

Hermite Normal Form

By Jose Divasón and Jesús Aransay*

February 6, 2026

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	1
1.1	Some previous properties	1
1.1.1	Rings	1
1.1.2	Polynomials	2
1.1.3	Units	2
1.1.4	Upper triangular matrices	3
1.1.5	More properties of mod type	5
1.1.6	Div and Mod	5
1.2	Units, associated and congruent relations	6
1.3	Associates and residues functions	7
1.3.1	Concrete instances in Euclidean rings	9
1.3.2	Concrete case of the integer ring	9
1.4	Definition of Hermite Normal Form	9
1.4.1	Echelon form up to row k	11
1.4.2	Hermite Normal Form up to row k	12
1.5	Definition of an algorithm to compute the Hermite Normal Form	13

*This research has been funded by the research grant FPI-UR-12 of the Universidad de La Rioja and by the project MTM2014-54151-P from Ministerio de Economía y Competitividad (Gobierno de España).

1.6	Proving the correctness of the algorithm	13
1.6.1	The proof	13
1.6.2	Proving that the Hermite Normal Form is computed by means of elementary operations	41
1.6.3	The final theorem	43
1.7	Proving the uniqueness of the Hermite Normal Form	44
1.8	Examples of execution	49
2	Hermite Normal Form refined to immutable arrays	49
2.1	Definition of the algorithm over immutable arrays	49
2.2	Proving the equivalence between definitions of both represen- tations	50
2.3	Examples of execution using immutable arrays	52

1 Hermite Normal Form

```

theory Hermite
  imports
    Echelon-Form.Echelon-Form-Inverse
    Echelon-Form.Examples-Echelon-Form-Abstract
    HOL-Computational-Algebra.Euclidean-Algorithm
begin

```

1.1 Some previous properties

1.1.1 Rings

```

subclass (in bezout-ring-div) euclidean-ring
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
  thus ?thesis
  by (intro dvd-trans[OF is-unit-triv one-dvd]) simp
qed

```

```

lemma poly-dvd-antisym2:
  fixes p q :: 'a::field poly
  assumes dvd1: p dvd q and dvd2: q dvd p
  shows p div [:coeff p (degree p):] = q div [:coeff q (degree q):]
proof (cases p = 0)

```

```

case True
thus ?thesis
  by (metis dvd1 dvd-0-left-iff)
next
case False have q: q ≠ 0
  by (metis False dvd2 dvd-0-left-iff)
have degree: degree p = degree q
  using ⟨p dvd q⟩ ⟨q dvd p⟩ ⟨p ≠ 0⟩ ⟨q ≠ 0⟩
  by (intro order-antisym dvd-imp-degree-le)
from ⟨p dvd q⟩ obtain a where a: q = p * a ..
with ⟨q ≠ 0⟩ have a ≠ 0 by auto
with degree a ⟨p ≠ 0⟩ have degree a = 0
  by (simp add: degree-mult-eq)
with a show ?thesis
proof (cases a, auto split: if-splits, metis ⟨a ≠ 0⟩)
  fix aa :: 'a
  assume a1: aa ≠ 0
  assume q = smult aa p
  have [aa * coeff p (degree p):] dvd smult aa p
    using a1 by (metis (no-types) coeff-dvd-poly coeff-smult degree-smult-eq)
  thus p div [coeff p (degree p):] = smult aa p div [aa * coeff p (degree p):]
    using a1 by (simp add: False coeff-dvd-poly dvd-div-div-eq-mult)
qed
qed

```

1.1.3 Units

```

lemma unit-prod:
  assumes finite S
  shows is-unit (prod (λi. U $ i $ i) S) = (∀ i ∈ S. is-unit (U $ i $ i))
  using assms
proof (induct)
  case empty
  thus ?case by auto
next
  case (insert a S)
  have prod (λi. U $ i $ i) (insert a S) = U $ a $ a * prod (λi. U $ i $ i) S
    by (simp add: insert.hyps(2))
  thus ?case using is-unit-mult-iff insert.hyps by auto
qed

```

1.1.4 Upper triangular matrices

```

lemma is-unit-diagonal:
  fixes U::'a::{comm-ring-1, algebraic-semidom} ^'n::{finite, wellorder} ^'n::{finite, wellorder}
  assumes U: upper-triangular U
  and det-U: is-unit (det U)
  shows ∀ i. is-unit (U $ i $ i)
proof -

```

```

have is-unit (prod (λi. U $ i $ i) UNIV)
  using det-U det-upperdiagonal[of U] U
  unfolding upper-triangular-def by auto
hence ∀i∈UNIV. is-unit (U $ i $ i) using unit-prod[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 (∑k∈UNIV. A $ i $ k * B $ k $ j) = 0
  proof (rule sum.neutral, clarify)
    fix x
    show A $ i $ x * B $ x $ j = 0
    proof (cases x<i)
      case True
        thus ?thesis using A unfolding upper-triangular-def by auto
      next
        case False
          hence x>j using ji by auto
          thus ?thesis using A B ji unfolding upper-triangular-def by auto
    qed
  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 with A show det (minorM A j i) = 0
  unfolding minorM-eq det-sq-matrix-eq[symmetric] from-vec-to-vec det-minor-row
  by (subst Square-Matrix.det-upperdiagonal)
  (auto simp: upd-row.rep-eq from-vec.rep-eq row-def axis-def upper-triangular-def
intro!: prod-zero)
qed

```

```

lemma upper-triangular-inverse:
  fixes A::('a::{euclidean-semiring,comm-ring-1},'n::{wellorder, finite}}) vec, 'n)
vec
  assumes A: upper-triangular A
  and inv-A: invertible A

```

shows *upper-triangular* (*matrix-inv* A)
using *upper-triangular-adjugate*[$OF\ A$]
unfolding *invertible-imp-matrix-inv*[$OF\ inv-A$]
unfolding *matrix-scalar-mult-def upper-triangular-def* **by** *auto*

lemma *upper-triangular-mult-diagonal*:

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

assumes A : *upper-triangular* A

and B : *upper-triangular* B

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

proof –

have $UNIV-rw$: $UNIV = (insert\ i\ (UNIV - \{i\}))$ **by** *auto*

have $sum-0$: $(\sum_{k \in UNIV - \{i\}} A\ \$\ i\ \$\ k * B\ \$\ k\ \$\ i) = 0$

proof (*rule sum.neutral, rule*)

fix x **assume** x : $x \in UNIV - \{i\}$

show $A\ \$\ i\ \$\ x * B\ \$\ x\ \$\ i = 0$

proof (*cases x < i*)

case *True*

thus *?thesis* **using** A **unfolding** *upper-triangular-def* **by** *auto*

next

case *False*

hence $x > i$ **using** x **by** *auto*

thus *?thesis* **using** B **unfolding** *upper-triangular-def* **by** *auto*

qed

qed

have $(A**B)\ \$\ i\ \$\ i = (\sum_{k \in UNIV} A\ \$\ i\ \$\ k * B\ \$\ k\ \$\ i)$

unfolding *matrix-matrix-mult-def* **by** *simp*

also have $\dots = (\sum_{k \in (insert\ i\ (UNIV - \{i\}))} A\ \$\ i\ \$\ k * B\ \$\ k\ \$\ i)$

using $UNIV-rw$ **by** *simp*

also have $\dots = (A\ \$\ i\ \$\ i * B\ \$\ i\ \$\ i) + (\sum_{k \in UNIV - \{i\}} A\ \$\ i\ \$\ k * B\ \$\ k\ \$\ i)$

by (*rule sum.insert, simp-all*)

finally show *?thesis* **unfolding** $sum-0$ **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 Div and Mod

lemma *dvd-minus-eq-mod*:

fixes $c::'a::\text{unique-euclidean-ring}$
assumes $c \neq 0$ **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 $c::'a::\text{unique-euclidean-ring}$
assumes $c \neq 0$ **and** $a \text{ mod } c = b \text{ mod } c$
shows $c \text{ dvd } a - b$
using *assms* **by** (*simp add: mod-eq-dvd-iff*)

lemma *dvd-cong-not-eq-mod*:
fixes $c::'a::\text{unique-euclidean-ring}$
assumes $xa \text{ mod } c \neq xb$ **and** $c \text{ dvd } xa \text{ mod } c - xb$ **and** $c \neq 0$
shows $xb \text{ mod } c \neq xb$
using *assms*
by (*metis (no-types, lifting) diff-add-cancel dvdE mod-mod-trivial mod-mult-self4*)

lemma *diff-mod-cong-0*:
fixes $c::'a::\text{unique-euclidean-ring}$
assumes $xa \text{ mod } c \neq xb \text{ mod } c$ **and** $c \text{ dvd } xa \text{ mod } c - xb \text{ mod } c$ **shows** $c = 0$
using *assms dvd-cong-not-eq-mod mod-mod-trivial* **by** *blast*

lemma *cng-diff-mod*:
fixes $c::'a::\text{unique-euclidean-ring}$
assumes $xa \neq xb$ **and** $c \text{ dvd } xa - xb$ **and** $xa = xa \text{ mod } c$ **shows** $xb \neq xb \text{ mod } c$
by (*metis assms diff-eq-diff-eq diff-numeral-special(12) dvd-0-left dvd-minus-eq-mod*)

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

1.2 Units, associated and congruent relations

context *semiring-1*
begin

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

end

context *ring-1*
begin

definition $\text{cong}::'a \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{bool}$
where $\text{cong } a \ c \ b = (\exists k. (a - c) = b * k)$

```

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

end

context normalization-semidom
begin

lemma Units-eq:  $Units = \{x. x\ dvd\ 1\}$  unfolding Units-def dvd-def ..

lemma normalize-Units:  $x \in Units \implies normalize\ x = 1$ 
  by (intro is-unit-normalize) (simp-all add: Units-eq)

lemma associated-eq:  $(normalize\ a = normalize\ b) \longleftrightarrow (\exists u \in Units. a = u * b)$ 
proof
  assume A:  $normalize\ a = normalize\ b$ 
  show  $\exists u \in Units. a = u * b$ 
  proof (cases a = 0  $\vee$  b = 0)
    case False
    from A have  $a = (unit-factor\ a\ div\ unit-factor\ b) * b$ 
    by (metis mult-not-zero normalize-0 normalize-mult-unit-factor mult.left-commute

      unit-div-mult-self unit-factor-is-unit)
    moreover from False have  $unit-factor\ a\ div\ unit-factor\ b \in Units$ 
    by (simp add: Units-eq unit-div)
    ultimately show ?thesis by blast
  next
  case True
  with A normalize-eq-0-iff[of a] normalize-eq-0-iff[of b] have  $a = 0\ b = 0$  by
auto
  thus ?thesis by (auto intro!: exI[of - 1] simp: Units-def)
  qed
qed (auto simp: normalize-Units Units-def)

end

context unique-euclidean-ring
begin

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

lemma equiv-associated:
  shows equiv UNIV associated-rel
  unfolding associated-rel-def equiv-def refl-on-def sym-def trans-def by simp

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

lemma refl-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 *relf-congruent-rel sym-congruent-rel trans-congruent-rel* **by** *auto*

end

1.3 Associates and residues functions

context *normalization-semidom*
begin

definition *ass-function :: ('a ⇒ 'a) ⇒ bool*
where *ass-function f*
 $= ((\forall a. \text{normalize } a = \text{normalize } (f a)) \wedge \text{pairwise } (\lambda a b. \text{normalize } a \neq \text{normalize } b) (\text{range } f))$

definition *Complete-set-non-associates S*
 $= (\exists f. \text{ass-function } f \wedge f'UNIV = S \wedge (\text{pairwise } (\lambda a b. \text{normalize } a \neq \text{normalize } b) S))$

end

context *ring-1*
begin

definition *res-function :: ('a ⇒ 'a ⇒ 'a) ⇒ bool*
where *res-function f* $= (\forall c. (\forall a b. \text{cong } a b c \longleftrightarrow f c a = f c b) \wedge \text{pairwise } (\lambda a b. \neg \text{cong } a b c) (\text{range } (f c)) \wedge (\forall a. \exists k. f c a = a + k*c))$

definition *Complete-set-residues g*
 $= (\exists f. \text{res-function } f \wedge (\forall c. (\text{pairwise } (\lambda a b. \neg \text{cong } a b c) (f c'UNIV)) \wedge g c = f c'UNIV))$
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* **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 *normalize x ≠ normalize 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 pairwise-def*
by (*metis normalize-eq-0-iff*)

lemma *ass-function-0'*:
assumes *r*: *ass-function ass*
shows (*ass x div x = 0*) = (*x = 0*)
proof *safe*
assume ***: *ass x div x = 0*
from *r* **have** ****: *normalize (ass x) = normalize x*
by (*simp add: ass-function-def*)
from *associatedD2*[*OF this*] **have** *x dvd ass x*
by *simp*
with ** *** **show** *x = 0*
by (*auto simp: dvd-div-eq-0-iff*)
qed *auto*

lemma *res-function-Complete-set-residues*:
assumes *f*: *res-function f*
shows *Complete-set-residues* ($\lambda c. (f\ c)$ 'UNIV)
unfolding *Complete-set-residues-def*
apply (*rule exI*[*of - f*]) **using** *f* **unfolding** *res-function-def* **by** *blast*

lemma *in-Res-not-congruent*:
assumes *res-g*: *Complete-set-residues g*
and *x*: *x ∈ g b* **and** *y*: *y ∈ g b* **and** *x-not-y*: *x ≠ y*
shows \neg *cong x y b*
using *assms*
unfolding *Complete-set-residues-def*
unfolding *pairwise-def*
by *auto*

1.3.1 Concrete instances in Euclidean rings

definition *ass-function-euclidean* ($p::'a::\{\text{normalization-euclidean-semiring}, \text{euclidean-ring}\}$)
 $= \text{normalize } p$

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

lemma *ass-function-euclidean*: *ass-function* *ass-function-euclidean*

unfolding *ass-function-def* *image-def* *ass-function-euclidean-def* *pairwise-def* **by** *auto*

lemma *res-function-euclidean*:

res-function (*res-function-euclidean* $:: 'a :: \text{unique-euclidean-ring} \Rightarrow -$)

by (*auto simp add: pairwise-def res-function-def cong-eq image-def res-function-euclidean-def dvd-minus-eq-mod*)

(*auto simp add: dvd-cong-not-eq-mod eq-mod-dvd-minus diff-mod-cong-0 cong-diff-mod exists-k-mod*)

1.3.2 Concrete case of the integer ring

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

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

by (*unfold ass-function-int-def ass-function-euclidean-def*) *simp*

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

unfolding *ass-function-int-def* *image-def*

by (*auto,metis 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 *echelon-form-upt-row* A $k =$

$$\left(\begin{array}{l} (\forall i. \text{to-nat } i < k \wedge \text{is-zero-row } i \ A \longrightarrow \neg (\exists j. j > i \wedge \text{to-nat } j < k \wedge \neg \text{is-zero-row } j \ A)) \wedge \\ (\forall i \ j. i < j \wedge \text{to-nat } j < k \wedge \neg \text{is-zero-row } i \ A \wedge \neg \text{is-zero-row } j \ A \longrightarrow (\text{LEAST } n. A \ \$ \ i \ \$ \ n \neq 0) < (\text{LEAST } n. A \ \$ \ j \ \$ \ n \neq 0)) \end{array} \right)$$

lemma *echelon-form-upt-row-condition1-explicit:*

assumes *echelon-form-upt-row* A k

and *to-nat* $i < k$ **and** *is-zero-row* $i \ A$

shows $\neg (\exists j. j > i \wedge \text{to-nat } j < k \wedge \neg \text{is-zero-row } j \ A)$

using *assms* **unfolding** *echelon-form-upt-row-def* **by** *blast*

lemma *echelon-form-upt-row-condition1-explicit'*:
assumes *echelon-form-upt-row A k*
and *to-nat i < k and is-zero-row i A and i ≤ j and to-nat j < k*
shows *is-zero-row j A*
proof (*cases i=j*)
 case *True* **thus** *?thesis* **using** *assms* **by** *auto*
next
 case *False* **thus** *?thesis* **using** *assms* **unfolding** *echelon-form-upt-row-def* **by**
simp
qed

lemma *echelon-form-upt-row-condition1-explicit-neg*:
assumes *echelon-form-upt-row A k*
and *ia: ¬ is-zero-row i A and ia-i: ia < i*
and *i: to-nat i < k*
shows *¬ is-zero-row ia A*
proof –
 have *to-nat ia < k* **by** (*metis ia-i i less-trans to-nat-mono*)
 thus *?thesis* **using** *assms* **unfolding** *echelon-form-upt-row-def* **by** *blast*
qed

lemma *echelon-form-upt-row-condition2-explicit*:
assumes *echelon-form-upt-row A k*
and *ia < j and to-nat j < k and ¬ is-zero-row ia A and ¬ is-zero-row j A*
shows (*LEAST n. A \$ ia \$ n ≠ 0*) < (*LEAST n. A \$ j \$ n ≠ 0*)
using *assms* **unfolding** *echelon-form-upt-row-def* **by** *auto*

lemma *echelon-form-upt-row-intro*:
assumes($\forall i. \text{to-nat } i < k \wedge \text{is-zero-row } i A \longrightarrow \neg (\exists j. i < j \wedge \text{to-nat } j < k \wedge \neg \text{is-zero-row } j A)$)
and ($\forall i j. i < j \wedge \text{to-nat } j < k \wedge \neg \text{is-zero-row } i A \wedge \neg \text{is-zero-row } j A \longrightarrow (\text{LEAST } n. A \$ i \$ n \neq 0) < (\text{LEAST } n. A \$ j \$ n \neq 0)$)
shows *echelon-form-upt-row A k*
using *assms* **unfolding** *echelon-form-upt-row-def* **by** *simp*

lemma *echelon-form-echelon-form-upt-row*: *echelon-form A = echelon-form-upt-row A (nrows A)*
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).

definition *Hermite-upt-row A k associates residues =*
(*Complete-set-non-associates associates* \wedge

Complete-set-residues residues \wedge
echelon-form-upt-row A k \wedge
 $(\forall i. \text{to-nat } i < k \wedge \neg \text{is-zero-row } i \ A \longrightarrow A \$ i \$ (\text{LEAST } n. A \$ i \$ n \neq 0)$
 $\in \text{associates}) \wedge$
 $(\forall i. \text{to-nat } i < k \wedge \neg \text{is-zero-row } i \ A \longrightarrow (\forall j < i. A \$ j \$ (\text{LEAST } n. A \$ i \$ n \neq 0)$
 $n \neq 0) \in \text{residues } (A \$ i \$ (\text{LEAST } n. A \$ i \$ n \neq 0))))$
 $)$

The definition of Hermite Normal Form is now introduced:

definition *Hermite*:: $'a::\{\text{bezout-ring-div,normalization-semidom}\}$ *set* $\Rightarrow ('a \Rightarrow 'a$
set) \Rightarrow
 $((('a, 'b::\{\text{mod-type}\}) \text{vec}, 'c::\{\text{mod-type}\}) \text{vec} \Rightarrow \text{bool}$
where *Hermite associates residues* $A = ($
Complete-set-non-associates associates
 $\wedge (\text{Complete-set-residues residues})$
 $\wedge \text{echelon-form } A$
 $\wedge (\forall i. \neg \text{is-zero-row } i \ A \longrightarrow A \$ i \$ (\text{LEAST } n. A \$ i \$ n \neq 0) \in \text{associates})$
 $\wedge (\forall i. \neg \text{is-zero-row } i \ A \longrightarrow (\forall j. j < i \longrightarrow A \$ j \$ (\text{LEAST } n. A \$ i \$ n \neq 0) \in$
residues $(A \$ i \$ (\text{LEAST } n. A \$ i \$ n \neq 0))))$
 $)$

lemma *Hermite-Hermite-upt-row*: *Hermite ass res* $A = \text{Hermite-upt-row } A$ (*nrows*
 A) *ass res*

by (*simp add: Hermite-def Hermite-upt-row-def to-nat-less-card is-zero-row-def*
nrows-def ncols-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 $(\forall i. \neg \text{is-zero-row } i \ A \longrightarrow A \$ i \$ (\text{LEAST } n. A \$ i \$ n \neq 0) \in \text{associates})$
and $(\forall i. \neg \text{is-zero-row } i \ A \longrightarrow (\forall j. j < i \longrightarrow A \$ j \$ (\text{LEAST } n. A \$ i \$ n \neq 0)$
 $\in \text{residues } (A \$ i \$ (\text{LEAST } n. A \$ i \$ n \neq 0))))$
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 .

primrec *Hermite-reduce-above* :: 'a::unique-euclidean-ring ^ cols::mod-type ^ rows::mod-type => nat
=>'rows=>'cols=> ('a=>'a=>'a) => 'a ^ cols::mod-type ^ rows::mod-type

where *Hermite-reduce-above* A 0 i j res = A
| *Hermite-reduce-above* A (*Suc* n) i j res = (let i' =(*from-nat* n ::'rows);
 A_{ij} = A \$ i \$ j ;
 $A_{i'j}$ = A \$ i' \$ j
in
Hermite-reduce-above (*row-add* A i' i (((res A_{ij} ($A_{i'j}$)) - ($A_{i'j}$)) *div* A_{ij})) n i
 j res)

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

definition *Hermite-of-upt-row-i* A i *ass* res = *foldl* (*Hermite-of-row-i* *ass* res) A
(*map from-nat* [$0..<i$])

definition *Hermite-of* A *ass* res *bezout* =
(let A' = *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 : $n \leq$ *to-nat* a

shows (*Hermite-reduce-above* A n i j 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*)

(*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 : $n \leq$ *to-nat* i **and** a : *to-nat* a < n

shows (*Hermite-reduce-above* A n i j res) \$ a \$ b

= *row-add* A a i ((res (A \$ i \$ j) (A \$ a \$ j) - (A \$ a \$ j)) *div* (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)
define A' where A' = row-add A (from-nat n) i
  ((res (A $ i $ j) (A $ from-nat n $ j) - A $ from-nat n $ j) div A $ i $ j)
have n: n < nrows A
  unfolding nrows-def by (metis Suc.prem1 Suc-le-eq less-trans to-nat-less-card)
show ?case
proof (cases to-nat a = n)
  case False
  have a-less-n: to-nat a < n by (metis False Suc.prem2 less-antisym)
  have Hermite-reduce-above A (Suc n) i j res $ a $ b = Hermite-reduce-above
A' n i j res $ a $ b
  by (simp add: Let-def A'-def)
  also have ... = row-add A' a i ((res (A' $ i $ j) (A' $ a $ j) - A' $ a $ j) div
A' $ i $ j) $ a $ b
  by (rule Suc.hyps[OF - a-less-n], simp add: Suc.prem1 Suc-leD)
  also have ... = row-add A a i ((res (A $ i $ j) (A $ a $ j) - A $ a $ j) div A
$ i $ j) $ a $ b
  unfolding row-add-def A'-def
  using a-less-n Suc.prem n to-nat-from-nat-id[OF n[unfolding nrows-def]]
  by auto
  finally show ?thesis .
next
case True
hence a-eq-fn-n: a = from-nat n by auto
have Hermite-reduce-above A (Suc n) i j res $ a $ b = Hermite-reduce-above
A' n i j res $ a $ b
  by (simp add: Let-def A'-def)
  also have ... = A' $ a $ b
  by (rule Hermite-reduce-above-preserves, simp add: True)
  finally show ?thesis unfolding A'-def a-eq-fn-n .
qed
qed

```

```

lemma Hermite-of-row-preserves-below:
  assumes i-a: i < a
  shows (Hermite-of-row-i ass res A i) $ a $ b = A $ a $ b
proof (auto simp add: Hermite-of-row-i-def Let-def)
  let ?M=(mult-row A i (ass (A $ i $ (LEAST n. A $ i $ n ≠ 0)) div A $ i $
(LEAST n. A $ i $ n ≠ 0)))
  let ?H=Hermite-reduce-above ?M (to-nat i) i (LEAST n. A $ i $ n ≠ 0) res
  have ?H $ a $ b = ?M $ a $ b
  by (rule Hermite-reduce-above-preserves)
  (metis i-a not-le not-less-iff-gr-or-eq to-nat-mono')
  also have ... = A $ a $ b unfolding mult-row-def using i-a by fastforce
  finally show ?H $ a $ b = A $ a $ b .

```

qed

lemma *Hermite-of-row-preserves-previous-cols*:

assumes $b: b < (\text{LEAST } n. A \$ i \$ n \neq 0)$

and *not-zero-i-A*: $\neg \text{is-zero-row } i \ A$

and e : *echelon-form* A

shows $(\text{Hermite-of-row-}i \ \text{ass } \text{res } A \ i) \$ a \$ b = A \$ a \$ b$

proof (*auto simp add: Hermite-of-row-i-def Let-def*)

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

let $?M = (\text{mult-row } A \ i \ (\text{ass } (A \$ i \$?n) \ \text{div } A \$ i \$?n))$

let $?H = \text{Hermite-reduce-above } ?M \ (\text{to-nat } i) \ i \ (\text{LEAST } n. A \$ i \$ n \neq 0) \ \text{res}$

have $Aib: A \$ i \$ b = 0$ **by** (*metis (mono-tags) b not-less-Least*)

show $?H \$ a \$ b = A \$ a \$ b$

proof (*cases a ≥ i*)

case *True*

have $?H \$ a \$ b = ?M \$ a \$ b$

by (*rule Hermite-reduce-above-preserves (metis True to-nat-mono)*)

also have $\dots = A \$ a \$ b$ **using** Aib **unfolding** *mult-row-def* **by** *auto*

finally show *?thesis* .

next

let $?R = \text{row-add } ?M \ a \ i \ ((\text{res } (?M \$ i \$?n) \ (?M \$ a \$?n) - ?M \$ a \$?n) \ \text{div } ?M \$ i \$?n)$

case *False*

hence $ia: i > a$ **by** *simp*

have $?H \$ a \$ b = ?R \$ a \$ b$ **by** (*rule Hermite-reduce-above-works, auto simp add: ia to-nat-mono*)

also have $\dots = A \$ a \$ b$ **using** $ia \ Aib$ **unfolding** *row-add-def mult-row-def* **by** *auto*

finally show *?thesis* .

qed

qed

lemma *echelon-form-Hermite-of-condition1*:

fixes $\text{res } \text{ass } i \ A$

defines $M: M \equiv \text{mult-row } A \ i \ (\text{ass } (A \$ i \$ (\text{LEAST } n. A \$ i \$ n \neq 0)) \ \text{div } A \$ i \$ (\text{LEAST } n. A \$ i \$ n \neq 0))$

defines $H: H \equiv \text{Hermite-reduce-above } M \ (\text{to-nat } i) \ i \ (\text{LEAST } n. A \$ i \$ n \neq 0) \ \text{res}$

assumes e : *echelon-form* A

and a : *ass-function* ass

and *not-zero-iA*: $\neg \text{is-zero-row } i \ A$

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  $\dots = A \$ j \$ a$  unfolding  $M$  mult-row-def using  $ij$  by auto
  also have  $\dots = 0$  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
next
case  $False$  note not-zero-ia-A=False
let ?n=(LEAST n. A $ ia $ n  $\neq$  0)
have A-ia-n: A $ ia $ ?n  $\neq$  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  $\dots \neq 0$  unfolding  $M$  mult-row-def using A-ia-n ass-function-0'[OF
a] by auto
  finally have  $H \$ ia \$ ?n \neq 0$  .
  hence not-zero-ia-H:  $\neg$  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
case  $False$ 
let ?m=(LEAST m. A $ i $ m  $\neq$  0)
let ?R=row-add M ia i ((res (M $ i $ ?m) (M $ ia $ ?m) - M $ ia $ ?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-iaA])
  have A-im: A $ i $ ?n = 0 by (metis (full-types) nm not-less-Least)
  have  $H \$ ia \$ ?n = ?R \$ ia \$ ?n$ 
    unfolding  $H$ 
    by (rule Hermite-reduce-above-works, auto simp add: ia-less-i to-nat-mono)
  also have  $\dots \neq 0$ 
    using ia-less-i A-im A-ia-n unfolding row-add-def M mult-row-def by auto
  finally have  $H \$ ia \$ ?n \neq 0$  .
  hence not-zero-ia-H:  $\neg$  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
qed
qed
lemma row-zero-A-imp-row-zero-H:

```

```

fixes res ass i A
defines M:  $M \equiv \text{mult-row } A \ i \ (\text{ass } (A \ \$ \ i \ \$ \ (\text{LEAST } n. \ A \ \$ \ i \ \$ \ n \neq 0)) \ \text{div } A \ \$ \ i \ \$ \ (\text{LEAST } n. \ A \ \$ \ i \ \$ \ n \neq 0))$ 
defines H:  $H \equiv \text{Hermite-reduce-above } M \ (\text{to-nat } i) \ i \ (\text{LEAST } n. \ A \ \$ \ i \ \$ \ n \neq 0) \ \text{res}$ 
assumes e: echelon-form A
and not-zero-iA:  $\neg \text{is-zero-row } i \ A$ 
and zero-j-A: is-zero-row j A
shows 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-ja:  $A \ \$ \ j \ \$ \ a = 0$ 
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 ≤ 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  $\dots = 0$  unfolding M mult-row-def using True A-ja by auto
finally show ?thesis .
next
let ?n=(LEAST n. A $ i $ n ≠ 0)
let ?R=row-add M j i ((res (M $ i $ ?n) (M $ j $ ?n) - M $ j $ ?n) div M $ i $ ?n)
case False
hence ji: j < i by simp
have  $H \ \$ \ j \ \$ \ a = ?R \ \$ \ j \ \$ \ a$ 
unfolding H by (rule Hermite-reduce-above-works, auto simp add: ji to-nat-mono)
also have  $\dots = 0$ 
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 ?thesis .
qed
qed

```

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

```

fixes res ass i A
defines M:  $M \equiv \text{mult-row } A \ i \ (\text{ass } (A \ \$ \ i \ \$ \ (\text{LEAST } n. \ A \ \$ \ i \ \$ \ n \neq 0)) \ \text{div } A \ \$ \ i \ \$ \ (\text{LEAST } n. \ A \ \$ \ i \ \$ \ n \neq 0))$ 
defines H:  $H \equiv \text{Hermite-reduce-above } M \ (\text{to-nat } i) \ i \ (\text{LEAST } n. \ A \ \$ \ i \ \$ \ n \neq 0) \ \text{res}$ 
assumes i-ia: i < ia
and not-zero-ia-H:  $\neg \text{is-zero-row } ia \ H$ 
shows  $(\text{LEAST } n. \ A \ \$ \ ia \ \$ \ n \neq 0) = (\text{LEAST } n. \ H \ \$ \ ia \ \$ \ n \neq 0)$ 
proof (rule Least-equality)
let ?n=(LEAST n. H $ ia $ n ≠ 0)

```

```

have A $ ia $ ?n = M $ ia $ ?n unfolding M mult-row-def using i-ia by auto
also have ... = H $ ia $ ?n
  unfolding H
  by (rule Hermite-reduce-above-preserves[symmetric])
  (metis i-ia dual-order.strict-iff-order to-nat-mono^)
also have ... ≠ 0 by (metis (mono-tags) LeastI is-zero-row-def' not-zero-ia-H)
finally show A $ ia $ (LEAST n. H $ ia $ n ≠ 0) ≠ 0 .
next
fix y
assume A-ia-y: A $ ia $ y ≠ 0
have H $ ia $ y = M $ ia $ y unfolding H
  by (rule Hermite-reduce-above-preserves)
  (metis i-ia dual-order.strict-iff-order to-nat-mono^)
also have ... ≠ 0 unfolding M mult-row-def using i-ia A-ia-y by auto
finally show (LEAST n. H $ ia $ n ≠ 0) ≤ y by (rule Least-le)
qed

lemma Hermite-reduce-above-Least-eq:
fixes res ass i A
defines M: M ≡ mult-row A i (ass (A $ i $ (LEAST n. A $ i $ n ≠ 0)) div A
$ i $ (LEAST n. A $ i $ n ≠ 0))
defines H: H ≡ Hermite-reduce-above M (to-nat i) i (LEAST n. A $ i $ n ≠
0) res
assumes a: ass-function ass
and not-zero-ia: ¬ is-zero-row i A
shows (LEAST n. A $ i $ n ≠ 0) = (LEAST n. H $ i $ n ≠ 0)
proof (rule Least-equality[symmetric])
let ?n=(LEAST n. A $ i $ n ≠ 0)
have Ain: A $ i $ ?n ≠ 0
  by (metis (mono-tags, lifting) LeastI is-zero-row-def' not-zero-ia)
have H $ i $ ?n = M $ i $ ?n
  unfolding H
  by (rule Hermite-reduce-above-preserves, simp)
also have ... ≠ 0 unfolding M mult-row-def by (auto simp add: Ain ass-function-0'[OF
a])
finally show H $ i $ ?n ≠ 0 .
fix y assume H-iy: H $ i $ y ≠ 0
show (LEAST n. A $ i $ n ≠ 0) ≤ y
proof (rule Least-le, rule ccontr, simp)
  assume Aiy: A $ i $ y = 0
  have H $ i $ y = M $ i $ y
  unfolding H
  by (rule Hermite-reduce-above-preserves, simp)
also have ... = (ass (A $ i $ ?n) div A $ i $ ?n) * A $ i $ y
  unfolding M mult-row-def by auto
also have ... = 0 unfolding Aiy by simp
finally show False using H-iy by contradiction
qed
qed

```

lemma *Hermite-reduce-above-Least-eq-ge*:

fixes *res ass i A*

defines *M*: $M \equiv \text{mult-row } A \ i \ (\text{ass } (A \ \$ \ i \ \$ \ (\text{LEAST } n. \ A \ \$ \ i \ \$ \ n \neq 0)) \ \text{div } A \ \$ \ i \ \$ \ (\text{LEAST } n. \ A \ \$ \ i \ \$ \ n \neq 0))$

defines *H*: $H \equiv \text{Hermite-reduce-above } M \ (\text{to-nat } i) \ i \ (\text{LEAST } n. \ A \ \$ \ i \ \$ \ n \neq 0) \ \text{res}$

assumes *e*: *echelon-form A*

and *not-zero-iA*: $\neg \text{is-zero-row } i \ A$

and *not-zero-ia-A*: $\neg \text{is-zero-row } ia \ A$

and *not-zero-ia-H*: $\neg \text{is-zero-row } ia \ H$

and *ia-less-i*: $ia < i$

shows $(\text{LEAST } n. \ H \ \$ \ ia \ \$ \ n \neq 0) = (\text{LEAST } n. \ A \ \$ \ ia \ \$ \ n \neq 0)$

proof –

let *?least-H* = $(\text{LEAST } n. \ H \ \$ \ ia \ \$ \ n \neq 0)$

let *?least-A* = $(\text{LEAST } n. \ A \ \$ \ ia \ \$ \ n \neq 0)$

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

let *?Ain* = $A \ \$ \ i \ \$ \ ?n$

let *?R* = $\text{row-add } M \ ia \ i \ ((\text{res } (M \ \$ \ i \ \$ \ ?n) \ (M \ \$ \ ia \ \$ \ ?n) - M \ \$ \ ia \ \$ \ ?n) \ \text{div } M \ \$ \ i \ \$ \ ?n)$

have *A-ia-least-A*: $A \ \$ \ ia \ \$ \ ?least-A \neq 0$

by (*metis* (*mono-tags*, *lifting*) *LeastI is-zero-row-def' not-zero-ia-A*)

have *H-ia-least-H*: $H \ \$ \ ia \ \$ \ ?least-H \neq 0$

by (*metis* (*mono-tags*, *lifting*) *LeastI is-zero-row-def' not-zero-ia-H*)

have *A-i-least-ia-0*: $A \ \$ \ i \ \$ \ (\text{LEAST } n. \ A \ \$ \ ia \ \$ \ n \neq 0) = 0$

proof –

have $(\text{LEAST } n. \ A \ \$ \ ia \ \$ \ n \neq 0) < (\text{LEAST } n. \ A \ \$ \ i \ \$ \ n \neq 0)$

using *e echelon-form-condition1 echelon-form-condition2-explicit ia-less-i not-zero-ia* **by** *blast*

thus *?thesis* **using** *not-less-Least* **by** *blast*

qed

have *H-ia-least-A*: $H \ \$ \ ia \ \$ \ ?least-A \neq 0$

proof –

have $H \ \$ \ ia \ \$ \ ?least-A = ?R \ \$ \ ia \ \$ \ ?least-A$

unfolding *H*

by (*rule Hermite-reduce-above-works*, *simp-all add: ia-less-i to-nat-mono*)

also have $\dots \neq 0$ **using** *ia-less-i unfolding row-add-def M mult-row-def*

by (*auto simp add: A-i-least-ia-0 A-ia-least-A*)

finally show *?thesis* .

qed

hence *least-H-le-least-A*: $?least-H \leq ?least-A$

by (*metis* (*mono-tags*) *not-less not-less-Least*)

have *A-i-least-H*: $A \ \$ \ i \ \$ \ ?least-H = 0$

proof –

have $(\text{LEAST } n. \ A \ \$ \ ia \ \$ \ n \neq 0) < (\text{LEAST } n. \ A \ \$ \ i \ \$ \ n \neq 0)$

using *e echelon-form-condition1 echelon-form-condition2-explicit ia-less-i not-zero-ia* **by** *blast*

thus *?thesis* **using** *not-less-Least least-H-le-least-A*

by (metis (mono-tags) dual-order.strict-trans2)
 qed
 have $A \ \$ \ ia \ \$ \ ?least-H \neq 0$
 proof -
 have *ia-not-i*: $ia \neq i$ using *ia-less-i* by simp
 have $?R \ \$ \ ia \ \$ \ ?least-H = H \ \$ \ ia \ \$ \ ?least-H$
 unfolding *H*
 by (rule *Hermite-reduce-above-works*[symmetric], simp-all add: *ia-less-i to-nat-mono*)
 also have $\dots \neq 0$ by (rule *H-ia-least-H*)
 finally have *R-ia-least-H*: $?R \ \$ \ ia \ \$ \ ?least-H \neq 0$.
 hence $A \ \$ \ ia \ \$ \ ?least-H + (res \ (ass \ (?Ain) \ div \ ?Ain \ * \ ?Ain) \ (A \ \$ \ ia \ \$ \ (LEAST \ n. \ A \ \$ \ i \ \$ \ n \neq 0)) - A \ \$ \ ia \ \$ \ (LEAST \ n. \ A \ \$ \ i \ \$ \ n \neq 0))$
 $div \ (ass \ (?Ain) \ div \ ?Ain \ * \ ?Ain) \ * \ (ass \ (?Ain) \ div \ ?Ain \ * \ A \ \$ \ i \ \$ \ ?least-H) \neq 0$
 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*: $?least-A \leq ?least-H$ by (metis (poly-guards-query) *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*: $M \equiv mult-row \ A \ i \ (ass \ (A \ \$ \ i \ \$ \ (LEAST \ n. \ A \ \$ \ i \ \$ \ n \neq 0)) \ div \ A \ \$ \ i \ \$ \ (LEAST \ n. \ A \ \$ \ i \ \$ \ n \neq 0))$
 defines *H*: $H \equiv Hermite-reduce-above \ M \ (to-nat \ i) \ i \ (LEAST \ n. \ A \ \$ \ i \ \$ \ n \neq 0) \ res$
 assumes *e*: *echelon-form A*
 and *a*: *ass-function ass*
 and *not-zero-iA*: $\neg is-zero-row \ i \ A$
 and *not-zero-ia-A*: $\neg is-zero-row \ ia \ A$
 and *not-zero-ia-H*: $\neg is-zero-row \ ia \ H$
 shows $(LEAST \ n. \ H \ \$ \ ia \ \$ \ n \neq 0) = (LEAST \ n. \ A \ \$ \ ia \ \$ \ n \neq 0)$
 proof (cases *ia < i*)
 case *True*
 show *?thesis*
 unfolding *H M*
 by (rule *Hermite-reduce-above-Least-eq-ge*[*OF e not-zero-iA not-zero-ia-A - True*])
 (metis *H M not-zero-ia-H*)
 next
 case *False*
 hence *i-le-ia*: $i \leq ia$ by simp
 show *?thesis*
 proof (cases *ia = i*)

```

case True
show ?thesis
  unfolding True H M
  by (rule Hermite-reduce-above-Least-eq[symmetric, OF a not-zero-iA])
next
case False
hence i-ia: i < ia using i-le-ia by simp
show ?thesis
  unfolding H M
  by (rule Hermite-reduce-above-Least-eq-le[symmetric, OF i-ia], metis H M
not-zero-ia-H)
qed
qed

```

lemma *echelon-form-Hermite-of-condition2:*

```

fixes res ass i A
defines M: M ≡ mult-row A i (ass (A $ i $ (LEAST n. A $ i $ n ≠ 0)) div A
$ i $ (LEAST n. A $ i $ n ≠ 0))
defines H: H ≡ Hermite-reduce-above M (to-nat i) 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 ia-less-j: ia < j
and not-zero-ia-H: ¬ is-zero-row ia H
and not-zero-j-H: ¬ is-zero-row j H
shows (LEAST n. H $ ia $ n ≠ 0) < (LEAST n. H $ j $ n ≠ 0)
proof –
let ?n = (LEAST n. A $ i $ n ≠ 0)
have Ain: A $ i $ ?n ≠ 0
  by (metis (mono-tags) LeastI is-zero-row-def' not-zero-iA)
have not-zero-j-A: ¬ is-zero-row j A
  using row-zero-A-imp-row-zero-H[OF e not-zero-iA] not-zero-j-H
  unfolding H M by blast
have not-zero-ia-A: ¬ is-zero-row ia A
  using row-zero-A-imp-row-zero-H[OF e not-zero-iA] not-zero-ia-H
  unfolding H M by blast
have Least-le-A: (LEAST n. A $ ia $ n ≠ 0) < (LEAST n. A $ j $ n ≠ 0)
  by (rule echelon-form-condition2-explicit[OF e ia-less-j not-zero-ia-A not-zero-j-A])
show ?thesis
proof (cases i < ia)
  case True
  have ij: i < j by (metis True ia-less-j less-trans)
  have Least-A-ia-H-ia: (LEAST n. A $ ia $ n ≠ 0) = (LEAST n. H $ ia $ n
  ≠ 0)
  unfolding H M
  by (rule Hermite-reduce-above-Least-eq-le[OF True], metis H M not-zero-ia-H)
  moreover have Least-A-ia-H-ia: (LEAST n. A $ j $ n ≠ 0) = (LEAST n. H

```

```

$ j $ n ≠ 0)
  unfolding H M
  by (rule Hermite-reduce-above-Least-eq-le[OF ij], metis H M not-zero-j-H)
  ultimately show ?thesis using Least-le-A by simp
next
case False
hence ia-le-i: ia ≤ i by simp
show ?thesis
proof (cases i=ia)
case True thus ?thesis
  using Hermite-reduce-above-Least-eq[OF a not-zero-iA] Least-le-A
  using Hermite-reduce-above-Least-eq-le[OF ia-less-j]
  using not-zero-j-H unfolding H M by fastforce
next
case False
hence ia-less-i: ia < i using ia-le-i by simp
have Least-H-ia-A-ia: (LEAST n. H $ ia $ n ≠ 0) = (LEAST n. A $ ia $ n
≠ 0)
  unfolding H M
  by (rule Hermite-reduce-above-Least-eq-ge[OF e not-zero-iA not-zero-ia-A -
ia-less-i])
  (metis H M not-zero-ia-H)
  show ?thesis
  proof (cases j < i)
  case True
  have (LEAST n. H $ j $ n ≠ 0) = (LEAST n. A $ j $ n ≠ 0)
    unfolding H M
    by (rule Hermite-reduce-above-Least-eq-ge[OF e not-zero-iA not-zero-j-A -
True])
  (metis H M not-zero-j-H)
  thus ?thesis by (simp add: Least-H-ia-A-ia Least-le-A)
  next
  case False
  hence j-ge-i: j ≥ i by auto
  show ?thesis
  proof (cases j=i)
  case True
  have (LEAST n. H $ j $ n ≠ 0) = (LEAST n. A $ j $ n ≠ 0)
    unfolding H M
    using Hermite-reduce-above-Least-eq True a not-zero-iA by fastforce
  thus ?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 (LEAST n. H $ j $ n ≠ 0) = (LEAST n. A $ j $ n ≠ 0)
    unfolding H M
    by (rule Hermite-reduce-above-Least-eq-le[symmetric, OF j-sg-i])
  (metis H M not-zero-j-H)
  thus ?thesis by (simp add: Least-H-ia-A-ia Least-le-A)

```

qed
 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 (*LEAST* *n*. *A* \$ *ia* \$ *n* \neq 0) < (*LEAST* *n*. *A* \$ *j* \$ *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 \$ i \$ (LEAST n. A \$ i \$ n \neq 0)) div A \$ i \$ (LEAST
n. A \$ i \$ n \neq 0)))
(to-nat i) i (LEAST n. A \$ i \$ n \neq 0) res)
and *ia < j*
thus *is-zero-row j* (*Hermite-reduce-above*
(mult-row A i (ass (A \$ i \$ (LEAST n. A \$ i \$ n \neq 0)) div A \$ i \$ (LEAST
n. A \$ i \$ n \neq 0)))
(to-nat i) i (LEAST n. A \$ i \$ n \neq 0) res)
using *echelon-form-Hermite-of-condition1*[*OF e a*] **by** *blast*
next
fix *ia j*
let *?H*=(*Hermite-reduce-above* (*mult-row A i (ass (A \$ i \$ (LEAST n. A \$ i \$*
n \neq 0))
div A \$ i \$ (LEAST n. A \$ i \$ n \neq 0))) (to-nat i) i (LEAST n. A \$ i \$ n \neq
0) res)
assume \neg *is-zero-row i A*
and *ia < j*
and \neg *is-zero-row ia ?H*
and \neg *is-zero-row j ?H*
thus (*LEAST* *n*. *?H* \$ *ia* \$ *n* \neq 0) < (*LEAST* *n*. *?H* \$ *j* \$ *n* \neq 0)
using *echelon-form-Hermite-of-condition2*[*OF e a*] **by** *blast*
 qed

lemma *echelon-form-fold-Hermite-of-row-i*:
assumes *e*: echelon-form *A* **and** *a*: ass-function *ass* **and** *r*: res-function *res*
shows echelon-form (foldl (Hermite-of-row-i *ass* *res*) *A* (map from-nat [0..*k*]))
proof (induct *k*)
case 0
thus ?*case* **by** (simp add: *e*)
next
case (Suc *k*)
show ?*case* **by** (simp, rule echelon-form-Hermite-of-row[OF *a* *r* Suc.hyps])
qed

lemma *echelon-form-Hermite-of-upt-row-i*:
assumes *e*: echelon-form *A* **and** *a*: ass-function *ass* **and** *r*: res-function *res*
shows echelon-form (Hermite-of-upt-row-i *A* *k* *ass* *res*)
unfolding Hermite-of-upt-row-i-def
using echelon-form-fold-Hermite-of-row-i *assms* **by** auto

lemma *echelon-form-Hermite-of*:
fixes *A*::'a::{\bezout-ring-div,normalization-semidom,unique-euclidean-ring} ^ cols::{\mod-type} ^ rows::{\mod-type}
assumes *a*: ass-function *ass*
and *r*: res-function *res*
and *b*: is-bezout-ext *bezout*
shows echelon-form (Hermite-of *A* *ass* *res* *bezout*)
unfolding Hermite-of-def Hermite-of-upt-row-i-def *Let-def* *nrows-def*
by (rule echelon-form-fold-Hermite-of-row-i[OF echelon-form-echelon-form-of[OF *b*] *a* *r*])

lemma *in-ass-Hermite-of-row*:
assumes *a*: ass-function *ass*
and res-function *res*
and *not-zero-i-A*: \neg 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 (?*Ain*) div ?*Ain* * ?*Ain*
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* **by** (*simp add: associatedD2*)
qed
also have ... ∈ *range* *ass* **by** *simp*
finally show ?*H* \$ *i* \$ (*LEAST* *n*. ?*H* \$ *i* \$ *n* ≠ 0) ∈ *range* *ass* .
qed

lemma *Hermite-of-upt-row-preserves-below*:
assumes *i*: *to-nat* *a* ≥ *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*
by (*simp add: Hermite-of-upt-row-i-def*,
metis Hermite-of-upt-row-i-def Hermite-of-row-preserves-below Suc.hyps Suc.prem)
Suc-leD Suc-le-eq from-nat-mono from-nat-to-nat-id to-nat-less-card)
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*: ¬ *is-zero-row* *a* *A*
and *not-zero-i-A*: ¬ *is-zero-row* *i* *A*
and *e*: *echelon-form* *A*
and *a*: *ass-function* *ass*
and *n*: *n* ≤ *to-nat* *i*
shows ¬ *is-zero-row* *a* (*Hermite-reduce-above* *M* *n* *i* (*LEAST* *n*. *A* \$ *i* \$ *n* ≠ 0)
res)
proof –
let ?*H* = (*Hermite-reduce-above* *M* *n* *i* (*LEAST* *n*. *A* \$ *i* \$ *n* ≠ 0) *res*)
let ?*n* = *LEAST* *n*. *A* \$ *a* \$ *n* ≠ 0
let ?*m* = *LEAST* *n*. *A* \$ *i* \$ *n* ≠ 0
have *Aan*: *A* \$ *a* \$?*n* ≠ 0
by (*metis* (*mono-tags*) *LeastI not-zero-a-A is-zero-row-def'*)
have *Aim*: *A* \$ *i* \$?*m* ≠ 0
by (*metis* (*mono-tags*) *LeastI not-zero-i-A is-zero-row-def'*)
show ?*thesis*
proof (*cases n ≤ to-nat a*)
case *True*

have $?H \ \$ \ a \ \$ \ ?n = M \ \$ \ a \ \$ \ ?n$
by (*metis Hermite-reduce-above-preserves True*)
also have $\dots \neq 0$ **unfolding** M *mult-row-def* **using** *ass-function-0'*[$OF \ a$] Aan
by *auto*
finally show $?thesis$ **unfolding** *is-zero-row-def is-zero-row-upt-k-def ncols-def*
by *auto*
next
let $?R = row-add \ M \ a \ i$
 $((res \ (M \ \$ \ i \ \$ \ ?m) \ (M \ \$ \ a \ \$ \ ?m) - M \ \$ \ a \ \$ \ ?m) \ div \ M \ \$ \ i \ \$ \ ?m)$
case *False*
hence $a-n$: *to-nat* $a < n$ **by** *simp*
have ai : $a < i$
by (*metis False dual-order.trans n nat-less-le not-less-iff-gr-or-eq to-nat-mono*)
have ($LEAST \ n. \ A \ \$ \ a \ \$ \ n \neq 0$) $< ?m$
by (*rule echelon-form-condition2-explicit*[$OF \ e \ ai \ not-zero-a-A \ not-zero-i-A$])
hence Ain : $A \ \$ \ i \ \$ \ (LEAST \ n. \ A \ \$ \ a \ \$ \ n \neq 0) = 0$
by (*metis (full-types) not-less-Least*)
have $a-not-i$: $a \neq i$ **by** (*metis False n*)
have $?H \ \$ \ a \ \$ \ ?n = ?R \ \$ \ a \ \$ \ ?n$
by (*rule Hermite-reduce-above-works*[$OF \ n \ a-n$])
also have $\dots \neq 0$ **using** $a-not-i$ $Aan \ Ain$ **unfolding** *row-add-def* M *mult-row-def*
by *auto*
finally show $?thesis$ **unfolding** *is-zero-row-def is-zero-row-upt-k-def ncols-def*
by *auto*
qed
qed

lemma *Least-Hermite-of-row-i*:

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

lemma *Least-Hermite-of-row-i2*:

assumes i : \neg *is-zero-row* i A **and** k : \neg *is-zero-row* k A
and e : *echelon-form* A
and a : *ass-function* ass
shows $(LEAST\ n.\ Hermite-of-row-i\ ass\ res\ A\ k\ \$\ i\ \$\ n\ \neq\ 0) = (LEAST\ n.\ A\ \$\ i\ \$\ n\ \neq\ 0)$
proof –
let $?M = mult-row\ A\ k\ (ass\ (A\ \$\ k\ \$\ (LEAST\ n.\ A\ \$\ k\ \$\ n\ \neq\ 0))\ div\ A\ \$\ k\ \$\ (LEAST\ n.\ A\ \$\ k\ \$\ n\ \neq\ 0))$
let $?H = Hermite-reduce-above\ ?M\ (to-nat\ k)\ k\ (LEAST\ n.\ A\ \$\ k\ \$\ n\ \neq\ 0)\ res$
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** $\dots = (LEAST\ n.\ A\ \$\ i\ \$\ n\ \neq\ 0)$
by $(rule\ Hermite-reduce-above-Least[OF\ e\ a\ k\ i])$
 $(simp\ add:\ a\ e\ i\ k\ not-zero-Hermite-reduce-above)$
finally **show** $?thesis$.
qed

lemma *Hermite-of-row-i-works*:

fixes i A ass
defines $n:n \equiv (LEAST\ n.\ A\ \$\ i\ \$\ n\ \neq\ 0)$
defines $M:M \equiv (mult-row\ A\ i\ (ass\ (A\ \$\ i\ \$\ n)\ div\ A\ \$\ i\ \$\ n))$
assumes ai : $a < i$
and i : \neg *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* $M\ n$ **using** i **by** *simp*
also **have** $\dots = row-add\ M\ a\ i\ ((res\ (M\ \$\ i\ \$\ n)\ (M\ \$\ a\ \$\ n))$
 $- M\ \$\ a\ \$\ n)\ div\ M\ \$\ i\ \$\ n)\ \$\ a\ \$\ b$
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 $n:n \equiv (LEAST\ n.\ A\ \$\ i\ \$\ n\ \neq\ 0)$
defines $M:M \equiv (mult-row\ A\ i\ (ass\ (A\ \$\ i\ \$\ n)\ div\ A\ \$\ i\ \$\ n))$
assumes i : \neg *is-zero-row* i A
shows *Hermite-of-row-i* $ass\ res\ A\ i\ \$\ i\ \$\ b = M\ \$\ i\ \$\ b$
proof –
let $?H = Hermite-reduce-above\ M\ (to-nat\ i)\ i\ n\ res$
have *Hermite-of-row-i* $ass\ res\ A\ i\ \$\ i\ \$\ b = ?H\ \$\ i\ \$\ b$
unfolding *Hermite-of-row-i-def* *Let-def* $M\ n$ **using** i **by** *simp*
also **have** $\dots = M\ \$\ i\ \$\ b$ **by** $(rule\ Hermite-reduce-above-preserves,\ simp)$

finally show *?thesis* .
qed

lemma *Hermite-of-upt-row-preserves-nonzero-rows-ge*:
assumes *i*: \neg *is-zero-row* *i* *A* **and** *i2*: *to-nat* $i \geq k$
shows \neg *is-zero-row* *i* (*Hermite-of-upt-row-i* *A* *k* *ass* *res*)
proof –
let *?n* = *LEAST* *n*. $A \ \$ \ i \ \$ \ n \neq 0$
have *Ain*: $A \ \$ \ i \ \$ \ ?n \neq 0$ **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 $\dots \neq 0$ **by** (*metis* (*mono-tags*) *LeastI* *i* *is-zero-row-def*)
finally have *Hermite-of-upt-row-i* *A* *k* *ass* *res* $\ \$ \ i \ \$ \ (LEAST \ n. \ A \ \$ \ i \ \$ \ n \neq 0)$
 $\neq 0$.
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*: \neg *is-zero-row* *i* *A*
and *i2*: *to-nat* $i \geq k$
shows $(LEAST \ n. \ Hermite-of-upt-row-i \ A \ k \ ass \ res \ \$ \ i \ \$ \ n \neq 0) = (LEAST \ n. \ A \ \$ \ i \ \$ \ n \neq 0)$
proof (*rule* *Least-equality*)
let *?n* = *LEAST* *n*. $A \ \$ \ i \ \$ \ n \neq 0$
have *Ain*: $A \ \$ \ i \ \$ \ ?n \neq 0$ **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 $\dots \neq 0$ **by** (*metis* (*mono-tags*) *LeastI* *i* *is-zero-row-def*)
finally show *Hermite-of-upt-row-i* *A* *k* *ass* *res* $\ \$ \ i \ \$ \ (LEAST \ n. \ A \ \$ \ i \ \$ \ n \neq 0)$
 $\neq 0$.
fix *y*
assume *H*: *Hermite-of-upt-row-i* *A* *k* *ass* *res* $\ \$ \ i \ \$ \ y \neq 0$
show $(LEAST \ n. \ A \ \$ \ i \ \$ \ n \neq 0) \leq y$
by (*rule* *Least-le*, *metis* *Hermite-of-upt-row-preserves-below* *H* *i2*)
qed

lemma *Hermite-of-upt-row-i-Least*:
assumes *iA*: \neg *is-zero-row* *i* *A*
and *e*: *echelon-form* *A*
and *a*: *ass-function* *ass*
and *r*: *res-function* *res*
and *k*: $k \leq nrows \ A$
shows $(LEAST \ n. \ Hermite-of-upt-row-i \ A \ k \ ass \ res \ \$ \ i \ \$ \ n \neq 0) = (LEAST \ n. \ A \ \$ \ i \ \$ \ n \neq 0)$

```

proof (cases to-nat  $i \geq k$ )
  case True
  thus ?thesis using Hermite-of-upt-row-i-Least-ge  $iA$  by blast
next
  case False
  hence  $i$ -less- $k$ : to-nat  $i < k$  by simp
  thus ?thesis using  $e$   $iA$   $k$ 
  proof (induct  $k$  arbitrary:  $A$ )
    case 0
    thus ?case
    unfolding Hermite-of-upt-row-i-def by simp
  next
  case (Suc  $k$ )
  have  $k$ :  $k \leq n$ rows  $A$  using Suc.premis unfolding nrows-def by simp
  have  $k2$ :  $k < n$ rows  $A$  using Suc.premis unfolding nrows-def by simp
  define  $A'$  where  $A' = \text{foldl}$  (Hermite-of-row-i ass res)  $A$  (map from-nat  $[0..<k]$ )
  have  $A'$ -def2:  $A' = \text{Hermite-of-upt-row-i}$   $A$   $k$  ass res
    unfolding Hermite-of-upt-row-i-def  $A'$ -def ..
  have  $e$ : echelon-form  $A'$ 
    unfolding  $A'$ -def2
  by (rule echelon-form-Hermite-of-upt-row-i[OF -  $a$   $r$ ], auto simp add: Suc.premis)
  show ?case
  proof (cases to-nat  $i = k$ )
    case True
    have  $i$ -fn- $k$ : from-nat  $k = i$  by (metis True from-nat-to-nat-id)
    have not-zero- $i$ - $A'$ :  $\neg$  is-zero-row  $i$   $A'$ 
      unfolding  $A'$ -def2
      by (rule Hermite-of-upt-row-preserves-nonzero-rows-ge, auto simp add:
        Suc.premis 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  $A'$ -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  $A'$ -def2
      by (rule Hermite-of-upt-row-i-Least-ge, auto simp add: True Suc.premis)
    finally show ?thesis .
  next
  case False
  hence  $i$ -less- $k$ : to-nat  $i < k$  using Suc.premis by simp
  hence  $i$ -less- $k2$ :  $i < \text{from-nat } k$ 
    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.prem k)
    next
      case False
      have not-zero-i-A':  $\neg$  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.prem
k)
      finally show ?thesis .
    qed
  qed
qed
qed

```

lemma *Hermite-of-upt-row-preserves-nonzero-rows:*

```

  assumes i:  $\neg$  is-zero-row i A
  and e: echelon-form A
  and a: ass-function ass
  and r: res-function res
  and k:  $k \leq$  nrows A
  shows  $\neg$  is-zero-row i (Hermite-of-upt-row-i A k ass res)
proof -
  let ?n=LEAST n. A $ i $ n  $\neq$  0
  have Ain: A $ i $ ?n  $\neq$  0 by (metis (mono-tags) LeastI i is-zero-row-def')
  show ?thesis
  proof (cases to-nat i  $\geq$  k)
    case True thus ?thesis using Hermite-of-upt-row-preserves-nonzero-rows-ge i
by blast
  next
    case False
    hence i-less-k: to-nat i < k by auto
    thus ?thesis using i k
  proof (induct k)
    case 0
    thus ?case by (metis less-nat-zero-code)
  next
    case (Suc k)
    have k-nrows:  $k \leq$  nrows A using Suc.prem unfolding nrows-def by auto
    have k-nrows2:  $k <$  nrows A using Suc.prem unfolding nrows-def by auto
    define A' where 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 ..
  have least-A'-A: (LEAST n. A' $ i $ n ≠ 0) = (LEAST n. A $ i $ n ≠ 0)
    unfolding A'-def2
    by (rule Hermite-of-upt-row-i-Least[OF - e a r], auto simp add: k-nrows
      Suc.prem)
  have e: echelon-form A'
    unfolding A'-def2 by (simp add: a e echelon-form-Hermite-of-upt-row-i r)
  show ?case
  proof (cases to-nat i = k)
    let ?M = mult-row A' i (ass (A' $ i $ (LEAST n. A' $ i $ n ≠ 0)) div A' $
i $ (LEAST n. A' $ i $ n ≠ 0))
    case True
    hence fn-k-i: from-nat k = i by (metis from-nat-to-nat-id)
    have not-zero-i-A': ¬ is-zero-row i A'
      by (unfold A'-def2)
      (rule Hermite-of-upt-row-preserves-nonzero-rows-ge, auto simp add: True
        Suc.prem)
    have A'-i-l: (A' $ i $ (LEAST n. A' $ i $ n ≠ 0)) ≠ 0
      by (metis (mono-tags) LeastI is-zero-row-def' not-zero-i-A')
    have Hermite-of-upt-row-i A (Suc k) ass res $ i $ ?n =
      Hermite-of-row-i ass res A' (from-nat k) $ i $ ?n
      unfolding Hermite-of-upt-row-i-def A'-def by simp
    also have ... = ?M $ i $ ?n unfolding fn-k-i
      by (rule Hermite-of-row-i-works2[OF not-zero-i-A'])
    also have ... ≠ 0
      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.prem(1) less-antisym)
  hence i-k2: i < from-nat k using i-k Suc.prem
    by (metis from-nat-mono from-nat-to-nat-id k-nrows2 n-rows-def)
  have not-zero-i-A': ¬ is-zero-row i A'
    unfolding A'-def2 by (rule Suc.hyps[OF i-k Suc.prem(2) k-nrows])
  show ?thesis
  proof (cases is-zero-row (from-nat k) A')
    case False
    have Ain: A' $ i $ (LEAST n. A $ i $ n ≠ 0) ≠ 0 unfolding
least-A'-A[symmetric]
      by (metis (mono-tags) LeastI is-zero-row-def' not-zero-i-A')
    have Akn: A' $ (from-nat k) $ (LEAST n. A $ i $ n ≠ 0) = 0
      proof -
        have (LEAST n. A' $ i $ n ≠ 0) < (LEAST n. A' $ (from-nat k) $ n
          ≠ 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=(LEAST n. A' $ from-nat k $ n ≠ 0)
  let ?M=mult-row A' (from-nat k)
    (ass (A' $ from-nat k $ ?m) div
     A' $ from-nat k $ ?m)
  have Hermite-of-upt-row-i A (Suc k) ass res $ i $ ?n =
    Hermite-of-row-i ass res A' (from-nat k) $ i $ ?n
  unfolding Hermite-of-upt-row-i-def A'-def by simp
  also have ... = row-add (mult-row A' (from-nat k)
    (ass (A' $ from-nat k $ ?m) div A' $ from-nat k $ ?m)) i (from-nat k)
    ((res (?M $ from-nat k $ ?m) (?M $ i $ ?m) - ?M $ i $ ?m) div ?M $
from-nat k $ ?m) $ i $ ?n
    by (rule Hermite-of-row-i-works[OF i-k2 False])
  also have ... ≠ 0 using i-k2 Ain 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
  using True by (simp add: Hermite-of-upt-row-i-def Hermite-of-row-i-def
A'-def Let-def)
  thus ?thesis using not-zero-i-A' unfolding A'-def2 by simp
  qed
  qed
  qed
  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 $ i $ (LEAST n. A $ i $ n ≠ 0) ∈ 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.prem1 unfolding nrows-def by simp
  have k2: k < nrows A using Suc.prem2 unfolding nrows-def by simp

```

```

define  $A'$  where  $A' = \text{foldl } (\text{Hermite-of-row-}i \text{ ass res}) A (\text{map from-nat } [0..<k])$ 
have  $A'\text{-def2}: A' = \text{Hermite-of-upt-row-}i A k \text{ ass res}$  unfolding  $\text{Hermite-of-upt-row-}i\text{-def}$ 
 $A'\text{-def} \dots$ 
define  $M$  where  $M = \text{mult-row } A' i (\text{ass } (A' \$ i \$ (\text{LEAST } n. A' \$ i \$ n \neq 0)))$ 
 $\text{div } A' \$ i \$ (\text{LEAST } n. A' \$ i \$ n \neq 0)$ 
have  $\text{not-zero-}A': \neg \text{is-zero-row } i A'$ 
using  $\text{Hermite-of-upt-row-preserves-nonzero-rows}[OF \text{ not-zero-}i\text{-}A \text{ e a r } k]$ 
unfolding  $A'\text{-def Hermite-of-upt-row-}i\text{-def}$  by  $\text{simp}$ 
have  $e\text{-}A': \text{echelon-form } A'$  by  $(\text{metis } A'\text{-def a e echelon-form-fold-Hermite-of-row-}i$ 
 $r)$ 
have  $\text{least-eq}: (\text{LEAST } n. (\text{Hermite-of-row-}i \text{ ass res } A' i) \$ i \$ n \neq 0) = (\text{LEAST}$ 
 $n. A' \$ i \$ n \neq 0)$ 
by  $(\text{rule } \text{Least-Hermite-of-row-}i[OF \text{ not-zero-}A' \text{ e-}A' a])$ 
have  $\text{least-eq2}: (\text{LEAST } n. A' \$ i \$ n \neq 0) = (\text{LEAST } n. A \$ i \$ n \neq 0)$ 
unfolding  $A'\text{-def2}$ 
by  $(\text{rule } \text{Hermite-of-upt-row-}i\text{-Least}[OF \text{ not-zero-}i\text{-}A \text{ e a r } k])$ 
show  $?case$ 
proof  $(\text{cases to-nat } i = k)$ 
case  $\text{True}$ 
have  $\text{fn-}k\text{-}i: \text{from-nat } k = i$  by  $(\text{metis } \text{True from-nat-to-nat-id})$ 
have  $(\text{Hermite-of-upt-row-}i A (\text{Suc } k) \text{ ass res}) \$ i \$ (\text{LEAST } n. A \$ i \$ n \neq$ 
 $0) =$ 
 $(\text{Hermite-of-row-}i \text{ ass res } A' i) \$ i \$ (\text{LEAST } n. A \$ i \$ n \neq 0)$ 
unfolding  $\text{Hermite-of-upt-row-}i\text{-def}$ 
by  $(\text{simp add: } A'\text{-def fn-}k\text{-}i)$ 
also have  $\dots = (\text{Hermite-of-row-}i \text{ ass res } A' i) \$ i \$ (\text{LEAST } n. (\text{Hermite-of-row-}i$ 
 $\text{ ass res } A' i) \$ i \$ n \neq 0)$ 
unfolding  $\text{least-eq least-eq2} \dots$ 
also have  $\dots \in \text{range ass}$  by  $(\text{rule } \text{in-ass-Hermite-of-row}[OF a r \text{ not-zero-}A])$ 
finally show  $?thesis \dots$ 
next
case  $\text{False}$ 
hence  $i\text{-less-}k: \text{to-nat } i < k$  using  $\text{Suc.prem s}$  by  $\text{auto}$ 
hence  $i\text{-less-}k2: i < \text{from-nat } k$  using  $\text{Suc.prem s}$ 
by  $(\text{metis from-nat-mono from-nat-to-nat-id } k2 \text{ n rows-def})$ 
show  $?thesis$ 
proof  $(\text{cases is-zero-row } (\text{from-nat } k) A')$ 
case  $\text{True}$ 
have  $\text{Hermite-of-upt-row-}i A (\text{Suc } k) \text{ ass res} = \text{Hermite-of-upt-row-}i A k \text{ ass}$ 
 $\text{res}$ 
using  $\text{True}$  by  $(\text{simp add: Hermite-of-upt-row-}i\text{-def Hermite-of-row-}i\text{-def}$ 
 $A'\text{-def Let-def})$ 
thus  $?thesis$  using  $\text{Suc.hyps not-zero-}i\text{-}A k i\text{-less-}k$  by  $\text{auto}$ 
next
case  $\text{False}$ 
have  $(\text{Hermite-of-upt-row-}i A (\text{Suc } k) \text{ ass res}) \$ i \$ (\text{LEAST } n. A \$ i \$ n \neq$ 
 $0)$ 
 $= (\text{Hermite-of-row-}i \text{ ass res } A' (\text{from-nat } k)) \$ i \$ (\text{LEAST } n. A \$ i \$ n \neq$ 
 $0)$ 

```

```

    unfolding Hermite-of-upt-row-i-def A'-def by auto
  also have ... = A' $ i $ (LEAST n. A $ i $ n ≠ 0)
  proof (rule Hermite-of-row-preserves-previous-cols[OF - False e-A'])
    show (LEAST n. A $ 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 ass
  unfolding A'-def using Suc.premys 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 ≤ nrow 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
  by (metis Hermite-of-upt-row-preserves-below ik)
  also have ... = 0 using i unfolding is-zero-row-def is-zero-row-upt-k-def nrow-def
  by simp
  finally show Hermite-of-upt-row-i A k ass res $ i $ j = 0 .
  qed

```

```

lemma Hermite-of-upt-row-preserves-zero-rows:
  fixes A::'a::{bezout-ring-div,normalization-semidom,unique-euclidean-ring} ^ cols:: {mod-type} ^ rows:: {mod-type}
  assumes i: is-zero-row i A
  and e: echelon-form A and a: ass-function ass and r: res-function res and k: k
≤ nrow A
  shows is-zero-row i (Hermite-of-upt-row-i A k ass res)
proof (cases to-nat i ≥ k)
  case True
  show ?thesis by (rule Hermite-of-upt-row-preserves-zero-rows-ge[OF i k True])
next
  case False
  hence i-k: to-nat i < k by simp
  show ?thesis
  using k i-k
  proof (induct k)

```

```

case 0
thus ?case unfolding 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
define A' where  $A' = \text{foldl } (\text{Hermite-of-row-}i \text{ ass res}) A (\text{map from-nat } [0..<k])$ 
have A'-def2:  $A' = \text{Hermite-of-upt-row-}i A k \text{ ass res}$ 
  unfolding Hermite-of-upt-row-i-def A'-def ..
show ?case unfolding 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 (Hermite-of-upt-row-i A (Suc k) ass res) =
    (Hermite-of-row-i ass res A' i)
  unfolding Hermite-of-upt-row-i-def
  by (simp add: A'-def fn-k-i)
  moreover have is-zero-row i (Hermite-of-upt-row-i A k ass res)
  by (rule Hermite-of-upt-row-preserves-zero-rows-ge[OF i k], simp add: True)
  ultimately show (Hermite-of-upt-row-i A (Suc k) ass res) $ i $ j = 0
  unfolding is-zero-row-def' A'-def2 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
  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 $ 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 $ j = Hermite-of-upt-row-i A k ass res $ i $ 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  $\neg \text{is-zero-row } i (\text{Hermite-of-upt-row-}i A k \text{ ass res})$ 
  using echelon-form-condition1
  by (metis A'-def2 False e a r echelon-form-Hermite-of-upt-row-i i-less-k2)
  ultimately show ?thesis by contradiction
qed
qed
qed

```

qed

lemma *Hermite-of-preserves-zero-rows*:

fixes $A::'a::\{\text{bezout-ring-div,normalization-semidom,unique-euclidean-ring}\}^{\sim}\text{cols}::\{\text{mod-type}\}^{\sim}\text{rows}::\{\text{mod-type}\}^{\sim}$
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 i *echelon-form-echelon-form-of*[OF b] a r])
(*auto simp add: nrows-def*)

lemma *Hermite-of-Least*:

fixes $A::'a::\{\text{bezout-ring-div,normalization-semidom,unique-euclidean-ring}\}^{\sim}\text{cols}::\{\text{mod-type}\}^{\sim}\text{rows}::\{\text{mod-type}\}^{\sim}$
assumes i : \neg *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* $\$ i \$ n \neq 0$) = (*LEAST* n .
(*echelon-form-of* A *bezout*) $\$ i \$ n \neq 0$)
proof –
have *non-zero-i-eA*: \neg *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])
show *?thesis*
unfolding *Hermite-of-def* *Let-def*
by (*rule* *Hermite-of-upt-row-i-Least*[OF *non-zero-i-eA* e a r], *simp add: nrows-def*)
qed

lemma *in-associates-Hermite-of*:

fixes $A::'a::\{\text{bezout-ring-div,normalization-semidom,unique-euclidean-ring}\}^{\sim}\text{cols}::\{\text{mod-type}\}^{\sim}\text{rows}::\{\text{mod-type}\}^{\sim}$
assumes a : *ass-function* ass
and r : *res-function* res
and b : *is-bezout-ext* *bezout*
and i : \neg *is-zero-row* i (*Hermite-of* A ass res *bezout*)
shows *Hermite-of* A ass res *bezout* $\$ i \$ (LEAST n. Hermite-of A ass res bezout \$ i \$ n \neq 0) \in \text{range } ass$
proof –
have *non-zero-i-eA*: \neg *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*[*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. A \$ 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)*
define *M* **where** *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 $\forall b \text{ ba. } \exists bb. \text{ba} + bb * b = \text{res } b \text{ ba}$
using *r* **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' nonzero-mult-div-cancel-left 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: to-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 ≠ 0)*
∈ range (res (Hermite-of-upt-row-i A k ass res \$ i \$ (LEAST n. A \$ i \$ n ≠ 0)))
using *k not-zero-i-A k2*
proof (*induct k*)
case *0* **thus** *?case* **by** *auto*
next
case (*Suc k*)
let *?n=(LEAST n. A \$ i \$ n ≠ 0)*
have *k: k ≤ nrows A* **using** *Suc.prem*s **unfolding** *nrows-def* **by** *simp*
have *k2: k < nrows A* **using** *Suc.prem*s **unfolding** *nrows-def* **by** *simp*
define *A'* **where** *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 ..
define M where M = mult-row A' i (ass (A' $ i $ (LEAST n. A' $ i $ n ≠ 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 ≠ 0) = (LEAST
n. A' $ i $ n ≠ 0)
by (rule Least-Hermite-of-row-i[OF not-zero-A' e-A' a])
have least-eq2: (LEAST n. A' $ i $ n ≠ 0) = (LEAST n. A $ i $ n ≠ 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 H-rw: (Hermite-of-upt-row-i A (Suc k) ass res $ i $ (LEAST n. A $ i $
n ≠ 0))
= (Hermite-of-row-i ass res A' i $ i $ (LEAST n. A' $ i $ n ≠ 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 ≠
0) =
(Hermite-of-row-i ass res A' i) $ j $ (LEAST n. A $ i $ n ≠ 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.premis by auto
hence i-less-k2: i < from-nat k using Suc.premis
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 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 $ 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 $ 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 $ 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 ≠ 0)))
  unfolding A'-def2
  by (rule Suc.hyps[OF i-less-k], auto simp add: Suc.prem k)
  also have ... = range (res (Hermite-of-upt-row-i A (Suc k) ass res $ i $
(LEAST n. A $ i $ n ≠ 0)))
  unfolding H-rw A'-def2 ..
  finally show ?thesis .
  qed
  qed
  qed

```

lemma *in-residues-Hermite-of:*

```

fixes A::'a::{bezout-ring-div,normalization-semidom,unique-euclidean-ring} ~'cols::{mod-type} ~'rows::{mod-t
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* $\$ i \$ n \neq 0$)
 $\in \text{range} (\text{res} (\text{Hermite-of A ass res bezout} \$ i \$ (\text{LEAST } n. \text{Hermite-of A ass res bezout} \$ i \$ n \neq 0)))$
proof –
have *non-zero-i-eA*: $\neg \text{is-zero-row } i$ (*echelon-form-of A bezout*)
using *Hermite-of-preserves-zero-rows*[*OF - a r b*] **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*) ($\lambda c. \text{range} (\text{res } c)$) (*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: r res-function-Complete-set-residues*)
show *echelon-form* (*Hermite-of A ass res bezout*)
by (*simp add: a b echelon-form-Hermite-of r*)
fix *i*
assume *i*: $\neg \text{is-zero-row } i$ (*Hermite-of A ass res bezout*)
show *Hermite-of A ass res bezout* $\$ i \$$ (*LEAST n. Hermite-of A ass res bezout* $\$ i \$ n \neq 0$) $\in \text{range } \text{ass}$
by (*rule in-associates-Hermite-of*[*OF a r b i*])
next
fix *i j* **assume** *i*: $\neg \text{is-zero-row } i$ (*Hermite-of A ass res bezout*) **and** *j*: $j < i$
show *Hermite-of A ass res bezout* $\$ j \$$ (*LEAST n. Hermite-of A ass res bezout* $\$ i \$ n \neq 0$)
 $\in \text{range} (\text{res} (\text{Hermite-of A ass res bezout} \$ i \$ (\text{LEAST } n. \text{Hermite-of A ass res bezout} \$ i \$ n \neq 0)))$
by (*rule in-residues-Hermite-of*[*OF 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  $\exists P. \text{invertible } P \wedge \text{Hermite-reduce-above } A \ n \ i \ j \ \text{res} = P ** A$ 
using  $n$ 
proof (induct n arbitrary: A)
  case  $0$  thus  $?case$  by (auto, metis invertible-def matrix-mul-lid)
next
  case ( $\text{Suc } n$ )
  let  $?R = (\text{row-add } A \ (\text{from-nat } n) \ i \ ((\text{res } (A \ \$ \ i \ \$ \ j) \ (A \ \$ \ \text{from-nat } n \ \$ \ j) - A \ \$ \ \text{from-nat } n \ \$ \ j) \ \text{div } A \ \$ \ i \ \$ \ j))$ 
  obtain  $Q$  where  $\text{inv-Q}: \text{invertible } Q$  and  $H\text{-QR}: \text{Hermite-reduce-above } ?R \ n \ i \ j \ \text{res} = Q ** ?R$ 
  using  $\text{Suc.hyps } \text{Suc.premis}$  by auto
  let  $?P = (\text{row-add } (\text{mat } 1) \ (\text{from-nat } n) \ i \ ((\text{res } (A \ \$ \ i \ \$ \ j) \ (A \ \$ \ \text{from-nat } n \ \$ \ j) - A \ \$ \ \text{from-nat } n \ \$ \ j) \ \text{div } A \ \$ \ i \ \$ \ j))$ 
  have  $\text{inv-P}: \text{invertible } ?P$ 
  proof (rule invertible-row-add)
    show  $\text{mod-type-class.from-nat } n \neq i$ 
    by (metis Suc.premis Suc-le-eq add-to-nat-def from-nat-mono less-irrefl monoid-add-class.add.right-neutral to-nat-0 to-nat-less-card)
  qed
  have  $\text{inv-QP}: \text{invertible } (Q ** ?P)$  by (metis inv-P inv-Q invertible-mult)
  have  $\text{Hermite-reduce-above } A \ (\text{Suc } n) \ i \ j \ \text{res} = \text{Hermite-reduce-above } ?R \ n \ i \ j \ \text{res}$ 
  by (auto simp add: Let-def)
  also have  $\dots = Q ** ?R$  unfolding  $H\text{-QR}$  ..
  also have  $\dots = Q ** (?P ** A)$  by (subst row-add-mat-1[symmetric], rule refl)
  also have  $\dots = (Q ** ?P) ** A$  by (simp add: matrix-mul-assoc)
  finally show  $?case$  using  $\text{inv-QP}$  by auto
qed

```

lemma *invertible-Hermite-of-row-i:*

```

assumes  $a: \text{ass-function } \text{ass}$ 
shows  $\exists P. \text{invertible } P \wedge \text{Hermite-of-row-i } \text{ass} \ \text{res } A \ i = P ** A$ 
unfolding  $\text{Hermite-of-row-i-def}$ 
proof (auto simp add: Let-def, metis invertible-def matrix-mul-lid)
  let  $?n = \text{LEAST } n. A \ \$ \ i \ \$ \ n \neq 0$ 
  let  $?M = \text{mult-row } A \ i \ (\text{ass } (A \ \$ \ i \ \$ \ ?n) \ \text{div } A \ \$ \ i \ \$ \ ?n)$ 
  let  $?P = \text{mult-row } (\text{mat } 1) \ i \ (\text{ass } (A \ \$ \ i \ \$ \ ?n) \ \text{div } A \ \$ \ i \ \$ \ ?n)$ 
  let  $?Ain = A \ \$ \ i \ \$ \ ?n$ 
  have  $\text{ass-dvd}: \text{ass } ?Ain \ \text{dvd } ?Ain$  using  $a$  unfolding  $\text{ass-function-def}$  by (simp add: associatedD1)
  have  $\text{ass-dvd}': ?Ain \ \text{dvd } \text{ass } ?Ain$  using  $a$  unfolding  $\text{ass-function-def}$  by (simp add: associatedD1)
  assume  $iA: \neg \text{is-zero-row } i \ A$ 
  have  $\text{Ain-0}: A \ \$ \ i \ \$ \ ?n \neq 0$  by (metis (mono-tags) LeastI iA is-zero-row-def')
  have  $\text{ass-Ain-0}: \text{ass } (A \ \$ \ i \ \$ \ ?n) \neq 0$  by (metis Ain-0 ass-dvd dvd-0-left-iff)
  have  $\text{inv-P}: \text{invertible } ?P$ 
  proof (rule invertible-mult-row[of - A \$ i \$ ?n div ass (A \$ i \$ ?n)])

```

have $ass\ ?Ain\ div\ ?Ain\ * (\ ?Ain\ div\ ass\ ?Ain) = (ass\ ?Ain\ div\ ?Ain\ * \ ?Ain)$
div ass ?Ain
by (*rule div-mult-swap[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*
finally show $ass\ ?Ain\ div\ ?Ain\ * (\ ?Ain\ div\ ass\ ?Ain) = 1$.
have $\ ?Ain\ div\ ass\ (\ ?Ain) * (ass\ (\ ?Ain)\ div\ ?Ain) = (\ ?Ain\ div\ ass\ (\ ?Ain)) * ass$
 $(\ ?Ain))\ div\ ?Ain$
by (*rule div-mult-swap[OF ass-dvd]*)
also have $... = \ ?Ain\ div\ ?Ain$ **unfolding** *dvd-div-mult-self[OF ass-dvd]* ..
also have $... = 1$ **using** *Ain-0* **by** *simp*
finally show $\ ?Ain\ div\ ass\ (\ ?Ain) * (ass\ (\ ?Ain)\ div\ ?Ain) = 1$.
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** *fast*
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 $\ \exists P.\ invertible\ P \wedge Hermite-reduce-above\ ?M\ (to\ nat\ i)\ i\ ?n\ res =$
 $P\ **\ A$
using *inv-QP* **by** *auto*
qed

lemma *invertible-Hermite-of-upt-row-i:*

assumes $a:$ *ass-function ass*
shows $\ \exists P.\ invertible\ P \wedge Hermite-of-upt-row-i\ A\ k\ ass\ res = P\ **\ A$
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)*
 $= P\ **\ (Hermite-of-upt-row-i\ A\ k\ ass\ res)$ **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 (Suc k) ass res*
 $= Hermite-of-row-i\ ass\ res\ (Hermite-of-upt-row-i\ A\ k\ ass\ res)\ (from\ nat\ k)$

unfolding *Hermite-of-upt-row-i-def* **by** *auto*
also have ... = $P ** (Hermite-of-upt-row-i A k ass res)$ **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 $A :: 'a :: \{bezout-ring-div, normalization-semidom, unique-euclidean-ring\} \rightsquigarrow cols :: \{mod-type\} \rightsquigarrow rows :: \{mod-t$
assumes a : *ass-function ass*
and b : *is-bezout-ext bezout*
shows $\exists P. invertible P \wedge Hermite-of A ass res bezout = P ** A$
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*
= $P ** (echelon-form-of A bezout)$ **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 A ass res bezout*
= *Hermite-of-upt-row-i (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 ?*thesis* **using** *inv-PQ* **by** *blast*
qed

1.6.3 The final theorem

lemma *Hermite*:

assumes a : *ass-function ass*
and r : *res-function res*
and b : *is-bezout-ext bezout*
shows $\exists P. invertible P \wedge (Hermite-of A ass res bezout) = P ** A \wedge$
Hermite (range ass) ($\lambda c. range (res c)$) (Hermite-of A ass res bezout)
using *invertible-Hermite-of[OF a b] Hermite-Hermite-of[OF a r b]* **by** *fast*

1.7 Proving the uniqueness of the Hermite Normal Form

lemma *diagonal-least-nonzero*:

fixes $H :: (('a :: \{bezout-ring-div, normalization-euclidean-semiring, unique-euclidean-ring\},$
 $'b :: mod-type) vec, 'b) vec$
assumes H : *Hermite associates residues H*
and *inv-H: invertible H* **and** *up-H: upper-triangular H*
shows (*LEAST* $n. H \$ i \$ n \neq 0$) = i
proof (*rule Least-equality*)
show $H \$ i \$ i \neq 0$

```

    by (metis (full-types) inv-H invertible-iff-is-unit is-unit-diagonal not-is-unit-0
up-H)
  fix y
  assume Hiy: H $ i $ y ≠ 0
  show i ≤ y
    using up-H unfolding upper-triangular-def
    by (metis (poly-guards-query) Hiy not-less)
qed

```

lemma *diagonal-in-associates:*

```

  fixes H :: ('a :: {bezout-ring-div, normalization-euclidean-semiring, unique-euclidean-ring},
'b :: mod-type) vec, 'b) vec

```

```

  assumes H: Hermite associates residues H
  and inv-H: invertible H and up-H: upper-triangular H
  shows H $ i $ i ∈ associates

```

proof –

```

  have H $ i $ i ≠ 0
    by (metis (full-types) inv-H invertible-iff-is-unit is-unit-diagonal not-is-unit-0
up-H)
  hence ¬ is-zero-row i H unfolding is-zero-row-def is-zero-row-upt-k-def ncols-def
  by auto
  thus ?thesis using H unfolding Hermite-def unfolding diagonal-least-nonzero[OF
H inv-H up-H]
  by auto

```

qed

lemma *above-diagonal-in-residues:*

```

  fixes H :: ('a :: {bezout-ring-div, normalization-euclidean-semiring, unique-euclidean-ring},
'b :: mod-type) vec, 'b) vec

```

```

  assumes H: Hermite associates residues H
  and inv-H: invertible H and up-H: upper-triangular H
  and j-i: j < i
  shows H $ j $ (LEAST n. H $ i $ n ≠ 0) ∈ residues (H $ i $ (LEAST n. H $
i $ n ≠ 0))

```

proof –

```

  have H $ i $ i ≠ 0
    by (metis (full-types) inv-H invertible-iff-is-unit is-unit-diagonal not-is-unit-0
up-H)
  hence ¬ is-zero-row i H unfolding is-zero-row-def is-zero-row-upt-k-def ncols-def
  by auto
  thus ?thesis using H j-i 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, normalization-euclidean-semiring, unique-euclidean-ring} ^ 'n :: mod-type ^ 'n :: mod-

```

```

assumes A-PH:  $A = P ** H$ 
and A-QK:  $A = Q ** K$ 
and inv-A: invertible A
and inv-P: invertible P
and inv-Q: invertible Q
and H: Hermite associates residues H
and K: Hermite associates residues K
shows  $H = K$ 
proof –
  have cs-residues: Complete-set-residues residues using H unfolding Hermite-def
by simp
  have inv-H: invertible H
    by (metis A-PH inv-A inv-P invertible-def invertible-mult matrix-mul-assoc
matrix-mul-lid)
  have inv-K: invertible K
    by (metis A-QK inv-A inv-Q invertible-def invertible-mult matrix-mul-assoc
matrix-mul-lid)
  define U where  $U = (\text{matrix-inv } P)**Q$ 
  have inv-U: invertible U
    by (metis U-def inv-P inv-Q invertible-def invertible-mult matrix-inv-left ma-
trix-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  $\det K *k U = H ** \text{adjugate } K$ 
    unfolding H-UK matrix-mul-assoc[symmetric] mult-adjugate-det matrix-mul-mat
  ..
  have upper-triangular-H: upper-triangular H
    by (metis H Hermite-def echelon-form-imp-upper-triangular)
  have upper-triangular-K: upper-triangular K
    by (metis K Hermite-def echelon-form-imp-upper-triangular)
  have upper-triangular-U: upper-triangular U
    by (metis H-UK inv-K matrix-inv-right matrix-mul-assoc matrix-mul-rid up-
per-triangular-H
      upper-triangular-K upper-triangular-inverse upper-triangular-mult)
  have unit-det-U: is-unit (det U) by (metis inv-U invertible-iff-is-unit)
  have is-unit-diagonal-U:  $(\forall i. \text{is-unit } (U \$ i \$ i))$ 
    by (rule is-unit-diagonal[OF upper-triangular-U unit-det-U])
  have Uii-1:  $(\forall i. (U \$ i \$ i) = 1)$  and Hii-Kii:  $(\forall i. (H \$ i \$ i) = (K \$ i \$ i))$ 
proof (auto)
  fix i
  have Hii:  $H \$ i \$ i \in \text{associates}$ 
    by (rule diagonal-in-associates[OF H inv-H upper-triangular-H])
  have Kii:  $K \$ i \$ i \in \text{associates}$ 
    by (rule diagonal-in-associates[OF K inv-K upper-triangular-K])
  have ass-Hii-Kii:  $\text{normalize } (H \$ i \$ i) = \text{normalize } (K \$ i \$ i)$ 
    by (meson associatedI inv-H inv-K invertible-iff-is-unit is-unit-diagonal
      unit-imp-dvd upper-triangular-H upper-triangular-K)
  show Hii-eq-Kii:  $H \$ i \$ i = K \$ i \$ i$ 
    by (metis Hermite-def Hii K Kii ass-Hii-Kii in-Ass-not-associated)

```

```

have H $ i $ i = U $ i $ i * K $ i $ i
by (metis H-UK upper-triangular-K upper-triangular-U upper-triangular-mult-diagonal)
thus U $ i $ i = 1 unfolding Hii-eq-Kii mult-cancel-right1
  by (metis Hii-eq-Kii inv-H invertible-iff-is-unit
        is-unit-diagonal not-is-unit-0 upper-triangular-H)
qed
have zero-above:  $\forall j s. j \geq 1 \wedge j < \text{ncols } A - \text{to-nat } s \longrightarrow U \$ s \$ (s + \text{from-nat } j) = 0$ 
proof (clarify)
  fix j s assume  $1 \leq j$  and  $j < \text{ncols } A - (\text{to-nat } (s::'n))$ 
  thus  $U \$ s \$ (s + \text{from-nat } j) = 0$ 
  proof (induct j rule: less-induct)
    fix p
    assume induct-step:  $(\bigwedge y. y < p \implies 1 \leq y \implies y < \text{ncols } A - \text{to-nat } s \implies U \$ s \$ (s + \text{from-nat } y) = 0)$ 
    and p1:  $1 \leq p$  and p2:  $p < \text{ncols } A - \text{to-nat } s$ 
    have s-less:  $s < s + \text{from-nat } p$  using p1 p2 unfolding ncols-def
    by (metis One-nat-def add commute add-diff-cancel-right' add-lessD1 add-to-nat-def

        from-nat-to-nat-id less-diff-conv neq-iff not-le
        to-nat-from-nat-id to-nat-le zero-less-Suc)
    show  $U \$ s \$ (s + \text{from-nat } p) = 0$ 
    proof -
      have UNIV-rw:  $UNIV = \text{insert } s (UNIV - \{s\})$  by auto
      have UNIV-s-rw:  $UNIV - \{s\} = \text{insert } (s + \text{from-nat } p) ((UNIV - \{s\}) - \{s + \text{from-nat } p\})$ 
      using p1 p2 s-less unfolding ncols-def by (auto simp: algebra-simps)
      have sum-rw:  $(\sum k \in UNIV - \{s\}. U \$ s \$ k * K \$ k \$ (s + \text{from-nat } p))$ 
        =  $U \$ s \$ (s + \text{from-nat } p) * K \$ (s + \text{from-nat } p) \$ (s + \text{from-nat } p)$ 
        +  $(\sum k \in (UNIV - \{s\}) - \{s + \text{from-nat } p\}. U \$ s \$ k * K \$ k \$ (s + \text{from-nat } p))$ 
      using UNIV-s-rw sum.insert by (metis (erased, lifting) Diff-iff finite singletonI)
      have sum-0:  $(\sum k \in (UNIV - \{s\}) - \{s + \text{from-nat } p\}. U \$ s \$ k * K \$ k \$ (s + \text{from-nat } p)) = 0$ 
      proof (rule sum.neutral, rule)
        fix x assume  $x \in UNIV - \{s\} - \{s + \text{from-nat } p\}$ 
        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
          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
                define a where  $a = \text{to-nat } x - \text{to-nat } s$ 

```

```

from  $x-g-s$  have  $to-nat\ s < to-nat\ x$  by (rule  $to-nat-mono$ )
hence  $xa: x=s+(from-nat\ a)$  unfolding  $a-def$   $add-to-nat-def$ 
by (simp  $add: less-imp-diff-less$   $to-nat-less-card$   $algebra-simps$ 
 $to-nat-from-nat-id$ )
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  $1 \leq a$ 
by (auto simp  $add: a-def\ p1\ p2$ ) (metis  $Suc-leI\ to-nat-mono\ x-g-s$ 
 $zero-less-diff$ )
show  $a < ncols\ A - to-nat\ s$  using  $a-p\ p2$  by auto
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
qed
qed
have  $H\ \$\ s\ \$\ (s + from-nat\ p) = (\sum_{k \in UNIV} U\ \$\ s\ \$\ k * K\ \$\ k\ \$\ (s +$ 
 $from-nat\ p))$ 
unfolding  $H-UK\ matrix-matrix-mult-def$  by auto
also have  $... = (\sum_{k \in insert\ s\ (UNIV - \{s\})} U\ \$\ s\ \$\ k * K\ \$\ k\ \$\ (s +$ 
 $from-nat\ p))$ 
using  $UNIV-rw$  by simp
also have  $... = U\ \$\ s\ \$\ s * K\ \$\ s\ \$\ (s + from-nat\ p)$ 
 $+ (\sum_{k \in UNIV - \{s\}} U\ \$\ s\ \$\ k * K\ \$\ k\ \$\ (s + from-nat\ p))$ 
by (rule  $sum.insert, simp-all$ )
also have  $... = U\ \$\ s\ \$\ s * K\ \$\ s\ \$\ (s + from-nat\ p)$ 
 $+ U\ \$\ s\ \$\ (s + from-nat\ p) * K\ \$\ (s + from-nat\ p)\ \$\ (s + from-nat\ p)$ 
unfolding  $sum-rw\ sum-0$  by simp
finally have  $H-s-sp: H\ \$\ s\ \$\ (s + from-nat\ p)$ 
 $= U\ \$\ s\ \$\ (s + from-nat\ p) * K\ \$\ (s + from-nat\ p)\ \$\ (s + from-nat\ p) +$ 
 $K\ \$\ s\ \$\ (s + from-nat\ p)$ 
using  $Uii-1$  by auto
hence  $cong-HK: cong\ (H\ \$\ s\ \$\ (s + from-nat\ p))\ (K\ \$\ s\ \$\ (s + from-nat$ 

```

$p)) (K \text{ \$ } (s+\text{from-nat } p) \text{ \$ } (s + \text{from-nat } p))$
unfolding *cong-def* **by** *auto*
have *H-s-sp-residues*: $(H \text{ \$ } s \text{ \$ } (s + \text{from-nat } p)) \in \text{residues } (K \text{ \$ } (s+\text{from-nat } p) \text{ \$ } (s + \text{from-nat } p))$
using *above-diagonal-in-residues*[*OF H inv-H upper-triangular-H s-less*]
unfolding *diagonal-least-nonzero*[*OF H inv-H upper-triangular-H*]
by (*metis Hii-Kii*)
have *K-s-sp-residues*: $(K \text{ \$ } s \text{ \$ } (s + \text{from-nat } p)) \in \text{residues } (K \text{ \$ } (s+\text{from-nat } p) \text{ \$ } (s + \text{from-nat } p))$
using *above-diagonal-in-residues*[*OF K inv-K upper-triangular-K s-less*]
unfolding *diagonal-least-nonzero*[*OF K inv-K upper-triangular-K*] .
have *Hs-sp-Ks-sp*: $(H \text{ \$ } s \text{ \$ } (s + \text{from-nat } p)) = (K \text{ \$ } s \text{ \$ } (s + \text{from-nat } p))$

using *cong-HK in-Res-not-congruent*[*OF cs-residues H-s-sp-residues K-s-sp-residues*]
by *fast*
have *is-unit* $(K \text{ \$ } (s + \text{from-nat } p) \text{ \$ } (s + \text{from-nat } p))$
by (*metis Hii-Kii inv-H invertible-iff-is-unit is-unit-diagonal upper-triangular-H*)
hence $K \text{ \$ } (s + \text{from-nat } p) \text{ \$ } (s + \text{from-nat } p) \neq 0$ **by** (*metis not-is-unit-0*)
thus *?thesis* **unfolding** *from-nat-1* **using** *H-s-sp* **unfolding** *Hs-sp-Ks-sp*
by *auto*
qed
qed
qed
have $U = \text{mat } 1$
proof (*unfold mat-def vec-eq-iff, auto*)
fix *ia* **show** $U \text{ \$ } ia \text{ \$ } ia = 1$ **using** *Uii-1* **by** *simp*
fix *i* **assume** *i-ia*: $i \neq ia$
show $U \text{ \$ } i \text{ \$ } ia = 0$
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*
define *a* **where** $a = \text{to-nat } ia - \text{to-nat } i$
have *ia-eq*: $ia = i + \text{from-nat } a$ **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 $1 \leq a$ **unfolding** *a-def*
by (*metis diff-is-0-eq i-less-ia less-one not-less to-nat-mono*)
moreover **have** $a < \text{ncols } A - \text{to-nat } i$
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::unique-euclidean-ring iarray iarray =>
  nat => nat => nat => ('a=>'a=>'a) => 'a iarray iarray
where Hermite-reduce-above-iarrays A 0 i j res = A
  | Hermite-reduce-above-iarrays A (Suc n) i j res = (let i'=n;
    Aij = A !! i !! j;
    Ai'j = A !! i' !! j
  in
    Hermite-reduce-above-iarrays (row-add-iarray A i' i (((res Aij (Ai'j)) - (Ai'j))
      div Aij)) n i j res)

```

```

definition Hermite-of-row-i-iarray ass res A i = (
  if is-zero-iarray (A !! i)
  then A
  else
    let j = least-non-zero-position-of-vector (A !! i); Aij= (A !! i !! j);
    A' = mult-row-iarray A i ((ass Aij) div Aij)
    in Hermite-reduce-above-iarrays A' i i j res)

```

```

definition Hermite-of-upt-row-i-iarrays A i ass res = foldl (Hermite-of-row-i-iarray
  ass res) A [0..<i]

```

definition *Hermite-of-iarrays* A *ass res bezout* =
 (let A' = *echelon-form-of-iarrays* A *bezout*
 in *Hermite-of-upt-row-i-iarrays* A' (*nrows-iarray* A) *ass res*)

2.2 Proving the equivalence between definitions of both representations

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

lemma *matrix-to-iarray-Hermite-of-row-i*[*code-unfold*]:
fixes $A::'a::\{\text{unique-euclidean-ring}\}^{\wedge}\text{cols}::\{\text{mod-type}\}^{\wedge}\text{rows}::\{\text{mod-type}\}$
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)
case *True* **thus** ?*thesis*
unfolding *Hermite-of-row-i-def* *Hermite-of-row-i-iarray-def* *Let-def* *zero-rw*
by *auto*
next
case *False*
have $Ain: A \$ i \$ (\text{LEAST } n. A \$ i \$ n \neq 0) \neq 0$
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
  show ?thesis using vec-to-iarray-least-non-zero-position-of-vector-from-index[OF
v-rw]
  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:
  fixes A::'a::{unique-euclidean-ring} ^'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.premis 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-upt-row-i A i ass res)) (to-nat (from-nat i::'rows)))
  unfolding matrix-to-iarray-Hermite-of-row-i ..
  also have ... = (Hermite-of-row-i-iarray ass res (matrix-to-iarray (Hermite-of-upt-row-i

```

```

A i ass res)) i)
  using to-nat-from-nat-id[OF i[unfolding nrows-def]] by simp
  also have ... = (Hermite-of-row-i-iarray ass res
    (Hermite-of-upt-row-i-iarrays (matrix-to-iarray A) i ass res) i)
  using Suc.hyps i unfolding nrows-def by simp
  also have ... = Hermite-of-upt-row-i-iarrays (matrix-to-iarray A) (Suc i) ass
res
  unfolding Hermite-of-upt-row-i-iarrays-def by simp
  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[code] let A = IArray[IArray[37,8,6::int],IArray[5,4,-8],IArray[3,24,-7]]
  in (Hermite-of-iarrays 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-iarray (Hermite-of A ass-function-euclidean res-function-euclidean
euclid-ext2)

```

```

end

```