

The Cayley-Hamilton theorem

Stephan Adelsberger Stefan Hetzl Florian Pollak

June 17, 2024

Abstract

This document contains a proof of the Cayley-Hamilton theorem based on the development of matrices in `HOL/Multivariate_Analysis`.

Contents

1	Introduction	1
----------	---------------------	----------

1 Introduction

The Cayley-Hamilton theorem states that every square matrix is a zero of its own characteristic polynomial, in symbols: $\chi_A(A) = 0$. It is a central theorem of linear algebra and plays an important role for matrix normal form theory.

In this document we work with matrices over a commutative ring R and give a direct algebraic proof of the theorem. The starting point of the proof is the following fundamental property of the adjugate matrix

$$\text{adj}(B) \cdot B = B \cdot \text{adj}(B) = \det(B)I_n \tag{1}$$

where I_n denotes the $n \times n$ -identity matrix and $\det(B)$ the determinant of B . Recall that the characteristic polynomial is defined as $\chi_A(X) = \det(XI_n - A)$, i.e. as the determinant of a matrix whose entries are polynomials. Considering the adjugate of this matrix we obtain

$$(XI_n - A) \cdot \text{adj}(XI_n - A) = \chi_A(X)I_n \tag{2}$$

directly from (1). Now, $\text{adj}(XI_n - A)$ being a matrix of polynomials of degree at most $n - 1$ can be written as

$$\text{adj}(XI_n - A) = \sum_{i=0}^{n-1} X^i B_i \text{ for } B_i \in R^{n \times n}. \tag{3}$$

A straightforward calculation starting from (2) using (3) then shows that

$$\chi_A(X)I_n = X^n B_{n-1} + \sum_{i=1}^{n-1} X^i (B_{i-1} - A \cdot B_i) - A \cdot B_0. \quad (4)$$

Now let c_i be the coefficient of X^i in $\chi_A(X)$. Then equating the coefficients in (4) yields

$$\begin{aligned} B_{n-1} &= I_n, \\ B_{i-1} - A \cdot B_i &= c_i I_n \text{ for } 1 \leq i \leq n-1, \text{ and} \\ -A \cdot B_0 &= c_0 I_n. \end{aligned}$$

Multiplying the i -th equation with A^i from the left gives

$$\begin{aligned} A^n \cdot B_{n-1} &= A^n, \\ A^i \cdot B_{i-1} - A^{i+1} \cdot B_i &= c_i A^i \text{ for } 1 \leq i \leq n-1, \text{ and} \\ -A \cdot B_0 &= c_0 I_n \end{aligned}$$

which shows that

$$\chi_A(A)I_n = A^n + c_{n-1}A^{n-1} + \cdots + c_1 A + c_0 I_n = 0$$

and hence $\chi_A(A) = 0$ which finishes this proof sketch.

There are numerous other proofs of the Cayley-Hamilton theorem, in particular the one formalized in Coq by Sidi Ould Biha [1, 2]. This proof also starts with the fundamental property of the adjugate matrix but instead of the above calculation relies on the existence of a ring isomorphism between $\mathcal{M}_n(R[X])$, the matrices of polynomials over R , and $(\mathcal{M}_n(R))[X]$, the polynomials whose coefficients are matrices over R . On the upside, this permits a briefer and more abstract argument (once the background theory contains all prerequisites) but on the downside one has to deal with the mathematically subtle evaluation of polynomials over the non-commutative(!) ring $\mathcal{M}_n(R)$. As described nicely in [2] this evaluation is no longer a ring homomorphism. However, its use in the proof of the Cayley-Hamilton theorem is sufficiently restricted so that one can work around this problem.

Sections ??, ??, and ?? contain basic results about matrices and polynomials which are needed for the proof of the Cayley-Hamilton theorem in addition to the results which are available in the library. Section ?? contains basic results about matrices of polynomials, including the definition of the characteristic polynomial and proofs of some of its basic properties. Finally, Section ?? contains the proof of the Cayley-Hamilton theorem as outlined above.

theory *Square-Matrix*

```

imports
  HOL-Analysis.Determinants
  HOL-Analysis.Cartesian-Euclidean-Space
begin

lemma smult-axis:  $x * s \text{ axis } i \ y = \text{axis } i \ (x * y::\text{mult-zero})$ 
  by (simp add: axis-def vec-eq-iff)

typedef ('a, 'n) sq-matrix = UNIV :: ('n  $\Rightarrow$  'n  $\Rightarrow$  'a) set
  morphisms to-fun of-fun
  by (rule UNIV-witness)

syntax -sq-matrix :: type  $\Rightarrow$  type  $\Rightarrow$  type ((-  $\rightsquigarrow$  / -) [15, 16] 15)

parse-translation <
  let
    fun vec t u = Syntax.const @{type-syntax sq-matrix} $ t $ u;
    fun sq-matrix-tr [t, u] =
      (case Term-Position.strip-positions u of
        v as Free (x, -) =>
          if Lexicon.is-tid x then
            vec t (Syntax.const @{syntax-const -ofsort} $ v $
              Syntax.const @{class-syntax finite})
          else vec t u
        | - => vec t u)
  in
    [(@{syntax-const -sq-matrix}, K sq-matrix-tr)]
  end
  >

setup-lifting type-definition-sq-matrix

lift-definition map-sq-matrix :: ('a  $\Rightarrow$  'c)  $\Rightarrow$  'a  $\rightsquigarrow$  'b  $\Rightarrow$  'c  $\rightsquigarrow$  'b is
   $\lambda f \ M \ i \ j. f \ (M \ i \ j) .$ 

lift-definition from-vec :: 'a  $\rightsquigarrow$  'n  $\rightsquigarrow$  'n  $\Rightarrow$  'a  $\rightsquigarrow$  'n is
   $\lambda M \ i \ j. M \ \$ \ i \ \$ \ j .$ 

lift-definition to-vec :: 'a  $\rightsquigarrow$  'n  $\Rightarrow$  'a  $\rightsquigarrow$  'n  $\rightsquigarrow$  'n is
   $\lambda M. \chi \ i \ j. M \ i \ j .$ 

lemma from-vec-eq-iff: from-vec M = from-vec N  $\longleftrightarrow$  M = N
  by transfer (auto simp: vec-eq-iff fun-eq-iff)

lemma to-vec-from-vec[simp]: to-vec (from-vec M) = M
  by transfer (simp add: vec-eq-iff)

lemma from-vec-to-vec[simp]: from-vec (to-vec M) = M
  by transfer (simp add: vec-eq-iff fun-eq-iff)

```

lemma *map-sq-matrix-compose*[simp]: $\text{map-sq-matrix } f (\text{map-sq-matrix } g M) = \text{map-sq-matrix } (\lambda x. f (g x)) M$
by *transfer simp*

lemma *map-sq-matrix-ident*[simp]: $\text{map-sq-matrix } (\lambda x. x) M = M$
by *transfer (simp add: vec-eq-iff)*

lemma *map-sq-matrix-cong*:
 $M = N \implies (\bigwedge i j. f (\text{to-fun } N i j) = g (\text{to-fun } N i j)) \implies \text{map-sq-matrix } f M = \text{map-sq-matrix } g N$
by *transfer simp*

lift-definition *diag* :: 'a::zero \Rightarrow 'aⁿ is
 $\lambda k i j. \text{if } i = j \text{ then } k \text{ else } 0$.

lemma *diag-eq-iff*: $\text{diag } x = \text{diag } y \longleftrightarrow x = y$
by *transfer (simp add: fun-eq-iff)*

lemma *map-sq-matrix-diag*[simp]: $f 0 = 0 \implies \text{map-sq-matrix } f (\text{diag } c) = \text{diag } (f c)$
by *transfer (simp add: fun-eq-iff)*

lift-definition *smult-sq-matrix* :: 'a::times \Rightarrow 'aⁿ \Rightarrow 'aⁿ (**infixr** *_S 75) is
 $\lambda c M i j. c * M i j$.

lemma *smult-map-sq-matrix*:
 $(\bigwedge y. f (x * y) = z * f y) \implies \text{map-sq-matrix } f (x *_S A) = z *_S \text{map-sq-matrix } f A$
by *transfer simp*

lemma *map-sq-matrix-smult*: $c *_S \text{map-sq-matrix } f A = \text{map-sq-matrix } (\lambda x. c * f x) A$
by *transfer simp*

lemma *one-smult*[simp]: $(1::\text{monoid-mult}) *_S x = x$
by *transfer simp*

lemma *smult-diag*: $x *_S \text{diag } y = \text{diag } (x * y::\text{mult-zero})$
by *transfer (simp add: fun-eq-iff)*

instantiation *sq-matrix* :: (semigroup-add, finite) semigroup-add
begin

lift-definition *plus-sq-matrix* :: 'aⁿ \Rightarrow 'aⁿ \Rightarrow 'aⁿ is
 $\lambda A B i j. A i j + B i j$.

instance
by *standard (transfer, simp add: field-simps)*

end

lemma *map-sq-matrix-add*:

$(\bigwedge a b. f (a + b) = f a + f b) \implies \text{map-sq-matrix } f (A + B) = \text{map-sq-matrix } f A + \text{map-sq-matrix } f B$
by *transfer simp*

lemma *add-map-sq-matrix*: $\text{map-sq-matrix } f A + \text{map-sq-matrix } g A = \text{map-sq-matrix } (\lambda x. f x + g x) A$

by *transfer simp*

instantiation *sq-matrix* :: (*monoid-add*, *finite*) *monoid-add*
begin

lift-definition *zero-sq-matrix* :: '*a*'[~]'*b*' is $\lambda i j. 0$.

instance

by *standard (transfer, simp)+*

end

lemma *diag-0*: $\text{diag } 0 = 0$

by *transfer simp*

lemma *diag-0-eq*: $\text{diag } x = 0 \iff x = 0$

by *transfer (simp add: fun-eq-iff)*

lemma *zero-map-sq-matrix*: $f 0 = 0 \implies \text{map-sq-matrix } f 0 = 0$

by *transfer simp*

lemma *map-sq-matrix-0[simp]*: $\text{map-sq-matrix } (\lambda x. 0) A = 0$

by *transfer simp*

instance *sq-matrix* :: (*ab-semigroup-add*, *finite*) *ab-semigroup-add*

by *standard (transfer, simp add: field-simps)+*

instantiation *sq-matrix* :: (*minus*, *finite*) *minus*

begin

lift-definition *minus-sq-matrix* :: '*a*'[~]'*b*' \Rightarrow '*a*'[~]'*b*' \Rightarrow '*a*'[~]'*b*' is

$\lambda A B i j. A i j - B i j$.

instance ..

end

instantiation *sq-matrix* :: (*group-add*, *finite*) *group-add*

begin

lift-definition *uminus-sq-matrix* :: ' $a \rightsquigarrow b \Rightarrow 'a \rightsquigarrow b$ ' is
uminus .

instance
 by *standard* (*transfer*, *simp*)+

end

lemma *map-sq-matrix-diff*:
 $(\bigwedge a b. f (a - b) = f a - f b) \Longrightarrow \text{map-sq-matrix } f (A - B) = \text{map-sq-matrix } f A - \text{map-sq-matrix } f B$
 by *transfer* (*simp add: vec-eq-iff*)

lemma *smult-diff*: **fixes** $a :: 'a::\text{comm-ring-1}$ **shows** $a *_S (A - B) = a *_S A - a *_S B$
 by *transfer* (*simp add: field-simps*)

instance *sq-matrix* :: (*cancel-semigroup-add*, *finite*) *cancel-semigroup-add*
 by (*standard*; *transfer*, *simp add: field-simps fun-eq-iff*)

instance *sq-matrix* :: (*cancel-ab-semigroup-add*, *finite*) *cancel-ab-semigroup-add*
 by (*standard*; *transfer*, *simp add: field-simps*)

instance *sq-matrix* :: (*comm-monoid-add*, *finite*) *comm-monoid-add*
 by (*standard*; *transfer*, *simp add: field-simps*)

lemma *map-sq-matrix-sum*:
 $f 0 = 0 \Longrightarrow (\bigwedge a b. f (a + b) = f a + f b) \Longrightarrow$
 $\text{map-sq-matrix } f (\sum i \in I. A i) = (\sum i \in I. \text{map-sq-matrix } f (A i))$
 by (*induction I rule: infinite-finite-induct*)
 (*auto simp: zero-map-sq-matrix map-sq-matrix-add*)

lemma *sum-map-sq-matrix*: $(\sum i \in I. \text{map-sq-matrix } (f i) A) = \text{map-sq-matrix } (\lambda x. \sum i \in I. f i x) A$
 by (*induction I rule: infinite-finite-induct*) (*simp-all add: add-map-sq-matrix*)

lemma *smult-zero[simp]*: **fixes** $a :: 'a::\text{ring-1}$ **shows** $a *_S 0 = 0$
 by *transfer* (*simp add: vec-eq-iff*)

lemma *smult-right-add*: **fixes** $a :: 'a::\text{ring-1}$ **shows** $a *_S (x + y) = a *_S x + a *_S y$
 by *transfer* (*simp add: vec-eq-iff field-simps*)

lemma *smult-sum*: **fixes** $a :: 'a::\text{ring-1}$ **shows** $(\sum i \in I. a *_S f i) = a *_S (\text{sum } f I)$
 by (*induction I rule: infinite-finite-induct*)
 (*simp-all add: smult-right-add vec-eq-iff*)

instance *sq-matrix* :: (*ab-group-add*, *finite*) *ab-group-add*
 by *standard* (*transfer*, *simp add: field-simps*)+

instantiation *sq-matrix* :: (semiring-0, finite) semiring-0
begin

lift-definition *times-sq-matrix* :: 'a[~]b ⇒ 'a[~]'b ⇒ 'a[~]b **is**
 $\lambda M N i j. \sum_{k \in UNIV}. M i k * N k j$.

instance

proof

fix *a b c* :: 'a[~]'b **show** $a * b * c = a * (b * c)$
by *transfer*
(auto simp: fun-eq-iff sum-distrib-left sum-distrib-right field-simps intro:
sum.swap)
qed (transfer, simp add: vec-eq-iff sum.distrib field-simps)+
end

lemma *diag-mult*: $diag\ x * A = x *_S A$
by *transfer* (simp add: if-distrib[**where** $f = \lambda x. x * a$ **for** *a*] sum.If-cases)

lemma *mult-diag*:

fixes *x* :: 'a::comm-ring-1
shows $A * diag\ x = x *_S A$
by *transfer* (simp add: if-distrib[**where** $f = \lambda x. a * x$ **for** *a*] mult.commute sum.If-cases)

lemma *smult-mult1*: **fixes** *a* :: 'a::comm-ring-1 **shows** $a *_S (A * B) = (a *_S A) * B$
by *transfer* (simp add: sum-distrib-left field-simps)

lemma *smult-mult2*: **fixes** *a* :: 'a::comm-ring-1 **shows** $a *_S (A * B) = A * (a *_S B)$
by *transfer* (simp add: sum-distrib-left field-simps)

lemma *map-sq-matrix-mult*:

fixes *f* :: 'a::semiring-1 ⇒ 'b::semiring-1
assumes $f: \bigwedge a\ b. f\ (a + b) = f\ a + f\ b \bigwedge a\ b. f\ (a * b) = f\ a * f\ b\ f\ 0 = 0$
shows $map\text{-sq-matrix}\ f\ (A * B) = map\text{-sq-matrix}\ f\ A * map\text{-sq-matrix}\ f\ B$
proof (transfer fixing: *f*)
fix *A B* :: 'c ⇒ 'c ⇒ 'a
{ **fix** *I i j* **have** $f\ (\sum_{k \in I}. A\ i\ k * B\ k\ j) = (\sum_{k \in I}. f\ (A\ i\ k) * f\ (B\ k\ j))$
by (induction *I* rule: infinite-finite-induct) (auto simp add: *f*) }
then show $(\lambda i\ j. f\ (\sum_{k \in UNIV}. A\ i\ k * B\ k\ j)) = (\lambda i\ j. \sum_{k \in UNIV}. f\ (A\ i\ k) * f\ (B\ k\ j))$
by *simp*
qed

lemma *from-vec-mult[simp]*: $from\text{-vec}\ (M ** N) = from\text{-vec}\ M * from\text{-vec}\ N$
by *transfer* (simp add: matrix-matrix-mult-def fun-eq-iff vec-eq-iff)

instantiation *sq-matrix* :: (semiring-1, finite) semiring-1

begin

lift-definition *one-sq-matrix* :: 'a[~]b is

$\lambda i j. \text{if } i = j \text{ then } 1 \text{ else } 0 .$

instance

by *standard* (*transfer*, *simp add: fun-eq-iff sum.If-cases*

*if-distrib[where f= $\lambda x. x * b$ for b]* *if-distrib[where f= $\lambda x. b * x$ for b]*)+

end

instance *sq-matrix* :: (*semiring-1*, *finite*) *numeral* ..

lemma *diag-1*: *diag 1 = 1*

by *transfer simp*

lemma *diag-1-eq*: *diag x = 1 \longleftrightarrow x = 1*

by *transfer (simp add: fun-eq-iff)*

instance *sq-matrix* :: (*ring-1*, *finite*) *ring-1*

by *standard simp-all*

interpretation *sq-matrix*: *vector-space smult-sq-matrix*

by *standard* (*transfer*, *simp add: vec-eq-iff field-simps*)+

instantiation *sq-matrix* :: (*real-vector*, *finite*) *real-vector*

begin

lift-definition *scaleR-sq-matrix* :: *real* \Rightarrow 'a[~]b \Rightarrow 'a[~]b is

$\lambda r A i j. r *_{\mathbb{R}} A i j .$

instance

by *standard* (*transfer*, *simp add: scaleR-add-right scaleR-add-left*)+

end

instance *sq-matrix* :: (*semiring-1*, *finite*) *Rings.dvd* ..

lift-definition *transpose* :: 'a[~]n \Rightarrow 'a[~]n is

$\lambda M i j. M j i .$

lemma *transpose-transpose[simp]*: *transpose (transpose A) = A*

by *transfer simp*

lemma *transpose-diag[simp]*: *transpose (diag c) = diag c*

by *transfer (simp add: fun-eq-iff)*

lemma *transpose-zero[simp]*: *transpose 0 = 0*

by *transfer simp*

lemma *transpose-one*[simp]: $\text{transpose } 1 = 1$
by *transfer* (*simp add: fun-eq-iff*)

lemma *transpose-add*[simp]: $\text{transpose } (A + B) = \text{transpose } A + \text{transpose } B$
by *transfer simp*

lemma *transpose-minus*[simp]: $\text{transpose } (A - B) = \text{transpose } A - \text{transpose } B$
by *transfer simp*

lemma *transpose-uminus*[simp]: $\text{transpose } (- A) = - \text{transpose } A$
by *transfer* (*simp add: fun-eq-iff*)

lemma *transpose-mult*[simp]:
 $\text{transpose } (A * B :: 'a::\text{comm-semiring-0}^{\sim}n) = \text{transpose } B * \text{transpose } A$
by *transfer* (*simp add: field-simps*)

lift-definition *trace* :: $'a::\text{comm-monoid-add}^{\sim}n \Rightarrow 'a$ **is**
 $\lambda M. \sum_{i \in \text{UNIV}. M \ i \ i .$

lemma *trace-diag*[simp]: $\text{trace } (\text{diag } c :: 'a::\text{semiring-1}^{\sim}n) = \text{of-nat } \text{CARD}(n) * c$
by *transfer simp*

lemma *trace-0*[simp]: $\text{trace } 0 = 0$
by *transfer simp*

lemma *trace-1*[simp]: $\text{trace } (1 :: 'a::\text{semiring-1}^{\sim}n) = \text{of-nat } \text{CARD}(n)$
by *transfer simp*

lemma *trace-plus*[simp]: $\text{trace } (A + B) = \text{trace } A + \text{trace } B$
by *transfer* (*simp add: sum.distrib*)

lemma *trace-minus*[simp]: $\text{trace } (A - B) = (\text{trace } A - \text{trace } B :: -::\text{ab-group-add})$
by *transfer* (*simp add: sum-subtractf*)

lemma *trace-uminus*[simp]: $\text{trace } (- A) = - (\text{trace } A :: -::\text{ab-group-add})$
by *transfer* (*simp add: sum-negf*)

lemma *trace-smult*[simp]: $\text{trace } (s *_{\mathcal{S}} A) = (s * \text{trace } A :: -::\text{semiring-0})$
by *transfer* (*simp add: sum-distrib-left*)

lemma *trace-transpose*[simp]: $\text{trace } (\text{transpose } A) = \text{trace } A$
by *transfer simp*

lemma *trace-mult-symm*:
fixes $A \ B :: 'a::\text{comm-semiring-0}^{\sim}n$
shows $\text{trace } (A * B) = \text{trace } (B * A)$
by *transfer* (*auto intro: sum.swap simp: mult commute*)

lift-definition $\det :: 'a::comm-ring-1 \rightsquigarrow 'n \Rightarrow 'a$ **is**
 $\lambda A. (\sum p | p \text{ permutes } UNIV. \text{ of-int } (sign\ p) * (\prod i \in UNIV. A\ i\ (p\ i))) .$

lemma $\det\text{-eq}$: $\det A = (\sum p | p \text{ permutes } UNIV. \text{ of-int } (sign\ p) * (\prod i \in UNIV. \text{ to-fun } A\ i\ (p\ i)))$
by *transfer rule*

lemma $\text{permutes-UNIV-permutation}$: $\text{permutation } p \longleftrightarrow p \text{ permutes } (UNIV:::\text{finite})$
by *(auto simp: permutation-permutes permutes-def)*

lemma $\det\text{-0[simp]}$: $\det\ 0 = 0$
by *transfer (simp add: zero-power)*

lemma $\det\text{-transpose}$: $\det (\text{transpose } A) = \det A$
apply *transfer*
apply *(subst sum-permutations-inverse)*
apply *(rule sum.cong[OF refl])*
apply *(simp add: sign-inverse permutes-UNIV-permutation)*
apply *(subst prod.reindex-bij-betw[symmetric])*
apply *(rule permutes-imp-bij)*
apply *assumption*
apply *(simp add: permutes-inverses)*
done

lemma $\det\text{-diagonal}$:
fixes $A :: 'a::comm-ring-1 \rightsquigarrow 'n$
shows $(\bigwedge i\ j. i \neq j \Longrightarrow \text{ to-fun } A\ i\ j = 0) \Longrightarrow \det A = (\prod i \in UNIV. \text{ to-fun } A\ i\ i)$
proof *transfer*
fix $A :: 'n \Rightarrow 'n \Rightarrow 'a$ **assume** $\text{ neq: } \bigwedge i\ j. i \neq j \Longrightarrow A\ i\ j = 0$
let $?pp = \lambda p. \text{ of-int } (sign\ p) * (\prod i \in UNIV. A\ i\ (p\ i))$

{ **fix** $p :: 'n \Rightarrow 'n$ **assume** $p: p \text{ permutes } UNIV\ p \neq id$
then obtain i **where** $i: i \neq p\ i$
unfolding $id\text{-def}$ **by** *metis*
with $\text{ neq}[OF\ i]$ **have** $(\prod i \in UNIV. A\ i\ (p\ i)) = 0$
by *(intro prod-zero) auto* **}**

then have $(\sum p | p \text{ permutes } UNIV. ?pp\ p) = (\sum p \in \{id\}. ?pp\ p)$
by *(intro sum.mono-neutral-cong-right) (auto intro: permutes-id)*
then show $(\sum p | p \text{ permutes } UNIV. ?pp\ p) = (\prod i \in UNIV. A\ i\ i)$
by *(simp add: sign-id)*

qed

lemma $\det\text{-1[simp]}$: $\det (1::'a::comm-ring-1 \rightsquigarrow 'n) = 1$
by *(subst det-diagonal) (transfer, simp)+*

lemma $\det\text{-lowerdiagonal}$:
fixes $A :: 'a::comm-ring-1 \rightsquigarrow 'n::\{\text{finite, wellorder}\}$
shows $(\bigwedge i\ j. i < j \Longrightarrow \text{ to-fun } A\ i\ j = 0) \Longrightarrow \det A = (\prod i \in UNIV. \text{ to-fun } A\ i\ i)$
proof *transfer*

fix $A :: 'n \Rightarrow 'n \Rightarrow 'a$ **assume** $ld: \bigwedge i j. i < j \implies A i j = 0$
let $?pp = \lambda p. \text{of-int } (\text{sign } p) * (\prod_{i \in UNIV}. A i (p i))$

{ **fix** $p :: 'n \Rightarrow 'n$ **assume** $p: p \text{ permutes } UNIV \ p \neq id$
with $\text{permutes-natset-le}[OF\ p(1)]$ **obtain** i **where** $i: p\ i > i$
by (metis not-le)
with $ld[OF\ i]$ **have** $(\prod_{i \in UNIV}. A i (p i)) = 0$
by $(\text{intro prod-zero})\ auto\ }$
then have $(\sum p \mid p \text{ permutes } UNIV. ?pp\ p) = (\sum p \in \{id\}. ?pp\ p)$
by $(\text{intro sum.mono-neutral-cong-right})\ (auto\ \text{intro: permutes-id})$
then show $(\sum p \mid p \text{ permutes } UNIV. ?pp\ p) = (\prod_{i \in UNIV}. A i i)$
by $(\text{simp add: sign-id})$
qed

lemma *det-upperdiagonal*:

fixes $A :: 'a::\text{comm-ring-1} \rightsquigarrow 'n::\{\text{finite, wellorder}\}$
shows $(\bigwedge i j. j < i \implies \text{to-fun } A\ i\ j = 0) \implies \text{det } A = (\prod_{i \in UNIV}. \text{to-fun } A\ i\ i)$
using $\text{det-lowerdiagonal}[of\ \text{transpose } A]$
unfolding $\text{det-transpose transpose.rep-eq}$.

lift-definition *perm-rows* $:: 'a \rightsquigarrow 'b \Rightarrow ('b \Rightarrow 'b) \Rightarrow 'a \rightsquigarrow 'b$ **is**
 $\lambda M\ p\ i\ j. M\ (p\ i)\ j$.

lift-definition *perm-cols* $:: 'a \rightsquigarrow 'b \Rightarrow ('b \Rightarrow 'b) \Rightarrow 'a \rightsquigarrow 'b$ **is**
 $\lambda M\ p\ i\ j. M\ i\ (p\ j)$.

lift-definition *upd-rows* $:: 'a \rightsquigarrow 'b \Rightarrow 'b\ \text{set} \Rightarrow ('b \Rightarrow 'a \rightsquigarrow 'b) \Rightarrow 'a \rightsquigarrow 'b$ **is**
 $\lambda M\ S\ v\ i\ j. \text{if } i \in S \text{ then } v\ i\ \$\ j \text{ else } M\ i\ j$.

lift-definition *upd-cols* $:: 'a \rightsquigarrow 'b \Rightarrow 'b\ \text{set} \Rightarrow ('b \Rightarrow 'a \rightsquigarrow 'b) \Rightarrow 'a \rightsquigarrow 'b$ **is**
 $\lambda M\ S\ v\ i\ j. \text{if } j \in S \text{ then } v\ j\ \$\ i \text{ else } M\ i\ j$.

lift-definition *upd-row* $:: 'a \rightsquigarrow 'b \Rightarrow 'b \Rightarrow 'a \rightsquigarrow 'b \Rightarrow 'a \rightsquigarrow 'b$ **is**
 $\lambda M\ i'\ v\ i\ j. \text{if } i = i' \text{ then } v\ \$\ j \text{ else } M\ i\ j$.

lift-definition *upd-col* $:: 'a \rightsquigarrow 'b \Rightarrow 'b \Rightarrow 'a \rightsquigarrow 'b \Rightarrow 'a \rightsquigarrow 'b$ **is**
 $\lambda M\ j'\ v\ i\ j. \text{if } j = j' \text{ then } v\ \$\ i \text{ else } M\ i\ j$.

lift-definition *row* $:: 'a \rightsquigarrow 'b \Rightarrow 'b \Rightarrow 'a \rightsquigarrow 'b$ **is**
 $\lambda M\ i. \chi\ j. M\ i\ j$.

lift-definition *col* $:: 'a \rightsquigarrow 'b \Rightarrow 'b \Rightarrow 'a \rightsquigarrow 'b$ **is**
 $\lambda M\ j. \chi\ i. M\ i\ j$.

lemma *perm-rows-transpose*: $\text{perm-rows } (\text{transpose } M) p = \text{transpose } (\text{perm-cols } M\ p)$
by *transfer simp*

lemma *perm-cols-transpose*: $\text{perm-cols } (\text{transpose } M) p = \text{transpose } (\text{perm-rows } M\ p)$

p)
by *transfer simp*

lemma *upd-row-transpose*: $\text{upd-row } (\text{transpose } M) \ i \ p = \text{transpose } (\text{upd-col } M \ i \ p)$
by *transfer simp*

lemma *upd-col-transpose*: $\text{upd-col } (\text{transpose } M) \ i \ p = \text{transpose } (\text{upd-row } M \ i \ p)$
by *transfer simp*

lemma *upd-rows-transpose*: $\text{upd-rows } (\text{transpose } M) \ i \ p = \text{transpose } (\text{upd-cols } M \ i \ p)$
by *transfer simp*

lemma *upd-cols-transpose*: $\text{upd-cols } (\text{transpose } M) \ i \ p = \text{transpose } (\text{upd-rows } M \ i \ p)$
by *transfer simp*

lemma *upd-rows-empty[simp]*: $\text{upd-rows } M \ \{\} \ f = M$
by *transfer simp*

lemma *upd-cols-empty[simp]*: $\text{upd-cols } M \ \{\} \ f = M$
by *transfer simp*

lemma *upd-rows-single[simp]*: $\text{upd-rows } M \ \{i\} \ f = \text{upd-row } M \ i \ (f \ i)$
by *transfer (simp add: fun-eq-iff)*

lemma *upd-cols-single[simp]*: $\text{upd-cols } M \ \{i\} \ f = \text{upd-col } M \ i \ (f \ i)$
by *transfer (simp add: fun-eq-iff)*

lemma *upd-rows-insert*: $\text{upd-rows } M \ (\text{insert } i \ I) \ f = \text{upd-row } (\text{upd-rows } M \ I \ f) \ i \ (f \ i)$
by *transfer (auto simp: fun-eq-iff)*

lemma *upd-rows-insert-rev*: $\text{upd-rows } M \ (\text{insert } i \ I) \ f = \text{upd-rows } (\text{upd-row } M \ i \ (f \ i)) \ I \ f$
by *transfer (auto simp: fun-eq-iff)*

lemma *upd-rows-upd-row-swap*: $i \notin I \implies \text{upd-rows } (\text{upd-row } M \ i \ x) \ I \ f = \text{upd-row } (\text{upd-rows } M \ I \ f) \ i \ x$
by *transfer (simp add: fun-eq-iff)*

lemma *upd-cols-insert*: $\text{upd-cols } M \ (\text{insert } i \ I) \ f = \text{upd-col } (\text{upd-cols } M \ I \ f) \ i \ (f \ i)$
by *transfer (auto simp: fun-eq-iff)*

lemma *upd-cols-insert-rev*: $\text{upd-cols } M \ (\text{insert } i \ I) \ f = \text{upd-cols } (\text{upd-col } M \ i \ (f \ i)) \ I \ f$
by *transfer (auto simp: fun-eq-iff)*

lemma *upd-cols-upd-col-swap*: $i \notin I \implies \text{upd-cols } (\text{upd-col } M \ i \ x) \ I \ f = \text{upd-col } (\text{upd-cols } M \ I \ f) \ i \ x$

(*upd-cols M I f*) *i x*
by *transfer (simp add: fun-eq-iff)*

lemma *upd-rows-cong[cong]*:
 $M = N \implies T = S \implies (\bigwedge s. s \in S = \text{simp} \implies f s = g s) \implies \text{upd-rows } M T f = \text{upd-rows } N S g$
unfolding *simp-implies-def*
by *transfer (auto simp: fun-eq-iff)*

lemma *upd-cols-cong[cong]*:
 $M = N \implies T = S \implies (\bigwedge s. s \in S = \text{simp} \implies f s = g s) \implies \text{upd-cols } M T f = \text{upd-cols } N S g$
unfolding *simp-implies-def*
by *transfer (auto simp: fun-eq-iff)*

lemma *row-upd-row-If*: $\text{row } (\text{upd-row } M i x) j = (\text{if } i = j \text{ then } x \text{ else } \text{row } M j)$
by *transfer (simp add: vec-eq-iff fun-eq-iff)*

lemma *row-upd-row[simp]*: $\text{row } (\text{upd-row } M i x) i = x$
by *(simp add: row-upd-row-If)*

lemma *col-upd-col-If*: $\text{col } (\text{upd-col } M i x) j = (\text{if } i = j \text{ then } x \text{ else } \text{col } M j)$
by *transfer (simp add: vec-eq-iff)*

lemma *col-upd-col[simp]*: $\text{col } (\text{upd-col } M i x) i = x$
by *(simp add: col-upd-col-If)*

lemma *upd-row-row[simp]*: $\text{upd-row } M i (\text{row } M i) = M$
by *transfer (simp add: fun-eq-iff)*

lemma *upd-row-upd-row-cancel[simp]*: $\text{upd-row } (\text{upd-row } M i x) i y = \text{upd-row } M i y$
by *transfer (simp add: fun-eq-iff)*

lemma *upd-col-upd-col-cancel[simp]*: $\text{upd-col } (\text{upd-col } M i x) i y = \text{upd-col } M i y$
by *transfer (simp add: fun-eq-iff)*

lemma *upd-col-col[simp]*: $\text{upd-col } M i (\text{col } M i) = M$
by *transfer (simp add: fun-eq-iff)*

lemma *row-transpose*: $\text{row } (\text{transpose } M) i = \text{col } M i$
by *transfer simp*

lemma *col-transpose*: $\text{col } (\text{transpose } M) i = \text{row } M i$
by *transfer simp*

lemma *det-perm-cols*:
fixes $A :: 'a::\text{comm-ring-1} \rightsquigarrow n$
assumes p *permutes UNIV*

shows $\det (\text{perm-cols } A \ p) = \text{of-int } (\text{sign } p) * \det A$
proof (*transfer fixing: p*)
fix $A :: 'n \Rightarrow 'n \Rightarrow 'a$
from p **have** $(\sum q \mid q \text{ permutes } UNIV. \text{of-int } (\text{sign } q) * (\prod i \in UNIV. A \ i \ (p \ (q \ i)))) =$
 $(\sum q \mid q \text{ permutes } UNIV. \text{of-int } (\text{sign } (\text{inv } p \circ q)) * (\prod i \in UNIV. A \ i \ (q \ i)))$
by (*intro sum.reindex-bij-witness[where j= $\lambda q. p \circ q$ and i= $\lambda q. \text{inv } p \circ q$]*)
(auto simp: comp-assoc[symmetric] permutes-inv-o permutes-compose permutes-inv)
with p **show** $(\sum q \mid q \text{ permutes } UNIV. \text{of-int } (\text{sign } q) * (\prod i \in UNIV. A \ i \ (p \ (q \ i)))) =$
 $\text{of-int } (\text{sign } p) * (\sum p \mid p \text{ permutes } UNIV. \text{of-int } (\text{sign } p) * (\prod i \in UNIV. A \ i \ (p \ i)))$
by (*auto intro!: sum.cong simp: sum-distrib-left sign-compose permutes-inv sign-inverse permutes-UNIV-permutation*)
qed

lemma *det-perm-rows*:
fixes $A :: 'a::\text{comm-ring-1} \rightsquigarrow 'n$
assumes $p: p \text{ permutes } UNIV$
shows $\det (\text{perm-rows } A \ p) = \text{of-int } (\text{sign } p) * \det A$
using *det-perm-cols[OF p, of transpose A]* **by** (*simp add: det-transpose perm-cols-transpose*)

lemma *det-row-add*: $\det (\text{upd-row } M \ i \ (a + b)) = \det (\text{upd-row } M \ i \ a) + \det (\text{upd-row } M \ i \ b)$
by *transfer (simp add: prod.If-cases sum.distrib[symmetric] field-simps)*

lemma *det-row-mul*: $\det (\text{upd-row } M \ i \ (c * s \ a)) = c * \det (\text{upd-row } M \ i \ a)$
by *transfer (simp add: prod.If-cases sum-distrib-left field-simps)*

lemma *det-row-uminus*: $\det (\text{upd-row } M \ i \ (- a)) = - \det (\text{upd-row } M \ i \ a)$
by (*simp add: vector-sneg-minus1 det-row-mul*)

lemma *det-row-minus*: $\det (\text{upd-row } M \ i \ (a - b)) = \det (\text{upd-row } M \ i \ a) - \det (\text{upd-row } M \ i \ b)$
unfolding *diff-conv-add-uminus det-row-add det-row-uminus ..*

lemma *det-row-0*: $\det (\text{upd-row } M \ i \ 0) = 0$
using *det-row-mul[of M i 0]* **by** *simp*

lemma *det-row-sum*: $\det (\text{upd-row } M \ i \ (\sum s \in S. a \ s)) = (\sum s \in S. \det (\text{upd-row } M \ i \ (a \ s)))$
by (*induction S rule: infinite-finite-induct*) (*simp-all add: det-row-0 det-row-add*)

lemma *det-col-add*: $\det (\text{upd-col } M \ i \ (a + b)) = \det (\text{upd-col } M \ i \ a) + \det (\text{upd-col } M \ i \ b)$
using *det-row-add[of transpose M i a b]* **by** (*simp add: upd-row-transpose det-transpose*)

lemma *det-col-mul*: $\det (\text{upd-col } M \ i \ (c * s \ a)) = c * \det (\text{upd-col } M \ i \ a)$

using *det-row-mul*[of *transpose M i c a*] **by** (*simp add: upd-row-transpose det-transpose*)

lemma *det-col-uminus*: $\det (\text{upd-col } M \ i \ (- \ a)) = - \ \det (\text{upd-col } M \ i \ a)$
by (*simp add: vector-sneg-minus1 det-col-mul*)

lemma *det-col-minus*: $\det (\text{upd-col } M \ i \ (a - b)) = \det (\text{upd-col } M \ i \ a) - \det (\text{upd-col } M \ i \ b)$
unfolding *diff-conv-add-uminus det-col-add det-col-uminus ..*

lemma *det-col-0*: $\det (\text{upd-col } M \ i \ 0) = 0$
using *det-col-mul*[of *M i 0*] **by** *simp*

lemma *det-col-sum*: $\det (\text{upd-col } M \ i \ (\sum_{s \in S} a \ s)) = (\sum_{s \in S} \det (\text{upd-col } M \ i \ (a \ s)))$
by (*induction S rule: infinite-finite-induct*) (*simp-all add: det-col-0 det-col-add*)

lemma *det-identical-cols*:

assumes $i \neq i'$ **shows** $\text{col } A \ i = \text{col } A \ i' \implies \det A = 0$

proof (*transfer fixing: i i'*)

fix $A :: 'a \Rightarrow 'a \Rightarrow 'b$ **assume** $(\chi \ j. A \ j \ i) = (\chi \ i. A \ i \ i')$

then have [*simp*]: $\bigwedge j \ q. A \ j \ (\text{Transposition.transpose } i \ i' \ (q \ j)) = A \ j \ (q \ j)$

by (*simp add: vec-eq-iff Transposition.transpose-def*)

let $?p = \lambda p. \text{of-int } (\text{sign } p) * (\prod_{i \in \text{UNIV}} A \ i \ (p \ i))$

let $?s = \lambda q. \text{Transposition.transpose } i \ i' \circ q$

let $?E = \{p. p \ \text{permutes } \text{UNIV} \wedge \text{evenperm } p\}$

have [*simp*]: *inj-on* $?s \ ?E$

by (*auto simp: inj-on-def fun-eq-iff Transposition.transpose-def*)

note $p = \text{permutes-UNIV-permutation evenperm-comp permutes-swap-id evenperm-swap permutes-compose}$

sign-compose sign-swap-id

from $\langle i \neq i' \rangle$ **have** $*$: *evenperm* q **if** $q \notin ?s' ?E$ q *permutes UNIV* **for** q

using *that* **by** (*auto simp add: comp-assoc[symmetric] image-iff p elim!: allE*[of $- \ ?s \ q$])

have $(\sum p \mid p \ \text{permutes } \text{UNIV}. \ ?p \ p) = (\sum p \in ?E \cup ?s' ?E. \ ?p \ p)$

by (*auto simp add: permutes-compose permutes-swap-id intro: * sum.cong*)

also have $\dots = (\sum p \in ?E. \ ?p \ p) + (\sum p \in ?s' ?E. \ ?p \ p)$

by (*intro sum.union-disjoint*) (*auto simp: p* $\langle i \neq i' \rangle$)

also have $(\sum p \in ?s' ?E. \ ?p \ p) = (\sum p \in ?E. - \ ?p \ p)$

using $\langle i \neq i' \rangle$ **by** (*subst sum.reindex*) (*auto intro!: sum.cong simp: p*)

finally show $(\sum p \mid p \ \text{permutes } \text{UNIV}. \ ?p \ p) = 0$

by (*simp add: sum-negf*)

qed

lemma *det-identical-rows*: $i \neq i' \implies \text{row } A \ i = \text{row } A \ i' \implies \det A = 0$

using *det-identical-cols*[of $i \ i'$ *transpose A*] **by** (*simp add: det-transpose col-transpose*)

lemma *det-cols-sum*:

$det (upd\text{-}cols\ M\ T\ (\lambda i. \sum s \in S. a\ i\ s)) = (\sum f \in T \rightarrow_E S. det (upd\text{-}cols\ M\ T\ (\lambda i. a\ i\ (f\ i))))$

proof (*induct T arbitrary: M rule: infinite-finite-induct*)

case (*insert i T*)

have $(\sum f \in insert\ i\ T \rightarrow_E S. det (upd\text{-}cols\ M\ (insert\ i\ T)\ (\lambda i. a\ i\ (f\ i)))) = (\sum s \in S. \sum f \in T \rightarrow_E S. det (upd\text{-}cols\ (upd\text{-}col\ M\ i\ (a\ i\ s))\ T\ (\lambda i. a\ i\ (f\ i))))$

unfolding *sum.cartesian-product PiE-insert-eq* **using** $\langle i \notin T \rangle$

by (*subst sum.reindex[OF inj-combinator[OF $\langle i \notin T \rangle$]]*)

(*auto intro!: sum.cong arg-cong[where f=det] upd-cols-cong*

simp: upd-cols-insert-rev simp-implies-def)

also have $\dots = det (upd\text{-}col\ (upd\text{-}cols\ M\ T\ (\lambda i. sum\ (a\ i\ S))\ i\ (\sum s \in S. a\ i\ s))$

unfolding *insert(3)[symmetric]* **by** (*simp add: upd-cols-upd-col-swap[OF $\langle i \notin T \rangle$] det-col-sum*)

finally show *?case*

by (*simp add: upd-cols-insert*)

qed *auto*

lemma *det-rows-sum*:

$det (upd\text{-}rows\ M\ T\ (\lambda i. \sum s \in S. a\ i\ s)) = (\sum f \in T \rightarrow_E S. det (upd\text{-}rows\ M\ T\ (\lambda i. a\ i\ (f\ i))))$

using *det-cols-sum[of transpose M T a S]* **by** (*simp add: upd-cols-transpose det-transpose*)

lemma *det-rows-mult*: $det (upd\text{-}rows\ M\ T\ (\lambda i. c\ i\ *s\ a\ i)) = (\prod i \in T. c\ i) * det (upd\text{-}rows\ M\ T\ a)$

by *transfer (simp add: prod.If-cases sum-distrib-left field-simps prod.distrib)*

lemma *det-cols-mult*: $det (upd\text{-}cols\ M\ T\ (\lambda i. c\ i\ *s\ a\ i)) = (\prod i \in T. c\ i) * det (upd\text{-}cols\ M\ T\ a)$

using *det-rows-mult[of transpose M T c a]* **by** (*simp add: det-transpose upd-rows-transpose*)

lemma *det-perm-rows-If*: $det (perm\text{-}rows\ B\ f) = (if\ f\ permutes\ UNIV\ then\ of\text{-}int\ (sign\ f) * det\ B\ else\ 0)$

proof *cases*

assume $\neg f\ permutes\ UNIV$

moreover

with *bij-imp-permutes[of f UNIV]* **have** $\neg inj\ f$

using *finite-UNIV-inj-surj[of f]* **by** (*auto simp: bij-betw-def*)

then obtain $i\ j$ **where** $f\ i = f\ j\ i \neq j$

by (*auto simp: inj-on-def*)

moreover

then have $row\ (perm\text{-}rows\ B\ f)\ i = row\ (perm\text{-}rows\ B\ f)\ j$

by *transfer (auto simp: vec-eq-iff)*

ultimately show *?thesis*

by (*simp add: det-identical-rows*)

qed (*simp add: det-perm-rows*)

lemma *det-mult*: $\det (A * B) = \det A * \det B$
proof –
 have $A * B = \text{upd-rows } 0 \text{ UNIV } (\lambda i. \sum_{j \in \text{UNIV}} \text{to-fun } A \ i \ j * \text{row } B \ j)$
 by *transfer simp*
 moreover have $\bigwedge f. \text{upd-rows } 0 \text{ UNIV } (\lambda i. \text{Square-Matrix.row } B \ (f \ i)) = \text{perm-rows } B \ f$
 by *transfer simp*
 moreover have $\det A = (\sum p \mid p \text{ permutes UNIV. of-int } (\text{sign } p) * (\prod_{i \in \text{UNIV}} \text{to-fun } A \ i \ (p \ i)))$
 by *transfer rule*
 ultimately show *?thesis*
 by (*auto simp add: det-rows-sum det-rows-mult sum-distrib-right det-perm-rows-If split: if-split-asm intro!: sum.mono-neutral-cong-right*)
qed

lift-definition *minor* :: $'a \rightsquigarrow 'b \Rightarrow 'b \Rightarrow 'a :: \text{semiring-1} \rightsquigarrow 'b$ **is**
 $\lambda A \ i \ j \ k \ l. \text{if } k = i \wedge l = j \text{ then } 1 \text{ else if } k = i \vee l = j \text{ then } 0 \text{ else } A \ k \ l.$

lemma *minor-transpose*: $\text{minor } (\text{transpose } A) \ i \ j = \text{transpose } (\text{minor } A \ j \ i)$
by *transfer (auto simp: fun-eq-iff)*

lemma *minor-eq-row-col*: $\text{minor } M \ i \ j = \text{upd-row } (\text{upd-col } M \ j \ (\text{axis } i \ 1)) \ i \ (\text{axis } j \ 1)$
by *transfer (simp add: fun-eq-iff axis-def)*

lemma *minor-eq-col-row*: $\text{minor } M \ i \ j = \text{upd-col } (\text{upd-row } M \ i \ (\text{axis } j \ 1)) \ j \ (\text{axis } i \ 1)$
by *transfer (simp add: fun-eq-iff axis-def)*

lemma *row-minor*: $\text{row } (\text{minor } M \ i \ j) \ i = \text{axis } j \ 1$
by (*simp add: minor-eq-row-col*)

lemma *col-minor*: $\text{col } (\text{minor } M \ i \ j) \ j = \text{axis } i \ 1$
by (*simp add: minor-eq-col-row*)

lemma *det-minor-row'*:

$\text{row } B \ i = \text{axis } j \ 1 \implies \det (\text{minor } B \ i \ j) = \det B$

proof (*induction {k. to-fun B k j ≠ 0} – {i} arbitrary: B rule: infinite-finite-induct*)

case *empty*

then have $\bigwedge k. k \neq i \implies \text{to-fun } B \ k \ j = 0$

by (*auto simp add: card-eq-0-iff*)

with *empty.prem*s **have** $\text{axis } i \ 1 = \text{col } B \ j$

by *transfer (auto simp: vec-eq-iff axis-def)*

with *empty.prem*s[*symmetric*] **show** *?case*

by (*simp add: minor-eq-row-col*)

next

case (*insert r NZ*)

then have $r: r \neq i \text{ to-fun } B \ r \ j \neq 0$

by *auto*

```

let ?B' = upd-row B r (row B r - (to-fun B r j) *s row B i)
have det (minor ?B' i j) = det ?B'
proof (rule insert.hyps)
  show NZ = {k. to-fun ?B' k j ≠ 0} - {i}
    using insert.hyps(2,4) insert.prem
    by transfer (auto simp add: axis-def set-eq-iff)
  show row ?B' i = axis j 1
    using r insert by (simp add: row-upd-row-If)
qed
also have minor ?B' i j = minor B i j
  using r insert.prem by transfer (simp add: fun-eq-iff axis-def)
also have det ?B' = det B
  using ⟨r ≠ i⟩
  by (simp add: det-row-minus det-row-mul det-identical-rows[OF ⟨r ≠ i⟩] row-upd-row-If)
finally show ?case .
qed simp

```

```

lemma det-minor-row: det (minor B i j) = det (upd-row B i (axis j 1))
proof -
  have det (minor (upd-row B i (axis j 1)) i j) = det (upd-row B i (axis j 1))
    by (rule det-minor-row') simp
  then show ?thesis
    by (simp add: minor-eq-col-row)
qed

```

```

lemma det-minor-col: det (minor B i j) = det (upd-col B j (axis i 1))
  using det-minor-row[of transpose B j i]
  by (simp add: minor-transpose det-transpose upd-row-transpose)

```

lift-definition cofactor :: 'a[~]b ⇒ 'a::comm-ring-1[~]b is
 $\lambda A i j. \det (\text{minor } A i j)$.

```

lemma cofactor-transpose: cofactor (transpose A) = transpose (cofactor A)
  by (simp add: cofactor-def minor-transpose det-transpose transpose.rep-eq to-fun-inject[symmetric]
of-fun-inverse)

```

definition adjugate A = transpose (cofactor A)

```

lemma adjugate-transpose: adjugate (transpose A) = transpose (adjugate A)
  by (simp add: adjugate-def cofactor-transpose)

```

theorem adjugate-mult-det: adjugate A * A = diag (det A)

proof (intro to-fun-inject[THEN iffD1] fun-eq-iff[THEN iffD2] allI)

```

  fix i k
  have to-fun (adjugate A * A) i k = (∑ j∈UNIV. to-fun A j k * det (minor A j
i))

```

```

  by (simp add: adjugate-def times-sq-matrix.rep-eq transpose.rep-eq cofactor-def
mult.commute of-fun-inverse)

```

```

  also have ... = det (upd-col A i (∑ j∈UNIV. to-fun A j k *s axis j 1))

```

by (simp add: det-minor-col det-col-mul det-col-sum)
 also have $(\sum_{j \in UNIV}. \text{to-fun } A \ j \ k \ *s \ \text{axis } j \ 1) = \text{col } A \ k$
 by transfer (simp add: smult-axis vec-eq-iff, simp add: axis-def sum.If-cases)
 also have $\text{det } (\text{upd-col } A \ i \ (\text{col } A \ k)) = (\text{if } i = k \ \text{then } \text{det } A \ \text{else } 0)$
 by (auto simp: col-upd-col-If det-identical-cols[of i k])
 also have $\dots = \text{to-fun } (\text{diag } (\text{det } A)) \ i \ k$
 by (simp add: diag.rep-eq)
 finally show $\text{to-fun } (\text{adjugate } A * A) \ i \ k = \text{to-fun } (\text{diag } (\text{det } A)) \ i \ k$.
 qed

lemma mult-adjugate-det: $A * \text{adjugate } A = \text{diag } (\text{det } A)$

proof -

have $\text{transpose } (\text{transpose } (A * \text{adjugate } A)) = \text{transpose } (\text{diag } (\text{det } A))$

unfolding transpose-mult adjugate-transpose[symmetric] adjugate-mult-det det-transpose

..

then show ?thesis

by simp

qed

end

theorem Cayley-Hamilton:

fixes $A :: 'a::\text{comm-ring-1} \ \widehat{\sim} \ 'n$

shows $\text{poly-mat } (\text{charpoly } A) \ A = 0$

proof -

Part 1

define n where $n = \text{CARD } ('n) - 1$

then have $d\text{-charpoly}: n + 1 = \text{degree } (\text{charpoly } A)$ and

$d\text{-adj}: n = \text{max-degree } (\text{adjugate } (\mathbf{X} - \mathbf{C} \ A))$

define B where $B \ i = \text{map-sq-matrix } (\lambda p. \text{coeff } p \ i) \ (\text{adjugate } (\mathbf{X} - \mathbf{C} \ A))$ for i

have $A\text{-eq-}B: \text{adjugate } (\mathbf{X} - \mathbf{C} \ A) = (\sum_{i \leq n}. X^{i} *_{S} \mathbf{C} \ (B \ i))$

Part 2

have $\text{charpoly } A *_{S} 1 = X *_{S} \text{adjugate } (\mathbf{X} - \mathbf{C} \ A) - \mathbf{C} \ A * \text{adjugate } (\mathbf{X} - \mathbf{C} \ A)$

also have $\dots = (\sum_{i \leq n}. X^{i+1} *_{S} \mathbf{C} \ (B \ i)) - (\sum_{i \leq n}. X^{i} *_{S} \mathbf{C} \ (A * B \ i))$

also have $(\sum_{i \leq n}. X^{i+1} *_{S} \mathbf{C} \ (B \ i)) =$
 $(\sum_{i < n}. X^{i+1} *_{S} \mathbf{C} \ (B \ i)) + X^{n+1} *_{S} \mathbf{C} \ (B \ n)$

also have $(\sum_{i \leq n}. X^{i} *_{S} \mathbf{C} \ (A * B \ i)) =$
 $(\sum_{i < n}. X^{i+1} *_{S} \mathbf{C} \ (A * B \ (i+1))) + \mathbf{C} \ (A * B \ 0)$

finally have $d\text{diag-charpoly}$:

$\text{charpoly } A *_{S} 1 = X^{n+1} *_{S} \mathbf{C} \ (B \ n) +$
 $(\sum_{i < n}. X^{i+1} *_{S} \mathbf{C} \ (B \ i - A * B \ (i+1))) - \mathbf{C} \ (A * B \ 0)$

```

let ?p = λi. coeff (charpoly A) i *S A ^ i
let ?AB = λi. A ^ (i + 1) * B i
have (∑ i ≤ n + 1. ?p i) = ?p 0 + (∑ i < n. ?p (i + 1)) + ?p (n + 1)
also have ?p 0 = - ?AB 0
also have (∑ i < n. ?p (i + 1)) = (∑ i = 0 .. < n. ?AB i - ?AB (i + 1))
also have ... = ?AB 0 - ?AB n
also have ?AB n = ?p (n + 1)
also have coeff (charpoly A) (n + 1) = 1
finally show ?thesis
qed
  
```

References

- [1] S. Ould Biha. Formalisation des mathématiques : une preuve du théorème de Cayley-Hamilton. In *JFLA (Journées Francophones des Langages Applicatifs)*, pages 1–14. INRIA, 2008. available at <http://hal.inria.fr/inria-00202795/PDF/ouldbiha.pdf>.
- [2] S. Ould Biha. *Composants mathématiques pour la théorie des groupes*. PhD thesis, Université de Nice – Sophia Antipolis, 2010. available at <http://hal.inria.fr/tel-00493524>.