Diagonal-form

By Jose Divasón

October 30, 2012

Contents

theory *Diagonal-form*
imports *Main*
~~/*src*/*HOL*/*Matrix*/*Matrix*
~~/*src*/*HOL*/*Divides*
~~/*src*/*HOL*−*Algebra*/*Ring*
~~/*src*/*HOL*−*Algebra*/*IntRing*
begin

This function interchanges the columns n and m of an infinite matrix

definition *interchange-columns-infmatrix* :: *'a infmatrix* => *nat* ⇒ *nat* ⇒ *'a infmatrix* **where**
interchange-columns-infmatrix A n m == (λ*i j*. *if j*=*n then A i m else if j* = *m then A i n else A i j*)

This function interchanges the columns n and m of a finite matrix

definition *interchange-columns-matrix* :: *int matrix* => *nat* ⇒ *nat* ⇒ *int matrix*
where *interchange-columns-matrix A n m* == *Abs-matrix* (*interchange-columns-infmatrix* (*Rep-matrix A*) *n m*)

lemma *Rep-Abs-aux-interchange'*:
shows *Rep-matrix* (*Abs-matrix* (λ*i j*. *if j* = *n then Rep-matrix A i m else if j* = *m then Rep-matrix A i n else Rep-matrix A i j*)) =
(λ*i j*. *if j* = *n then Rep-matrix A i m else if j* = *m then Rep-matrix A i n else Rep-matrix A i j*)
⟨*proof*⟩

lemma *Rep-Abs-aux-interchange2'*:
shows *Rep-matrix* (*Abs-matrix* (λ*i j*. *if i* = *n then Rep-matrix A m j else if i* = *m then Rep-matrix A n j else Rep-matrix A i j*)) =
(λ*i j*. *if i* = *n then Rep-matrix A m j else if i* = *m then Rep-matrix A n j else Rep-matrix A i j*)
⟨*proof*⟩

If we interchange two columns twice, we will obtain the original matrix.

lemma *interchange-columns-matrix-id*:
shows *interchange-columns-matrix* (*interchange-columns-matrix A n m*) *n m* = *A*
⟨*proof*⟩

If we interchange two columns a and b, the elements situated in column a now there will be in column b

lemma *interchange-columns*:
shows ∀ *i*. *Rep-matrix*(*interchange-columns-matrix A a b*) *i a* = *Rep-matrix A i b*
⟨*proof*⟩

lemma *interchange-columns'*:
shows ∀ *i*. *Rep-matrix*(*interchange-columns-matrix A a b*) *i b* = *Rep-matrix A i a*
⟨*proof*⟩

lemma *interchange-columns-eq*:
shows ∀ *i s*. (*s*≠*n* ∧ *s*≠*m*) ⟶ *Rep-matrix* (*interchange-columns-matrix* (*A*::*int matrix*) *n m*) *i s* = (*Rep-matrix A*) *i s*
⟨*proof*⟩

This function interchanges two rows of an infinite matrix

definition *interchange-rows-infmatrix* :: *'a infmatrix* => *nat* ⇒ *nat* ⇒ *'a infmatrix*
where *interchange-rows-infmatrix A n m* == (λ*i j*. *if i*=*n then A m j else if i* = *m then A n j else A i j*)

This function interchanges two rows of a finite matrix

definition *interchange-rows-matrix* :: *int matrix* => *nat* ⇒ *nat* ⇒ *int matrix*
where *interchange-rows-matrix A n m*== *Abs-matrix* (*interchange-rows-infmatrix* (*Rep-matrix A*) *n m*)

lemma *interchange-rows-matrix-id*:
shows *interchange-rows-matrix* (*interchange-rows-matrix A n m*) *n m* = *A*
⟨*proof*⟩

lemma *interchange-rows*:
shows ∀ *j*. *Rep-matrix*(*interchange-rows-matrix A a b*) *a j* = *Rep-matrix A b j*
⟨*proof*⟩

lemma *interchange-rows'*:
shows ∀ *j*. *Rep-matrix*(*interchange-rows-matrix A a b*) *b j* = *Rep-matrix A a j*
⟨*proof*⟩

lemma *interchange-rows-matrix-eq*:

shows $\forall i\ s.\ (s \neq n \wedge s \neq m) \longrightarrow$ *Rep-matrix* (*interchange-rows-matrix A n m*) *s i* = (*Rep-matrix A*) *s i*
⟨*proof*⟩

lemma *interchange-columns-interchange-rows*:
shows *Rep-matrix* (*interchange-columns-matrix* (*interchange-rows-matrix A a b*) *n m*) *a n* = (*Rep-matrix A*) *b m*
⟨*proof*⟩

For a given matrix, this function moves the element (i,j) to the position (k, l)

definition *move-element* :: *int matrix* => *nat* ⇒ *nat* ⇒ *nat* ⇒ *nat* ⇒ *int matrix*
where *move-element A i j k l* = (*interchange-columns-matrix* (*interchange-rows-matrix A i k*) *j l*)

lemma *move-element*:
shows *Rep-matrix* (*move-element A i j k l*) *k l* = (*Rep-matrix A*) *i j*
⟨*proof*⟩

This function adds to row i the row j multiplied by q

definition *row-add-infmatrix* :: (*'a*::{*plus*,*times*}) *infmatrix* => *nat* ⇒ *nat* ⇒ *'a* ⇒ *'a infmatrix*
where *row-add-infmatrix A i j q* = (λ*a b. if a*=*i then* ((*A i b*) + *q*∗(*A j b*)) *else A a b*)

definition *row-add-matrix* :: *int matrix* ⇒ *nat* ⇒ *nat* ⇒ *int* ⇒ *int matrix*
where *row-add-matrix A i j q* = *Abs-matrix* (*row-add-infmatrix* (*Rep-matrix A*) *i j q*)

lemma *Rep-aux-row-add*:
shows *Rep-matrix* (*Abs-matrix* (λ*a b. if a* = *i then Rep-matrix* (*A*::(*'a*::*ring*) *matrix*) *i b* + *q* ∗ *Rep-matrix A j b else Rep-matrix A a b*))
=(λ*a b. if a* = *i then Rep-matrix A i b* + *q* ∗ *Rep-matrix A j b else Rep-matrix A a b*)
⟨*proof*⟩

lemma *row-add-matrix-eq*:
shows ∀ *s. Rep-matrix* (*row-add-matrix A i j q*) *i s* = (*Rep-matrix A*) *i s* + *q*∗(*Rep-matrix A*) *j s*
⟨*proof*⟩

lemma *Rep-matrix-row-add-matrix-preserve*:
shows ∀ *s t. s*≠*i* ⟶ *Rep-matrix* (*row-add-matrix A i j q*) *s t* = (*Rep-matrix A*) *s t*
⟨*proof*⟩

This function adds to column i the column j multiplied by q

definition *column-add-infmatrix* :: ($'a$::{*plus*,*times*}) *infmatrix* => *nat* ⇒ *nat* ⇒ $'a$ ⇒ $'a$ *infmatrix*
where *column-add-infmatrix A i j q* = (λ*a b*. *if b*=*i then* ((*A a i*) + *q*∗(*A a j*)) *else A a b*)

definition *column-add-matrix* :: *int matrix* ⇒ *nat* ⇒ *nat* ⇒ *int* ⇒ *int matrix*
where *column-add-matrix A i j q* = *Abs-matrix* (*column-add-infmatrix* (*Rep-matrix A*) *i j q*)

lemma *Rep-aux-column-add*:
shows *Rep-matrix* (*Abs-matrix* (λ*a b*. *if b* = *i then Rep-matrix* (*A*::($'a$::*ring*) *matrix*) *a i* + *q* ∗ *Rep-matrix A a j else Rep-matrix A a b*))
=(λ*a b*. *if b* = *i then Rep-matrix A a i* + *q* ∗ *Rep-matrix A a j else Rep-matrix A a b*)
⟨*proof*⟩

lemma *column-add-matrix-eq*:
shows ∀ *s*. *Rep-matrix* (*column-add-matrix A i j q*) *s i* = (*Rep-matrix A*) *s i* + *q*∗(*Rep-matrix A*) *s j*
⟨*proof*⟩

lemma *Rep-matrix-column-add-matrix-preserve*:
shows ∀ *s t*. *t*≠*i* ⟶ *Rep-matrix* (*column-add-matrix A i j q*) *s t* = (*Rep-matrix A*) *s t*
⟨*proof*⟩

Function that multiply a column by a number

definition *column-multiply-infmatrix* :: ($'a$::{*plus*,*times*}) *infmatrix* => *nat* ⇒ $'a$ ⇒ $'a$ *infmatrix*
where *column-multiply-infmatrix A i q* = (λ*a b*. *if b*=*i then q*∗(*A a i*) *else A a b*)

definition *column-multiply-matrix* :: *int matrix* ⇒ *nat* ⇒ *int* ⇒ *int matrix*
where *column-multiply-matrix A i q* = *Abs-matrix* (*column-multiply-infmatrix* (*Rep-matrix A*) *i q*)

lemma *Rep-abs-aux-column-multiply*:
shows *Rep-matrix* (*Abs-matrix* (λ*a b*. *if b* = *i then q* ∗ *Rep-matrix* (*A*::($'a$::*ring*) *matrix*) *a i else Rep-matrix A a b*))
=(λ*a b*. *if b* = *i then q* ∗ *Rep-matrix A a i else Rep-matrix A a b*)
⟨*proof*⟩

lemma *column-multiply-matrix-eq*:
shows ∀ *s*. *Rep-matrix* (*column-multiply-matrix A i q*) *s i* = *q*∗((*Rep-matrix A*) *s i*)
⟨*proof*⟩

lemma *column-multiply-matrix-eq2*:
shows $\forall s\ t.\ t \neq i \longrightarrow$ *Rep-matrix* (*column-multiply-matrix A i q*) *s t* = (*Rep-matrix A*) *s t*
find-theorems *Rep-matrix ?A*
⟨*proof*⟩

This function multiplies a row i by a number q.

definition *row-multiply-infmatrix* :: ('*a*::{*plus*,*times*}) *infmatrix* => *nat* ⇒ '*a* ⇒ '*a infmatrix*
where *row-multiply-infmatrix A i q* = (λ*a b*. *if a*=*i then q*∗(*A i b*) *else A a b*)

definition *row-multiply-matrix* :: *int matrix* ⇒ *nat* ⇒ *int* ⇒ *int matrix*
where *row-multiply-matrix A i q* = *Abs-matrix* (*row-multiply-infmatrix* (*Rep-matrix A*) *i q*)

lemma *Rep-abs-aux-row-multiply*:
shows *Rep-matrix* (*Abs-matrix* (λ*a b*. *if a* = *i then q* ∗ *Rep-matrix* (*A*::('*a*::*ring*) *matrix*) *i b else Rep-matrix A a b*))
=(λ*a b*. *if a* = *i then q* ∗ *Rep-matrix A i b else Rep-matrix A a b*)
⟨*proof*⟩

lemma *row-multiply-matrix-eq*:
shows ∀ *s*. *Rep-matrix* (*row-multiply-matrix A i q*) *i s* = *q*∗((*Rep-matrix A*) *i s*)
⟨*proof*⟩

lemma *row-multiply-matrix-eq2*:
shows $\forall s\ t.\ s \neq i \longrightarrow$ *Rep-matrix* (*row-multiply-matrix A i q*) *s t* = (*Rep-matrix A*) *s t*
⟨*proof*⟩

definition *partRowReduce-infmatrix* :: ('*a*::*ring-div*) *infmatrix* ⇒ *nat* ⇒ *nat* ⇒ '*a infmatrix*
where *partRowReduce-infmatrix A k l* = (λ*i j*. *if i*>*k then* (*row-add-infmatrix A i k* (−((*A i l*) *div* (*A k l*)))) *i j else A i j*)

definition *partRowReduce-matrix* :: *int matrix* ⇒ *nat* ⇒ *nat* ⇒ *int matrix*
where *partRowReduce-matrix A k l* = *Abs-matrix* (*partRowReduce-infmatrix* (*Rep-matrix A*) *k l*)

lemma *aux-partRowReduce*:
shows *Rep-matrix* (*Abs-matrix* (λ*i j*. *if k* < *i then row-add-infmatrix*
(*Rep-matrix* (*A*::('*a*::*ring-div*) *matrix*)) *i k* (− (*Rep-matrix A i l div Rep-matrix A k l*)) *i j else Rep-matrix A i j*))

$= (\lambda i\ j.\ \textit{if}\ k < i\ \textit{then row-add-infmatrix}\ (\textit{Rep-matrix}\ A)\ i\ k\ (-\ (\textit{Rep-matrix}\ A\ i\ l\ \textit{div Rep-matrix}\ A\ k\ l))\ i\ j\ \textit{else Rep-matrix}\ A\ i\ j)$
⟨*proof*⟩

lemma *partRowReduce′*:
shows $\forall\, s\ t.\ s \leq k \longrightarrow \textit{Rep-matrix}\ (\textit{partRowReduce-matrix}\ A\ k\ l)\ s\ t = (\textit{Rep-matrix}\ A)\ s\ t$
⟨*proof*⟩

lemma *partRowReduce*:
shows $\forall\, i\ j.\ i > k \longrightarrow \textit{Rep-matrix}\ (\textit{partRowReduce-matrix}\ A\ k\ l)\ i\ j = \textit{row-add-infmatrix}\ (\textit{Rep-matrix}\ A)\ i\ k\ (-\ (\textit{Rep-matrix}\ A\ i\ l\ \textit{div Rep-matrix}\ A\ k\ l))\ i\ j$
⟨*proof*⟩

lemma *mod-abs*:
assumes *b-not-zero*: $b \neq 0$
shows $\textit{abs}\ ((a::\textit{int})\ \textit{mod}\ b) < \textit{abs}\ (b)$
⟨*proof*⟩

lemma *partRowReduce-abs*:
assumes *i-g-k*: $i > k$
and *Akk-not-zero*: $\textit{Rep-matrix}\ (A::\textit{int matrix})\ k\ k \neq 0$
shows $\textit{abs}\ (\textit{Rep-matrix}\ (\textit{partRowReduce-matrix}\ A\ k\ k)\ i\ k) < \textit{abs}\ ((\textit{Rep-matrix}\ A)\ k\ k)$
⟨*proof*⟩

definition *partColumnReduce-infmatrix* :: $('a::\textit{ring-div})\ \textit{infmatrix} \Rightarrow \textit{nat} \Rightarrow \textit{nat} \Rightarrow 'a\ \textit{infmatrix}$
where $\textit{partColumnReduce-infmatrix}\ A\ k\ l = (\lambda i\ j.\ \textit{if}\ j > l\ \textit{then}\ (\textit{column-add-infmatrix}\ A\ j\ l\ (-((A\ k\ j)\ \textit{div}\ (A\ k\ l))))\ i\ j\ \textit{else}\ A\ i\ j)$

definition *partColumnReduce-matrix* :: $\textit{int matrix} \Rightarrow \textit{nat} \Rightarrow \textit{nat} \Rightarrow \textit{int matrix}$
where $\textit{partColumnReduce-matrix}\ A\ k\ l = \textit{Abs-matrix}\ (\textit{partColumnReduce-infmatrix}\ (\textit{Rep-matrix}\ A)\ k\ l)$

lemma *aux-partColumnReduce*:
shows $\textit{Rep-matrix}\ (\textit{Abs-matrix}\ (\lambda i\ j.\ \textit{if}\ l < j\ \textit{then column-add-infmatrix}\ (\textit{Rep-matrix}\ A)\ j\ l\ (-\ (\textit{Rep-matrix}\ A\ k\ j\ \textit{div Rep-matrix}\ A\ k\ l))\ i\ j\ \textit{else Rep-matrix}\ A\ i\ j))$
$=(\lambda i\ j.\ \textit{if}\ l < j\ \textit{then column-add-infmatrix}\ (\textit{Rep-matrix}\ (A::('a::\textit{ring-div})\ \textit{matrix}))\ j\ l\ (-\ (\textit{Rep-matrix}\ A\ k\ j\ \textit{div Rep-matrix}\ A\ k\ l))\ i\ j\ \textit{else Rep-matrix}\ A\ i\ j)$
⟨*proof*⟩

lemma *partColumnReduce′*:
shows $\forall\, s\ t.\ t \leq l \longrightarrow \textit{Rep-matrix}\ (\textit{partColumnReduce-matrix}\ A\ k\ l)\ s\ t = (\textit{Rep-matrix}\ A)\ s\ t$

⟨*proof*⟩

lemma *partColumnReduce*:
shows $\forall\, i\, j.\ j>l \longrightarrow$ *Rep-matrix* (*partColumnReduce-matrix A k l*) *i j* = (*column-add-infmatrix* (*Rep-matrix A*) *j l* (−((*Rep-matrix A k j*) *div* (*Rep-matrix A k l*)))) *i j*
⟨*proof*⟩

lemma *partColumnReduce-abs*:
assumes *j-g-k*: $j>k$
and *Akk-not-zero*: *Rep-matrix* (*A*::*int matrix*) *k k* $\neq$ *0*
shows *abs* (*Rep-matrix* (*partColumnReduce-matrix A k k*) *k j*) < *abs* ((*Rep-matrix A*) *k k*)
⟨*proof*⟩

definition *fst-bis*:: ('*a* × '*b* × '*c*)⇒'*a*
where *fst-bis A* = *fst A*

lemma *fst-bis*[*simp*]:*fst-bis*(*a*,*b*,*c*)=*a* ⟨*proof*⟩

definition *snd-bis* :: ('*a* × '*b* × '*c*)⇒'*b*
where *snd-bis A* = *fst* (*snd A*)

lemma *snd-bis*[*simp*]:*snd-bis*(*a*,*b*,*c*)=*b* ⟨*proof*⟩

definition *trd-bis* :: ('*a* × '*b* × '*c*)⇒'*c*
where *trd-bis A* = *snd* (*snd A*)

lemma *trd-bis*[*simp*]:*trd-bis* (*a*,*b*,*c*)=*c* ⟨*proof*⟩

instantiation *prod* :: (*ord*, *ord*) *ord*
begin

definition *less-prod* :: '*a* × '*b* ⇒ '*a* × '*b* ⇒ *bool*
where *less-prod P Q* = (*fst P* < *fst Q* ∨ ((*fst P* = *fst Q*) ∧ (*snd P* < *snd Q*)))

definition *less-eq-prod* :: '*a* × '*b* ⇒ '*a* × '*b* ⇒ *bool*
where *less-eq-prod P Q* == ((*less P Q*) ∨ ((*fst P* = *fst Q*) ∧ (*snd P* = *snd Q*)))

instance
⟨*proof*⟩

end

For a given matrix A and a natural number k, this function returns us (False, 0, 0) if the submatrix from position A k k (included) are null. In other case, this function returns (True, pos1, pos2), where (pos1, pos2) is the position of the element which has the least notzero absolute value

definition *minNonzero-nc* :: (*int matrix*) ⇒ *nat* ⇒ (*bool* × *nat* × *nat*)
where *minNonzero-nc A k* = (*if* (∀ *i j*. *i*∈{*k*..<*nrows A*} ∧ *j*∈{*k*..<*ncols A*} ⟶ *Rep-matrix A i j* = *0*) *then* (*False*,*0*::*nat*,*0*::*nat*)
else (*True*,*LEAST pos*. (*fst pos*)≥*k* ∧ (*snd pos*)≥*k* ∧ *Rep-matrix A* (*fst pos*) (*snd pos*) ≠ *0*
∧ (∀ *m n*. *m*≥*k* ∧ *n*≥*k* ∧ *Rep-matrix A m n* ≠ *0* ⟶ (*abs*(*Rep-matrix A m n*)) ≥ (*abs*(*Rep-matrix A* (*fst pos*) (*snd pos*))))))

If the matrix has non zero elements from position A k k, then the boolean that the function minNonzero_nc A k returns will be True.

lemma *minNonzero-true*:
assumes ¬ (∀ (*i*::*nat*) *j*::*nat*. *i* ∈ {*k*..<*nrows A*} ∧ *j* ∈ {*k*..<*ncols A*} ⟶ *Rep-matrix A i j* = *0*)
shows *fst-bis* (*minNonzero-nc A k*) = *True*
⟨*proof*⟩

Analogously, if function (minNonzero_nc A k) returns True, then not every element are zero in the submatrix from element A k k.

lemma *minNonzero-true-inv*:
assumes *fst-bis* (*minNonzero-nc A k*) = *True*
shows ¬ (∀ (*i*::*nat*) *j*::*nat*. *i* ∈ {*k*..<*nrows A*} ∧ *j* ∈ {*k*..<*ncols A*} ⟶ *Rep-matrix A i j* = *0*)
⟨*proof*⟩

This function moves the element of the submatrix (k:nrows,k:ncols) with the least nonzero absolute value to the position (k,k)

definition *move-minNonzero-nc* :: (*int matrix*) ⇒ *nat* ⇒ (*int matrix*)
where *move-minNonzero-nc A k* = (*move-element A* (*snd-bis* (*minNonzero-nc A k*)) (*trd-bis* (*minNonzero-nc A k*)) *k k*)

partial-function (*tailrec*) *iterate-partRowReduce* :: (*int matrix*) ⇒ *nat* ⇒ (*int matrix*)
where *iterate-partRowReduce A k* = (*if* (∀ *m*. *m*>*k* ⟶ *Rep-matrix A m k* = *0*) *then A else*
iterate-partRowReduce (*partRowReduce-matrix* (*move-minNonzero-nc A k*) *k k*) *k*)

declare *iterate-partRowReduce.simps* [*simp del*]

lemma *partRowReduce-abs2*:
assumes *Akk-not-zero*: *Rep-matrix* (*A*::*int matrix*) *k k* ≠ *0*
and *eq*: *abs*(*Rep-matrix A k k*) ≤ *abs*(*Rep-matrix A i k*)
and *i*:*i*>*k*
shows *abs* (*Rep-matrix* (*partRowReduce-matrix A k k*) *i k*) < *abs* (*Rep-matrix A i k*)
⟨*proof*⟩

instantiation *prod* :: (*order*, *order*) *order*
begin

instance
⟨*proof*⟩

end

instantiation *prod* :: (*linorder*, *linorder*) *linorder*
begin

instance
⟨*proof*⟩

end

In the file Wellfounded.thy is has been proved that well orders with the lexicographic order are themselves well orders, so we just have to rewrite our definitions of $op <$ and $op \leq$ adequetely and then use the result $[\![wf\ ?ra;\ wf\ ?rb]\!] \Longrightarrow wf\ (?ra <*lex*> ?rb)$.

instantiation *prod* :: (*wellorder*, *wellorder*) *wellorder*
begin
find-theorems *less-nat-induct*
instance
⟨*proof*⟩

end

definition
preImage :: ('*a* => '*b*) => '*b set* => '*a set* **where**
preImage f A = {*x* . *f x* ∈ *A*}

definition
pre :: ('*a* => '*b*) => '*b* => '*a set* **where**
pre f a = {*x* . *f x* = *a*}

definition
equalOn :: ['*a set*,'*a* => '*b*,'*a* => '*b*] => *bool* **where**
equalOn A f g = (!*x*:*A*. *f x* = *g x*)

lemma *preImage-insert*: *preImage f* (*insert a A*) = *pre f a Un preImage f A*
⟨*proof*⟩

lemma *preImageI*: *f x* : *A* ==> *x* : *preImage f A*
⟨*proof*⟩

lemma *preImageE*: x : *preImage* f A ==> f x : A
⟨*proof*⟩

lemma *equalOn-Un*: *equalOn* $(A \cup B)$ f g = (*equalOn* A f g ∧ *equalOn* B f g)
⟨*proof*⟩

lemma *equalOnD*: *equalOn* A f g ==> $(\forall x \in A . f x = g x)$
⟨*proof*⟩

lemma *equalOnI*:$(\forall x \in A . f x = g x)$ ==> *equalOn* A f g
⟨*proof*⟩

lemma *equalOn-UnD*: *equalOn* $(A \cup B)$ f g ==> *equalOn* A f g ∧ *equalOn* B f g
⟨*proof*⟩
lemma *inj-inv-singleton*[*simp*]: [| *inj* f; f z = y |] ==> $\{x. f x = y\} = \{z\}$
⟨*proof*⟩

lemma *finite-pre*[*simp*]: *inj* f ==> *finite* (*pre* f x)
⟨*proof*⟩

lemma *finite-preImage*[*simp*]: [| *finite* A; *inj* f |] ==> *finite* (*preImage* f A)
⟨*proof*⟩

lemma *pre*:
assumes l: $\exists x{\in}A. \forall y{\in}A. P x y$
shows $\{x{\in}A. \forall y{\in}A. P x y\} \neq \{\}$
⟨*proof*⟩

definition *Rep-matrix'* :: ('*a*::*zero matrix*)⇒(*nat*×*nat*)⇒'*a*
where *Rep-matrix'* A x = *Rep-matrix* A (*fst* x) (*snd* x)

Next function returns us the absolute values of the nonzero elements of a matrix A

definition *abs-set* :: (*int matrix*) ⇒ *nat set*
where *abs-set* A = (*nat*∘*abs*∘(*Rep-matrix'* A))‘ (*nonzero-positions* (*Rep-matrix* A))

If there exist nonzero elements in A, then the set returned by function (abs_set A) won't be empty.

lemma
assumes *nonzero-positions* (*Rep-matrix* A)≠{}
shows *abs-set* A≠{}
⟨*proof*⟩

If a element is belongs to (abs_set A), then it is not a zero.

lemma *abs-notzero*:

assumes x: $x \in$ *abs-set* A
shows $x \neq 0$
⟨*proof*⟩

lemma
assumes f:*finite* A
and ne:$(A::('a::\{wellorder\}\ set)) \neq \{\}$
shows *Least* $A \in A$
⟨*proof*⟩
thm *mem-def*
⟨*proof*⟩

lemma *nat-times-nat-least*:
assumes ne:$(A::(nat \times nat)\ set) \neq \{\}$
shows *Least* $A \in A$
⟨*proof*⟩
thm *mem-def*
⟨*proof*⟩

If the set returned by (abs_set A) is not empty, then the LEAST element belongs to this set. It is indispensable that the set has a well-order.

lemma *Least-in*:
assumes *abs-set* $A \neq \{\}$
shows *Least* (*abs-set* A) $\in$ (*abs-set* A)
⟨*proof*⟩

Then, each element belonging to (abs_set A) is greater than or equal than the LEAST.

lemma *Least-le-abs-set*:
assumes y:$y \in$ *abs-set* A — Con esto ya se que no es vacio
shows *Least* (*abs-set* A) $\leq y$
⟨*proof*⟩

Now we want to obtain the position of the LEAST element of (abs_set A). We use the definition of preimage. We have to take into account that the LEAST could be in several positions, that is, we can have several preimages (we obtain a set of positions).

lemma *pre-abs-set*:
assumes *abs-set* $A \neq \{\}$
shows *pre* (*nat*$\circ$*abs* $\circ$ *Rep-matrix'* A) (*Least* (*abs-set* A)) $\neq \{\}$
⟨*proof*⟩

lemma *Least-in-pre*:
assumes *abs-set* $A \neq \{\}$
shows *Least* (*pre* (*nat*$\circ$*abs* $\circ$ *Rep-matrix'* A) (*Least* (*abs-set* A))) $\in$ *pre* (*nat*$\circ$*abs* $\circ$ *Rep-matrix'* A) (*Least* (*abs-set* A))
⟨*proof*⟩

In positions of preimage set, there will be the elements with the least absolute value.

lemma *pre-is-the-least-abs*:
assumes *abs-set A*≠{}
and *i*:*Rep-matrix A i j* ≠ *0*
and *x*:*x* ∈ *pre* (*nat*∘*abs* ∘ *Rep-matrix′ A*) (*Least* (*abs-set A*))
shows *nat*(*abs*(*Rep-matrix′ A x*)) ≤ *nat*(*abs*(*Rep-matrix′ A* (*i*,*j*)))
⟨*proof*⟩

lemma
assumes *a*:*abs-set A*≠{}
and *x*:*x* ∈ *pre* (*nat*∘*abs* ∘ *Rep-matrix′ A*) (*Least* (*abs-set A*))
shows *x* ∈ *nonzero-positions* (*Rep-matrix A*)
⟨*proof*⟩

Now we have to do the same but for a submatrix obtained from a position (k,k):

definition *abs-set-k* :: (*int matrix*) ⇒ *nat* ⇒ *nat set*
where *abs-set-k A k* = (*nat*∘*abs*∘(*Rep-matrix′ A*))‘ (*nonzero-positions* (*Rep-matrix A*)∩{(*i*,*j*). *i*≥*k* ∧ *j*≥*k*})

lemma *nonzero-implies-abs-k-ne*:
assumes *nonzero-positions* (*Rep-matrix A*)∩{(*i*,*j*). *i*≥*k* ∧ *j*≥*k*}≠{}
shows *abs-set-k A k* ≠ {}
⟨*proof*⟩

lemma *abs-k-notzero*:
assumes *x*: *x* ∈ *abs-set-k A k*
shows *x*≠*0*
⟨*proof*⟩

lemma *Least-in-abs-k*:
assumes *abs-set-k A k* ≠ {}
shows *Least* (*abs-set-k A k*) ∈ (*abs-set-k A k*)
⟨*proof*⟩

lemma *Least-le-abs-set-k*:
assumes *y*:*y* ∈ *abs-set-k A k* — Con esto ya se que no es vacio
shows *Least* (*abs-set-k A k*) ≤ *y*
⟨*proof*⟩

lemma *pre-abs-set-k*:
assumes *abs-set-k A k*≠{}
shows *pre* (*nat*∘*abs* ∘ *Rep-matrix′ A*) (*Least* (*abs-set-k A k*)) ≠ {}
⟨*proof*⟩

lemma *pre-abs-set-k′*:
assumes *abs-set-k A k*≠{}
shows *pre* (*nat*∘*abs* ∘ *Rep-matrix′ A*) (*Least* (*abs-set-k A k*)) ∩ {(*i*,*j*). *i*≥*k* ∧ *j*≥*k*} ≠ {}
⟨*proof*⟩

lemma *Least-in-pre-abs-k*:
assumes *abs-set-k A k*≠{}
shows *Least* (*pre* (*nat*∘*abs* ∘ *Rep-matrix′ A*) (*Least* (*abs-set-k A k*))) ∈ *pre* (*nat*∘*abs* ∘ *Rep-matrix′ A*) (*Least* (*abs-set-k A k*))
⟨*proof*⟩

lemma *Least-in-pre-abs-k′*:
assumes *abs-set-k A k*≠{}
shows *Least* (*pre* (*nat*∘*abs* ∘ *Rep-matrix′ A*) (*Least* (*abs-set-k A k*)) ∩ {(*i*,*j*). *i*≥*k* ∧ *j*≥*k*})
∈ (*pre* (*nat*∘*abs* ∘ *Rep-matrix′ A*) (*Least* (*abs-set-k A k*))) ∩ {(*i*,*j*). *i*≥*k* ∧ *j*≥*k*}
⟨*proof*⟩

lemma *pre-is-the-least-abs-k*:
assumes *abs-set-k A k*≠{}
and *i*:*Rep-matrix A i j* ≠ *0*
and *i2*:*i*≥*k*
and *i3*: *j*≥*k*
and *x*:*x* ∈ *pre* (*nat*∘*abs* ∘ *Rep-matrix′ A*) (*Least* (*abs-set-k A k*))
shows *nat*(*abs*(*Rep-matrix′ A x*)) ≤ *nat*(*abs*(*Rep-matrix′ A* (*i*,*j*)))
⟨*proof*⟩

lemma *pre-is-the-least-abs-k′*:
assumes *abs*:*abs-set-k A k*≠{}
and *i*:*Rep-matrix A i j* ≠ *0*
and *i2*:*i*≥*k*
and *i3*: *j*≥*k*
and *x*:*x* ∈ *pre* (*nat*∘*abs* ∘ *Rep-matrix′ A*) (*Least* (*abs-set-k A k*))∩{(*i*,*j*).*i*≥*k* ∧ *j*≥*k*}
shows *nat*(*abs*(*Rep-matrix′ A x*)) ≤ *nat*(*abs*(*Rep-matrix′ A* (*i*,*j*)))
⟨*proof*⟩

lemma *aux-nonzero*:
assumes *y*∈{*y*. ∃*x*∈*nonzero-positions* (*Rep-matrix A*). *y* = *nat* |*Rep-matrix′ A x*|}
shows *y*≠*0*
⟨*proof*⟩

lemma *aux-least*: (*Least A*) = (*LEAST x*. *x*∈*A*)
⟨*proof*⟩

lemma *pre-in-nonzero-k*:

assumes a:*abs-set-k A* $k \neq \{\}$
and x:$x \in$ *pre* (*nat*$\circ$*abs* $\circ$ *Rep-matrix'* *A*) (*Least* (*abs-set-k A k*)) $\cap$ $\{(i,j).\ i \geq k \wedge j \geq k\}$
shows $x \in$ *nonzero-positions* (*Rep-matrix A*)$\cap\{(i,j).\ i \geq k \wedge j \geq k\}$
⟨*proof*⟩

lemma *nat-abs-int*:
assumes (*nat*(*abs*(*a*::*int*))) $\leq$ *nat*(*abs*(*b*::*int*))
shows *abs*(*a*)$\leq$ *abs*(*b*)
⟨*proof*⟩

lemma *nat-abs-int'*:
assumes *abs*(*a*)$\leq$ *abs*(*b*)
shows (*nat*(*abs*(*a*::*int*))) $\leq$ *nat*(*abs*(*b*::*int*))
⟨*proof*⟩

lemma *minNonzero-least*:
assumes *P-def*: *P*=(λ*pos*. $k \leq$ *fst pos* $\wedge$ $k \leq$ *snd pos* $\wedge$ *Rep-matrix A* (*fst pos*) (*snd pos*)$\neq 0$ $\wedge$
($\forall$ *m n*. $k \leq m \wedge k \leq n \wedge$ *Rep-matrix A m n* $\neq 0 \longrightarrow$ |*Rep-matrix A* (*fst pos*) (*snd pos*)| $\leq$ |*Rep-matrix A m n*|))
and *ex-minNonzero*: *fst-bis* (*minNonzero-nc* (*A*::*int matrix*) *k*) = *True*
shows *P* (*LEAST k*. *P k*)
⟨*proof*⟩

We move the element with the least nonzero absolute value to position (k,k). Then, we will have there a non zero element.

lemma *move-minNonzero-not-zero*:
assumes *ex-minNonzero*: *fst-bis* (*minNonzero-nc A k*) = *True*
shows *Rep-matrix* (*move-minNonzero-nc A k*) *k k* $\neq 0$
⟨*proof*⟩

That element will be less than any other nonzero element (in absolute value)

lemma *minNonzero*:
assumes i:$i \geq k$ **and** j:$j \geq k$
and *ex-minNonzero*: *fst-bis* (*minNonzero-nc* (*A*::*int matrix*) *k*) = *True* — No se puede quitar y hacer por casos, solo si k = 0
and *ij-not-zero*: *Rep-matrix A i j* $\neq 0$
shows *abs*(*Rep-matrix A* (*snd-bis*(*minNonzero-nc A k*)) (*trd-bis*(*minNonzero-nc A k*))) $\leq$ *abs*(*Rep-matrix A i j*)
⟨*proof*⟩

lemma *move-minNonzero*:
assumes i:$i \geq k$ **and** j:$j \geq k$
and *Aik-not-zero*: *Rep-matrix* (*A*::*int matrix*) *i j* $\neq 0$
shows *abs*(*Rep-matrix* (*move-minNonzero-nc A k*) *k k*) $\leq$ *abs*(*Rep-matrix A i j*)
⟨*proof*⟩

lemma *abs-part-min-strict-decreasing′*:
assumes *Aik-not-zero*: *Rep-matrix* (*A*::*int matrix*) *i k* ≠ *0*
and *i*:*i*>*k*
shows *abs* (*Rep-matrix* (*partRowReduce-matrix* (*move-minNonzero-nc A k*) *k k*) *i k*) < *abs* (*Rep-matrix A i k*)
⟨*proof*⟩

lemma *abs-part-min-strict-decreasing-ik-kk*:
assumes *Akk-not-zero*: *Rep-matrix* (*A*::*int matrix*) *k k* ≠ *0*
and *i*:*i*>*k*
shows *abs* (*Rep-matrix* (*partRowReduce-matrix* (*move-minNonzero-nc A k*) *k k*) *i k*) < *abs* (*Rep-matrix A k k*)
⟨*proof*⟩

lemma *Akk-strict-decreasing-move-minNonzero*:
assumes *i*: *i*>*k*
and *menor*: *abs*(*Rep-matrix* (*A*::*int matrix*) *i k*) < *abs*(*Rep-matrix A k k*)
and *Aik-not-zero*: *Rep-matrix A i k* ≠ *0*
shows *abs* (*Rep-matrix* (*move-minNonzero-nc A k*) *k k*) < *abs* (*Rep-matrix A k k*)
⟨*proof*⟩

lemma *Akk-strict-decreasing-move-minNonzero′*:
assumes *i*: *i*>*k*
and *j*: *j*≥*k*
and *menor*: *abs*(*Rep-matrix* (*A*::*int matrix*) *i j*) < *abs*(*Rep-matrix A k k*)
and *Aij-not-zero*: *Rep-matrix A i j* ≠ *0*
shows *abs* (*Rep-matrix* (*move-minNonzero-nc A k*) *k k*) < *abs* (*Rep-matrix A k k*)
⟨*proof*⟩

lemma *Akk-strict-decreasing-move-minNonzero2*:
assumes *i*: *i*≥*k*
and *j*: *j*>*k*
and *menor*: *abs*(*Rep-matrix* (*A*::*int matrix*) *i j*) < *abs*(*Rep-matrix A k k*)
and *Aij-not-zero*: *Rep-matrix A i j* ≠ *0*
shows *abs* (*Rep-matrix* (*move-minNonzero-nc A k*) *k k*) < *abs* (*Rep-matrix A k k*)
⟨*proof*⟩

lemma *abs-part-min-strict-decreasing-kk*:
assumes *Akk-not-zero*: *Rep-matrix* (*A*::*int matrix*) *k k* ≠ *0*
and *i*>*k*
and *Rep-matrix A i k* ≠ *0*
and *menor*: *abs*(*Rep-matrix* (*A*::*int matrix*) *i k*) < *abs*(*Rep-matrix A k k*)

shows *abs* (*Rep-matrix* (*partRowReduce-matrix* (*move-minNonzero-nc A k*) *k k*) *k k*) < *abs* (*Rep-matrix A k k*)
⟨*proof*⟩

lemma *partRowReduce-eq-abs*:
|*Rep-matrix* (*partRowReduce-matrix A k k*) *k k*| = |*Rep-matrix A k k*|
⟨*proof*⟩

lemma *abs-part-move-ik-kk*:
assumes *i*: *i*>*k*
and *t*: *fst-bis*(*minNonzero-nc* (*A*::*int matrix*) *k*)=*True*
shows *abs* (*Rep-matrix* (*partRowReduce-matrix* (*move-minNonzero-nc A k*) *k k*) *i k*)
< *abs* (*Rep-matrix* (*partRowReduce-matrix* (*move-minNonzero-nc A k*) *k k*) *k k*)
⟨*proof*⟩

definition *f′* :: (*int matrix* × *nat*) ⇒ *nat*
where *f′ A* = *nat*(*abs*(*Rep-matrix* (*move-minNonzero-nc* (*fst A*) (*snd A*)) (*snd A*) (*snd A*)))

lemma *iterate-partRowReduce*:
fixes *A*:: *int matrix*
assumes *Akk-not-zero*: *Rep-matrix A k k* ≠ *0*
shows ∀ *i*>*k*. *Rep-matrix* (*iterate-partRowReduce A k*) *i k* = *0*
⟨*proof*⟩

lemma *iterate-decrease-kk*:
assumes ¬(∀ *i*>*k*. *Rep-matrix A i k* = *0*)
and ¬(∀ *i*>*k*. *Rep-matrix* (*partRowReduce-matrix*(*move-minNonzero-nc A k*) *k k*) *i k* = *0*)
shows |*Rep-matrix* (*iterate-partRowReduce A k*) *k k*| < |*Rep-matrix* (*move-minNonzero-nc A k*) *k k*|
⟨*proof*⟩

lemma *iterate-decrease-kk′*:
assumes *Rep-matrix A k k* ≠ *0*
shows |*Rep-matrix* (*iterate-partRowReduce A k*) *k k*| ≤ |*Rep-matrix A k k*|
⟨*proof*⟩

lemma *move-minNonzero-not-zero-Akk*:
assumes *Rep-matrix A k k* ≠ *0*
shows *Rep-matrix* (*move-minNonzero-nc A k*) *k k* ≠ *0*
⟨*proof*⟩

lemma *iterate-nonzero-kk*:
assumes *Rep-matrix A k k* $\neq$ *0*
shows |*Rep-matrix* (*iterate-partRowReduce A k*) *k k*| $\neq$ *0*
⟨*proof*⟩

lemma *partColumnReduce-eq-abs*:
|*Rep-matrix* (*partColumnReduce-matrix A k k*) *k k*| = |*Rep-matrix A k k*|
⟨*proof*⟩

lemma *partColumnReduce-iterate-partRowReduce*:
assumes *Rep-matrix A k k* $\neq$ *0*
shows $\forall$ *i*>*k*. *Rep-matrix* (*partColumnReduce-matrix* (*iterate-partRowReduce A k*) *k k*) *i k* = *0*
⟨*proof*⟩

partial-function (*tailrec*) *Diagonalize* :: (*int matrix*) $\Rightarrow$ *nat* $\Rightarrow$ (*int matrix*)
where *Diagonalize A k* = (*if* (($\forall$ *m*. *m*>*k* $\longrightarrow$ *Rep-matrix A m k* = *0*) $\wedge$ ($\forall$ *n*. *n*>*k* $\longrightarrow$ *Rep-matrix A k n* = *0*)) *then A else*
Diagonalize (*partColumnReduce-matrix* (*iterate-partRowReduce* (*move-minNonzero-nc A k*) *k*) *k k*) *k*)

declare *Diagonalize.simps* [*simp del*]

lemma *Diagonalize-iterate-zero*:
assumes *minNonzero*: *fst-bis* (*minNonzero-nc A k*) = *True*
shows ($\forall$ *m*. *m*>*k* $\longrightarrow$ *Rep-matrix* (*Diagonalize A k*) *m k* = *0*)
⟨*proof*⟩

lemma *Diagonalize*:
assumes *minNonzero*: *fst-bis* (*minNonzero-nc A k*) = *True*
shows ($\forall$ *m*. *m*>*k* $\longrightarrow$ *Rep-matrix* (*Diagonalize A k*) *m k* = *0*) $\wedge$ ($\forall$ *n*. *n*>*k* $\longrightarrow$ *Rep-matrix* (*Diagonalize A k*) *k n* = *0*)
⟨*proof*⟩

lemma *Diagonalize-result*:
shows ($\forall$ *m*. *m*>*k* $\longrightarrow$ *Rep-matrix* (*Diagonalize A k*) *m k* = *0*) $\wedge$ ($\forall$ *n*. *n*>*k* $\longrightarrow$ *Rep-matrix* (*Diagonalize A k*) *k n* = *0*)
⟨*proof*⟩

Next function returns True if a matrix A is diagonal until position k (not included)

definition *is-Diagonal* :: (*int matrix* $\Rightarrow$ *nat* $\Rightarrow$ *bool*)
where *is-Diagonal A k* = ($\forall$ *a b*. ((*0* $\leq$ *a* $\wedge$ *a*<*k*) $\vee$ (*0* $\leq$ *b* $\wedge$ *b*<*k*)) $\wedge$ *a*$\neq$*b* $\longrightarrow$ *Rep-matrix A a b* = *0*)

lemma *is-Diagonal-0*:
shows *is-Diagonal A 0* = *True*
⟨*proof*⟩

lemma *Diagonalize-interchange-rows*:
assumes *es-H*:*is-Diagonal A k*
and *i*:*i*≥*k*
shows *is-Diagonal* (*interchange-rows-matrix A i k*) *k*
⟨*proof*⟩

lemma *Diagonalize-interchange-columns*:
assumes *es-H*:*is-Diagonal A k*
and *i*:*i*≥*k*
shows *is-Diagonal* (*interchange-columns-matrix A i k*) *k*
⟨*proof*⟩

lemma *k-le-snd-minNonzero*:
assumes *ex-minNonzero*: *fst-bis* (*minNonzero-nc A k*) = *True*
shows *k* ≤ *snd-bis* (*minNonzero-nc A k*)
⟨*proof*⟩

lemma *k-le-trd-minNonzero*:
assumes *ex-minNonzero*: *fst-bis* (*minNonzero-nc A k*) = *True*
shows *k* ≤ *trd-bis* (*minNonzero-nc A k*)
⟨*proof*⟩

lemma *Diagonalize-move-element*:
assumes *is-Diagonal A k*
and *i*≥*k*
and *j*≥*k*
shows *is-Diagonal* (*move-element A i j k k*) *k*
⟨*proof*⟩

lemma *Diagonalize-move-minNonzero*:
assumes *es-H*: *is-Diagonal A k*
and *f*: *fst-bis* (*minNonzero-nc A k*) = *True*
shows *is-Diagonal* (*move-minNonzero-nc A k*) *k*
⟨*proof*⟩

lemma *is-Diagonal-partRowReduce-move-minNonzero*:
assumes *es-H*:*is-Diagonal A k*
and *f*: *fst-bis* (*minNonzero-nc A k*) = *True*
shows *is-Diagonal* (*partRowReduce-matrix* (*move-minNonzero-nc A k*) *k k*) *k*
⟨*proof*⟩

lemma *is-Diagonal-iterate-partRowReduce*:
assumes *is-Diagonal A k*
shows *is-Diagonal* (*iterate-partRowReduce A k*) *k*

⟨*proof*⟩

lemma *is-Diagonal-partColumnReduce*:
assumes *es-H*: *is-Diagonal A k*
shows *is-Diagonal* (*partColumnReduce-matrix A k k*) *k*
⟨*proof*⟩

lemma *is-Diagonal-partColumnReduce-matrix-iterate-partRow*:
assumes *is-Diagonal A k*
and *fst-bis* (*minNonzero-nc A k*) = *True*
shows *is-Diagonal* (*partColumnReduce-matrix* (*iterate-partRowReduce* (*move-minNonzero-nc A k*) *k*) *k k*) *k*
⟨*proof*⟩

lemma *is-Diagonal-Diagonalize-k*:
assumes *es-H*: *is-Diagonal A k*
shows *is-Diagonal* (*Diagonalize A k*) *k*
⟨*proof*⟩

lemma *is-Diagonal-Diagonalize-suc-k*:
assumes *es-H*: *is-Diagonal A k*
shows *is-Diagonal* (*Diagonalize A k*) (*k+1*)
⟨*proof*⟩

lemma *not-is-Diagonal-greather*:
assumes ¬ *is-Diagonal A k*
and $k \leq j$
shows ¬ *is-Diagonal A j*
⟨*proof*⟩

lemma *is-Diagonal-less*:
assumes *is-Diagonal A k*
and $j \leq k$
shows *is-Diagonal A j*
⟨*proof*⟩

This function diagonalizes a matrix up to position k (included).

fun *Diagonalize-up-to-k* :: *int matrix* => *nat* => *int matrix*
where
Diagonalize-up-to-k A 0 = *Diagonalize A 0*
| *Diagonalize-up-to-k A* (*Suc k*) = *Diagonalize* (*Diagonalize-up-to-k A k*) (*Suc k*)

This functions returns us True if a matrix is diagonal up to position k (included). False, in other case.

definition *Diagonalize-p* :: (*int matrix* ⇒ *nat* ⇒ *bool*)

where *Diagonalize-p A k* = *is-Diagonal A* (*k* + *1*)

lemma *Diagonalize-p-zero*:
shows *Diagonalize-p* (*Diagonalize A 0*) *0*
⟨*proof*⟩

lemma *Diagonalize-p-incr*:
assumes *Diagonalize-p A k*
shows *Diagonalize-p* (*Diagonalize A* (*Suc k*)) (*Suc k*)
⟨*proof*⟩

lemma
Diagonalize-p-mon:
assumes *Diagonalize-p A n*
and $m \leq n$
shows *Diagonalize-p A m*
⟨*proof*⟩

lemma *Diagonalize-p-Diagonalize-less*:
assumes *m-le-n*: $m \leq n$
shows *Diagonalize-p* (*Diagonalize-up-to-k A n*) *m*
⟨*proof*⟩

corollary *Diagonalize-p-Diagonalize-n*:
shows *Diagonalize-p* (*Diagonalize-up-to-k A n*) *n*
⟨*proof*⟩

corollary *Diagonalize-p-Diagonalize-max*:
shows *Diagonalize-p* (*Diagonalize-up-to-k A* (*max* (*nrows A*) (*ncols A*))) (*max* (*nrows A*) (*ncols A*))
⟨*proof*⟩

definition *P-interchange-rows* :: *int matrix* => *nat* ⇒ *nat* ⇒ *int matrix*
where *P-interchange-rows A n m* == *interchange-rows-matrix* (*one-matrix* (*nrows A*)) *n m*

definition *P-interchange-rows′* :: *int matrix* ⇒ *nat* ⇒ *nat* ⇒ *int matrix*
where *P-interchange-rows′ A n m* ≡ *Abs-matrix*(λ*i j*. *if* (*i*=*n* ∧ *j*=*m*) *then 1 else* (*if* (*i*=*m* ∧ *j*=*n*) *then 1 else* (*if* (*i*=*j* ∧ *i*≠*m* ∧ *i*≠*n* ∧ *i* < *nrows A*) *then 1 else 0*)))

definition *Q-interchange-columns* :: *int matrix* => *nat* ⇒ *nat* ⇒ *int matrix*
where *Q-interchange-columns A n m* == *interchange-columns-matrix* (*one-matrix* (*ncols A*)) *n m*

definition *Q-interchange-columns′* :: *int matrix* ⇒ *nat* ⇒ *nat* ⇒ *int matrix*
where *Q-interchange-columns′ A n m* ≡ *Abs-matrix*(λ*i j*. *if* (*i*=*n* ∧ *j*=*m*) *then 1 else* (*if* (*i*=*m* ∧ *j*=*n*) *then 1 else* (*if* (*i*=*j* ∧ *i*≠*m* ∧ *i*≠*n* ∧ *j* < *ncols A*) *then 1 else 0*)))

definition *P-row-add* :: *int matrix* ⇒ *nat* ⇒ *nat* ⇒ *int* ⇒ *int matrix*
where *P-row-add A i j q* = *row-add-matrix* (*one-matrix* (*nrows A*)) *i j q*

definition *P-row-add′* :: *int matrix* ⇒ *nat* ⇒ *nat* ⇒ *int* ⇒ *int matrix*
where *P-row-add′ A n m q* = *Abs-matrix*(λ*i j*. *if* (*i*=*n* ∧ *j*=*m*) *then q else* (*if* (*i*=*j* ∧ *i*<*nrows A*) *then 1 else 0*))

definition *Q-column-add* :: *int matrix* ⇒ *nat* ⇒ *nat* ⇒ *int* ⇒ *int matrix*
where *Q-column-add A i j q* = *column-add-matrix* (*one-matrix* (*ncols A*)) *i j q*

definition *Q-column-add′* :: *int matrix* ⇒ *nat* ⇒ *nat* ⇒ *int* ⇒ *int matrix*
where *Q-column-add′ A n m q* = *Abs-matrix*(λ*i j*. *if* (*i*=*m* ∧ *j*=*n*) *then q else* (*if* (*i*=*j* ∧ *j*<*ncols A*) *then 1 else 0*))

lemma *P-row-add-eq*:
assumes *n*<*nrows A*
and *m*<*nrows A*
and *n*≠*m*
shows *P-row-add′* (*A*::*int matrix*) *n m q* = *P-row-add A n m q*
⟨*proof*⟩

lemma *P-interchange-row-eq*:
assumes *n*<*nrows A*
and *m*<*nrows A*
shows *P-interchange-rows′* (*A*::*int matrix*) *n m*= *P-interchange-rows A n m*
⟨*proof*⟩

lemma *Q-interchange-column-eq*:
assumes *n*<*ncols A*
and *m*<*ncols A*
shows *Q-interchange-columns′* (*A*::*int matrix*) *n m* = *Q-interchange-columns A n m*
⟨*proof*⟩

lemma *Q-column-add-eq*:
assumes *n*<*ncols A*
and *m*<*ncols A*
and *n*≠*m*
shows *Q-column-add′* (*A*::*int matrix*) *n m q* = *Q-column-add A n m q*
⟨*proof*⟩

lemma *Rep-abs-aux-P-interchange-rows*:
assumes $n<nrows\ A$ **and** $m<nrows\ A$
shows *Rep-matrix* (*Abs-matrix* ($\lambda i\ j$. *if* $i = n \wedge j = m$ *then 1 else if* $i = m \wedge j = n$ *then 1 else if* $i = j \wedge i \neq m \wedge i \neq n \wedge i < nrows\ A$ *then 1 else 0*))
$=(\lambda i\ j$. *if* $i = n \wedge j = m$ *then 1 else if* $i = m \wedge j = n$ *then 1 else if* $i = j \wedge i \neq m \wedge i \neq n \wedge i < nrows\ A$ *then 1 else 0*)
⟨*proof*⟩

lemma *one-matrix-exists-notnull*:
assumes $a<b$
shows $\exists\, i$. *Rep-matrix* ((*one-matrix b*)::*int matrix*) $a\ i \neq 0$
⟨*proof*⟩

lemma *interchange-rows-nrows*:
assumes n:$n<nrows\ A$
and m:$m<nrows\ A$
and i: $\exists\, i$. *Rep-matrix A* (*min n m*) $i \neq 0$
shows *nrows* (*interchange-rows-matrix* (*A*::*int matrix*) *n m*) = *nrows A*
⟨*proof*⟩

lemma *interchange-rows-ncols*:
assumes n:$n<nrows\ A$
and m:$m<nrows\ A$
shows *ncols* (*interchange-rows-matrix* (*A*::*int matrix*) *n m*) = *ncols A*
⟨*proof*⟩

lemma *P-interchange-rows-nrows-eq*:
assumes $n<nrows\ A$
and $m<nrows\ A$
shows *nrows* (*P-interchange-rows* (*A*::*int matrix*) *n m*) = *ncols* (*P-interchange-rows* (*A*::*int matrix*) *n m*)
⟨*proof*⟩

lemma *P-interchange-rows′-nrows-eq*:
assumes $n<nrows\ A$
and $m<nrows\ A$
shows *nrows* (*P-interchange-rows′* (*A*::*int matrix*) *n m*) = *ncols* (*P-interchange-rows′* (*A*::*int matrix*) *n m*)
⟨*proof*⟩

lemma *P-interchange-rows-nrows-eq2*:
assumes $n<nrows\ A$

and $m<nrows\ A$
shows $nrows\ (P\text{-}interchange\text{-}rows\ (A::int\ matrix)\ n\ m) = nrows\ A$
⟨*proof*⟩

corollary *P-interchange-rows′-nrows*:
assumes $n<nrows\ A$
and $m<nrows\ A$
shows $nrows\ (P\text{-}interchange\text{-}rows'\ (A::int\ matrix)\ n\ m)=nrows\ (P\text{-}interchange\text{-}rows\ (A::int\ matrix)\ n\ m)$
⟨*proof*⟩

corollary *P-interchange-rows′-nrows-A*:
assumes $n<nrows\ A$
and $m<nrows\ A$
shows $nrows\ (P\text{-}interchange\text{-}rows'\ (A::int\ matrix)\ n\ m)=nrows\ A$
⟨*proof*⟩

corollary *P-interchange-rows′-nrows-A-2*:
assumes $n<nrows\ A$
and $m<nrows\ A$
shows $ncols\ (P\text{-}interchange\text{-}rows'\ (A::int\ matrix)\ n\ m)=nrows\ A$
⟨*proof*⟩

lemma *P-interchange-rows-inverse*:
assumes n:$n<nrows\ A$
and m:$m<nrows\ A$
shows $inverse\text{-}matrix\ (P\text{-}interchange\text{-}rows'\ (A::int\ matrix)\ n\ m)\ (P\text{-}interchange\text{-}rows'\ A\ n\ m)$
⟨*proof*⟩

lemma *P-row-add-nrows*:
assumes n:$n<nrows\ A$
and m:$m<nrows\ A$
and *n-not-m*: $n\neq m$
shows $nrows\ (P\text{-}row\text{-}add\ (A::int\ matrix)\ n\ m\ q) = nrows\ A$
⟨*proof*⟩

lemma *P-row-add-ncols*:
assumes n:$n<nrows\ A$
and m:$m<nrows\ A$
and *n-not-m*: $n\neq m$
shows $ncols\ (P\text{-}row\text{-}add\ (A::int\ matrix)\ n\ m\ q) = nrows\ A$
⟨*proof*⟩

corollary *P-row-add-nrows-eq-ncols*:
assumes n:$n<nrows\ A$

and *m*:*m*<*nrows A*
and *n-not-m*: *n*≠*m*
shows *nrows* (*P-row-add* (*A*::*int matrix*) *n m q*) = *ncols* (*P-row-add* (*A*::*int matrix*) *n m q*)
⟨*proof*⟩

corollary *P-row-add′-nrows-eq-ncols*:
assumes *n*:*n*<*nrows A*
and *m*:*m*<*nrows A*
and *n-not-m*: *n*≠*m*
shows *nrows* (*P-row-add′* (*A*::*int matrix*) *n m q*) = *ncols* (*P-row-add′* (*A*::*int matrix*) *n m q*)
⟨*proof*⟩

lemma *Rep-abs-aux-P-row-add′*:
assumes *n*<*nrows A*
and *m*<*nrows A*
shows *Rep-matrix* (*Abs-matrix* (λ*i j*. *if i* = *n* ∧ *j* = *m then q else if i* = *j* ∧ *i* < *nrows A then 1*::*int else 0*))
=(λ*i j*. *if i* = *n* ∧ *j* = *m then q else if i* = *j* ∧ *i* < *nrows A then 1 else 0*)
⟨*proof*⟩

corollary *P-row-add′-nrows-eq-nrows-inverse*:
assumes *n*:*n*<*nrows A*
and *m*:*m*<*nrows A*
and *n-not-m*: *n*≠*m*
shows *nrows* (*P-row-add′* (*A*::*int matrix*) *n m q*) = *nrows* (*P-row-add′ A n m* (−*q*))
⟨*proof*⟩

corollary *P-row-add′-nrows-eq-ncols-inverse*:
assumes *n*:*n*<*nrows A*
and *m*:*m*<*nrows A*
and *n-not-m*: *n*≠*m*
shows *ncols* (*P-row-add′* (*A*::*int matrix*) *n m q*) = *ncols* (*P-row-add′ A n m* (−*q*))
⟨*proof*⟩

lemma *foldseq-aux*:
fixes *s*:: *nat* => *int*
shows *foldseq* (*op* +) *s* (*Suc n*) = (*op* +) (*s* (*Suc n*)) (*foldseq* (*op* +) *s n*)
⟨*proof*⟩

lemma *foldseq-finsum*:
fixes *s*:: *nat* => *int*

shows *foldseq* (*op* +) *s n* = *finsum* $\mathcal{Z}$ *s* {..*n*}
⟨*proof*⟩

lemma *abelian-monoid-z*:
shows *abelian-monoid* $\mathcal{Z}$
⟨*proof*⟩

lemma *finsum-singleton2*:
shows *finsum* $\mathcal{Z}$ *s* {*a*} = *s a*
⟨*proof*⟩

lemma *foldseq-almostzero-2*:
fixes *s*:: *nat* => *int*
assumes *s0*:∀ *i*. *i* ≠ *j* ∧ *i* ≠ *k* ⟶ *s i* = *0*
and *j*:*j*≤*n* **and** *k*:*k*≤*n* **and** *j-k*: *j*≠*k*
shows *foldseq* (*op* +) *s n* = (*s j*) + (*s k*)
⟨*proof*⟩ **find-theorems** *finsum*
⟨*proof*⟩ **thm** *Ring.abelian-monoid.finsum-zero*[*of* $\mathcal{Z}$]
⟨*proof*⟩

lemma *P-row-add-inverse*:
assumes *n*:*n*<*nrows A*
and *m*:*m*<*nrows A*
and *n-not-m*: *n*≠*m*
shows *inverse-matrix* (*P-row-add′* (*A*::*int matrix*) *n m q*) (*P-row-add′ A n m* (−*q*))
⟨*proof*⟩

lemma *Rep-matrix-mult-finsum*:
shows *Rep-matrix* (*A* ∗ *B*) *j i* = *finsum* $\mathcal{Z}$ (λ*k*. *Rep-matrix A j k* ∗ *Rep-matrix B k i*) {..(*max* (*ncols A*) (*nrows B*))}
⟨*proof*⟩

definition *is-invertible* :: *int matrix* => *bool*
where *is-invertible A* = (∃ *B*. *inverse-matrix A B*)

lemma *interchange-rows-PA*:
assumes *n*:*n*<*nrows A*
and *m*:*m*<*nrows A*
shows *interchange-rows-matrix* (*A*::*int matrix*) *n m* = *P-interchange-rows′ A n m* ∗ *A*
⟨*proof*⟩
find-theorems *finsum*
⟨*proof*⟩

lemma *interchange-rows-PAQ*:
assumes *n*<*nrows A*

and $m<nrows\ A$
shows $\exists\, P\ Q.$ *is-invertible* $P \wedge$ *is-invertible* $Q \wedge$ *interchange-rows-matrix* $A\ n\ m$
$= P*A*Q$
⟨*proof*⟩

lemma *interchange-rows-PAQ'*:
assumes $n{:}n<nrows\ (A{::}int\ matrix)$
and $m{:}m<nrows\ A$
shows *interchange-rows-matrix* $A\ n\ m = (P\text{-}interchange\text{-}rows'\ A\ n\ m) *A*(one\text{-}matrix(ncols\ A))$
⟨*proof*⟩

lemma *row-add-PA*:
assumes $n{:}n<nrows\ A$
and $m{:}m<nrows\ A$
and *n-not-m*: $n\neq m$
shows *row-add-matrix* $(A{::}int\ matrix)\ n\ m\ k = P\text{-}row\text{-}add'\ A\ n\ m\ k * A$
⟨*proof*⟩

lemma *rows-add-PAQ*:
assumes $n<nrows\ A$
and $m<nrows\ A$
and $n\neq m$
shows $\exists\, P\ Q.$ *is-invertible* $P \wedge$ *is-invertible* $Q \wedge$ *row-add-matrix* $A\ n\ m\ k= P*A*Q$
⟨*proof*⟩

lemma *mult-matrix-inverse*:
assumes $A{:}is\text{-}invertible\ (A{::}int\ matrix)$
and $B{:}is\text{-}invertible\ B$
and $n{:}ncols\ A = nrows\ B$ — PARA LA MULTIPLICACION. NO SE SI SE PUEDE SACAR A PARTIR DE TEOREMAS YA HECHOS.
shows *is-invertible* $(A*B) \wedge nrows\ (A*B)=nrows\ A \wedge ncols(A*B)=ncols\ B$
⟨*proof*⟩

corollary *mult-matrix-inverse2*:
assumes $A{:}is\text{-}invertible\ (A{::}int\ matrix)$
and $B{:}is\text{-}invertible\ B$
and $n{:}ncols\ A = nrows\ B$
shows *is-invertible* $(A*B)$ ⟨*proof*⟩

lemma *Rep-abs-aux-Q-interchange-cols*:
assumes $n<ncols\ A$ **and** $m<ncols\ A$
shows *Rep-matrix* $(Abs\text{-}matrix\ (\lambda i\ j.\ if\ i = n \wedge j = m\ then\ 1\ else\ if\ i = m \wedge j = n\ then\ 1\ else\ if\ i = j \wedge i \neq m \wedge i \neq n \wedge j < ncols\ A\ then\ 1\ else\ 0))$
$=(\lambda i\ j.\ if\ i = n \wedge j = m\ then\ 1\ else\ if\ i = m \wedge j = n\ then\ 1\ else\ if\ i = j \wedge i \neq m \wedge i \neq n \wedge j < ncols\ A\ then\ 1\ else\ 0)$
⟨*proof*⟩

lemma *one-matrix-exists-notnull-cols*:
assumes $a<b$
shows $\exists\, i.$ *Rep-matrix* ((*one-matrix* b)::*int matrix*) i $a \neq 0$
⟨*proof*⟩

lemma *interchange-columns-ncols*:
assumes n:$n<$*ncols* A
and m:$m<$*ncols* A
and i: $\exists\, i.$ *Rep-matrix* A i (*min* n m) $\neq 0$
shows *ncols* (*interchange-columns-matrix* (A::*int matrix*) n m) = *ncols* A
⟨*proof*⟩

lemma *interchange-columns-nrows*:
assumes n:$n<$*ncols* A
and m:$m<$*ncols* A
shows *nrows* (*interchange-columns-matrix* (A::*int matrix*) n m) = *nrows* A
⟨*proof*⟩

lemma *Q-interchange-columns-nrows-eq*:
assumes $n<$*ncols* A
and $m<$*ncols* A
shows *nrows* (*Q-interchange-columns* (A::*int matrix*) n m) = *ncols* (*Q-interchange-columns* (A::*int matrix*) n m)
⟨*proof*⟩

thm *P-interchange-row-eq*

lemma *Q-interchange-rows′-nrows-eq*:
assumes $n<$*ncols* A
and $m<$*ncols* A
shows *nrows* (*Q-interchange-columns′* (A::*int matrix*) n m) = *ncols* (*Q-interchange-columns′* (A::*int matrix*) n m)
⟨*proof*⟩

thm *P-interchange-rows-nrows-eq2*

lemma *Q-interchange-rows-ncols-eq*:
assumes $n<$*ncols* A
and $m<$*ncols* A
shows *ncols* (*Q-interchange-columns* (A::*int matrix*) n m) = *ncols* A
⟨*proof*⟩

corollary *Q-interchange-columns′-nrows*:
assumes $n<ncols\ A$
and $m<ncols\ A$
shows $nrows\ (Q\text{-}interchange\text{-}columns'\ (A::int\ matrix)\ n\ m)=nrows\ (Q\text{-}interchange\text{-}columns\ (A::int\ matrix)\ n\ m)$
⟨*proof*⟩

corollary *Q-interchange-columns′-ncols-A*:
assumes $n<ncols\ A$
and $m<ncols\ A$
shows $ncols\ (Q\text{-}interchange\text{-}columns'\ (A::int\ matrix)\ n\ m)=ncols\ A$
⟨*proof*⟩

corollary *Q-interchange-columns′-ncols-A-2*:
assumes $n<ncols\ A$
and $m<ncols\ A$
shows $nrows\ (Q\text{-}interchange\text{-}columns'\ (A::int\ matrix)\ n\ m)=ncols\ A$
⟨*proof*⟩

lemma *Q-interchange-columns-inverse*:
assumes $n{:}n<ncols\ A$
and $m{:}\ m<ncols\ (A::int\ matrix)$
shows $inverse\text{-}matrix\ (Q\text{-}interchange\text{-}columns'\ A\ n\ m)\ (Q\text{-}interchange\text{-}columns'\ A\ n\ m)$
⟨*proof*⟩

lemma *interchange-columns-AQ*:
assumes $n{:}n<ncols\ A$
and $m{:}\ m<ncols\ (A::int\ matrix)$
shows $interchange\text{-}columns\text{-}matrix\ A\ n\ m = A * Q\text{-}interchange\text{-}columns'\ A\ n\ m$
⟨*proof*⟩

lemma *interchange-columns-PAQ*:
assumes $n<ncols\ A$
and $m<ncols\ A$
shows $\exists\, P\ Q.\ is\text{-}invertible\ P \wedge is\text{-}invertible\ Q \wedge interchange\text{-}columns\text{-}matrix\ A\ n\ m = P*A*Q$
⟨*proof*⟩

lemma *interchange-columns-PAQ′*:
assumes $n{:}n<ncols\ (A::int\ matrix)$
and $m{:}m<ncols\ A$

shows *interchange-columns-matrix A n m* = *one-matrix* (*nrows A*)∗*A*∗*Q-interchange-columns′ A n m*
⟨*proof*⟩

lemma *Q-column-add-ncols*:
assumes *n*:*n*<*ncols A*
and *m*:*m*<*ncols A*
and *n-not-m*: *n*≠*m*
shows *ncols* (*Q-column-add* (*A*::*int matrix*) *n m q*) = *ncols A*
⟨*proof*⟩

lemma *Q-column-add-nrows*:
assumes *n*:*n*<*ncols A*
and *m*:*m*<*ncols A*
and *n-not-m*: *n*≠*m*
shows *nrows* (*Q-column-add* (*A*::*int matrix*) *n m q*) = *ncols A*
⟨*proof*⟩

corollary *Q-column-add-nrows-eq-ncols*:
assumes *n*:*n*<*ncols A*
and *m*:*m*<*ncols A*
and *n-not-m*: *n*≠*m*
shows *nrows* (*Q-column-add* (*A*::*int matrix*) *n m q*) = *ncols* (*Q-column-add* (*A*::*int matrix*) *n m q*)
⟨*proof*⟩

corollary *Q-column-add′-nrows-eq-ncols*:
assumes *n*:*n*<*ncols A*
and *m*:*m*<*ncols A*
and *n-not-m*: *n*≠*m*
shows *nrows* (*Q-column-add′* (*A*::*int matrix*) *n m q*) = *ncols* (*Q-column-add′* (*A*::*int matrix*) *n m q*)
⟨*proof*⟩

lemma *Rep-abs-aux-Q-column-add′*:
assumes *n*<*ncols A*
and *m*<*ncols A*
shows *Rep-matrix* (*Abs-matrix* (λ*i j*. *if i* = *m* ∧ *j* = *n then q else if i* = *j* ∧ *j* < *ncols A then 1 else 0*))
= (λ*i j*. *if i* = *m* ∧ *j* = *n then q else if i* = *j* ∧ *j* < *ncols A then 1 else 0*)
⟨*proof*⟩

corollary *Q-column-add′-ncols-eq-ncols-inverse*:
assumes *n*:*n*<*ncols A*
and *m*:*m*<*ncols A*
and *n-not-m*: *n*≠*m*
shows *ncols* (*Q-column-add′* (*A*::*int matrix*) *n m q*) = *ncols* (*Q-column-add′ A n*

m $(-q))$
⟨*proof*⟩

corollary *Q-column-add′-ncols-eq-nrows-inverse*:
assumes n:n<*ncols A*
and m:m<*ncols A*
and *n-not-m*: $n \neq m$
shows *nrows* (*Q-column-add′* (*A*::*int matrix*) *n m q*) = *nrows* (*Q-column-add′ A n m* $(-q)$)
⟨*proof*⟩

lemma *Q-column-add-inverse*:
assumes n:n<*ncols A*
and m:m<*ncols A*
and *n-not-m*: $n \neq m$
shows *inverse-matrix* (*Q-column-add′* (*A*::*int matrix*) *n m q*) (*Q-column-add′ A n m* $(-q)$)
⟨*proof*⟩

lemma *column-add-AQ*:
assumes n:n<*ncols A*
and m:m<*ncols A*
and *n-not-m*: $n \neq m$
shows *column-add-matrix* (*A*::*int matrix*) *n m k* = *A* ∗ *Q-column-add′ A n m k*
⟨*proof*⟩

lemma *column-add-PAQ*:
assumes n<*ncols A*
and m<*ncols A*
and $n \neq m$
shows ∃ *P Q*. *is-invertible P* ∧ *is-invertible Q* ∧ *column-add-matrix A n m k*= *P*∗*A*∗*Q*
⟨*proof*⟩

definition *P-interchange-rows-new* :: *nat* => *nat* ⇒ *nat* ⇒ *int matrix*
where *P-interchange-rows-new a n m* == *interchange-rows-matrix* (*one-matrix a*) *n m*

definition *P-interchange-rows′-new* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *int matrix*
where *P-interchange-rows′-new a n m* ≡ *Abs-matrix*(λ*i j*. *if* (*i*=*n* ∧ *j*=*m*) *then 1 else* (*if* (*i*=*m* ∧ *j*=*n*) *then 1 else* (*if* (*i*=*j* ∧ *i*≠*m* ∧ *i*≠*n* ∧ *i* < *a*) *then 1 else 0*)))

lemma *P-interchange-row-new-eq*:
assumes *n*:*n* < *a*
and *m*:*m* < *a*
shows *P-interchange-rows′-new a n m* = *P-interchange-rows-new a n m*
⟨*proof*⟩

lemma *P-interchange-rows-new-ncols*:
assumes *n*<*a* **and** *m*<*a*
shows *ncols* (*P-interchange-rows′-new a n m*) = *a*
⟨*proof*⟩

lemma *P-interchange-rows-new-nrows*:
assumes *n*<*a* **and** *m*<*a*
shows *nrows* (*P-interchange-rows′-new a n m*) = *a*
⟨*proof*⟩

lemma *P-interchange-rows-new-inverse*:
assumes *n*:*n*<*a*
and *m*:*m*<*a*
shows *inverse-matrix* (*P-interchange-rows′-new a n m*) (*P-interchange-rows′-new a n m*)
⟨*proof*⟩

lemma *Rep-Abs-P-interchange-rows-new*:
assumes *n*:*n*<*z* **and** *m*:*m*<*z*
shows *Rep-matrix* (*Abs-matrix* (λ*i j*. *if i* = *n* ∧ *j* = *m then 1*::*int else if i* = *m* ∧ *j* = *n then 1 else if i* = *j* ∧ *i* ≠ *m* ∧ *i* ≠ *n* ∧ *i* < *z then 1 else 0*))
= (λ*i j*. *if i* = *n* ∧ *j* = *m then 1 else if i* = *m* ∧ *j* = *n then 1*::*int else if i* = *j* ∧ *i* ≠ *m* ∧ *i* ≠ *n* ∧ *i* < *z then 1 else 0*)
⟨*proof*⟩

lemma *interchange-rows-PA-new*:
assumes *n*:*n*<*nrows A*
and *m*: *m*<*nrows A*
and *na*: *nrows A* ≤ *z*
shows *interchange-rows-matrix* (*A*::*int matrix*) *n m* = *P-interchange-rows′-new z n m* ∗ *A*
⟨*proof*⟩

definition *Q-interchange-columns-new* :: *nat* => *nat* ⇒ *nat* ⇒ *int matrix*
where *Q-interchange-columns-new a n m* == *interchange-columns-matrix* (*one-matrix a*) *n m*

definition *Q-interchange-columns′-new* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *int matrix*
where *Q-interchange-columns′-new a n m* ≡ *Abs-matrix*(λ*i j*. *if* (*i*=*n* ∧ *j*=*m*) *then 1 else* (*if* (*i*=*m* ∧ *j*=*n*) *then 1 else* (*if* (*i*=*j* ∧ *i*≠*m* ∧ *i*≠*n* ∧ *j* < *a*) *then 1 else 0*)))

lemma *Q-interchange-columns-new-eq*:
assumes *n*:*n* < *a*
and *m*:*m* < *a*
shows *Q-interchange-columns′-new a n m* = *Q-interchange-columns-new a n m*
⟨*proof*⟩

lemma *Q-interchange-columns-new-inverse*:
assumes *n*:*n* < *a*
and *m*: *m* < *a*
shows *inverse-matrix* (*Q-interchange-columns′-new a n m*) (*Q-interchange-columns′-new a n m*)
⟨*proof*⟩

lemma *Rep-Abs-Q-interchange-columns-new*:
assumes *n*:*n*<*z* **and** *m*:*m*<*z*
shows *Rep-matrix* (*Abs-matrix* (λ*i j*. *if i* = *n* ∧ *j* = *m then 1*::*int else if i* = *m* ∧ *j* = *n then 1 else if i* = *j* ∧ *i* ≠ *m* ∧ *i* ≠ *n* ∧ *j* < *z then 1 else 0*)) =
(λ*i j*. *if i* = *n* ∧ *j* = *m then 1 else if i* = *m* ∧ *j* = *n then 1*::*int else if i* = *j* ∧ *i* ≠ *m* ∧ *i* ≠ *n* ∧ *j* < *z then 1 else 0*)
⟨*proof*⟩

lemma *Q-interchange-columns-new-nrows*:
assumes *n*<*a* **and** *m*<*a*
shows *nrows* (*Q-interchange-columns′-new a n m*) = *a*
⟨*proof*⟩

lemma *interchange-columns-AQ-new*:
assumes *n*:*n*<*ncols A*
and *m*: *m*<*ncols A*
and *na*: *ncols A* ≤ *z*
shows *interchange-columns-matrix* (*A*::*int matrix*) *n m* = *A*∗*Q-interchange-columns′-new z n m*
⟨*proof*⟩

definition *P-row-add-new* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *int* ⇒ *int matrix*
where *P-row-add-new a i j q* = *row-add-matrix* (*one-matrix a*) *i j q*

definition *P-row-add′-new* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *int* ⇒ *int matrix*
where *P-row-add′-new a n m q* = *Abs-matrix*(λ*i j*. *if* (*i*=*n* ∧ *j*=*m*) *then q else* (*if* (*i*=*j* ∧ *i*<*a*) *then 1 else 0*))

lemma *P-row-add-new-eq*:
assumes *n*:*n* < *a*
and *m*:*m* < *a*
and *n-not-m*: *n*≠*m*
shows *P-row-add′-new a n m q* = *P-row-add-new a n m q*
⟨*proof*⟩

lemma *P-row-add-new-inverse*:
assumes *n*:*n*<*a*
and *m*:*m*<*a*
and *n-not-m*: *n*≠*m*
shows *inverse-matrix* (*P-row-add′-new a n m q*) (*P-row-add′-new a n m* (−*q*))
⟨*proof*⟩

lemma *RepAbs-row-add-new*:
assumes *n*<*z* **and** *m*<*z*
shows *Rep-matrix* (*Abs-matrix* (λ*i j*. *if i* = *n* ∧ *j* = *m then k else if i* = *j* ∧ *i* < *z then 1*::*int else 0*))
= (λ*i j*. *if i* = *n* ∧ *j* = *m then k else if i* = *j* ∧ *i* < *z then 1*::*int else 0*)
⟨*proof*⟩

lemma *P-row-add′-new-ncols*:
assumes *n*: *n*<*a* **and** *m*: *m*<*a* **and** *n-not-m*: *n*≠*m*
shows *ncols* (*P-row-add′-new a n m q*) = *a*
⟨*proof*⟩

lemma *row-add-new-PA*:
assumes *n*:*n*<*nrows* (*A*::*int matrix*)
and *m*:*m*<*nrows A*
and *na*: *nrows A* ≤ *z*
and *n-not-m*: *n*≠*m*
shows *row-add-matrix A n m q* = *P-row-add′-new z n m q* ∗ *A*
⟨*proof*⟩

definition *Q-column-add-new* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *int* ⇒ *int matrix*
where *Q-column-add-new a i j q* = *column-add-matrix* (*one-matrix a*) *i j q*

definition *Q-column-add′-new* :: *nat* ⇒ *nat* ⇒ *nat* ⇒ *int* ⇒ *int matrix*
where *Q-column-add′-new a n m q* = *Abs-matrix*(λ*i j*. *if* (*i*=*m* ∧ *j*=*n*) *then q else* (*if* (*i*=*j* ∧ *j* < *a*) *then 1 else 0*))

lemma *Q-column-add-new-eq*:
assumes *n*:*n* < *a*
and *m*:*m* < *a*
and *n-not-m*: *n*≠*m*
shows *Q-column-add-new a n m q* = *Q-column-add′-new a n m q*
⟨*proof*⟩

lemma *Q-column-add-new-inverse*:
assumes *n*:*n*<*a*
and *m*:*m*<*a*
and *n-not-m*: *n*≠*m*
shows *inverse-matrix* (*Q-column-add′-new a n m q*) (*Q-column-add′-new a n m* (−*q*))
⟨*proof*⟩

lemma *Q-column-add′-new-nrows*:
assumes *n*: *n*<*a* **and** *m*: *m*<*a* **and** *n-not-m*: *n*≠*m*
shows *nrows* (*Q-column-add′-new a n m q*) = *a*
⟨*proof*⟩

lemma *RepAbs-column-add-new*:
assumes *n*:*n*<*z* **and** *m*:*m*<*z*
shows *Rep-matrix* (*Abs-matrix* (λ*i j*. *if i* = *m* ∧ *j* = *n then k else if i* = *j* ∧ *j* < *z then 1*::*int else 0*))
= (λ*i j*. *if i* = *m* ∧ *j* = *n then k else if i* = *j* ∧ *j* < *z then 1*::*int else 0*)
⟨*proof*⟩

lemma *column-add-new-AQ*:
assumes *n*:*n*<*ncols* (*A*::*int matrix*)
and *m*:*m*<*ncols A*
and *na*: *ncols A* ≤ *z*
and *n-not-m*: *n*≠*m*
shows *column-add-matrix A n m q* = *A* ∗ *Q-column-add′-new z n m q*
⟨*proof*⟩

lemma *PQ-move-element*:
assumes *i*:*i*<*nrows* (*A*::*int matrix*) **and** *k*:*k*<*nrows A* **and** *j*:*j*<*ncols A* **and** *l*:*l*<*ncols A*
shows ∃ *P Q*. *is-invertible P* ∧ *is-invertible Q* ∧ *move-element A i j k l* = *P*∗*A*∗*Q*
⟨*proof*⟩

definition *is-square* :: (*int matrix*) ⇒ *nat* ⇒ *bool*
where *is-square A n* = (*nrows A* = *n* ∧ *ncols A* = *n*)

lemma *PQ-move-element′*:
assumes *i*:*i*<*nrows* (*A*::*int matrix*) **and** *k*:*k*<*nrows A* **and** *j*:*j*<*ncols A* **and** *l*:*l*<*ncols A*
shows ∀ *z t*. *z*≥*nrows A* ∧ *t*≥*ncols A*⟶ (∃ *P Q*. *is-invertible P* ∧ *is-invertible Q* ∧ *move-element A i j k l* = *P*∗*A*∗*Q* ∧ *is-square P z* ∧ *is-square Q t*)
⟨*proof*⟩

lemma *snd-minNonzero-less-nrows*:
assumes *minNonzero*: *fst-bis* (*minNonzero-nc A k*) = *True*
shows *snd-bis* (*minNonzero-nc A k*) < *nrows A*
⟨*proof*⟩

lemma *trd-minNonzero-less-ncols*:
assumes *minNonzero*: *fst-bis* (*minNonzero-nc A k*) = *True*
shows *trd-bis* (*minNonzero-nc A k*) < *ncols A*
⟨*proof*⟩

lemma *move-minNonzero-PAQ*:
assumes *minNonzero*: *fst-bis* (*minNonzero-nc A k*) = *True*
shows ∀ *z t*. *z*≥*nrows A* ∧ *t* ≥ *ncols A* ⟶ (∃ *P Q*. *is-invertible P* ∧ *is-invertible Q* ∧ (*move-minNonzero-nc A k*) = *P*∗*A*∗*Q* ∧ *is-square P z* ∧ *is-square Q t*)
⟨*proof*⟩

thm *partRowReduce-matrix-def* **thm** *partRowReduce-infmatrix-def* **thm** *iterate-partRowReduce.simps*
thm *row-add-matrix-def*

lemma *nrows-row-add-matrix-le*:
assumes *i*: *i*<*nrows A*
shows *nrows* (*row-add-matrix A i j k*) ≤ *nrows A*
⟨*proof*⟩

partial-function (*tailrec*) *row-add-matrix-tail-rec′* :: *int matrix* => *nat* => *nat* ⇒ *nat* => *int matrix*
where
row-add-matrix-tail-rec′ A i j k=
(*if nrows A* ≤ *i then A*
else row-add-matrix-tail-rec′
(*row-add-matrix A* (*i* + *1*) *k* (− (((*Rep-matrix A*) (*i* + *1*) *j*) *div*

((*Rep-matrix A*) *k j*))))
(*i* + *1*)
j
k)

lemma *aux-nrows*:
nrows (*row-add-matrix A* (*nrows* (*A*::*int matrix*)) *j* (− (*Rep-matrix A* (*nrows A*) *j div Rep-matrix A i j*))) ≤ *nrows A*
⟨*proof*⟩
find-theorems *nrows ?A* ≤ *?B*
⟨*proof*⟩

lemma *Rep-matrix-row-add-matrix-preserve′*:
assumes *s*≠*i* **shows** *Rep-matrix* (*row-add-matrix A i j q*) *s t* = *Rep-matrix A s t*
⟨*proof*⟩

lemma *row-add-infmatrix-preserve*:
assumes *s*≠*i* **shows** (*row-add-infmatrix A i j q*) *s t* = *A s t*
⟨*proof*⟩

lemma *aux-nrows′*:
nrows (*row-add-matrix A* (*nrows* (*A*::*int matrix*)) *z* (− (*Rep-matrix A* (*nrows A*) *j div Rep-matrix A z j*))) ≤ *nrows A*
⟨*proof*⟩

lemma
row-add-matrix-tail-rec-preserves′′:
assumes *s-le-i*: *s* ≤ *i*
and *z*:*z*≤*i*
shows *Rep-matrix* (*row-add-matrix-tail-rec′* (*A*::*int matrix*) *i j z*) *s t* = (*Rep-matrix A*) *s t*
⟨*proof*⟩

lemma *row-add-matrix-tail-rec-eq-2′′*:
assumes *nr-le-k*: *nrows* (*A*::*int matrix*) ≤ *k*
and *z*: *z*≤*i*
shows *Rep-matrix* (*row-add-matrix-tail-rec′ A i j z*) *k l* = *0*
⟨*proof*⟩

lemma *row-add-matrix-tail-rec-eq′*:
assumes *i-l-k*: *i*<*k*
and *z-le-i*: *z*≤*i*

shows *Rep-matrix* (*row-add-matrix-tail-rec′ A i j z*) *k l* =
row-add-infmatrix (*Rep-matrix A*) *k z* (− (*Rep-matrix A k j div Rep-matrix A z j*)) *k l*
⟨*proof*⟩

lemma
row-add-matrix-tail-rec-preserves′:
assumes *s-le-i*: $s \leq i$
shows *Rep-matrix* (*row-add-matrix-tail-rec′* (*A*::*int matrix*) *i j i*) *s t* = (*Rep-matrix A*) *s t*
⟨*proof*⟩

lemma *Rep-matrix-row-add-matrix-tail-rec′-eq-partRowReduce-matrix*:
shows *Rep-matrix* (*row-add-matrix-tail-rec′ A i j i*) = *Rep-matrix* (*partRowReduce-matrix A i j*)
⟨*proof*⟩

corollary *row-add-matrix-tail-rec′-eq-partRowReduce-matrix*:
shows *row-add-matrix-tail-rec′ A i j i* = *partRowReduce-matrix A i j*
⟨*proof*⟩

lemma *row-add-new-PA′*:
assumes *n*:*n*<*nrows* (*A*::*int matrix*)
and *m*:*m*<*nrows A*
and *na*: *nrows A* $\leq$ *z*
and *n-not-m*: $n \neq m$
shows ∃ *P*. *is-invertible P* ∧ *row-add-matrix A n m q* = *P* ∗ *A* ∧ *is-square P z*
⟨*proof*⟩

lemma *row-add-matrix-tail-rec′-PA*:
assumes *z*:*z*≥*nrows A*
and *not-zero*: *Rep-matrix A k k* ≠ *0*
and *i*: *k*≤*i*
shows (∃ *P*. *is-invertible P* ∧ (*row-add-matrix-tail-rec′ A i k k*) = *P*∗*A* ∧ *is-square P z*)
⟨*proof*⟩

lemma *partRowReduce-matrix-PA*:
assumes *z*:*z*≥*nrows A*
and *not-zero*: *Rep-matrix A k k* ≠ *0*
shows (∃ *P*. *is-invertible P* ∧ (*partRowReduce-matrix A k k*) = *P*∗*A* ∧ *is-square P z*)

⟨*proof*⟩

lemma *PQ-partRowReduce-move-minNonzero*:
assumes *z*: *z*≥*nrows A*
and *t*: *t* ≥ *ncols A*
and *not-all-zero*: *fst-bis* (*minNonzero-nc A k*) = *True*
shows ∃ *P Q*. *is-invertible P* ∧ *is-invertible Q* ∧ (*partRowReduce-matrix*(*move-minNonzero-nc A k*) *k k*) = *P* ∗ *A* ∗ *Q* ∧ *is-square P z* ∧ *is-square Q t*
⟨*proof*⟩

lemma *PQ-iterate-partRowReduce*:
assumes *z*: *z*≥*nrows A*
and *t*: *t*≥*ncols A*
and *k*: *k*<*nrows A*
and *k2*: *k*<*ncols A*
shows ∃ *P Q*. *is-invertible P* ∧ *is-invertible Q* ∧ *iterate-partRowReduce A k* = *P*∗*A*∗*Q* ∧ *is-square P z* ∧ *is-square Q t*
⟨*proof*⟩

lemma *ncols-column-add-matrix-le*:
assumes *i*: *i*<*ncols A*
shows *ncols* (*column-add-matrix A i j k*) ≤ *ncols A*
⟨*proof*⟩

partial-function (*tailrec*) *column-add-matrix-tail-rec′* :: *int matrix* => *nat* => *nat* ⇒ *nat* => *int matrix*
where
column-add-matrix-tail-rec′ A iterador i j=
(*if ncols A* ≤ (*iterador*) *then A*
else column-add-matrix-tail-rec′
(*column-add-matrix A* (*iterador* + *1*) *j* (− (((*Rep-matrix A*) *i* (*iterador* + *1*)) *div* ((*Rep-matrix A*) *i j*))))
(*iterador* + *1*)
i
j)

lemma *aux-ncols*:
ncols (*column-add-matrix A* (*ncols* (*A*::*int matrix*)) *j* (− (*Rep-matrix A i* (*ncols A*) *div Rep-matrix A i j*))) ≤ *ncols A*
⟨*proof*⟩

lemma *Rep-matrix-column-add-matrix-preserve′*:

assumes $t \neq i$ **shows** *Rep-matrix* (*column-add-matrix A i j q*) *s t* = *Rep-matrix A s t*
⟨*proof*⟩

lemma *column-add-infmatrix-preserve*:
assumes $t \neq i$ **shows** (*column-add-infmatrix A i j q*) *s t* = *A s t*
⟨*proof*⟩

lemma *aux-ncols'*:
ncols (*column-add-matrix A* (*ncols A*) *z* (− (*Rep-matrix A j* (*ncols A*) *div Rep-matrix A j z*))) ≤ *ncols* (*A*::*int matrix*)
⟨*proof*⟩

lemma
column-add-matrix-tail-rec-preserves'':
assumes *s-le-i*: $t \leq i$
and *z*:$z \leq i$
shows *Rep-matrix* (*column-add-matrix-tail-rec'* (*A*::*int matrix*) *i j z*) *s t* = (*Rep-matrix A*) *s t*
⟨*proof*⟩

lemma *column-add-matrix-tail-rec-eq-2''*:
assumes *nr-le-k*: *ncols* (*A*::*int matrix*) ≤ *l*
and *z*: $z \leq i$
shows *Rep-matrix* (*column-add-matrix-tail-rec' A i j z*) *k l* = *0*
⟨*proof*⟩

lemma *column-add-matrix-tail-rec-eq'*:
assumes *i-l-k*: $i < l$
and *z-le-i*: $z \leq i$
shows *Rep-matrix* (*column-add-matrix-tail-rec' A i j z*) *k l* =
column-add-infmatrix (*Rep-matrix A*) *l z* (− (*Rep-matrix A j l div Rep-matrix A j z*)) *k l*
⟨*proof*⟩

lemma
column-add-matrix-tail-rec-preserves':
assumes *s-le-i*: $t \leq i$
shows *Rep-matrix* (*column-add-matrix-tail-rec'* (*A*::*int matrix*) *i j i*) *s t* = (*Rep-matrix A*) *s t*
⟨*proof*⟩

lemma *Rep-matrix-column-add-matrix-tail-rec′-eq-partColumnReduce-matrix*:
shows *Rep-matrix* (*column-add-matrix-tail-rec′ A i j i*) = *Rep-matrix* (*partColumnReduce-matrix A j i*)
⟨*proof*⟩

corollary *column-add-matrix-tail-rec′-eq-partColumnReduce-matrix*:
shows *column-add-matrix-tail-rec′ A j i j* = *partColumnReduce-matrix A i j*
⟨*proof*⟩

lemma *column-add-new-AQ′*:
assumes *n*:*n*<*ncols* (*A*::*int matrix*)
and *m*:*m*<*ncols A*
and *na*: *ncols A* ≤ *z*
and *n-not-m*: *n*≠*m*
shows ∃ *Q*. *is-invertible Q* ∧ *column-add-matrix A n m k* = *A*∗*Q* ∧ *is-square Q z*
⟨*proof*⟩

lemma *column-add-matrix-tail-rec′-AQ*:
assumes *z*:*z*≥*ncols A*
and *not-zero*: *Rep-matrix A k k* ≠ *0*
and *i*: *k*≤*i*
shows (∃ *Q*. *is-invertible Q* ∧ (*column-add-matrix-tail-rec′ A i k k*) = *A*∗*Q* ∧ *is-square Q z*)
⟨*proof*⟩

lemma *Q-partColumnReduce*:
assumes *t*≥*ncols A*
and *Rep-matrix A k k* ≠ *0*
shows (∃ *Q*. *is-invertible Q* ∧ (*partColumnReduce-matrix A k k*) = *A*∗*Q* ∧ *is-square Q t*)
⟨*proof*⟩

lemma *nrows-move-minNonzero-nc*:
assumes *not-all-zero*: *fst-bis* (*minNonzero-nc A k*) = *True*
shows *nrows* (*move-minNonzero-nc A k*) ≤ *nrows A*
⟨*proof*⟩

lemma *ncols-move-minNonzero-nc*:
assumes *not-all-zero*: *fst-bis* (*minNonzero-nc A k*) = *True*
shows *ncols* (*move-minNonzero-nc A k*) ≤ *ncols A*
⟨*proof*⟩

lemma *PQ-partColumn-iterate-move*:

assumes z:$z \geq$*nrows A*
and t:$t \geq$*ncols A*
and *not-all-zero*: *fst-bis* (*minNonzero-nc A k*) = *True*
shows $\exists$ *P Q*. *is-invertible P* $\wedge$ *is-invertible Q* $\wedge$ (*partColumnReduce-matrix*(*iterate-partRowReduce*(*move-min*
A k) *k*) *k k*)
= *P* $*$ *A* $*$ *Q* $\wedge$ *is-square P z* $\wedge$ *is-square Q t*
⟨*proof*⟩

lemma *PQ-Diagonalize*:
assumes z: $z \geq$*nrows A*
and t: $t \geq$*ncols A*

shows $\exists$ *P Q*. *is-invertible P* $\wedge$ *is-invertible Q* $\wedge$ *Diagonalize A k* = *P*$*$*A*$*$*Q* $\wedge$ *is-square P z* $\wedge$ *is-square Q t*
⟨*proof*⟩

lemma *PQ-Diagonalize-up-to-k-not-null*:
assumes z:$z \geq$*nrows A*
and t:$t \geq$*ncols A*
and *nr*:*nrows A* > *0*
and *nc*: *ncols A* > *0* — Realmente me sobra, a partir de que nrows A ¿ 0 ya puedo sacar que ncols A ¿ 0
shows $\exists$ *P Q*. *is-invertible P* $\wedge$ *is-invertible Q* $\wedge$ *Diagonalize-up-to-k* (*A*::*int matrix*) *n* = *P*$*$*A*$*$*Q* $\wedge$ *is-square P z* $\wedge$ *is-square Q t*
⟨*proof*⟩
thm *Diagonalize.simps* **thm** *PQ-Diagonalize*

⟨*proof*⟩

lemma *Diagonalize-theorem*:
shows $\exists$ *P Q B*. *is-invertible P* $\wedge$ *is-invertible Q* $\wedge$ *B* = *P*$*$*A*$*$*Q* $\wedge$ *is-square P* (*nrows* (*A*::*int matrix*)) $\wedge$ *is-square Q* (*ncols A*) $\wedge$ *Diagonalize-p B* (*max* (*nrows A*) (*ncols A*))
⟨*proof*⟩

end