# The Cayley-Hamilton theorem

Stephan Adelsberger Stefan Hetzl Florian Pollak

March 17, 2025

**Abstract**

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

## Contents



## 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

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

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*: $x *s axis\ i\ y = axis\ i\ (x * y$::-::*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* ⇒ *type* ⇒ *type* (‹(- ^^/ -)› [*15*, *16*] *15*)

**syntax-types** *-sq-matrix* ⇌ *sq-matrix*

**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* :: $('a \Rightarrow 'c) \Rightarrow 'a^{\wedge\wedge}'b \Rightarrow 'c^{\wedge\wedge}'b$ **is**
$\lambda f\ M\ i\ j.\ f\ (M\ i\ j)$ **.**

**lift-definition** *from-vec* :: $'a^{\wedge}'n^{\wedge}'n \Rightarrow 'a^{\wedge\wedge}'n$ **is**
$\lambda M\ i\ j.\ M\ \$\ i\ \$\ j$ **.**

**lift-definition** *to-vec* :: $'a^{\wedge\wedge}'n \Rightarrow 'a^{\wedge}'n^{\wedge}'n$ **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*]: *map-sq-matrix f* (*map-sq-matrix g M*) = *map-sq-matrix* (λ*x*. *f* (*g x*)) *M*
**by** *transfer simp*

**lemma** *map-sq-matrix-ident*[*simp*]: *map-sq-matrix* (λ*x*. *x*) *M* = *M*
**by** *transfer* (*simp add*: *vec-eq-iff*)

**lemma** *map-sq-matrix-cong*:
*M* = *N* ⟹ (⋀*i j*. *f* (*to-fun N i j*) = *g* (*to-fun N i j*)) ⟹ *map-sq-matrix f M* = *map-sq-matrix g N*
**by** *transfer simp*

**lift-definition** *diag* :: *'a*::*zero* ⇒ *'a*^^*'n* **is**
λ*k i j*. *if i* = *j then k else 0* **.**

**lemma** *diag-eq-iff*: *diag x* = *diag y* ⟷ *x* = *y*
**by** *transfer* (*simp add*: *fun-eq-iff*)

**lemma** *map-sq-matrix-diag*[*simp*]: *f 0* = *0* ⟹ *map-sq-matrix f* (*diag c*) = *diag* (*f c*)
**by** *transfer* (*simp add*: *fun-eq-iff*)

**lift-definition** *smult-sq-matrix* :: *'a*::*times* ⇒ *'a*^^*'n* ⇒ *'a*^^*'n* (**infixr** ‹∗$_S$› *75*) **is**
λ*c M i j*. *c* ∗ *M i j* **.**

**lemma** *smult-map-sq-matrix*:
(⋀*y*. *f* (*x* ∗ *y*) = *z* ∗ *f y*) ⟹ *map-sq-matrix f* (*x* ∗$_S$ *A*) = *z* ∗$_S$ *map-sq-matrix f A*
**by** *transfer simp*

**lemma** *map-sq-matrix-smult*: *c* ∗$_S$ *map-sq-matrix f A* = *map-sq-matrix* (λ*x*. *c* ∗ *f x*) *A*
**by** *transfer simp*

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

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

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

**lift-definition** *plus-sq-matrix* :: *'a*^^*'b* ⇒ *'a*^^*'b* ⇒ *'a*^^*'b* **is**
λ*A B i j*. *A i j* + *B i j* **.**

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

**end**

**lemma** *map-sq-matrix-add*:
($\bigwedge a\ b.\ f\ (a + b) = f\ a + f\ b) \Longrightarrow$ *map-sq-matrix f* $(A + B)$ = *map-sq-matrix f A* + *map-sq-matrix f B*
**by** *transfer simp*

**lemma** *add-map-sq-matrix*: *map-sq-matrix f A* + *map-sq-matrix g A* = *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* :: $'a^{\frown\frown}{}'b$ **is** $\lambda i\ j.\ 0$ **.**

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

**end**

**lemma** *diag-0*: *diag 0* = *0*
**by** *transfer simp*

**lemma** *diag-0-eq*: *diag x* = *0* $\longleftrightarrow$ *x* = *0*
**by** *transfer* (*simp add*: *fun-eq-iff*)

**lemma** *zero-map-sq-matrix*: *f 0* = *0* $\Longrightarrow$ *map-sq-matrix f 0* = *0*
**by** *transfer simp*

**lemma** *map-sq-matrix-0*[*simp*]: *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* :: $'a^{\frown\frown}{}'b \Rightarrow {}'a^{\frown\frown}{}'b \Rightarrow {}'a^{\frown\frown}{}'b$ **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* :: $'a\hat{}\hat{}'b \Rightarrow 'a\hat{}\hat{}'b$ **is**
*uminus* **.**

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

**end**

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

**lemma** *smult-diff*: **fixes** $a$ :: $'a$::*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*:
$f\ 0 = 0 \Longrightarrow (\bigwedge a\ b.\ f\ (a + b) = f\ a + f\ b) \Longrightarrow$
*map-sq-matrix f* $(\sum i{\in}I.\ A\ i) = (\sum i{\in}I.$ *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*: $(\sum i{\in}I.$ *map-sq-matrix* $(f\ i)\ A) =$ *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** $a$ :: $'a$::*ring-1* **shows** $a *_S 0 = 0$
**by** *transfer* (*simp add*: *vec-eq-iff*)

**lemma** *smult-right-add*: **fixes** $a$ :: $'a$::*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** $a$ :: $'a$::*ring-1* **shows** $(\sum i{\in}I.\ a *_S f\ i) = a *_S$ (*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* :: $'a^{\frown\frown}{}'b \Rightarrow {}'a^{\frown\frown}{}'b \Rightarrow {}'a^{\frown\frown}{}'b$ **is**
$\lambda M\ N\ i\ j.\ \sum k{\in}UNIV.\ M\ i\ k * N\ k\ j$ **.**

**instance**
**proof**
**fix** *a b c* :: $'a^{\frown\frown}{}'b$ **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* $\Rightarrow$ $'b$::*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-sq-matrix f* (*A* $*$ *B*) = *map-sq-matrix f A* $*$ *map-sq-matrix f B*
**proof** (*transfer fixing*: *f*)
**fix** *A B* :: $'c \Rightarrow {}'c \Rightarrow {}'a$
**{ 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-vec* (*M* $**$ *N*) = *from-vec M* $*$ *from-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* :: $'a^{\frown\frown}'b$ **is**
$\lambda i\ j.$ *if* $i = j$ *then 1 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* $\Rightarrow$ $'a^{\frown\frown}'b \Rightarrow$ $'a^{\frown\frown}'b$ **is**
$\lambda r\ A\ i\ j.\ r *_R A\ i\ j$ **.**

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

**end**

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

**lift-definition** *transpose* :: $'a^{\frown\frown}'n \Rightarrow$ $'a^{\frown\frown}'n$ **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*]: *transpose 1 = 1*
**by** *transfer* (*simp add*: *fun-eq-iff*)

**lemma** *transpose-add*[*simp*]: *transpose* (*A + B*) = *transpose A + transpose B*
**by** *transfer simp*

**lemma** *transpose-minus*[*simp*]: *transpose* (*A − B*) = *transpose A − transpose B*
**by** *transfer simp*

**lemma** *transpose-uminus*[*simp*]: *transpose* (*− A*) = *− transpose A*
**by** *transfer* (*simp add*: *fun-eq-iff*)

**lemma** *transpose-mult*[*simp*]:
*transpose* (*A ∗ B* :: *'a*::*comm-semiring-0*^^*'n*) = *transpose B ∗ transpose A*
**by** *transfer* (*simp add*: *field-simps*)

**lift-definition** *trace* :: *'a*::*comm-monoid-add*^^*'n* ⇒ *'a* **is**
*λM*. ∑ *i*∈*UNIV*. *M i i* **.**

**lemma** *trace-diag*[*simp*]: *trace* (*diag c*::*'a*::*semiring-1*^^*'n*) = *of-nat CARD*(*'n*) ∗ *c*
**by** *transfer simp*

**lemma** *trace-0*[*simp*]: *trace 0 = 0*
**by** *transfer simp*

**lemma** *trace-1*[*simp*]: *trace* (*1*::*'a*::*semiring-1*^^*'n*) = *of-nat CARD*(*'n*)
**by** *transfer simp*

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

**lemma** *trace-minus*[*simp*]: *trace* (*A − B*) = (*trace A − trace B*::-::*ab-group-add*)
**by** *transfer* (*simp add*: *sum-subtractf*)

**lemma** *trace-uminus*[*simp*]: *trace* (*− A*) = *−* (*trace A*::-::*ab-group-add*)
**by** *transfer* (*simp add*: *sum-negf*)

**lemma** *trace-smult*[*simp*]: *trace* (*s ∗S A*) = (*s ∗ trace A*::-::*semiring-0*)
**by** *transfer* (*simp add*: *sum-distrib-left*)

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

**lemma** *trace-mult-symm*:
**fixes** *A B* :: *'a*::*comm-semiring-0*^^*'n*

**shows** *trace* $(A * B) = trace\ (B * A)$
**by** *transfer* (*auto intro*: *sum.swap simp*: *mult.commute*)

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

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

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

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

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

**{ fix** *p* :: *'n* ⇒ *'n* **assume** *p*: *p permutes UNIV* $p \neq id$
**then obtain** *i* **where** *i*: $i \neq p\ i$
**unfolding** *id-def* **by** *metis*
**with** *neq*[*OF i*] **have** $(\prod i{\in}UNIV.\ A\ i\ (p\ i)) = 0$
**by** (*intro prod-zero*) *auto* **}**
**then have** $(\sum p \mid p\ permutes\ UNIV.\ ?pp\ p) = (\sum p{\in}\{id\}.\ ?pp\ p)$
**by** (*intro sum.mono-neutral-cong-right*) (*auto intro*: *permutes-id*)
**then show** $(\sum p \mid p\ permutes\ UNIV.\ ?pp\ p) = (\prod i{\in}UNIV.\ A\ i\ i)$
**by** (*simp add*: *sign-id*)
**qed**

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

**lemma** *det-lowerdiagonal*:

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

**{ fix** $p :: 'n \Rightarrow 'n$ **assume** *p*: $p\ permutes\ UNIV\ p \neq id$
**with** *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** (*intro prod-zero*) *auto* **}**
**then have** $(\sum p \mid p\ permutes\ UNIV.\ ?pp\ p) = (\sum p{\in}\{id\}.\ ?pp\ p)$
**by** (*intro sum.mono-neutral-cong-right*) (*auto intro*: *permutes-id*)
**then show** $(\sum p \mid p\ permutes\ UNIV.\ ?pp\ p) = (\prod i{\in}UNIV.\ A\ i\ i)$
**by** (*simp add*: *sign-id*)
**qed**

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

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

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

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

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

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

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

**lift-definition** *row* $:: 'a^{\frown\frown}'b \Rightarrow 'b \Rightarrow 'a^{\frown}'b$ **is**
$\lambda M\ i.\ \chi\ j.\ M\ i\ j$ **.**

**lift-definition** *col* $:: 'a^{\frown\frown}'b \Rightarrow 'b \Rightarrow 'a^{\frown}'b$ **is**
$\lambda M\ j.\ \chi\ i.\ M\ i\ j$ **.**

**lemma** *perm-rows-transpose*: *perm-rows* (*transpose M*) $p$ = *transpose* (*perm-cols M p*)

**by** *transfer simp*

**lemma** *perm-cols-transpose*: *perm-cols* (*transpose M*) *p* = *transpose* (*perm-rows M p*)
**by** *transfer simp*

**lemma** *upd-row-transpose*: *upd-row* (*transpose M*) *i p* = *transpose* (*upd-col M i p*)
**by** *transfer simp*

**lemma** *upd-col-transpose*: *upd-col* (*transpose M*) *i p* = *transpose* (*upd-row M i p*)
**by** *transfer simp*

**lemma** *upd-rows-transpose*: *upd-rows* (*transpose M*) *i p* = *transpose* (*upd-cols M i p*)
**by** *transfer simp*

**lemma** *upd-cols-transpose*: *upd-cols* (*transpose M*) *i p* = *transpose* (*upd-rows M i p*)
**by** *transfer simp*

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

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

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

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

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

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

**lemma** *upd-rows-upd-row-swap*: *i* ∉ *I* ⟹ *upd-rows* (*upd-row M i x*) *I f* = *upd-row* (*upd-rows M I f*) *i x*
**by** *transfer* (*simp add*: *fun-eq-iff*)

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

**lemma** *upd-cols-insert-rev*: *upd-cols M* (*insert i I*) *f* = *upd-cols* (*upd-col M i* (*f i*)) *I f*

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

**lemma** *upd-cols-upd-col-swap*: $i \notin I \Longrightarrow$ *upd-cols* (*upd-col M i x*) *I f* = *upd-col* (*upd-cols M I f*) *i x*
**by** *transfer* (*simp add*: *fun-eq-iff*)

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

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

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

**lemma** *row-upd-row*[*simp*]: *row* (*upd-row M i x*) *i* = *x*
**by** (*simp add*: *row-upd-row-If*)

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

**lemma** *col-upd-col*[*simp*]: *col* (*upd-col M i x*) *i* = *x*
**by** (*simp add*: *col-upd-col-If*)

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

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

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

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

**lemma** *row-transpose*: *row* (*transpose M*) *i* = *col M i*
**by** *transfer simp*

**lemma** *col-transpose*: *col* (*transpose M*) *i* = *row M i*
**by** *transfer simp*

**lemma** *det-perm-cols*:
**fixes** $A :: 'a::comm\text{-}ring\text{-}1\hat{}'n$
**assumes** *p*: *p permutes UNIV*
**shows** *det* (*perm-cols A p*) = *of-int* (*sign p*) ∗ *det A*
**proof** (*transfer fixing*: *p*)
**fix** $A :: 'n \Rightarrow 'n \Rightarrow 'a$
**from** *p* **have** $(\sum q \mid q\ permutes\ UNIV.\ of\text{-}int\ (sign\ q) * (\prod i{\in}UNIV.\ A\ i\ (p\ (q\ i)))) =$
$(\sum q \mid q\ permutes\ UNIV.\ of\text{-}int\ (sign\ (inv\ p \circ q)) * (\prod i{\in}UNIV.\ A\ i\ (q\ i)))$
**by** (*intro sum.reindex-bij-witness*[**where** *j*=λ*q*. *p* ∘ *q* **and** *i*=λ*q*. *inv p* ∘ *q*])
(*auto simp*: *comp-assoc*[*symmetric*] *permutes-inv-o permutes-compose permutes-inv*)
**with** *p* **show** $(\sum q \mid q\ permutes\ UNIV.\ of\text{-}int\ (sign\ q) * (\prod i{\in}UNIV.\ A\ i\ (p\ (q\ i)))) =$
*of-int* (*sign p*) ∗ $(\sum p \mid p\ permutes\ UNIV.\ of\text{-}int\ (sign\ p) * (\prod i{\in}UNIV.\ A\ i\ (p\ i)))$
**by** (*auto intro*!: *sum.cong simp*: *sum-distrib-left sign-compose permutes-inv sign-inverse permutes-UNIV-permutation*)
**qed**

**lemma** *det-perm-rows*:
**fixes** $A :: 'a::comm\text{-}ring\text{-}1\hat{}'n$
**assumes** *p*: *p permutes UNIV*
**shows** *det* (*perm-rows A p*) = *of-int* (*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* (*upd-row M i* (*a* + *b*)) = *det* (*upd-row M i a*) + *det* (*upd-row M i b*)
**by** *transfer* (*simp add*: *prod.If-cases sum.distrib*[*symmetric*] *field-simps*)

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

**lemma** *det-row-uminus*: *det* (*upd-row M i* (− *a*)) = − *det* (*upd-row M i a*)
**by** (*simp add*: *vector-sneg-minus1 det-row-mul*)

**lemma** *det-row-minus*: *det* (*upd-row M i* (*a* − *b*)) = *det* (*upd-row M i a*) − *det* (*upd-row M i b*)
**unfolding** *diff-conv-add-uminus det-row-add det-row-uminus* **..**

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

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

**lemma** *det-col-add*: *det* (*upd-col M i* (*a* + *b*)) = *det* (*upd-col M i a*) + *det* (*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* (*upd-col M i* (*c* ∗*s a*)) = *c* ∗ *det* (*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* (*upd-col M i* (− *a*)) = − *det* (*upd-col M i a*)
**by** (*simp add*: *vector-sneg-minus1 det-col-mul*)

**lemma** *det-col-minus*: *det* (*upd-col M i* (*a* − *b*)) = *det* (*upd-col M i a*) − *det* (*upd-col M i b*)
**unfolding** *diff-conv-add-uminus det-col-add det-col-uminus* **..**

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

**lemma** *det-col-sum*: *det* (*upd-col M i* (∑ *s*∈*S*. *a s*)) = (∑ *s*∈*S*. *det* (*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* ≠ *i′* **shows** *col A i* = *col A i′* ⟹ *det A* = *0*
**proof** (*transfer fixing*: *i i′*)
**fix** *A* :: *′a* ⇒ *′a* ⇒ *′b* **assume** (*χ j*. *A j i*) = (*χ i*. *A i i′*)
**then have** [*simp*]: ⋀*j q*. *A j* (*Transposition.transpose i i′* (*q j*)) = *A j* (*q j*)
**by** (*simp add*: *vec-eq-iff Transposition.transpose-def*)

**let** *?p* = *λp*. *of-int* (*sign p*) ∗ (∏ *i*∈*UNIV*. *A i* (*p i*))
**let** *?s* = *λq*. *Transposition.transpose i i′* ∘ *q*
**let** *?E* = {*p*. *p permutes UNIV* ∧ *evenperm p*}

**have** [*simp*]: *inj-on ?s ?E*
**by** (*auto simp*: *inj-on-def fun-eq-iff Transposition.transpose-def*)

**note** *p* = *permutes-UNIV-permutation evenperm-comp permutes-swap-id evenperm-swap permutes-compose*
*sign-compose sign-swap-id*
**from** ‹*i* ≠ *i′*› **have** ∗: *evenperm q* **if** *q* ∉ *?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** (∑ *p* | *p permutes UNIV*. *?p p*) = (∑ *p* ∈ *?E* ∪ *?s*‘*?E*. *?p p*)
**by** (*auto simp add*: *permutes-compose permutes-swap-id intro*: ∗ *sum.cong*)
**also have** ... = (∑ *p*∈*?E*. *?p p*) + (∑ *p*∈*?s*‘*?E*. *?p p*)
**by** (*intro sum.union-disjoint*) (*auto simp*: *p* ‹*i* ≠ *i′*›)
**also have** (∑ *p*∈*?s*‘*?E*. *?p p*) = (∑ *p*∈*?E*. − *?p p*)
**using** ‹*i* ≠ *i′*› **by** (*subst sum.reindex*) (*auto intro*!: *sum.cong simp*: *p*)
**finally show** (∑ *p* | *p permutes UNIV*. *?p p*) = *0*
**by** (*simp add*: *sum-negf*)
**qed**

**lemma** *det-identical-rows*: $i \neq i' \Longrightarrow row\ A\ i = row\ A\ i' \Longrightarrow det\ A = 0$
**using** *det-identical-cols*[*of i i' transpose A*] **by** (*simp add*: *det-transpose col-transpose*)

**lemma** *det-cols-sum*:
$det\ (upd\text{-}cols\ M\ T\ (\lambda i.\ \sum s{\in}S.\ a\ i\ s)) = (\sum f{\in}T \to_E S.\ det\ (upd\text{-}cols\ M\ T\ (\lambda i.\ a\ i\ (f\ i))))$
**proof** (*induct T arbitrary*: *M rule*: *infinite-finite-induct*)
**case** (*insert i T*)
**have** $(\sum f{\in}insert\ i\ T \to_E S.\ det\ (upd\text{-}cols\ M\ (insert\ i\ T)\ (\lambda i.\ a\ i\ (f\ i)))) =$
$(\sum s{\in}S.\ \sum f{\in}T \to_E S.\ det\ (upd\text{-}cols\ (upd\text{-}col\ M\ i\ (a\ i\ s))\ T\ (\lambda i.\ a\ i\ (f\ i))))$
**unfolding** *sum.cartesian-product PiE-insert-eq* **using** ‹$i \notin T$›
**by** (*subst sum.reindex*[*OF inj-combinator*[*OF* ‹$i \notin T$›]])
(*auto intro*!: *sum.cong arg-cong*[**where** *f*=*det*] *upd-cols-cong*
*simp*: *upd-cols-insert-rev simp-implies-def*)
**also have** $\ldots = det\ (upd\text{-}col\ (upd\text{-}cols\ M\ T\ (\lambda i.\ sum\ (a\ i)\ S))\ i\ (\sum s{\in}S.\ a\ i\ s))$
**unfolding** *insert*(*3*)[*symmetric*] **by** (*simp add*: *upd-cols-upd-col-swap*[*OF* ‹$i \notin T$›] *det-col-sum*)
**finally show** *?case*
**by** (*simp add*: *upd-cols-insert*)
**qed** *auto*

**lemma** *det-rows-sum*:
$det\ (upd\text{-}rows\ M\ T\ (\lambda i.\ \sum s{\in}S.\ a\ i\ s)) = (\sum f{\in}T \to_E S.\ det\ (upd\text{-}rows\ M\ T\ (\lambda i.\ a\ i\ (f\ i))))$
**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** ¬ *f permutes UNIV*
**moreover**
**with** *bij-imp-permutes*[*of f UNIV*] **have** ¬ *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* = *upd-rows 0 UNIV* (λ*i*. ∑ *j*∈*UNIV*. *to-fun A i j* ∗*s row B j*)
**by** *transfer simp*
**moreover have** ⋀*f*. *upd-rows 0 UNIV* (λ*i*. *Square-Matrix.row B* (*f i*)) = *perm-rows B f*
**by** *transfer simp*
**moreover have** *det A* = (∑ *p* | *p permutes UNIV*. *of-int* (*sign p*) ∗ (∏ *i*∈*UNIV*. *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*^^*'b* ⇒ *'b* ⇒ *'b* ⇒ *'a*::*semiring-1*^^*'b* **is**
λ*A i j k l*. *if k* = *i* ∧ *l* = *j then 1 else if k* = *i* ∨ *l* = *j then 0 else A k l* .

**lemma** *minor-transpose*: *minor* (*transpose A*) *i j* = *transpose* (*minor A j i*)
**by** *transfer* (*auto simp*: *fun-eq-iff*)

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

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

**lemma** *row-minor*: *row* (*minor M i j*) *i* = *axis j 1*
**by** (*simp add*: *minor-eq-row-col*)

**lemma** *col-minor*: *col* (*minor M i j*) *j* = *axis i 1*
**by** (*simp add*: *minor-eq-col-row*)

**lemma** *det-minor-row′*:
*row B i* = *axis j 1* ⟹ *det* (*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** ⋀*k*. *k* ≠ *i* ⟶ *to-fun B k j* = *0*
**by** (*auto simp add*: *card-eq-0-iff*)
**with** *empty.prems* **have** *axis i 1* = *col B j*
**by** *transfer* (*auto simp*: *vec-eq-iff axis-def*)
**with** *empty.prems*[*symmetric*] **show** *?case*
**by** (*simp add*: *minor-eq-row-col*)
**next**

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

**lemma** *mult-adjugate-det*: *A* $*$ *adjugate A* = *diag* (*det A*)
**proof**−
**have** *transpose* (*transpose* (*A* $*$ *adjugate A*)) = *transpose* (*diag* (*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*::*comm-ring-1* ^^ *'n*
**shows** *poly-mat* (*charpoly A*) *A* = *0*
**proof** −

Part 1

**define** *n* **where** *n* = *CARD*(*'n*) − *1*
**then have** *d-charpoly*: *n* + *1* = *degree* (*charpoly A*) **and**
*d-adj*: *n* = *max-degree* (*adjugate* (**X** − **C** *A*))

**define** *B* **where** *B i* = *map-sq-matrix* (*λp. coeff p i*) (*adjugate* (**X** − **C** *A*)) **for** *i*
**have** *A-eq-B*: *adjugate* (**X** − **C** *A*) = ($\sum i{\leq}n$. *X*^*i* $*_S$ **C** (*B i*))

Part 2

**have** *charpoly A* $*_S$ *1* = *X* $*_S$ *adjugate* (**X** − **C** *A*) − **C** *A* $*$ *adjugate* (**X** − **C** *A*)
**also have** ... = ($\sum i{\leq}n$. *X*^(*i* + *1*) $*_S$ **C** (*B i*)) − ($\sum i{\leq}n$. *X*^*i* $*_S$ **C** (*A* $*$ *B i*))
**also have** ($\sum i{\leq}n$. *X*^(*i* + *1*) $*_S$ **C** (*B i*)) =
($\sum i{<}n$. *X*^(*i* + *1*) $*_S$ **C** (*B i*)) + *X*^(*n* + *1*) $*_S$ **C** (*B n*)
**also have** ($\sum i{\leq}n$. *X*^*i* $*_S$ **C** (*A* $*$ *B i*)) =
($\sum i{<}n$. *X*^(*i* + *1*) $*_S$ **C** (*A* $*$ *B* (*i* + *1*))) + **C** (*A* $*$ *B 0*)

**finally have** *diag-charpoly*:
*charpoly A* $*_S$ *1* = *X*^(*n* + *1*) $*_S$ **C** (*B n*) +
(∑ *i*<*n*. *X*^(*i* + *1*) $*_S$ **C** (*B i* − *A* ∗ *B* (*i* + *1*))) − **C** (*A* ∗ *B 0*)

Part 3

**let** *?p* = λ*i*. *coeff* (*charpoly A*) *i* $*_S$ *A*^*i*
**let** *?AB* = λ*i*. *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.