermite-of A ass res bezout $i \leq n \neq 0) = (LEAST n.
  (echelon-form-of A bezout) $i \leq n \neq 0)
  proof
  have non-zero-i-eA: ~ is-zero-row i (echelon-form-of A bezout)
    using Hermite-of-preserves-zero-rows[OF - a r b] i by auto
  have e: echelon-form (echelon-form-of A bezout) by (rule echelon-form-echelon-form-of[OF b])

37
lemma in-associates-Hermite-of:
  fixes A::'a::{bezout-ring-div} "cols::{mod-type} "'rows::{mod-type}
  assumes a: ass-function ass and r: res-function res and b: is-bezout-ext bezout
  shows Hermite-of A ass res bezout $ i $ (LEAST n. Hermite-of A ass res bezout $ i $ n ≠ 0) ∈ range ass
proof −
  have non-zero-i-eA: ¬ is-zero-row i (echelon-form-of A bezout)
    using Hermite-of-preserves-zero-rows[OF - a r b] i by auto
  have e: echelon-form (echelon-form-of A bezout)
    by (rule echelon-form-echelon-form-of[OF b])
  have least: (LEAST n. Hermite-of A ass res bezout $ i $ n ≠ 0) = (LEAST n.
    (echelon-form-of A bezout) $ i $ n ≠ 0)
    by (rule Hermite-of-Least[OF i a r b])
  show ?thesis unfolding least
  unfolding Hermite-of-def Let-def
    by (rule Hermite-of-upt-row-i-in-range[OF non-zero-i-eA e a r]
      (auto simp add: to-nat-less-card nrows-def))
qed

lemma Hermite-of-row-i-range-res:
  assumes ji: j<i and not-zero-i-A: ¬ is-zero-row i A and r: res-function res
  shows Hermite-of-row-i ass res A i $ j $ (LEAST n. Hermite-of A ass res bezout
    $ i $ n ≠ 0) ∈ range (res (Hermite-of-row-i ass res A i $ i $ (LEAST n. A $ i $ n ≠ 0))
proof −
  let ?n=(LEAST n. A $ i $ n ≠ 0)
  def M==(mult-row A i (ass (A $ i $ ?n) div A $ i $ ?n))
  let ?R=(row-add M j i ((res (M $ i $ ?n) (M $ j $ ?n)
    − M $ j $ ?n)) div M $ i $ ?n)
  have Hii: Hermite-of-row-i-ass res A i $ i $ ?n = M $ i $ ?n
    unfolding M-def by (rule Hermite-of-row-i-works2[OF not-zero-i-A])
  have rw: Hermite-of-row-i-ass res A i $ j $ ?n = ?R $ j $ ?n
    unfolding M-def by (rule Hermite-of-row-i-works[OF ji not-zero-i-A])
  show ?thesis
proof −
  have ∀ b ba. ∃ bb. ba + bb * b = res b ba
    using rw unfolding res-function-def by metis
  thus ?thesis using rw unfolding image-def Hii row-add-def by auto
    (metis (lifting) add-diff-cancel-left div-mult-self1-is-id mult.commute mult-eq-0-iff)
qed
qed
lemma Hermite-of-upt-row-i-in-range-res:
  fixes k ass res
  assumes not-zero-i-A: ∼ is-zero-row i A
  and e: echelon-form A
  and a: ass-function ass
  and r: res-function res
  and k: lo-nat i < k
  and k2: k ≤ nrows A
  and j: j < i
  shows Hermite-of-upt-row-i A k ass res $ j $ (LEAST n. A $ i $ $ n \neq 0 $)
  ∈ range (res (Hermite-of-upt-row-i A k ass res $ i $ (LEAST n. A $ i $ $ n \neq 0 $)));
  using k not-zero-i-A k2
proof (induct k)
case 0 thus ?case by auto
next
case (Suc k)
  let $ \bar{n} $= (LEAST n. A $ i $ $ n \neq 0 $)
  have k: k ≤ nrows A using Suc.prems unfolding nrows-def by simp
  have k2: k < nrows A using Suc.prems unfolding nrows-def by simp
  def A' := (foldl (Hermite-of-row-i ass res) A (map from-nat [0..<k]))
  have A'-def2: A' = Hermite-of-upt-row-i A k ass res unfolding Hermite-of-upt-row-i-def
  A'-def ..
  def M := (mult-row A' i (ass (A' $ i $ $ \bar{n} $) (LEAST n. A' $ i $ $ n \neq 0 $)) div A' $ i $ $ \bar{n} $) (LEAST n. A' $ i $ $ n \neq 0 $));
  have not-zero-A': ∼ is-zero-row i A'
    using Hermite-of-upt-row-preserves-nonzero-rows[OF not-zero-i-A e a r k]
    unfolding A'-def Hermite-of-upt-row-i-def by simp
  have e-A': echelon-form A' by (metis A'-def a e echelon-form-fold-Hermite-of-row-i r)
  have least-eq: (LEAST n. (Hermite-of-row-i ass res A' $ i $) $ i $ $ n \neq 0 $) = (LEAST n. A' $ i $ $ n \neq 0 $)
    by (rule Least-Hermite-of-row-i[OF not-zero-A' e-A' a])
  have least-eq2: (LEAST n. A' $ i $ $ n \neq 0 $) = (LEAST n. A $ i $ $ n \neq 0 $)
    unfolding A'-def2
    by (rule Least-Hermite-of-row-i-Least[OF not-zero-i-A e a r k])
  show ?case
proof (cases to-nat i = k)
case True
  have fn-k-i: from-nat k = i by (metis True from-nat-to-nat-id)
  have H-rw: (Hermite-of-upt-row-i A (Suc k) ass res $ i $ (LEAST n. A $ i $ $ n \neq 0 $))
    = (Hermite-of-row-i ass res A' $ i $ $ i $ (LEAST n. A' $ i $ $ n \neq 0 $))
    by (simp add: Hermite-of-upt-row-i-def A'-def fn-k-i least-eq2[unfolded A'-def])
  have (Hermite-of-upt-row-i A (Suc k) ass res) $ j $ (LEAST n. A $ i $ $ n \neq 0 $)
    = (Hermite-of-row-i ass res A' $ i $) $ j $ (LEAST n. A $ i $ $ n \neq 0 $)
    unfolding Hermite-of-upt-row-i-def
by (simp add: A'-def fn-k-i)
also have ... = (Hermite-of-row-i ass res A' i) $ j $ (LEAST n. A' $ i $ n ≠ 0)
unfolding least-eq2 ..
also have ... ∈ range (res (Hermite-of-row-i ass res A' i $ i $ (LEAST n. A' $ i $ n ≠ 0)))
by (rule Hermite-of-row-i-range-res[OF j not-zero-A' r])
also have ... = range ((res (Hermite-of-upt-row-i A (Suc k) ass res $ i $ (LEAST n. A $ i $ n ≠ 0)))))
unfolding H-rw ..
finally show ?thesis .

next
case False
hence i-less-k: to-nat i < k using Suc.prems by auto
hence i-less-k2: i < from-nat k using Suc.prems
by (metis from-nat-mono from-nat-to-nat-id k2 nrows-def)
show ?thesis
proof (cases is-zero-row (from-nat k) A')
case True
have Hermite-of-upt-row-i A (Suc k) ass res = Hermite-of-upt-row-i A k ass res
using True by (simp add: Hermite-of-upt-row-i-def A'-def Let-def)
thus ?thesis using Suc.hyps not-zero-i-A k i-less-k by auto
next
case False
have H-rw: (Hermite-of-upt-row-i A (Suc k) ass res $ i $ (LEAST n. A $ i $ n ≠ 0)) =
A' $ i $ (LEAST n. A $ i $ n ≠ 0)
proof (auto simp add: Hermite-of-upt-row-i-def A'-def[ symmetric],
rule Hermite-of-row-preserves-previous-cols[OF - False e-A'])
have (LEAST n. A'$ i $ i $ n ≠ 0) < (LEAST n. A'$ mod-type-class.from-nat k $ n ≠ 0)
by (rule echelon-form-condition2-explicit[OF e-A' i-less-k2 not-zero-A' False])
thus (LEAST n. A'$ i $ i $ n ≠ 0) < (LEAST n. A'$ mod-type-class.from-nat k $ n ≠ 0)
unfolding least-eq2 .
qed
have (Hermite-of-upt-row-i A (Suc k) ass res) $ j $ (LEAST n. A $ i $ n ≠ 0) =
(Hermite-of-row-i ass res A' (from-nat k)) $ j $ (LEAST n. A $ i $ n ≠ 0)
unfolding Hermite-of-upt-row-i-def A'-def by auto
also have ... = A' $ j $ (LEAST n. A $ i $ n ≠ 0)
proof (rule Hermite-of-row-preserves-previous-cols[OF - False e-A'])
show (LEAST n. A$ i $ i $ n ≠ 0) < (LEAST n. A$ mod-type-class.from-nat k $ n ≠ 0)
unfolding least-eq2[ symmetric]
by (rule echelon-form-condition2-explicit[OF e-A' i-less-k2 not-zero-A'])
False]
qed
also have ... ∈ range (res (Hermite-of-upt-row-i A k ass res $ i $ (LEAST n. A $ i $ n $ \neq 0))))
unfolding A'-def2
by (rule Suc.hyps[OF i-less-k], auto simp add: Suc.prems k)
also have ... = range (res (Hermite-of-upt-row-i A (Suc k) ass res $ i $ (LEAST n. A $ i $ n $ \neq 0))))
unfolding H-rw A'-def2 .
finally show ?thesis .
qed
qed
qed

lemma in-residues-Hermite-of:
fixes A::a::{bezout-ring-div} °{cols::{mod-type} °{rows::{mod-type}}
assumes a: ass-function ass
and r: res-function res
and b: is-bezout-ext bezout
and i: ¬ is-zero-row i (Hermite-of A ass res bezout)
and ji: j < i
shows Hermite-of A ass res bezout $ j $ (LEAST n. Hermite-of A ass res bezout $ i $ n $ \neq 0)
∈ range (res (Hermite-of A ass res bezout $ i $ (LEAST n. Hermite-of A ass res bezout $ i $ n $ \neq 0)))
proof –
have non-zero-i-eA: ¬ is-zero-row i (echelon-form-of A bezout)
  using Hermite-of-preserves-zero-rows[OF - a r b] i by auto
have e: echelon-form (echelon-form-of A bezout)
  by (rule echelon-form-echelon-form-of[OF b])
have least: (LEAST n. Hermite-of A ass res bezout $ i $ n $ \neq 0) = (LEAST n. (echelon-form-of A bezout) $ i $ n $ \neq 0)
  by (rule Hermite-of-Least[OF i a r b])
show ?thesis unfolding least
unfolding Hermite-of-def Let-def
by (rule Hermite-of-upt-row-i-in-range-res[OF non-zero-i-eA e a r - ji])
(auto simp add: to-nat-less-card nrows-def)
qed

lemma Hermite-Hermite-of:
assumes a: ass-function ass
and r: res-function res
and b: is-bezout-ext bezout
shows Hermite (range ass) (\lambda c. range (res c)) (Hermite-of A ass res bezout)
proof (rule Hermite-intro, auto)
show Complete-set-non-associates (range ass)
  by (simp add: ass-function-Complete-set-non-associates a)
show Complete-set-residues \((\lambda c. \text{range } (\text{res } c))\)
  by (simp add: \text{r resid-function-Complete-set-residues})
show echelon-form \((\text{Hermite-of } A \text{ ass res bezout})\)
  by (simp add: \text{a b echelon-form-Hermite-of } r)
fix \(i\)
assume \(i: \neg \text{is-zero-row } i \) (\text{Hermite-of } A \text{ ass res bezout})
show \(\text{Hermite-of } A \text{ ass res bezout } i \) (\text{LEAST } n. \text{Hermite-of } A \text{ ass res bezout } \$ i \$ \$ n \neq 0 \) \(\in\) \text{range ass}
  by (rule \text{in-associates-Hermite-of} [OF \text{a r b i}])

next
fix \(i\) \(j\)
assume \(i: \neg \text{is-zero-row } i \) (\text{Hermite-of } A \text{ ass res bezout}) \text{ and } \(j: j < i\)
show \(\text{Hermite-of } A \text{ ass res bezout } j \) (\text{LEAST } n. \text{Hermite-of } A \text{ ass res bezout } \$ i \$ \$ n \neq 0 \)
  \(\in\) \text{range } (\text{res } (\text{Hermite-of } A \text{ ass res bezout } i \$ i \$ \$ n \neq 0 \))
  by (rule \text{in-residues-Hermite-of} [OF \text{a r b i j}])
qed

1.6.2 Proving that the Hermite Normal Form is computed by means of elementary operations

lemma invertible-Hermite-reduce-above:
  assumes \(n: n \leq \text{to-nat } i\)
  shows \(\exists P. \text{invertible } P \land \text{Hermite-reduce-above } A \text{ n } i \text{ j res } = P \ast\ast A\)
  using \(n\)
proof (induct \(n\) arbitrary: \(A\))
case 0 thus \(?case\) by (auto, \text{metis invertible-def matrix-mul-lid})
next
case (Suc \(n\))
let \(?R=(\text{row-add } A \text{ (from-nat } n \) i ((\text{res } (A \$ i \$ j) (A \$ \text{from-nat } n \$ j) \text{ div } A \$ i \$ j)))\)
obtain \(Q\) where \(\text{inv-Q: invertible } Q \text{ and } \text{H-QR: Hermite-reduce-above } ?R \text{ n } i \text{ j res } = Q \ast\ast \?R\)
  using Suc.hyps Suc.prems by auto
let \(?P=(\text{row-add } \text{ (mat } 1 \) \text{ from-nat } n \) i ((\text{res } (A \$ i \$ j) (A \$ \text{from-nat } n \$ j) \text{ div } A \$ i \$ j))
\text{ div } A \$ \text{i \$ j}))\)
have \(\text{inv-P: invertible } \?P\)
proof (rule \text{invertible-row-add})
  show \(\text{mod-type-class,from-nat } n \neq i\)
    by (metis Suc.prems Suc-le-eq \text{add-to-nat-def from-nat-mono less-irrefl monoid-add-class.add.right-neutral to-nat-0 to-nat-less-card})
qed
have \(\text{inv-QP: invertible } (Q \ast\ast \?P)\) by (metis \text{inv-P inv-Q invertible-mult})
have \(\text{Hermite-reduce-above } A \text{ (Suc } n \) i \text{ j res } = \text{Hermite-reduce-above } \?R \text{ n } i \text{ j res}\)
  by (auto simp add: \text{Let-def})
also have \(\ldots = Q \ast\ast \?R\) unfolding \(\text{H-QR}\) ..
also have \(\ldots = Q \ast\ast (\?P \ast\ast A)\) by (subst \text{row-add-mat-1[symmetric], rule refl})
also have \(\ldots = (Q \ast\ast \?P) \ast\ast A\) by (simp add: matrix-mul-assoc)
finally show \(?case\) using \(\text{inv-QP}\) by auto
lemma invertible-Hermite-of-row-i:
  assumes a: ass-function ass
  shows ∃ P. invertible P ∧ Hermite-of-row-i ass res A i = P ** A

unfolding Hermite-of-row-i-def
proof (auto simp add: Let-def, metis invertible-def matrix-mul-lid)
  let ?n = LEAST n. A $ i $ n ≠ 0
  let ?M = mult-row A i (ass (A $ i $ ?n) div A $ i $ ?n)
  let ?P = mult-row (mat 1) i (ass (A $ i $ ?n) div A $ i $ ?n)

have ass-dvd: ass ?Ain dvd ?Ain using a unfolding ass-function-def associated-def by simp
have ass-dvd': ?Ain dvd ass ?Ain using a unfolding ass-function-def associated-def by simp
assume iA: ¬ is-zero-row i A
have Ain-0: A $ i $ ?n ≠ 0 by (metis mono-tags LeastI iA is-zero-row-def)
  have ass-Ain-0: ass (A $ i $ ?n) ≠ 0 by (metis Ain-0 ass-dvd dvd-0-left-iff)

  have ?Ain div ass ?Ain using a unfolding dvd-div-mult-self[OF ass-dvd]
  also have ... = (ass ?Ain) div ass ?Ain unfolding dvd-div-mult-self[OF ass-dvd'] ..
  also have ... = 1 using ass-Ain-0 by auto
    by (rule div-mult-swap[OF OF ass-dvd])
  also have ... = ?Ain div ?Ain unfolding dvd-div-mult-self[OF ass-dvd] ..
  also have ... = 1 using ?Ain-0 by simp
qed

obtain Q where inv-Q: invertible Q and H-QM: Hermite-reduce-above ?M (to-nat i) i ?n res = Q ** ?M
  using invertible-Hermite-reduce-above by blast
have inv-QP: invertible (Q**?P)
  by (metis inv-P inv-Q invertible-mult)
have Hermite-reduce-above ?M (to-nat i) i ?n res = Q ** ?M by (rule H-QM)
also have ... = Q ** (?P ** A) by (subst mult-row-mat-1[symmetric], rule refl)
also have ... = (?Q ** ?P) ** A by (simp add: matrix-mul-assoc)
finally show ∃ P. invertible P ∧ Hermite-reduce-above ?M (to-nat i) i ?n res = P ** A
  using inv-QP by auto
qed
lemma invertible-Hermite-of-upt-row-i:
  assumes a: ass-function ass
  shows $\exists P. \text{invertible } P \land \text{Hermite-of-upt-row-i } A \ k \ ass \ res = P \ ** \ A$
proof (induct k arbitrary: A)
  case 0
  thus $?case$ unfolding Hermite-of-upt-row-i-def by (auto, metis invertible-def matrix-mul-lid)
next
  case (Suc k)
  obtain Q where inv-Q: invertible Q and H-QA: Hermite-of-upt-row-i $A \ k \ ass \ res = Q \ ** \ A$
  using Suc.hyps by auto
  obtain P where inv-P: invertible P and H-PH: Hermite-of-row-i ass res (Hermite-of-upt-row-i $A \ k \ ass \ res$) (from-nat k)
  $= P \ ** \ (\text{Hermite-of-upt-row-i } A \ k \ ass \ res)$ using invertible-Hermite-of-row-i[OF a] by blast
  have inv-PQ: invertible $(P \ ** Q)$ by (simp add: inv-P inv-Q invertible-mult)
  have Hermite-of-upt-row-i $A \ (\text{Suc } k) \ ass \ res$
  $= \text{Hermite-of-upt-row-i } A \ k \ ass \ res$ (Hermite-of-upt-row-i $A \ k \ ass \ res$) (from-nat k)
  unfolding Hermite-of-upt-row-i-def by auto
  also have $... = P \ ** (\text{Hermite-of-upt-row-i } A \ k \ ass \ res)$ unfolding H-PH ..
  also have $... = P \ ** (Q \ ** A)$ unfolding H-QA ..
  also have $... = (P \ ** Q) \ ** A$ by (simp add: matrix-mul-assoc)
  finally show $?case$ using inv-PQ by blast
qed

lemma invertible-Hermite-of:
  fixes $A :: \{\text{bezout-ring-div}\} ^\ast \{\text{cols}\} ^\ast \{\text{mod-type}\} ^\ast \{\text{rows}\} ^\ast \{\text{mod-type}\}$
  assumes a: ass-function ass
  and b: is-bezout-ext bezout
  shows $\exists P. \text{invertible } P \land \text{Hermite-of } A \ ass \ res \ bezout \ = \ P \ ** \ A$
proof --
  obtain P where inv-P: invertible P and H-PH: Hermite-of-upt-row-i (echelon-form-of $A$ bezout) (nrows $A$) ass res
  $= P \ ** (\text{echelon-form-of } A \ bezout)$ using invertible-Hermite-of-upt-row-i[OF a] by blast
  obtain Q where inv-Q: invertible Q and E-QA: (echelon-form-of $A$ bezout) = $Q \ ** A$
  using echelon-form-of-invertible[OF b, of $A$] by auto
  have inv-PQ: invertible $(P \ ** Q)$ by (simp add: inv-P inv-Q invertible-mult)
  have Hermite-of-af $A \ ass \ res \ bezout$
  $= \text{Hermite-of-upt-row-i } (\text{echelon-form-of } A \ bezout)$ (nrows $A$) ass res
  unfolding Hermite-of-def Let-def ..
  also have $... = P \ ** (Q \ ** A)$ unfolding H-PH unfolding E-QA ..
  also have $... = (P \ ** Q) \ ** A$ by (simp add: matrix-mul-assoc)
finally show \( \texttt{thesis} \) using \( \texttt{inv-PQ} \) by \( \texttt{blast} \)

**1.6.3 The final theorem**

**lemma** \( \texttt{Hermite} \):

**assumes** \( a: \texttt{ass-function} \) \( \texttt{ass} \)

**and** \( r: \texttt{res-function} \) \( \texttt{res} \)

**and** \( b: \texttt{is-bezout-ext} \) \( \texttt{bezout} \)

**shows** \( \exists P. \) \( \texttt{invertible} \) \( P \) \( \land \) (\( \texttt{Hermite-of} \) \( A \) \( \texttt{ass res} \) \( \texttt{bezout} \)) \( = \) \( P \** A \land \texttt{Hermite} \) (\( \texttt{range} \) \( \texttt{ass} \)) (\( \lambda c. \) \( \texttt{range} \) (\( \texttt{res} \) \( c \)) (\( \texttt{Hermite-of} \) \( A \) \( \texttt{ass res} \) \( \texttt{bezout} \))

**using** \( \texttt{invertible-Hermite-of[OF a b]} \) \( \texttt{Hermite-Hermite-of[OF a r b]} \) **by** \( \texttt{fast} \)

**1.7 Proving the uniqueness of the Hermite Normal Form**

**lemma** \( \texttt{diagonal-least-nonzero} \):

**assumes** \( H: \texttt{Hermite} \) \( \texttt{associates residues} \) \( H \)

**and** \( \texttt{inv-H: invertible} \) \( H \) **and** \( \texttt{up-H: upper-triangular} \) \( H \)

**shows** (\( \texttt{LEAST} n. \) \( H \$ i \$ n \neq 0 \)) \( = i \)

**proof** \( \texttt{(rule Least-equality)} \)

**show** \( H \$ i \$ i \neq 0 \)

**by** (\( \texttt{metis (full-types) inv-H invertible-iff-is-unit is-unit-diagonal not-is-unit-0} \) **up-H**)

**fix** \( y \)

**assume** \( \texttt{Hiy:} \) \( H \$ i \$ y \neq 0 \)

**show** \( i \leq y \)

**using** \( \texttt{up-H unfolding upper-triangular-def} \)

**by** (\( \texttt{metis (poly-guards-query) Hiy not-less} \))

**qed**

**lemma** \( \texttt{diagonal-in-associates} \):

**assumes** \( H: \texttt{Hermite} \) \( \texttt{associates residues} \) \( H \)

**and** \( \texttt{inv-H: invertible} \) \( H \) **and** \( \texttt{up-H: upper-triangular} \) \( H \)

**shows** \( H \$ i \$ i \in \texttt{associates} \)

**proof** –

**have** \( H \$ i \$ i \neq 0 \)

**by** (\( \texttt{metis (full-types) inv-H invertible-iff-is-unit is-unit-diagonal not-is-unit-0} \) **up-H**)

**hence** \( \sim \) \( \texttt{is-zero-row i H unfolding is-zero-row-def is-zero-row-upt-k-def ncols-def} \)

**by** \( \texttt{auto} \)

**thus** \( \texttt{thesis} \) using \( \texttt{H unfolding Hermite-def unfolding diagonal-least-nonzero[OF H inv-H up-H]} \)

**by** \( \texttt{auto} \)

**qed**

**lemma** \( \texttt{above-diagonal-in-residues} \):

**assumes** \( H: \texttt{Hermite} \) \( \texttt{associates residues} \) \( H \)

**and** \( \texttt{inv-H: invertible} \) \( H \) **and** \( \texttt{up-H: upper-triangular} \) \( H \)

**and** \( j-i: j < i \)
shows $H \not\equiv j \pmod{(\text{LEAST } n. H \not\equiv i \pmod{n} \neq 0)} \in \text{residues (}H \not\equiv i \pmod{(\text{LEAST } n. H \not\equiv i \pmod{n} \neq 0)})$

proof –

- have $H \not\equiv i \not\equiv 0$
  - by (metis (full-types) invertible-iff-is-unit isn-unital-diagonal not-is-unit-0 up-H)
  
  hence $\neg\text{is-zero-row }i \text{ H unfolding is-zero-row-def is-zero-row-upt-k-def ncols-def}$
  by auto

  - thesis using $H \not\equiv j - i \text{ unfolding Hermite-def unfolding diagonal-least-nonzero[OF H inv-H up-H]}$
    - by auto

qed

The uniqueness of the Hermite Normal Form is proven following the proof presented in the book Integral Matrices (1972) by Morris Newman.

**Lemma Hermite-unique:**

fixes $K ::'a::bezout-ring-div''n::mod-type''n::mod-type

assumes $A-PH: A = P ** H$
and $A-QK: A = Q ** K$
and $inv-A: \text{invertible } A$
and $inv-P: \text{invertible } P$
and $inv-Q: \text{invertible } Q$
and $H: \text{Hermite associates residues } H$
and $K: \text{Hermite associates residues } K$

shows $H = K$

proof –

- have $\text{cs-residues: Complete-set-residues residues using } H \text{ unfolding Hermite-def}$
  by simp

  - have $\text{inv-H: invertible } H$
    - by (metis $A-PH$ $inv-A$ $inv-P$ invertible-def invertible-mult matrix-mul-assoc matrix-mul-lid)

  - have $\text{inv-K: invertible } K$
    - by (metis $A-QK$ $inv-A$ $inv-Q$ invertible-def invertible-mult matrix-mul-assoc matrix-mul-lid)

  - def $U \equiv (\text{matrix-inv P}) ** Q$

  - have $\text{inv-U: invertible } U$
    - by (metis $U$-def $inv-P$ $inv-Q$ invertible-def invertible-mult matrix-inv-left matrix-inv-right)

  - have $H-UK: H = U ** K$ using $A-PH$ $A-QK$ $inv-P$
    - by (metis $U$-def matrix-inv-left matrix-mul-assoc matrix-mul-lid)

  - have $\text{det } K \equiv H = H ** \text{adjugate } K$
    - by (metis $H-UK$ adjugate-def-symmetric matrix-mul-assoc matrix-scalar-mat-one)

  - have $\text{upper-triangular-H: upper-triangular } H$
    - by (metis $H$ Hermite-def echelon-form-imp-upper-triangular)

  - have $\text{upper-triangular-K: upper-triangular } K$
    - by (metis $K$ Hermite-def echelon-form-imp-upper-triangular)

  - have $\text{upper-triangular-U: upper-triangular } U$
    - by (metis $H-UK$ $inv-K$ matrix-inv-right matrix-mul-assoc matrix-mul-rd upper-triangular-H)

        upper-triangular-K upper-triangular-inverse upper-triangular-mult)
have \(\text{unit-det-} U\): \(\text{is-unit (det } U\) by (metis \text{inv-U invertible-iff-is-unit})

have \(\text{is-unit-diagonal-} U\): \((\forall i. \text{is-unit (} U \notin i i i)\)
by (rule \text{is-unit-diagonal}[OF upper-triangular-} U \text{ unit-det-} U\])

have \(U\text{-ii-1}: (\forall i. (U \notin i i i) = 1)\) and \(H\text{-ii-ki}: (\forall i. (H \notin i i i) = (K \notin i i i))\)
proof (auto)

fix \(i\)

have \(H\text{-ii}: H \notin i i i \in \text{associates}\)
by (rule \text{diagonal-in-associates}[OF \text{H inv-H upper-triangular-} H\])

have \(K\text{-ii}: K \notin i i i \in \text{associates}\)
by (rule \text{diagonal-in-associates}[OF \text{K inv-K upper-triangular-} K\])

have \(\text{ass-Hii-ki}: \text{associated (} H \notin i i i \in \text{Ass-not-associated})\)
by (metis \text{associated-def inv-H inv-K invertible-iff-is-unit is-unit-diagonal}
unit-imp-dvd upper-triangular-} H \text{ upper-triangular-} K\)

show \(H\text{-ii-ki}: H \notin i i i \in \text{K i} \in \text{K i}\)
by (metis \text{Hermite-def H Ki K Ki ass-Hii-ki in-Ass-not-associated})

have \(H \notin i i i = U \notin i i i \ast K \notin i i i\)
by (metis \text{H-UK upper-triangular-} K \text{ upper-triangular-} U \text{ upper-triangular-mult-diagonal})

thus \(U \notin i i i = 1\) unfolding \(H\text{-ii-ki multi-cancel-right1}\)
by (metis \text{Hii-}H\text{-ki inv-H invertible-iff-is-unit}
is-unit-diagonal not-is-unit-0 upper-triangular-} H\)

qed

have \(\text{zero-above: } \forall j s. j \geq 1 \land j < \text{ncols } A \rightarrow \text{to-nat } s \rightarrow U \notin i i i \ast (s + \text{from-nat } j) = 0\)
proof (clarify)

fix \(j s\) assume \(1 \leq j \land j < \text{ncols } A \rightarrow (\text{to-nat } (s::'n))\)
thus \(U \notin i i i \ast (s + \text{from-nat } j) = 0\)
proof (induct \(j\) rule: \text{less-induct})

fix \(p\)
assume \(\text{induct-step: } (\forall y. y < p \rightarrow 1 \leq y \rightarrow y < \text{ncols } A \rightarrow \text{to-nat } s \rightarrow U \notin i i i \ast (s + \text{from-nat } y) = 0)\)

and \(p1: 1 \leq p\) and \(p2: p < \text{ncols } A \rightarrow \text{to-nat } s\)

have \(s\text{-less: } s < s + \text{from-nat } p\) using \(p1\) \(p2\) unfolding \text{ncols-def}
by (metis \text{One-nat-def add.commute add-diff-cancel-right1 add-lessD1 add-to-nat-def}

from-nat-to-nat-id less-diff-conv \text{neq-iff not-le}
to-nat-from-nat-id to-nat-le zero-less-Suc)

show \(U \notin i i i \ast (s + \text{from-nat } p) = 0\)
proof -

have \(\text{UNIV-}r w\): \(\text{UNIV = insert } s (\text{UNIV} \text{-}\{s\})\) by auto
have \(\text{UNIV-}s-\text{rw}: \text{UNIV} \text{-}\{s\} = \text{insert } (s + \text{from-nat } p) (\text{(UNIV} \text{-}\{s\}) \text{-}\{s + \text{from-nat } p\})\)

using \(p1\) \(p2\) unfolding \text{ncols-def}
by (auto, metis \text{add-left-neutral diff-add-zero diff-0-eq diff-le-self le-less-trans}

less-diff-conv \text{not-less-}eq\text{-}to-nat\text{-}0 \text{ to-nat-from-nat-id})

have \(\text{setsum-}r w\): \(\text{(} \sum k \in \text{UNIV-}\{s\}. U \notin i i i \ast k \ast K \notin i i i k \ast (s + \text{from-nat } p))\)
= \(U \notin i i i \ast (s + \text{from-nat } p) \ast K \notin i i i (s + \text{from-nat } p) \ast (s + \text{from-nat } p)\)
+ \(\text{(} \sum k \in (\text{UNIV} \text{-}\{s\}) \text{-}\{s + \text{from-nat } p\}. U \notin i i i \ast k \ast K \notin i i i k \ast (s + \text{from-nat } p))\)

47
\begin{verbatim}
from-nat p)

  using UNIV-s-rw setsum.insert by (metis (erased, lifting) Diff-iff finite singletonI)

have setsum-0: \(\sum_{k \in (UNIV - \{s\}) - \{s + \text{from-nat } p\}}. U \$ s \$ k \& K \$ k \& (s + \text{from-nat } p) = 0\)
  proof (rule setsum.neutral, rule)
   fix x assume x: x \in UNIV - \{s\} - \{s + \text{from-nat } p\}
   show U \$ s \$ x \& K \$ x \& (s + \text{from-nat } p) = 0
     proof (cases x<s)
       case True
       thus \?thesis using upper-triangular-U unfolding upper-triangular-def by auto
     next
       case False
       hence x-g-s: x > s using x by (metis Diff-iff neq-iff singletonI)
       show \?thesis
         proof (cases x<s from-nat p)
           case True
           def a≡=to-nat x - to-nat s unfolding a-def
           have xa: x = s + (from-nat a) unfolding a-def
             by (metis a-def add-to-nat-def diff-le-self dual-order.strict-iff-order
                 from-nat-to-nat-id le-add-diff-inverse le-less-trans to-nat-from-nat-id
                 to-nat-less-card to-nat-mono x-g-s)
           have U \$ s \$ x =0
             proof (unfold xa, rule induct-step)
               show a-p: a < p unfolding a-def using p2 unfolding ncols-def
                 proof
                   have x < from-nat (to-nat s + to-nat (from-nat p::'n))
                     by (metis (no-types) True add-to-nat-def)
                   hence to-nat x - to-nat s < to-nat (from-nat p::'n)
                     by (simp add: add.commute less-diff-conv2 less-imp-le to-nat-le
                         x-g-s)
                   thus to-nat x - to-nat s < p
                     by (metis (no-types) from-nat-eq-imp-eq from-nat-to-nat-id
                         le-less-trans
                         less-imp-le not-le to-nat-less-card)
                   qed
                   show 1 \le a
                     by (auto simp add: a-def p1 p2) (metis Suc-leI to-nat-mono x-g-s
                         zero-less-diff)
                 qed
                 thus \?thesis by simp
               next
                 case False
                 hence x>s+from-nat p using x-g-s x by auto
                 thus \?thesis using upper-triangular-K unfolding upper-triangular-def
                   by auto
               qed

48
\end{verbatim}
have $H \otimes s \otimes (s + \text{from-nat } p) = (\sum_{k \in \text{UNIV}}. U \otimes s \otimes k \otimes k \otimes (s + \text{from-nat } p))$

unfolding $H$-$U$K matrix-matrix-mult-def by auto
also have $\ldots = (\sum_{k \in \text{insert } s \ (\text{UNIV} - \{s\})}. U \otimes s \otimes k \otimes k \otimes (s + \text{from-nat } p))$
using UNIV-rw by simp
also have $\ldots = U \otimes s \otimes s \otimes K \otimes s \otimes (s + \text{from-nat } p) + (\sum_{k \in \text{UNIV} - \{s\}}. U \otimes s \otimes k \otimes K \otimes k \otimes (s + \text{from-nat } p))$
by (rule setsum.insert, simp-all)
also have $\ldots = U \otimes s \otimes s \otimes K \otimes s \otimes (s + \text{from-nat } p) + U \otimes s \otimes (s + \text{from-nat } p) \otimes K \otimes (s + \text{from-nat } p) \otimes (s + \text{from-nat } p)$
unfolding setsum-rw setsum-0 by simp
finally have $H$-$s$-sp: $H \otimes s \otimes (s + \text{from-nat } p) = U \otimes s \otimes (s + \text{from-nat } p) \otimes K \otimes (s + \text{from-nat } p) + K \otimes s \otimes (s + \text{from-nat } p)$
using $\text{Uii}$-$1$ by auto
hence cong-HK: cong ($H \otimes s \otimes (s + \text{from-nat } p)$) ($K \otimes s \otimes (s + \text{from-nat } p)$) ($K \otimes (s + \text{from-nat } p) \otimes (s + \text{from-nat } p)$)
unfolding cong-def by auto
have $H$-$s$-sp-residues: ($H \otimes s \otimes (s + \text{from-nat } p)$) $\in$ residues ($K \otimes (s + \text{from-nat } p)$) $\times$ ($s + \text{from-nat } p$))
using above-diagonal-in-residues[$OF H \otimes inv-H \otimes upper$-$triangular-H \ s$-$less$]
unfolding diagonal-least-nonzero[$OF H \otimes inv-H \otimes upper$-$triangular-H$]
by (metis $\text{Hii}$-$Kii$)
have $K$-$s$-sp-residues: ($K \otimes s \otimes (s + \text{from-nat } p)$) $\in$ residues ($K \otimes (s + \text{from-nat } p)$) $\times$ ($s + \text{from-nat } p$))
using above-diagonal-in-residues[$OF K \otimes inv-K \otimes upper$-$triangular-K \ s$-$less$]
unfolding diagonal-least-nonzero[$OF K \otimes inv-K \otimes upper$-$triangular-K$] .
have $H$s-$sp$-$K$s-$sp$: ($H \otimes s \otimes (s + \text{from-nat } p)$) = ($K \otimes s \otimes (s + \text{from-nat } p)$)
using cong-HK in-Res-not-congruent[$OF cs$-$residues H$-$s$-$sp$-$residues$]

K$-$s$-$sp$-$residues$

by fast
have is-unit ($K \otimes (s + \text{from-nat } p) \otimes (s + \text{from-nat } p)$)
by (metis $\text{Hii}$-$Kii inv-H invertible-iff-is-unit-diagonal upper-triangular-H)
hence $K \otimes (s + \text{from-nat } p) \otimes (s + \text{from-nat } p) \neq 0$ by (metis not-is-unit-0)
thus $\text{?thesis}$ unfolding from-nat-1 using $H$-$sp$ unfolding $H$s-$sp$-$K$s-$sp$

by auto
qed
qed

have $U = \mat 1$
proof (unfold mat-def vec-eq-iff, auto)
fix $ia$ show $U \otimes ia \otimes ia = 1$ using $\text{Uii}$-$1$ by simp
fix $i$ assume $i$-$ia$: $i \neq ia$
show $U \otimes i \otimes ia = 0$
proof (cases $ia$<:i)
case True
  thus ?thesis using upper-triangular-U unfolding upper-triangular-def by auto
next
case False
  hence i-less-ia: i < ia using i-ia by auto
  def a == to-nat ia - to-nat i
  have ia-eq: ia = i + from-nat a unfolding a-def
  by (metis i-less-ia a-def add-to-nat-def dual-order.strict-iff-order from-nat-to-nat-id
  le-add-diff-inverse less-imp-diff-less to-nat-from-nat-id to-nat-less-card
  to-nat-mono)
  have 1 ≤ a unfolding a-def
  by (metis diff-is-0-eq i-less-ia less-one not-less to-nat-mono)
  moreover have a < ncols A - to-nat i
  unfolding a-def ncols-def
  by (metis False diff-less-mono not-less to-nat-less-card to-nat-mono')
  ultimately show ?thesis using zero-above unfolding ia-eq by blast
qed
qed
thus ?thesis using H-UK matrix-mul-lid by fast
qed

1.8 Examples of execution

value[code] let A = list-of-list-to-matrix ([[[37,8,6], [5,4,-8], [3,24,-7]] :: int ^ 3 ^ 3]
in matrix-to-list-of-list (Hermite-of A ass-function-euclidean res-function-euclidean euclid-ext2)

value[code] let A = list-of-list-to-matrix ([[[3,4,5], [-2,1]], [[-1,0,2], [0,1,4,1]]] :: real poly ^ 2 ^ 2
in matrix-to-list-of-list (Hermite-of A ass-function-euclidean res-function-euclidean euclid-ext2)

end

2 Hermite Normal Form refined to immutable arrays

theory Hermite-IArrays
imports
  Hermite
  ../Echelon-Form/Echelon-Form-IArrays
begin

2.1 Definition of the algorithm over immutable arrays

primrec Hermite-reduce-above-iarrays :: 'a::ring-div iarray iarray ⇒ nat ⇒ nat ⇒ nat ⇒ ('a⇒'a⇒'a) ⇒ 'a iarray iarray
where \( \text{Hermite-reduce-above-arrays} \ A \ 0 \ i \ j \ \text{res} = A \)
\[
\mid \text{Hermite-reduce-above-arrays} \ A \ (\text{Suc} \ n) \ i \ j \ \text{res} = (\text{let} \ i' = n; \\
A_{ij} = A !! i !! j; \\
A_{i'j} = A !! i' !! j \\
\text{in} \\
\text{Hermite-reduce-above-arrays} \ (\text{row-add-arrays} \ A \ i' \ i \ ((\text{res} \ A_{ij} (A_{i'j})) - (A_{i'j})) \ \text{div} \ A_{ij})) \ n \ i \ j \ \text{res})
\]

**Definition** \( \text{Hermite-of-row-i-arrays} \ A \ i \ \text{ass} \ \text{res} \) =
\[
\text{if} \ \text{is-zero-arrays} \ (A !! i) \\
\text{then} \ A \\
\text{else} \\
\text{let} \ j = \text{least-non-zero-position-of-vector} \ (A !! i); \ A_{ij} = (A !! i !! j); \\
A' = \text{mult-row-arrays} \ A \ i \ ((\text{ass} \ A_{ij}) \ \text{div} \ A_{ij}) \\
\text{in} \ \text{Hermite-reduce-above-arrays} \ A' \ i \ i \ j \ \text{res})
\]

**Definition** \( \text{Hermite-of-upt-row-i-arrays} \ A \ i \ \text{ass} \ \text{res} \) =
\[
\text{foldl} \ (\text{Hermite-of-row-i-arrays} \ \text{ass} \ \text{res}) \ A [0..<i]
\]

**Definition** \( \text{Hermite-of-i-arrays} \ A \ \text{ass} \ \text{res} \ \text{bezout} = \\
(let \ A' = \text{echelon-form-of-arrays} \ A \ \text{bezout} \\
in \ \text{Hermite-of-upt-row-i-arrays} \ A' \ (\text{nrows-arrays} \ A) \ \text{ass} \ \text{res})
\]

2.2 Proving the equivalence between definitions of both representations

**Lemma** \( \text{matrix-to-arrays-Hermite-reduce-above}: \\
\text{fixes} \ A::'a::{\text{ring-div}} ^\prime \text{cols}::{\text{mod-type}} ^\prime \text{rows}::{\text{mod-type}} \\
\text{assumes} \ n<\text{nrows} \ A \\
\text{shows} \ \text{matrix-to-arrays} \ (\text{Hermite-reduce-above} \ A \ n \ i \ j \ \text{res}) \\
= \ \text{Hermite-reduce-above-arrays} \ (\text{matrix-to-arrays} \ A) \ n \ (\text{to-nat} \ i) \ (\text{to-nat} \ j) \ \text{res} \\
\text{using} \ \text{assms} \\
\text{proof} \ (\text{induct} \ n \ \text{arbitrary}: \ A) \\
\text{case} \ 0 \ \text{thus} \ ?\text{case} \ \text{by} \ \text{auto} \\
\text{next} \\
\text{case} \ (\text{Suc} \ n) \\
\text{have} \ n: \ n<\text{nrows} \ A \\
\text{using} \ \text{Suc.prems} \ \text{unfolding} \ \text{nrows-def} \ \text{by} \ \text{simp} \\
\text{obtain} \ a::\text{rows} \ \text{where} \ n-tna: \ n = \text{to-nat} \ a \\
\text{by} \ (\text{metis} \ \text{Suc.prems} \ \text{Suc-lessD} \ \text{nrows-def} \ \text{to-nat-from-nat-id}) \\
\text{show} \ ?\text{case} \\
\text{unfolding} \ \text{Hermite-reduce-above.simps} \\
\text{unfolding} \ \text{Hermite-reduce-above-arrays.simps} \\
\text{unfolding} \ \text{Let-def sub-def[\text{symmetric}]} \\
\text{unfolding} \ n-tna \\
\text{unfolding} \ \text{matrix-to-arrays-row-add[\text{symmetric}]} \ \text{from-nat-to-nat-id} \\
\text{unfolding} \ \text{matrix-to-arrays-nth} \\
\text{unfolding} \ n-tna[\text{symmetric}] \\
\text{by} \ (\text{rule} \ \text{Suc.hyps}, \ \text{auto} \ \text{simp add:} \ \text{nrows-def n[unfolded nrows-def]})
\]
lemma matrix-to-iarray-Hermite-of-row-i[code-unfold]:

fixes A :: 'a::{ring-div} *'cols::{mod-type} *'rows::{mod-type}

shows matrix-to-iarray (Hermite-of-row-i ass res A i) = Hermite-of-row-i-iarray ass res (matrix-to-iarray A) (to-nat i)

proof -

have zero-rw: is-zero-iarray (matrix-to-iarray A !! to-nat i) = is-zero-row i A

by (simp add: is-zero-iarray-eq-iff is-zero-row-eq-row-zero vec-to-iarray-row)

show ?thesis

proof (cases is-zero-row i A)


by auto

next

case False

have Ain: A $ i $ (LEAST n. A $ i $ n ≠ 0) ≠ 0

using False

by (metis (mono-tags, lifting) LeastI is-zero-row-def)

have l: least-non-zero-position-of-vector (matrix-to-iarray A !! to-nat i) = to-nat (LEAST n. A $ i $ n ≠ 0)

proof -

have least-rw: (LEAST n. A $ i $ n ≠ 0 ∧ 0 ≤ n) = (LEAST n. A $ i $ n ≠ 0)

by (rule Least-equality, auto simp add: least-mod-type Ain Least-le)

have v-rw: ¬ vector-all-zero-from-index (to-nat (0::'cols), vec-to-iarray (A $ i))

using False least-mod-type

unfolding vector-all-zero-from-index-eq[of 0 A$i, symmetric] is-zero-row-def'

by auto


unfolding least-rw least-non-zero-position-of-vector-def to-nat-0 vec-matrix

qed

show ?thesis

unfolding Hermite-of-row-i-def Hermite-of-row-i-iarray-def Let-def

unfolding zero-rw[symmetric]

unfolding matrix-to-iarray-mult-row[symmetric]

unfolding l

unfolding matrix-to-iarray-nth

by (auto, rule matrix-to-iarray-Hermite-reduce-above, simp add: nrows-def to-nat-less-card)

qed

qed

lemma matrix-to-iarray-Hermite-of-upt-row-i:

52
fixes  A::'a::{ring-div}  "cols::{mod-type}"  "rows::{mod-type}"
assumes i: i ≤ nrows A
shows matrix-to-iarray (Hermite-of-upt-row-i A i ass res) = Hermite-of-upt-row-i-iarrays (matrix-to-iarray A) i ass res
using i
proof (induct i arbitrary: A)
case 0
thus ?case by (simp add: Hermite-of-upt-row-i-def Hermite-of-upt-row-i-iarrays-def)
next
case (Suc i)
have i: i < nrows A using Suc.prems unfolding nrows-def by simp
have matrix-to-iarray (Hermite-of-upt-row-i A (Suc i) ass res) = matrix-to-iarray (Hermite-of-row-i ass res (Hermite-of-upt-row-i A i ass res) (from-nat i))
  unfolding Hermite-of-upt-row-i-def by auto
also have ... = (Hermite-of-row-i-iarray ass res (matrix-to-iarray (Hermite-of-row-i ass res (Hermite-of-upt-row-i A i ass res)) i))
  unfolding matrix-to-iarray-Hermite-of-row-i ..
also have ... = (Hermite-of-control-case 0 w i ass res (matrix-to-iarray (Hermite-of-row-i-ass-res (Hermite-of-upt-row-i A i ass res) i)))
  unfolding matrix-to-iarray-echelon-form-of ..
finally show ?case.
qed

lemma matrix-to-iarray-Hermite-of[code-unfold]:
shows matrix-to-iarray (Hermite-of A ass res bezout) = Hermite-of-iarrays (matrix-to-iarray A) ass res bezout
proof
  have n: nrows A ≤ nrows (echelon-form-of A bezout) unfolding nrows-def by simp
  show ?thesis
    unfolding Hermite-of-def Hermite-of-iarrays-def Let-def
    unfolding matrix-to-iarray-Hermite-of-upt-row-i[OF n]
    unfolding matrix-to-iarray-echelon-form-of
    unfolding matrix-to-iarray-nrows ..
  qed

2.3 Examples of execution using immutable arrays

value[code] let A = list-of-list-to-matrix [[[37, 8, 6]], [5, 4, −8], [3, 24, −7]]::int^3^3
in matrix-to-iarray (Hermite-of A ass-function-euclidean res-function-euclidean euclid-ext2)
value[\text{code}] \text{ let } A = \text{ IArray[IArray[37, 8, 6::\text{int}], IArray[5, 4, -8], IArray[3, 2, 4, -7]]} \\
\text{ in (Hermite-of-arrays A ass-function-euclidean res-function-euclidean euclid-ext2)}

value[\text{code}] \text{ let } A = \text{ list-of-list-to-matrix ([[[:3, 4, 5:],[[:2, 1:],[[:1, 0, 2:],[[:0, 1, 4, 1::]]]]::real poly 2^2}} \\
\text{ in matrix-to-iarray (Hermite-of A ass-function-euclidean res-function-euclidean euclid-ext2)}

\text{end}