# Matrices, Jordan Normal Forms, and Spectral Radius Theory* 

René Thiemann and Akihisa Yamada

September 25, 2023


#### Abstract

Matrix interpretations are useful as measure functions in termination proving. In order to use these interpretations also for complexity analysis, the growth rate of matrix powers has to examined. Here, we formalized an important result of spectral radius theory, namely that the growth rate is polynomially bounded if and only if the spectral radius of a matrix is at most one.

To formally prove this result we first studied the growth rates of matrices in Jordan normal form, and prove the result that every complex matrix has a Jordan normal form by means of two algorithms: we first convert matrices into similar ones via Schur decomposition, and then apply a second algorithm which converts an upper-triangular matrix into Jordan normal form. We further showed uniqueness of Jordan normal forms which then gives rise to a modular algorithm to compute individual blocks of a Jordan normal form.

The whole development is based on a new abstract type for matrices, which is also executable by a suitable setup of the code generator. It completely subsumes our former AFP-entry on executable matrices [6], and its main advantage is its close connection to the HMArepresentation which allowed us to easily adapt existing proofs on determinants.

All the results have been applied to improve CeTA [7, 1], our certifier to validate termination and complexity proof certificates.


## Contents

## 1 Introduction

3.1 Instantiations . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

[^0]4 Vectors and Matrices ..... 21
4.1 Vectors ..... 22
4.2 Matrices ..... 31
4.3 Update Operators ..... 51
4.4 Block Vectors and Matrices ..... 52
4.5 Homomorphism properties ..... 73
5 Code Generation for Basic Matrix Operations ..... 88
6 Gauss-Jordan Algorithm ..... 93
6.1 Row Operations ..... 93
6.2 Gauss-Jordan Elimination ..... 97
7 Code Generation for Basic Matrix Operations ..... 126
8 Elementary Column Operations ..... 130
9 Determinants ..... 135
10 Code Equations for Determinants ..... 192
10.1 Properties of triangular matrices ..... 193
10.2 Algorithms for Triangulization ..... 194
10.3 Finding Non-Zero Elements ..... 198
10.4 Determinant Preserving Growth of Triangle ..... 199
10.5 Recursive Triangulization of Columns ..... 202
10.6 Triangulization ..... 203
10.7 Divisor will not be 0 ..... 204
10.8 Determinant Preservation Results ..... 206
10.9 Determinant Computation ..... 209
11 Converting Matrices to Strings ..... 211
12 Characteristic Polynomial ..... 212
13 Jordan Normal Form ..... 226
13.1 Application for Complexity ..... 243
14 Missing Vector Spaces ..... 244
15 Matrices as Vector Spaces ..... 274
16 Gram-Schmidt Orthogonalization ..... 302
16.1 Orthogonality with Conjugates ..... 302
16.2 The Algorithm ..... 304
16.3 Properties of the Algorithms ..... 304
17 Schur Decomposition ..... 313
18 Computing Jordan Normal Forms ..... 327
18.1 Pseudo Code Algorithm ..... 327
18.2 Real Algorithm ..... 328
18.3 Preservation of Dimensions ..... 331
18.4 Properties of Auxiliary Algorithms ..... 333
18.5 Proving Similarity ..... 337
18.6 Invariants for Proving that Result is in JNF ..... 343
18.7 Alternative Characterization of identify-blocks in Presence of local.ev-block ..... 349
18.8 Proving the Invariants ..... 352
18.9 Combination with Schur-decomposition ..... 394
18.10Application for Complexity ..... 396
19 Code Equations for All Algorithms ..... 397
20 Strassen's algorithm for matrix multiplication. ..... 398
21 Strassen's Algorithm as Code Equation ..... 404
22 Comparison of Matrices ..... 404
23 Matrix Conversions ..... 423
24 Derivation Bounds ..... 425
25 Complexity Carrier ..... 426
26 Converting Arctic Numbers to Strings ..... 428
27 Application: Complexity of Matrix Orderings ..... 429
27.1 Locales for Carriers of Matrix Interpretations and Polynomial Orders ..... 430
27.2 The Integers as Carrier ..... 431
27.3 The Rational and Real Numbers as Carrier ..... 432
27.4 The Arctic Numbers as Carrier ..... 434
28 Matrix Kernel ..... 435
29 Jordan Normal Form - Uniqueness ..... 461
30 Spectral Radius Theory ..... 468
31 Missing Lemmas of List ..... 472
32 Matrix Rank ..... 473
33 Subadditivity of rank ..... 481
34 Missing Lemmas of Sublist ..... 488
35 Pick ..... 490
36 Sublist ..... 495
37 weave ..... 497
38 Submatrices ..... 507
39 Submatrix ..... 507
40 Rank and Submatrices ..... 508

## 1 Introduction

The spectral radius of a square, complex valued matrix $A$ is defined as the largest norm of some eigenvalue $c$ with eigenvector $v$. It is a central notion to estimate how the values in $A^{n}$ for increasing $n$. If the spectral radius is larger than 1 , clearly the values grow exponentially, since then $A^{n} \cdot v=c^{n} \cdot v$ becomes exponentially large.
The other results, namely that the values in $A^{n}$ are bounded by a constant, if the spectral radius is smaller than 1 , and that there is a polynomial bound if the spectral radius is exactly 1 are only immediate for matrices which have an eigenbasis, a precondition which is not satisfied by every matrix.
However, these results are derivable via Jordan normal forms (JNFs): If $J$ is a JNF of $A$, then the growth rates of $A^{n}$ and $J^{n}$ are related by a constant as $A$ and $J$ are similar matrices. And for the values in $J^{n}$ there is a closed formula which gives the desired complexity bounds. To be more precise, the values in $J^{n}$ are bounded by $\mathcal{O}\left(\left.|c|\right|^{n} \cdot n^{k-1}\right)$ where $k$ is the size of the largest block of an eigenvalue $c$ which has maximal norm w.r.t. the set of all eigenvalues. And since every complex matrix has a JNF, we can derive the polynomial (resp. constant bounds), if the spectral radius is 1 (resp. smaller than 1).
These results are already applied in current complexity tools, and the motivation of this development was to extend our certifier CeTA to be able to validate corresponding complexity proofs. To this end, we formalized the following main results:

- an algorithm to compute the characteristic polynomial, since the eigenvalues are exactly the roots of this polynomial;
- the complexity bounds for JNFs; and
- an algorithm which computes JNFs for every matrix, provided that the list of eigenvalues is given. With the help of the fundamental theorem of algebra this shows that every complex matrix has a JNF.

Since CeTA is generated from Isabelle/HOL via code-generation, all the algorithms and results need to be available at code-generation time. Especially there is no possibility to create types on the fly which are chosen to fit the matrix dimensions of the input. To this end, we cannot use the matrix-representation of HOL multivariate analysis (HMA).

Instead, we provide a new matrix library which is based on HOL-algebra with its explicit carriers. In contrast to our earlier development [6], we do not immediately formalize everything as lists of lists, but use a more mathematical notion as triples of the form (dimension, dimension, characteristicfunction). This makes reasoning very similar to HMA, and a suitable implementation type can be chosen afterwards: we provide one via immutable arrays (we use IArray's from the HOL library), but one can also think of an implementation for sparse matrices, etc. Even the infinite carrier itself is executable where we rely upon Lochbihler's container framework [4] to have different set representations at the same time.

As a consequence of not using HMA, we could not directly reuse existing algorithms which have been formalized for this representation. For instance, we formalized our own version of Gauss-Jordan elimination which is not very different to the one of Divasón and Aransay in [2]: both define row-echelon form and apply elementary row transformations. Whereas Gauss-Jordan elimination has been developed from scratch as a case-study to see how suitable our matrix representation is, in other cases we often just copied and adjusted existing proofs from HMA. For instance, most of the library for determinants has been copied from the Isabelle distribution and adapted to our matrix representation.

As a result of our formalization, CeTA is now able to check polynomial bounds for matrix interpretations [3].

## 2 Material missing in the distribution

This theory provides some definitions and lemmas which we did not find in the Isabelle distribution.

```
theory Missing-Misc
    imports
        HOL-Library.FuncSet
        HOL-Combinatorics.Permutations
```


## begin

declare finite-image-iff [simp]
lemma inj-on-finite:
$\langle$ finite $(f$ ' $A) \longleftrightarrow$ finite $A\rangle$ if $\langle\operatorname{inj-on~} f A\rangle$
using that by (fact finite-image-iff)
The following lemma is slightly generalized from Determinants.thy in HMA
lemma finite-bounded-functions:
assumes $f S$ : finite $S$
shows finite $T \Longrightarrow$ finite $\{f .(\forall i \in T . f i \in S) \wedge(\forall i . i \notin T \longrightarrow f i=i)\}$
proof (induct $T$ rule: finite-induct)
case empty
have th: $\{f . \forall i . f i=i\}=\{i d\}$
by auto
show ?case
by (auto simp add: th)
next
case (insert a $T$ )
let ?f $=\lambda(y, g)$. if $i=a$ then $y$ else $g i$
let ? $S=$ ? $f$ ' $(S \times\{f .(\forall i \in T . f i \in S) \wedge(\forall i . i \notin T \longrightarrow f i=i)\})$
have $? S=\{f .(\forall i \in$ insert a $T . f i \in S) \wedge(\forall i . i \notin$ insert a $T \longrightarrow f i=i)\}$
apply (auto simp add: image-iff)
apply (rule-tac $x=x a$ in bexI)
apply (rule-tac $x=\lambda i$. if $i=a$ then $i$ else $x i$ in exI)
apply (insert insert, auto) done
with finite-imageI[OF finite-cartesian-product[OF fS insert.hyps(3)], of ?f]
show ?case
by metis
qed
lemma finite-bounded-functions':
assumes $f S$ : finite $S$
shows finite $T \Longrightarrow$ finite $\{f .(\forall i \in T . f i \in S) \wedge(\forall i . i \notin T \longrightarrow f i=j)\}$
proof (induct $T$ rule: finite-induct)
case empty
have th: $\{f . \forall i . f i=j\}=\{(\lambda x . j)\}$
by auto
show ?case
by (auto simp add: th)
next
case (insert a $T$ )
let ?f $=\lambda(y, g)$. if $i=a$ then $y$ else $g i$
let ?S $=$ ? f' $(S \times\{f .(\forall i \in T . f i \in S) \wedge(\forall i . i \notin T \longrightarrow f i=j)\})$
have ? $S=\{f .(\forall i \in$ insert a $T . f i \in S) \wedge(\forall i . i \notin$ insert a $T \longrightarrow f i=j)\}$ apply (auto simp add: image-iff)

```
    apply (rule-tac x=x a in bexI)
    apply (rule-tac x = \lambdai. if i=a then j else x i in exI)
    apply (insert insert, auto)
    done
    with finite-imageI[OF finite-cartesian-product[OF fS insert.hyps(3)], of ?f]
    show ?case
    by metis
qed
lemma permutes-less [simp]:
    assumes p:p permutes {0..<(n :: nat)}
    shows
        i<n\Longrightarrowpi<n
    i<n\Longrightarrow inv pi<n
    p(inv pi)=i
    inv p (pi) =i
    using assms
    by (simp-all add: permutes-inverses permutes-nat-less permutes-nat-inv-less)
lemma permutes-prod:
    assumes p: p permutes S
    shows (\prods\inS.f(ps)s)=(\prods\inS.fs(inv p s))
        (is ?l = ?r)
    using assms by (fact prod.permutes-inv)
lemma permutes-sum:
    assumes p: p permutes S
    shows (\sums\inS.f(ps)s)=(\sums\inS.fs(invps))
        (is ?l = ?r)
    using assms by (fact sum.permutes-inv)
context
    fixes }A\mathrm{ :: 'a set
        and }B:: 'b se
        and a-to-b :: 'a m'b
        and b-to-a :: 'b }\mp@subsup{|}{}{\prime
    assumes ab: ^a. a \inA\Longrightarrowa-to-b a \in B
        and ba: ^b. b G B\Longrightarrowb-to-a b\inA
        and ab-ba: ^a.a\inA\Longrightarrowb-to-a (a-to-b a)=a
    and ba-ab: \bigwedge b. b\inB\Longrightarrowa-to-b (b-to-a b)=b
begin
qualified lemma permutes-memb: fixes p:: 'b b 'b
    assumes p: p permutes B
    and a:a\inA
    defines ip\equiv Hilbert-Choice.inv p
    shows }a\inA a-to-b a\inB ip (a-to-b a)\inB p (a-to-b a) \in
        b-to-a (p (a-to-b a)) \inA b-to-a (ip (a-to-b a)) \inA
proof -
```

```
    let ?b = a-to-b a
    from p have ip: ip permutes B unfolding ip-def by (rule permutes-inv)
    note in-ip = permutes-in-image[OF ip]
    note in-p = permutes-in-image[OF p]
    show a: a\inA by fact
    show b: ?b }\inB\mathrm{ by (rule ab[OF a])
    show pb: p ?b B B unfolding in-p by (rule b)
    show ipb:ip ?b\inB unfolding in-ip by (rule b)
    show b-to-a (p ?b) \inA by (rule ba[OF pb])
    show b-to-a (ip ?b) \inA by (rule ba[OF ipb])
qed
lemma permutes-bij-main:
    {p.p permutes }A}\supseteq(\lambdapa. if a\inA then b-to-a(p(a-to-b a)) else a)'{p.
permutes B}
    (is ?A \supseteq?f'?B)
proof
    note d}=\mathrm{ permutes-def
    let ?g=\lambda qb. if b\inB then a-to-b (q(b-to-a b)) else b
    let ?inv = Hilbert-Choice.inv
    fix }
    assume p:p\in?f`?B
    then obtain q}\mathrm{ where q:q permutes B and p: p=?f q by auto
    let ?iq=? ?inv q
    from q have iq: ?iq permutes B by (rule permutes-inv)
    note in-iq = permutes-in-image[OF iq]
    note in-q = permutes-in-image[OF q]
    have qiB: \bigwedge b.b\inB\Longrightarrowq(?iq b)=b using q by (rule permutes-inverses)
    have iqB: \bigwedge b.b\inB\Longrightarrow? ?iq(qb)=b using q by (rule permutes-inverses)
    from q[unfolded d]
    have q1: \ b. b\not\inB\Longrightarrowqb=b
    and q2: \bigwedgeb. \exists! b'. q b ' = b by auto
    note memb = permutes-memb[OF q]
    show p}\in
    proof (rule, intro conjI impI allI, force)
    fix a
    show }\exists\mathrm{ ! a'. ?f q a' = a
    proof (cases a }\inA\mathrm{ )
            case True
            note a=memb[OF True]
            let ?a = b-to-a (?iq (a-to-b a))
            show ?thesis
            proof
                show ?f q ?a = a using a by (simp add: ba-ab qiB ab-ba)
            next
                fix }\mp@subsup{a}{}{\prime
            assume id:?f q a'=a
            show }\mp@subsup{a}{}{\prime}=?,
            proof (cases a' }\mp@subsup{a}{}{\prime}\inA
```

```
                case False
                thus ?thesis using id a by auto
                next
                    case True
                    note }\mp@subsup{a}{}{\prime}=memb[OF this
                    from id True have b-to-a (q (a-to-b a')) =a by simp
                    from arg-cong[OF this, of a-to-b] a' a
                    have q(a-to-b a')=a-to-b a by (simp add: ba-ab)
                    from arg-cong[OF this, of ?iq]
                    have a-to-b a'=? ?iq (a-to-b a) unfolding iqB[OF a'(2)].
                    from arg-cong[OF this, of b-to-a] show ?thesis unfolding ab-ba[OF True]
•
            qed
        qed
        next
        case False note a= this
        show ?thesis
        proof
            show ?f qa =a using a by simp
        next
            fix }\mp@subsup{a}{}{\prime
            assume id:?f q a' =a
            show a'=a
            proof (cases a' }\inA
                case False
                with id show ?thesis by simp
            next
                case True
                note a'= memb[OF True]
                with id False show ?thesis by auto
                qed
            qed
        qed
    qed
qed
end
lemma permutes-bij': assumes ab: \bigwedge a. a\inA\Longrightarrowa-to-b a 
    and ba: \bigwedge b. b B B\Longrightarrowb-to-a b\inA
    and ab-ba: \bigwedge a. a \inA\Longrightarrowb-to-a (a-to-b a)=a
    and ba-ab: \bigwedge b. b\inB\Longrightarrowa-to-b (b-to-a b)=b
    shows {p.p permutes A} = (\lambda p a. if a\inA then b-to-a (p (a-to-b a)) else a)`
{p.p permutes B}
    (is ?A = ?f`?B)
proof -
    note one-dir = ab ba ab-ba ba-ab
    note other-dir = ba ab ba-ab ab-ba
    let ?g = (\lambda p b. if b G B then a-to-b (p (b-to-a b)) else b)
```

```
define }PA\mathrm{ where }PA=?
define f}\mathrm{ where f=?f
define }g\mathrm{ where }g=?
{
    fix p
    assume p\inPA
    hence p: p permutes A unfolding PA-def by simp
    from p[unfolded permutes-def] have pnA: \a. a\not\inA\Longrightarrowpa=a by auto
    have ?f (?g p)=p
    proof (rule ext)
        fix }
        show ?f (?g p) a = pa
        proof (cases a }\inA\mathrm{ )
            case False
            thus ?thesis by (simp add: pnA)
        next
            case True note a= this
            hence pa\inA unfolding permutes-in-image[OF p].
            thus ?thesis using a by (simp add: ab-ba ba-ab ab)
        qed
    qed
    hence f (g p) = p unfolding f-def g-def .
}
hence f' g'PA=PA by force
hence id: ?f '?g'?A = ?A unfolding PA-def f-def g-def .
have ?f '?B}\subseteq\mathrm{ ?A by (rule permutes-bij-main[OF one-dir])
moreover have ?g'?A\subseteq?B by (rule permutes-bij-main[OF ba ab ba-ab ab-ba])
hence ?f '?g '?A}\subseteq\mathrm{ ?f '?B by auto
hence ?A\subseteq?f ' ?B unfolding id .
    ultimately show ?thesis by blast
qed
lemma permutes-others:
    assumes p:p permutes S and x:x\not\inS shows p x = x
    using px by (rule permutes-not-in)
lemma inj-on-nat-permutes: assumes i: inj-on f (S :: nat set)
    and fS:f}\inS->
    and fin: finite S
    and f:\bigwedgei.i\not\inS\Longrightarrowfi=i
    shows f permutes }
    unfolding permutes-def
proof (intro conjI allI impI, rule f)
    fix y
    from endo-inj-surj[OF fin-i] fS have fs: f' }S=S\mathrm{ by auto
    show }\exists\mathrm{ ! x. f x = y
    proof (cases y \inS)
    case False
    thus ?thesis by (intro ex1I[of-y], insert fS f, auto)
```

```
    next
        case True
        with fs obtain x where }x:x\inS\mathrm{ and fx: fx=y by force
        show ?thesis
        proof (rule ex1I, rule fx)
            fix }\mp@subsup{x}{}{\prime
            assume fx': f x ' = y
            with True f[of x] have }\mp@subsup{x}{}{\prime}\inS\mathrm{ by metis
            from inj-onD[OF i fx[folded fx] x this]
            show }\mp@subsup{x}{}{\prime}=x\mathrm{ by simp
    qed
    qed
qed
abbreviation (input) signof ::<(nat => nat) => ' }a\mathrm{ :: ring-1>
    where <signof p \equivof-int (sign p)〉
lemma signof-id:
    signof id = 1
    signof }(\lambdax.x)=
    by simp-all
lemma signof-inv: finite S \Longrightarrow permutes S \ signof (inv p) = signof p
    by (simp add: permutes-imp-permutation sign-inverse)
lemma signof-pm-one: signof p}\in{1,-1
    by (simp add: sign-def)
lemma signof-compose:
    assumes p permutes {0..<(n :: nat)}
    and q permutes {0 ..<(m :: nat)}
    shows signof (por ) = signof p * signof q
proof -
    from assms have pp: permutation p permutation q
        by (auto simp: permutation-permutes)
    then show signof ( }p\mathrm{ o q) = signof p* signof q
        by (simp add: sign-compose)
qed
end
```


## 3 Missing Ring

This theory contains several lemmas which might be of interest to the Isabelle distribution.

```
theory Missing-Ring
    imports
    Missing-Misc
```

HOL-Algebra.Ring
begin
context ordered-cancel-semiring
begin
subclass ordered-cancel-ab-semigroup-add ..
end
partially ordered variant
class ordered-semiring-strict $=$ semiring + comm-monoid-add + ordered-cancel-ab-semigroup-add $+$
assumes mult-strict-left-mono: $a<b \Longrightarrow 0<c \Longrightarrow c * a<c * b$
assumes mult-strict-right-mono: $a<b \Longrightarrow 0<c \Longrightarrow a * c<b * c$
begin
subclass semiring-0-cancel ..
subclass ordered-semiring
proof
fix $a b c::{ }^{\prime} a$
assume $A: a \leq b 0 \leq c$
from $A$ show $c * a \leq c * b$
unfolding le-less
using mult-strict-left-mono by (cases $c=0$ ) auto
from $A$ show $a * c \leq b * c$
unfolding le-less
using mult-strict-right-mono by (cases $c=0$ ) auto
qed
lemma mult-pos-pos[simp]: $0<a \Longrightarrow 0<b \Longrightarrow 0<a * b$
using mult-strict-left-mono [of 0 b al by simp
lemma mult-pos-neg: $0<a \Longrightarrow b<0 \Longrightarrow a * b<0$
using mult-strict-left-mono $\left[\begin{array}{lll}o f & b & 0\end{array}\right]$ by simp
lemma mult-neg-pos: $a<0 \Longrightarrow 0<b \Longrightarrow a * b<0$

Legacy - use mult-neg-pos
lemma mult-pos-neg2: $0<a \Longrightarrow b<0 \Longrightarrow b * a<0$
by (drule mult-strict-right-mono $\left[\begin{array}{lll}\text { of } & 0\end{array}\right]$, auto)
Strict monotonicity in both arguments
lemma mult-strict-mono:
assumes $a<b$ and $c<d$ and $0<b$ and $0 \leq c$
shows $a * c<b * d$
using assms apply (cases $c=0$ )

```
apply (simp)
apply (erule mult-strict-right-mono [THEN less-trans])
apply (force simp add: le-less)
apply (erule mult-strict-left-mono, assumption)
done
```

This weaker variant has more natural premises

## lemma mult-strict-mono':

    assumes \(a<b\) and \(c<d\) and \(0 \leq a\) and \(0 \leq c\)
    shows \(a * c<b * d\)
    by (rule mult-strict-mono) (insert assms, auto)
lemma mult-less-le-imp-less:
assumes $a<b$ and $c \leq d$ and $0 \leq a$ and $0<c$
shows $a * c<b * d$
using assms apply (subgoal-tac $a * c<b * c$ )
apply (erule less-le-trans)
apply (erule mult-left-mono)
apply simp
apply (erule mult-strict-right-mono)
apply assumption
done
lemma mult-le-less-imp-less:
assumes $a \leq b$ and $c<d$ and $0<a$ and $0 \leq c$
shows $a * c<b * d$
using assms apply (subgoal-tac $a * c \leq b * c$ )
apply (erule le-less-trans)
apply (erule mult-strict-left-mono)
apply simp
apply (erule mult-right-mono)
apply simp
done
end
class ordered-idom $=$ idom + ordered-semiring-strict +
assumes zero-less-one [simp]: $0<1$ begin
subclass semiring-1 ..
subclass comm-ring-1 ..
subclass ordered-ring ..
subclass ordered-comm-semiring by (unfold-locales, fact mult-left-mono)
subclass ordered-ab-semigroup-add ..
lemma of-nat-ge- $0[$ simp $]$ : of-nat $x \geq 0$
proof (induct $x$ )
case 0 thus ? case by auto
next case (Suc $x$ )

```
    hence 0 \leq of-nat x by auto
    also have of-nat x < of-nat (Suc x) by auto
    finally show?case by auto
qed
lemma of-nat-eq-0[simp]:of-nat }x=0\longleftrightarrowx=
proof(induct x,simp)
    case (Suc x)
        have of-nat (Suc x) > 0 apply(rule le-less-trans[of - of-nat x]) by auto
        thus ?case by auto
qed
lemma inj-of-nat:inj (of-nat :: nat }=>\mp@subsup{}{}{\prime}=\mathrm{ ')
proof(rule injI)
    fix x y show of-nat }x=of-nat y\Longrightarrowx=
    proof (induct x arbitrary: y)
    case 0 thus ?case
            proof (induct y)
                case 0 thus ?case by auto
                next case (Suc y)
                        hence of-nat (Suc y) = 0 by auto
                hence Suc y = 0 unfolding of-nat-eq-0 by auto
                    hence False by auto
                    thus ?case by auto
            qed
        next case (Suc x)
            thus ?case
            proof (induct y)
                case 0
                    hence of-nat (Suc x)=0 by auto
                    hence Suc x = 0 unfolding of-nat-eq-0 by auto
                    hence False by auto
                    thus ?case by auto
                next case (Suc y) thus ?case by auto
            qed
        qed
qed
subclass ring-char-0 by(unfold-locales, fact inj-of-nat)
end
context comm-monoid
begin
lemma finprod-reindex-bij-betw: bij-betw \(h\) S T
\(\Longrightarrow g \in h^{\prime} S \rightarrow\) carrier \(G\)
```

$\Longrightarrow$ finprod $G(\lambda x . g(h x)) S=$ finprod $G g T$
using finprod-reindex[of $g h S$ ] unfolding bij-betw-def by auto
lemma finprod-reindex-bij-witness:
assumes witness:
^a. $a \in S \Longrightarrow i(j a)=a$
$\bigwedge a . a \in S \Longrightarrow j a \in T$
$\bigwedge b . b \in T \Longrightarrow j(i b)=b$
$\bigwedge b . b \in T \Longrightarrow i b \in S$
assumes eq:
$\bigwedge a . a \in S \Longrightarrow h(j a)=g a$
assumes $g: g \in S \rightarrow$ carrier $G$
and $h: h \in j$ ' $S \rightarrow$ carrier $G$
shows finprod $G$ g $S=$ finprod $G h T$
proof -
have b: bij-betw j $S T$
using bij-betw-byWitness[where $A=S$ and $f=j$ and $f^{\prime}=i$ and $\left.A^{\prime}=T\right]$ witness
by auto
have fp: finprod $G g S=$ finprod $G(\lambda x . h(j x)) S$
by (rule finprod-cong, insert eq $g$, auto)
show ?thesis
using finprod-reindex-bij-betw[ $O F \quad b \quad h]$ unfolding $f p$.
qed
end
lemmas (in abelian-monoid) finsum-reindex-bij-witness $=$ add.finprod-reindex-bij-witness
locale csemiring $=$ semiring + comm-monoid $R$
context cring
begin
sublocale csemiring ..
end
lemma (in comm-monoid) finprod-one':
$(\bigwedge a . a \in A \Longrightarrow f a=\mathbf{1}) \Longrightarrow$ finprod $G f A=\mathbf{1}$
by (induct A rule: infinite-finite-induct, auto)
lemma (in comm-monoid) finprod-split:
finite $A \Longrightarrow f^{\prime} A \subseteq$ carrier $G \Longrightarrow a \in A \Longrightarrow$ finprod $G f A=f a \otimes$ finprod $G f$
( $A-\{a\}$ )
by (rule $\operatorname{trans}[O F \operatorname{trans}[O F-$ finprod-Un-disjoint $[$ of $\{a\} A-\{a\} f]]]$, auto, rule arg-cong $[$ of -- finprod $G f]$, auto)
lemma (in comm-monoid) finprod-finprod:
finite $A \Longrightarrow$ finite $B \Longrightarrow(\bigwedge a b, a \in A \Longrightarrow b \in B \Longrightarrow g a b \in$ carrier $G) \Longrightarrow$ finprod $G(\lambda a$. finprod $G(g a) B) A=$ finprod $G(\lambda(a, b) . g a b)(A \times B)$
proof (induct A rule: finite-induct)
case (insert $a^{\prime} A$ )

```
    note \(I H=\) this
    let ?l \(=\left(\bigotimes a \in\right.\) insert \(a^{\prime} A\). finprod \(G\left(\begin{array}{ll}g & a) B\end{array}\right)\)
    let ? \(r=\left(\bigotimes a \in\right.\) insert \(a^{\prime} A \times B\). case \(a\) of \(\left.(a, b) \Rightarrow g a b\right)\)
    have ?l \(=\) finprod \(G\left(\begin{array}{ll}g & \left.a^{\prime}\right) B \otimes\left(\otimes a \in A \text {. finprod } G\left(\begin{array}{ll}g & a\end{array}\right) B\right), ~(\$)\end{array}\right.\)
    using \(I H\) by \(\operatorname{simp}\)
    also have \((\bigotimes a \in A\). finprod \(G(g a) B)=\) finprod \(G(\lambda(a, b) . g\) a \(b)(A \times B)\)
    by (rule \(I H(3)\), insert \(I H\), auto)
    finally have \(i d l: ? l=\) finprod \(G\left(g a^{\prime}\right) B \otimes\) finprod \(G(\lambda(a, b) . g a b)(A \times B)\).
    from \(I H\) (2) have insert \(a^{\prime} A \times B=\left\{a^{\prime}\right\} \times B \cup A \times B\) by auto
    hence ? \(r=\left(\bigotimes a \in\left\{a^{\prime}\right\} \times B \cup A \times B\right.\). case \(a\) of \(\left.(a, b) \Rightarrow g a b\right)\) by simp
    also have \(\ldots=\left(\otimes a \in\left\{a^{\prime}\right\} \times B\right.\). case \(a\) of \(\left.(a, b) \Rightarrow g a b\right) \otimes(\otimes a \in A \times B\).
case \(a\) of \((a, b) \Rightarrow g a b)\)
    by (rule finprod-Un-disjoint, insert \(I H\), auto)
    also have \(\left(\otimes a \in\left\{a^{\prime}\right\} \times B\right.\). case \(a\) of \(\left.(a, b) \Rightarrow g a b\right)=\) finprod \(G\left(g a^{\prime}\right) B\)
    using \(I H(4) I H(5)\)
    proof (induct \(B\) rule: finite-induct)
    case (insert \(b^{\prime} B\) )
    note \(I H=\) this
    have \(i d:\left(\otimes a \in\left\{a^{\prime}\right\} \times B\right.\). case \(a\) of \(\left.(a, b) \Rightarrow g a b\right)=\) finprod \(G\left(g a^{\prime}\right) B\)
        by (rule \(\operatorname{IH}(3)[O F \operatorname{IH}(4)]\), auto)
    have id2: \(\bigwedge x F\). \(\left\{a^{\prime}\right\} \times\) insert \(x F=\) insert \(\left(a^{\prime}, x\right)\left(\left\{a^{\prime}\right\} \times F\right)\) by auto
    have id3: \(\left(\otimes a \in \operatorname{insert}\left(a^{\prime}, b^{\prime}\right)\left(\left\{a^{\prime}\right\} \times B\right)\right.\). case \(a\) of \(\left.(a, b) \Rightarrow g a b\right)\)
            \(=g a^{\prime} b^{\prime} \otimes\left(\otimes a \in\left(\left\{a^{\prime}\right\} \times B\right)\right.\). case \(a\) of \(\left.(a, b) \Rightarrow g a b\right)\)
            by (rule trans[OF finprod-insert], insert IH, auto)
    show ?case unfolding id2 id3 id
        by (rule sym, rule finprod-insert, insert IH, auto)
    qed \(\operatorname{simp}\)
    finally have \(i d r: ? r=\) finprod \(G\left(g a^{\prime}\right) B \otimes(\bigotimes a \in A \times B\). case a of \((a, b) \Rightarrow g\)
\(a b)\).
    show ?case unfolding idl idr ..
qed \(\operatorname{simp}\)
lemma (in comm-monoid) finprod-swap:
    assumes finite \(A\) finite \(B \bigwedge a b . a \in A \Longrightarrow b \in B \Longrightarrow g a b \in\) carrier \(G\)
    shows finprod \(G(\lambda(b, a) . g\) a \(b)(B \times A)=\) finprod \(G(\lambda(a, b) . g a b)(A \times B)\)
proof -
    have \([\) simp \(]:(\lambda(a, b) .(b, a))\) ' \((A \times B)=B \times A\) by auto
    have \([\) simp \(]:(\lambda x\). case case \(x\) of \((a, b) \Rightarrow(b, a)\) of \((a, b) \Rightarrow g b a)=(\lambda(a, b)\).
\(g a b)\)
            by (intro ext, auto)
    show ?thesis
        by (rule trans[OF trans[OF - finprod-reindex[of \(\lambda(a, b) . g b a \lambda(a, b) .(b, a)]]]\),
        insert assms, auto simp: inj-on-def)
qed
```

lemma (in comm-monoid) finprod-finprod-swap:
finite $A \Longrightarrow$ finite $B \Longrightarrow(\bigwedge a b . a \in A \Longrightarrow b \in B \Longrightarrow g a b \in$ carrier $G) \Longrightarrow$ finprod $G\left(\lambda a\right.$. finprod $G\left(\begin{array}{ll}g & a) \\ )\end{array}\right) A=$ finprod $G(\lambda b$. finprod $G(\lambda a . g a b)$
A) $B$
using finprod-finprod $[$ of $A B]$ finprod-finprod $[$ of $B A]$ finprod-swap $[o f ~ A ~ B]$ by $\operatorname{simp}$

```
lemmas (in semiring) finsum-zero' \(=\) add.finprod-one \({ }^{\prime}\)
lemmas (in semiring) finsum-split \(=a d d\). finprod-split
lemmas (in semiring) finsum-finsum-swap \(=\) add.finprod-finprod-swap
lemma (in csemiring) finprod-zero:
    finite \(A \Longrightarrow f \in A \rightarrow\) carrier \(R \Longrightarrow \exists a \in A . f a=\mathbf{0}\)
        \(\Longrightarrow\) finprod \(R f A=\mathbf{0}\)
proof (induct A rule: finite-induct)
    case (insert a A)
    from finprod-insert[OF insert(1-2), of f] insert(4)
    have ins: finprod \(R f(\) insert a \(A)=f a \otimes\) finprod \(R f A\) by simp
    have \(f A\) : finprod \(R f A \in\) carrier \(R\)
        by (rule finprod-closed, insert insert, auto)
    show ?case
    proof (cases fa=0)
        case True
        with \(f A\) show ?thesis unfolding ins by simp
    next
        case False
        with insert(5) have \(\exists a \in A . f a=\mathbf{0}\) by auto
        from \(\operatorname{insert}(3)[O F-t h i s]\) insert have finprod \(R f A=\mathbf{0}\) by auto
        with insert show ?thesis unfolding ins by auto
    qed
qed \(\operatorname{simp}\)
lemma (in semiring) finsum-product:
    assumes \(A\) : finite \(A\) and \(B\) : finite \(B\)
    and \(f: f \in A \rightarrow\) carrier \(R\) and \(g: g \in B \rightarrow\) carrier \(R\)
    shows finsum \(R f A \otimes\) finsum \(R g B=(\bigoplus i \in A . \bigoplus j \in B . f i \otimes g j)\)
    unfolding finsum-ldistr[OF A finsum-closed \([\) OF \(g] f]\)
proof (rule finsum-cong' \([\) OF refl \(]\) )
    fix \(a\)
    assume \(a: a \in A\)
    show \(f a \otimes\) finsum \(R g B=(\bigoplus j \in B . f a \otimes g j)\)
    by (rule finsum-rdistr \([O F B-g]\), insert a \(f\), auto)
qed (insert \(f g B\), auto intro: finsum-closed)
lemma (in semiring) Units-one-side- \(I\) :
    \(a \in\) carrier \(R \Longrightarrow p \in\) Units \(R \Longrightarrow p \otimes a=\mathbf{1} \Longrightarrow a \in\) Units \(R\)
    \(a \in\) carrier \(R \Longrightarrow p \in\) Units \(R \Longrightarrow a \otimes p=\mathbf{1} \Longrightarrow a \in\) Units \(R\)
    by (metis Units-closed Units-inv-Units Units-l-inv inv-unique)+
```

```
lemma permutes-funcset: p permutes }A\Longrightarrow(p`A->B)=(A->B
    by (simp add: permutes-image)
context comm-monoid
begin
lemma finprod-permute:
    assumes p: p permutes S
    and f:f\inS-> carrier G
    shows finprod G f S = finprod G (f\circp)S
proof -
    from <p permutes S〉 have inj p
        by (rule permutes-inj)
    then have inj-on pS
        by (auto intro: subset-inj-on)
    from finprod-reindex[OF - this, unfolded permutes-image [OF p],OF f]
    show ?thesis unfolding o-def .
qed
lemma finprod-singleton-set[simp]: assumes fa\incarrier G
    shows finprod Gf{a}=fa
proof -
    have finprod Gf{a}=fa\otimes finprod Gf {}
        by (rule finprod-insert, insert assms, auto)
    also have ... = f a using assms by auto
    finally show ?thesis .
qed
end
lemmas (in semiring) finsum-permute = add.finprod-permute
lemmas (in semiring) finsum-singleton-set = add.finprod-singleton-set
context cring
begin
lemma finsum-permutations-inverse:
    assumes f:f\in{p.p permutes S} }->\mathrm{ carrier R
    shows finsum R f {p. p permutes S} = finsum R (\lambdap.f(Hilbert-Choice.inv p))
{p.p permutes S}
    (is ?lhs = ?rhs)
proof -
    let ?inv = Hilbert-Choice.inv
    let ?S = {p.p permutes S}
    have th0: inj-on?inv?S
    proof (auto simp add: inj-on-def)
        fix qr
        assume q: q permutes S
            and r:r permutes S
            and qr: ?inv q= ?inv r
            then have ?inv (?inv q)=?inv (?inv r)
```

```
        by simp
    with permutes-inv-inv[OF q] permutes-inv-inv[OF r] show q}=
        by metis
    qed
    have th1: ?inv'?S = ?S
    using image-inverse-permutations by blast
    have th2: ?rhs = finsum R (f\circ?inv) ?S
    by (simp add: o-def)
    from finsum-reindex[OF - th0, of f] show ?thesis unfolding th1 th2 using f .
qed
lemma finsum-permutations-compose-right: assumes q:q permutes S
    and *:f\in{p.p permutes S} }->\mathrm{ carrier R
    shows finsum R f {p.p permutes S}= finsum R (\lambdap.f(p\circq)){p.p permutes
S}
    (is ?lhs = ?rhs)
proof -
    let ?S = {p.p permutes S }
    let ?inv = Hilbert-Choice.inv
    have th0:?rhs = finsum R (f\circ ( }\lambdap.p\circq))?
        by (simp add: o-def)
    have th1: inj-on ( }\lambdap.p\circq)?
    proof (auto simp add: inj-on-def)
        fix }p
        assume p permutes S
            and r:r permutes S
            and rp:p\circq=r\circq
    then have p\circ(q\circ\mathrm{ ?inv q) =r ○( }q\circ\mathrm{ ?inv q)}
            by (simp add:o-assoc)
    with permutes-surj[OF q, unfolded surj-iff] show p=r
        by simp
    qed
    have th3:(\lambdap.p\circq)'?S=?S
        using image-compose-permutations-right[OF q] by auto
    from finsum-reindex[OF - th1, of f]
    show ?thesis unfolding th0 th1 th3 using * .
qed
end
end
theory Conjugate
    imports HOL.Complex HOL-Library.Complex-Order
begin
class conjugate =
    fixes conjugate :: ' }a=>\mathrm{ ' 'a
    assumes conjugate-id[simp]: conjugate (conjugate a)=a
```

and conjugate-cancel-iff[simp]: conjugate $a=$ conjugate $b \longleftrightarrow a=b$

```
class conjugatable-ring = ring + conjugate +
    assumes conjugate-dist-mul: conjugate ( }a*b)=\mathrm{ conjugate a* conjugate b
    and conjugate-dist-add: conjugate ( }a+b)=\mathrm{ conjugate }a+\mathrm{ conjugate b
    and conjugate-neg: conjugate ( }-a)=- conjugate a
    and conjugate-zero[simp]: conjugate 0 = 0
begin
    lemma conjugate-zero-iff[simp]: conjugate }a=0\longleftrightarrowa=
        using conjugate-cancel-iff[of-0, unfolded conjugate-zero].
end
class conjugatable-field = conjugatable-ring + field
lemma sum-conjugate:
    fixes f :: 'b b 'a :: conjugatable-ring
    assumes finX: finite X
    shows conjugate (sum f X) = sum ( }\lambdax\mathrm{ . conjugate (fx)) X
    using finX by (induct set:finite, auto simp: conjugate-dist-add)
class conjugatable-ordered-ring = conjugatable-ring + ordered-comm-monoid-add
+
    assumes conjugate-square-positive: a* conjugate a \geq0
class conjugatable-ordered-field = conjugatable-ordered-ring + field
begin
    subclass conjugatable-field..
end
lemma conjugate-square-0:
    fixes a :: 'a :: {conjugatable-ordered-ring, semiring-no-zero-divisors}
    shows }a*\mathrm{ conjugate }a=0\Longrightarrowa=0 by aut
```


### 3.1 Instantiations

```
instantiation complex :: conjugatable-ordered-field
begin
    definition [simp]: conjugate \equivcnj
instance
    by intro-classes (auto simp:less-eq-complex-def)
end
instantiation real :: conjugatable-ordered-field
begin
    definition [simp]:conjugate (x::real) \equiv 
    instance by (intro-classes, auto)
end
```

```
instantiation rat :: conjugatable-ordered-field
begin
    definition [simp]: conjugate (x::rat) \equivx
    instance by (intro-classes, auto)
end
instantiation int :: conjugatable-ordered-ring
begin
    definition [simp]: conjugate (x::int) \equivx
    instance by (intro-classes, auto)
end
lemma conjugate-square-eq-0 [simp]:
    fixes }x\mathrm{ :: ' }a\mathrm{ :: {conjugatable-ring,semiring-no-zero-divisors}
    shows }x*\mathrm{ conjugate }x=0\longleftrightarrowx=0 conjugate x * x=0 \longleftrightarrow x=0
    by auto
lemma conjugate-square-greater-0 [simp]:
    fixes }x\mathrm{ :: ' }a\mathrm{ :: {conjugatable-ordered-ring,ring-no-zero-divisors}
    shows }x*\mathrm{ conjugate }x>0\longleftrightarrowx\not=
    using conjugate-square-positive[of x]
    by (auto simp:le-less)
lemma conjugate-square-smaller-0 [simp]:
    fixes x :: 'a :: {conjugatable-ordered-ring,ring-no-zero-divisors}
    shows \negx* conjugate x<0
    using conjugate-square-positive[of x] by auto
end
```


## 4 Vectors and Matrices

We define vectors as pairs of dimension and a characteristic function from natural numbers to elements. Similarly, matrices are defined as triples of two dimensions and one characteristic function from pairs of natural numbers to elements. Via a subtype we ensure that the characteristic function always behaves the same on indices outside the intended one. Hence, every matrix has a unique representation.

In this part we define basic operations like matrix-addition, -multiplication, scalar-product, etc. We connect these operations to HOL-Algebra with its explicit carrier sets.

```
theory Matrix
imports
    Polynomial-Interpolation.Ring-Hom
    Missing-Ring
    Conjugate
```

HOL-Algebra.Module
begin

### 4.1 Vectors

Here we specify which value should be returned in case an index is out of bounds. The current solution has the advantage that in the implementation later on, no index comparison has to be performed.

```
definition undef-vec :: nat \(\Rightarrow{ }^{\prime} a\) where
    undef-vec \(i \equiv[]!i\)
definition \(m k\)-vec \(::\) nat \(\Rightarrow\left(n a t \Rightarrow{ }^{\prime} a\right) \Rightarrow\left(n a t \Rightarrow{ }^{\prime} a\right)\) where
    \(m k\)-vec \(n f \equiv \lambda\) i. if \(i<n\) then \(f\) i else undef-vec \((i-n)\)
typedef 'a vec \(=\{(n, m k\)-vec \(n f) \mid n f::\) nat \(\Rightarrow\) 'a. True \(\}\)
    by auto
setup-lifting type-definition-vec
lift-definition dim-vec :: 'a vec \(\Rightarrow\) nat is \(f s t\).
lift-definition vec-index :: 'a vec \(\Rightarrow(n a t \Rightarrow\) 'a) (infixl \$ 100) is snd.
lift-definition vec \(::\) nat \(\Rightarrow\left(\right.\) nat \(\left.\Rightarrow{ }^{\prime} a\right) \Rightarrow{ }^{\prime} a\) vec
    is \(\lambda n f\). \((n, m k\)-vec \(n f)\) by auto
lift-definition vec-of-list :: 'a list \(\Rightarrow\) 'a vec is
    \(\lambda v\). (length \(v, m k\)-vec (length \(v)(n t h v)\) by auto
lift-definition list-of-vec :: 'a vec \(\Rightarrow\) 'a list is
    \(\lambda(n, v)\). map \(v[0 . .<n]\).
```

definition carrier-vec :: nat $\Rightarrow{ }^{\prime}$ 'a vec set where
carrier-vec $n=\{v$. dim-vec $v=n\}$
lemma carrier-vec-dim-vec $[$ simp $]: v \in$ carrier-vec (dim-vec $v$ ) unfolding carrier-vec-def
by auto
lemma dim-vec $[\operatorname{simp}]$ : dim-vec $($ vec $n f)=n$ by transfer simp
lemma vec-carrier $[$ simp $]$ : vec $n f \in$ carrier-vec $n$ unfolding carrier-vec-def by
auto
lemma index-vec[simp]: $i<n \Longrightarrow$ vec $n f \$ i=f i$ by transfer (simp add:
$m k$-vec-def)
lemma eq-vecI[intro]: ( $\bigwedge i . i<\operatorname{dim-vec} w \Longrightarrow v \$ i=w \$ i) \Longrightarrow \operatorname{dim}$-vec $v=$
dim-vec $w$
$\Longrightarrow v=w$
by (transfer, auto simp: mk-vec-def)
lemma carrier-dim-vec: $v \in$ carrier-vec $n \longleftrightarrow$ dim-vec $v=n$
unfolding carrier-vec-def by auto
lemma carrier-vec $D[$ simp $]: v \in$ carrier-vec $n \Longrightarrow$ dim-vec $v=n$ using carrier-dim-vec by auto
lemma carrier-vecI: dim-vec $v=n \Longrightarrow v \in$ carrier-vec $n$ using carrier-dim-vec by auto

```
instantiation vec :: (plus) plus
begin
```

definition plus-vec :: 'a vec $\Rightarrow{ }^{\prime} a$ vec $\Rightarrow$ ' $a$ :: plus vec where
$v_{1}+v_{2} \equiv$ vec (dim-vec $\left.v_{2}\right)\left(\lambda i . v_{1} \$ i+v_{2} \$ i\right)$
instance ..
end
instantiation vec :: (minus) minus
begin
definition minus-vec :: 'a vec $\Rightarrow{ }^{\prime} a$ vec $\Rightarrow$ ' $a$ :: minus vec where
$v_{1}-v_{2} \equiv$ vec (dim-vec $\left.v_{2}\right)\left(\lambda i . v_{1} \$ i-v_{2} \$ i\right)$
instance ..
end

## definition

```
zero-vec \(::\) nat \(\Rightarrow{ }^{\prime} a\) :: zero vec \(\left(0_{v}\right)\)
```

where $0_{v} n \equiv$ vec $n(\lambda i .0)$
lemma zero-carrier-vec[simp]: $0_{v} n \in$ carrier-vec $n$ unfolding zero-vec-def carrier-vec-def by auto
lemma index-zero-vec $[\operatorname{simp}]: i<n \Longrightarrow 0_{v} n \$ i=0 \operatorname{dim-vec}\left(0_{v} n\right)=n$ unfolding zero-vec-def by auto
lemma vec-of-dim- $0[$ simp $]$ : dim-vec $v=0 \longleftrightarrow v=0_{v} 0$ by auto

## definition

unit-vec $::$ nat $\Rightarrow$ nat $\Rightarrow\left({ }^{\prime} a::\right.$ zero-neq-one) vec
where unit-vec $n i=\operatorname{vec} n(\lambda j$. if $j=i$ then 1 else 0$)$
lemma index-unit-vec[simp]:
$i<n \Longrightarrow j<n \Longrightarrow$ unit-vec $n i \$ j=($ if $j=i$ then 1 else 0$)$
$i<n \Longrightarrow$ unit-vec $n i \$ i=1$
dim-vec (unit-vec $n i)=n$
unfolding unit-vec-def by auto
lemma unit-vec-eq[simp]:
assumes $i: i<n$
shows (unit-vec $n i=$ unit-vec $n j)=(i=j)$
proof -
have $i \neq j \Longrightarrow$ unit-vec $n i \$ i \neq$ unit-vec $n j \$ i$
unfolding unit-vec-def using $i$ by simp
then show ?thesis by metis

## qed

lemma unit-vec-nonzero[simp]:
assumes $i-n$ : $i<n$ shows unit-vec $n i \neq$ zero-vec $n$ (is ?l $\neq ? r$ )
proof -
have ?l $\$ i=1$ ? $\mathrm{r} \$ i=0$ using $i-n$ by auto
thus ? $l \neq ? r$ by auto
qed
lemma unit-vec-carrier[simp]: unit-vec $n i \in$ carrier-vec $n$ unfolding unit-vec-def carrier-vec-def by auto
definition unit-vecs:: nat $\Rightarrow{ }^{\prime} a$ :: zero-neq-one vec list where unit-vecs $n=\operatorname{map}($ unit-vec $n)[0 . .<n]$

List of first i units
fun unit-vecs-first:: nat $\Rightarrow$ nat $\Rightarrow{ }^{\prime} a::$ zero-neq-one vec list where unit-vecs-first $n 0=[]$ unit-vecs-first $n(S u c i)=$ unit-vecs-first $n i @[u n i t-v e c ~ n i]$
lemma unit-vecs-first: unit-vecs $n=$ unit-vecs-first $n n$
unfolding unit-vecs-def set-map set-upt
proof -
\{fix $m$
have $m \leq n \Longrightarrow \operatorname{map}$ (unit-vec $n$ ) $[0 . .<m]=$ unit-vecs-first $n m$ proof (induct $m$ )
case (Suc $m$ ) then have $m n: m \leq n$ by auto
show ?case unfolding upt-Suc using Suc(1)[OF mn] by auto
qed auto
\}
thus map (unit-vec $n$ ) $[0 . .<n]=$ unit-vecs-first $n n$ by auto
qed
list of last i units
fun unit-vecs-last:: nat $\Rightarrow$ nat $\Rightarrow{ }^{\prime} a$ :: zero-neq-one vec list
where unit-vecs-last n $0=[]$
unit-vecs-last $n($ Suc $i)=$ unit-vec $n(n-S u c i) \#$ unit-vecs-last $n i$
lemma unit-vecs-last-carrier: set (unit-vecs-last $n i) \subseteq$ carrier-vec $n$ by (induct $i ; a u t o$ )
lemma unit-vecs-last[code]: unit-vecs $n=$ unit-vecs-last $n n$ proof -
\{ fix $m$ assume $m=n$
have $m \leq n \Longrightarrow$ map (unit-vec $n$ ) $[n-m . .<n]=$ unit-vecs-last $n m$ proof (induction $m$ )
case (Suc m)
then have $n m: n-S u c m<n$ by auto
have ins: $[n-$ Suc $m . .<n]=(n-$ Suc $m) \#[n-m . .<n]$

```
            unfolding upt-conv-Cons[OF nm]
            by (auto simp: Suc.prems Suc-diff-Suc Suc-le-lessD)
            show ?case
            unfolding ins
            unfolding unit-vecs-last.simps
            unfolding list.map
            using Suc
            unfolding Suc by auto
        qed simp
    }
    thus unit-vecs n = unit-vecs-last n n
        unfolding unit-vecs-def by auto
qed
lemma unit-vecs-carrier: set (unit-vecs n) \subseteqcarrier-vec n
proof
    fix u :: 'a vec assume u:u\in set (unit-vecs n)
    then obtain i where }u=\mathrm{ unit-vec n i unfolding unit-vecs-def by auto
    then show }u\in\mathrm{ carrier-vec n
    using unit-vec-carrier by auto
qed
lemma unit-vecs-last-distinct:
        j\leqn\Longrightarrowi<n-j\Longrightarrowunit-vec n i\not\in set (unit-vecs-last n j)
    by (induction j arbitrary:i, auto)
lemma unit-vecs-first-distinct:
    i\leqj\Longrightarrowj<n\Longrightarrowunit-vec n j & set (unit-vecs-first n i)
    by (induction i arbitrary:j, auto)
definition map-vec where map-vec f v\equivvec (dim-vec v)(\lambdai.f (v$ i))
instantiation vec :: (uminus) uminus
begin
definition uminus-vec :: ' }a::\mathrm{ uminus vec }=>\mathrm{ ' 'a vec where
    -v\equivvec (dim-vec v) (\lambdai.- (v$i))
instance ..
end
definition smult-vec :: 'a :: times }=>\mp@subsup{}{}{\prime}a\mathrm{ vec }=>\mp@subsup{}{}{\prime}'a vec (infixl \cdotv 70) 
    where a vvv\equivvec(dim-vec v)(\lambdai.a*v$i)
definition scalar-prod :: 'a vec => 'a vec => 'a :: semiring-0 (infix • 70)
    where v}\cdotw\equiv\sumi\in{0..< dim-vec w}.v$i*w$
definition monoid-vec :: 'a itself }=>\mathrm{ nat }=>\mathrm{ ('a :: monoid-add vec) monoid where
    monoid-vec ty n}\equiv
        carrier = carrier-vec n,
        mult = (+),
```

$$
o n e=O_{v} n \mid
$$

definition module-vec ::
' $a$ :: semiring-1 itself $\Rightarrow$ nat $\Rightarrow\left({ }^{\prime} a,^{\prime} a\right.$ vec $)$ module where
module-vec ty $n \equiv 0$
carrier $=$ carrier-vec $n$,
mult $=$ undefined,
one $=$ undefined ,
zero $=0_{v} n$,
$a d d=(+)$,
smult $=\left(\cdot{ }_{v}\right)$ )

## lemma monoid-vec-simps:

mult (monoid-vec ty $n$ ) $=(+)$
carrier (monoid-vec ty $n$ ) $=$ carrier-vec $n$
one (monoid-vec ty $n$ ) $=0_{v} n$
unfolding monoid-vec-def by auto
lemma module-vec-simps:
add $($ module-vec ty $n)=(+)$
zero $($ module-vec ty $n)=0_{v} n$
carrier (module-vec ty $n$ ) $=$ carrier-vec $n$
smult (module-vec ty $n$ ) $=\left(\cdot{ }_{v}\right)$
unfolding module-vec-def by auto
definition finsum-vec :: ' $a$ :: monoid-add itself $\Rightarrow$ nat $\Rightarrow\left({ }^{\prime} c \Rightarrow{ }^{\prime} a\right.$ vec $) \Rightarrow{ }^{\prime} c$ set $\Rightarrow$ 'a vec where
finsum-vec ty $n=$ finprod $($ monoid-vec ty $n)$
lemma index-add-vec[simp]:
$i<$ dim-vec $v_{2} \Longrightarrow\left(v_{1}+v_{2}\right) \$ i=v_{1} \$ i+v_{2} \$ i \operatorname{dim-vec}\left(v_{1}+v_{2}\right)=\operatorname{dim}-v e c$
$v_{2}$
unfolding plus-vec-def by auto
lemma index-minus-vec [simp]:
$i<\operatorname{dim-vec} v_{2} \Longrightarrow\left(v_{1}-v_{2}\right) \$ i=v_{1} \$ i-v_{2} \$ i \operatorname{dim-vec}\left(v_{1}-v_{2}\right)=\operatorname{dim-vec}$ $v_{2}$
unfolding minus-vec-def by auto
lemma index-map-vec[simp]:
$i<$ dim-vec $v \Longrightarrow$ map-vec $f v \$ i=f(v \$ i)$
dim-vec (map-vec f $v$ ) $=$ dim-vec $v$
unfolding map-vec-def by auto
lemma map-carrier-vec[simp]: map-vec $h v \in$ carrier-vec $n=(v \in$ carrier-vec $n)$ unfolding map-vec-def carrier-vec-def by auto
lemma index-uminus-vec[simp]:
$i<$ dim-vec $v \Longrightarrow(-v) \$ i=-(v \$ i)$
dim-vec $(-v)=$ dim-vec $v$
unfolding uminus-vec-def by auto
lemma index-smult-vec[simp]:
$i<\operatorname{dim}$-vec $v \Longrightarrow(a \cdot v v) \$ i=a * v \$ i \operatorname{dim-vec}(a \cdot v v)=\operatorname{dim-vec} v$ unfolding smult-vec-def by auto
lemma add-carrier-vec[simp]:
$v_{1} \in$ carrier-vec $n \Longrightarrow v_{2} \in$ carrier-vec $n \Longrightarrow v_{1}+v_{2} \in$ carrier-vec $n$
unfolding carrier-vec-def by auto
lemma minus-carrier-vec $[$ simp $]$ :
$v_{1} \in$ carrier-vec $n \Longrightarrow v_{2} \in$ carrier-vec $n \Longrightarrow v_{1}-v_{2} \in$ carrier-vec $n$ unfolding carrier-vec-def by auto
lemma comm-add-vec[ac-simps]:
( $v_{1}::$ 'a $::$ ab-semigroup-add vec $) \in$ carrier-vec $n \Longrightarrow v_{2} \in$ carrier-vec $n \Longrightarrow v_{1}$
$+v_{2}=v_{2}+v_{1}$
by (intro eq-vecI, auto simp: ac-simps)
lemma assoc-add-vec[simp]:
( $v_{1}::$ 'a :: semigroup-add vec $) \in$ carrier-vec $n \Longrightarrow v_{2} \in$ carrier-vec $n \Longrightarrow v_{3} \in$ carrier-vec $n$

$$
\Longrightarrow\left(v_{1}+v_{2}\right)+v_{3}=v_{1}+\left(v_{2}+v_{3}\right)
$$

by (intro eq-vecI, auto simp: ac-simps)
lemma zero-minus-vec[simp]: $(v:: ' a::$ group-add vec $) \in$ carrier-vec $n \Longrightarrow 0_{v} n$ $-v=-v$
by (intro eq-vecI, auto)
lemma minus-zero-vec[simp]: (v :: 'a :: group-add vec) $\in$ carrier-vec $n \Longrightarrow v-0_{v}$ $n=v$
by (intro eq-vecI, auto)
lemma minus-cancel-vec $[\operatorname{simp}]:(v::$ 'a $::$ group-add vec $) \in$ carrier-vec $n \Longrightarrow v-$ $v=0_{v} n$
by (intro eq-vecI, auto)
lemma minus-add-uminus-vec: (v :: 'a :: group-add vec) $\in$ carrier-vec $n \Longrightarrow$ $w \in$ carrier-vec $n \Longrightarrow v-w=v+(-w)$
by (intro eq-vecI, auto)
lemma comm-monoid-vec: comm-monoid (monoid-vec TYPE ('a :: comm-monoid-add) n)
by (unfold-locales, auto simp: monoid-vec-def ac-simps)
lemma left-zero-vec[simp]: (v :: 'a :: monoid-add vec) $\in$ carrier-vec $n \Longrightarrow 0_{v} n+$ $v=v$ by auto
lemma right-zero-vec[simp]: (v :: 'a :: monoid-add vec $) \in$ carrier-vec $n \Longrightarrow v+$ $O_{v} n=v$ by auto
lemma uminus-carrier-vec[simp]:
$(-v \in$ carrier-vec $n)=(v \in$ carrier-vec $n)$
unfolding carrier-vec-def by auto
lemma uminus-r-inv-vec [simp]:
( $v::$ ' $a::$ group-add vec) $\in$ carrier-vec $n \Longrightarrow(v+-v)=0_{v} n$
by (intro eq-vecI, auto)
lemma uminus-l-inv-vec[simp]:
( $v::$ 'a $::$ group-add vec $) \in$ carrier-vec $n \Longrightarrow(-v+v)=0_{v} n$ by (intro eq-vecI, auto)
lemma add-inv-exists-vec:
( $v::$ 'a $::$ group-add vec $) \in$ carrier-vec $n \Longrightarrow \exists w \in$ carrier-vec $n . w+v=0_{v}$ $n \wedge v+w=0_{v} n$
by (intro bexI[of $--v]$, auto)
lemma comm-group-vec: comm-group (monoid-vec TYPE ('a :: ab-group-add) n) by (unfold-locales, insert add-inv-exists-vec, auto simp: monoid-vec-def ac-simps Units-def)
lemmas finsum-vec-insert $=$
comm-monoid.finprod-insert[OF comm-monoid-vec, folded finsum-vec-def, unfolded monoid-vec-simps]
lemmas finsum-vec-closed $=$ comm-monoid.finprod-closed[OF comm-monoid-vec, folded finsum-vec-def, unfolded monoid-vec-simps]
lemmas finsum-vec-empty $=$
comm-monoid.finprod-empty[OF comm-monoid-vec, folded finsum-vec-def, un-
folded monoid-vec-simps]
lemma smult-carrier-vec $[$ simp $]:\left(a \cdot_{v} v \in\right.$ carrier-vec $\left.n\right)=(v \in$ carrier-vec $n)$
unfolding carrier-vec-def by auto
lemma scalar-prod-left-zero $[$ simp $]: v \in$ carrier-vec $n \Longrightarrow 0_{v} n \cdot v=0$
unfolding scalar-prod-def
by (rule sum.neutral, auto)
lemma scalar-prod-right-zero $[$ simp $]: v \in$ carrier-vec $n \Longrightarrow v \cdot 0_{v} n=0$ unfolding scalar-prod-def
by (rule sum.neutral, auto)
lemma scalar-prod-left-unit[simp]: assumes $v:(v:: ' a$ :: semiring-1 vec $) \in$ car-

```
rier-vec n and i: i<n
    shows unit-vec n i \cdot v=v$i
proof -
    let ?f = \lambda k. unit-vec n i$k*v$k
    have id: (\sumk\in{0..<n}. ?f k)= unit-vec n i$ i*v$i+(\sumk\in{0..<n}-{i}.
?f k)
    by (rule sum.remove, insert i, auto)
    also have (\sum k\in{0..<n}-{i}. ?f k)=0
        by (rule sum.neutral, insert i, auto)
    finally
    show ?thesis unfolding scalar-prod-def using iv by simp
qed
lemma scalar-prod-right-unit[simp]: assumes i:i<n
    shows (v :: 'a :: semiring-1 vec) - unit-vec n i=v$ i
proof -
    let ?f = \lambdak.v $k* unit-vec ni $k
    have id: (\sumk\in{0..<n}. .f k)=v$i*unit-vec n i$i+( \sumk\in{0..<n}-{i}.
?f k)
    by (rule sum.remove, insert i, auto)
    also have ( }\sumk\in{0..<n}-{i}. ?f k)=
            by (rule sum.neutral, insert i, auto)
    finally
    show ?thesis unfolding scalar-prod-def using i by simp
qed
```

lemma add-scalar-prod-distrib: assumes $v: v_{1} \in$ carrier-vec $n v_{2} \in$ carrier-vec $n$
$v_{3} \in$ carrier-vec $n$
shows $\left(v_{1}+v_{2}\right) \cdot v_{3}=v_{1} \cdot v_{3}+v_{2} \cdot v_{3}$
proof -
have $\left(\sum i \in\left\{0 . .<\right.\right.$ dim-vec $\left.\left.v_{3}\right\} .\left(v_{1}+v_{2}\right) \$ i * v_{3} \$ i\right)=\left(\sum i \in\left\{0 . .<\right.\right.$ dim-vec $\left.v_{3}\right\}$.
$\left.v_{1} \$ i * v_{3} \$ i+v_{2} \$ i * v_{3} \$ i\right)$
by (rule sum.cong, insert $v$, auto simp: algebra-simps)
thus ?thesis unfolding scalar-prod-def using $v$ by (auto simp: sum.distrib)
qed
lemma scalar-prod-add-distrib: assumes $v: v_{1} \in$ carrier-vec $n v_{2} \in$ carrier-vec $n$ $v_{3} \in$ carrier-vec $n$
shows $v_{1} \cdot\left(v_{2}+v_{3}\right)=v_{1} \cdot v_{2}+v_{1} \cdot v_{3}$
proof -
have $\left(\sum i \in\left\{0 . .<\right.\right.$ dim-vec $\left.\left.v_{3}\right\} . v_{1} \$ i *\left(v_{2}+v_{3}\right) \$ i\right)=\left(\sum i \in\left\{0 . .<\right.\right.$ dim-vec $\left.v_{3}\right\}$.
$\left.v_{1} \$ i * v_{2} \$ i+v_{1} \$ i * v_{3} \$ i\right)$
by (rule sum.cong, insert $v$, auto simp: algebra-simps)
thus ?thesis unfolding scalar-prod-def using $v$ by (auto intro: sum.distrib)
qed
lemma smult-scalar-prod-distrib[simp]: assumes $v: v_{1} \in$ carrier-vec $n v_{2} \in$ car-rier-vec $n$
shows $\left(a \cdot{ }_{v} v_{1}\right) \cdot v_{2}=a *\left(v_{1} \cdot v_{2}\right)$
unfolding scalar-prod-def sum-distrib-left
by (rule sum.cong, insert $v$, auto simp: ac-simps)
lemma scalar-prod-smult-distrib[simp]: assumes $v: v_{1} \in$ carrier-vec $n v_{2} \in$ car-rier-vec $n$

```
shows v}\mp@subsup{v}{1}{}\cdot(a\cdotv\mp@subsup{v}{2}{})=(a::' 'a :: comm-ring) * (v, v v v )
unfolding scalar-prod-def sum-distrib-left
by (rule sum.cong, insert v, auto simp: ac-simps)
```

lemma comm-scalar-prod: assumes ( $v_{1}::$ ' $a::$ comm-semiring-0 vec) $\in$ carrier-vec $n v_{2} \in$ carrier-vec $n$ shows $v_{1} \cdot v_{2}=v_{2} \cdot v_{1}$
unfolding scalar-prod-def
by (rule sum.cong, insert assms, auto simp: ac-simps)
lemma add-smult-distrib-vec:
$\left(\left(a::^{\prime} a::\right.\right.$ ring $\left.)+b\right) \cdot{ }_{v} v=a \cdot{ }_{v} v+b \cdot{ }_{v} v$
unfolding smult-vec-def plus-vec-def
by (rule eq-vecI, auto simp: distrib-right)
lemma smult-add-distrib-vec:
assumes $v \in$ carrier-vec $n w \in$ carrier-vec $n$
shows $\left(a::^{\prime} a::\right.$ ring $) \cdot v(v+w)=a \cdot v v+a \cdot v w$
apply (rule eq-vecI)
unfolding smult-vec-def plus-vec-def
using assms distrib-left by auto
lemma smult-smult-assoc:
$a \cdot v\left(b \cdot{ }_{v} v\right)=\left(a * b::^{\prime} a::\right.$ ring $) \cdot{ }_{v} v$
apply (rule sym, rule eq-vecI)
unfolding smult-vec-def plus-vec-def using mult.assoc by auto
lemma one-smult-vec [simp]:
(1::'a::ring-1) $v_{v} v=v$ unfolding smult-vec-def
by (rule eq-vecI,auto)
lemma uminus-zero-vec[simp]: $-\left(0_{v} n\right)=\left(0_{v} n::\right.$ 'a :: group-add vec $)$
by (intro eq-vecI, auto)
lemma index-finsum-vec: assumes finite $F$ and $i: i<n$
and vs: vs $\in F \rightarrow$ carrier-vec $n$
shows finsum-vec TYPE('a :: comm-monoid-add) n vs $F \$ i=\operatorname{sum}(\lambda f$. vs $f \$$
i) $F$
using 〈finite $F$ 〉vs
proof (induct $F$ )
case (insert $f F$ )
hence $I H$ : finsum-vec $T Y P E(' a) n$ vs $F \$ i=\left(\sum f \in F\right.$. vs $\left.f \$ i\right)$
and vs: vs $\in F \rightarrow$ carrier-vec $n$ vs $f \in$ carrier-vec $n$ by auto
show ?case unfolding finsum-vec-insert[OF insert(1-2) vs]
unfolding sum.insert[OF insert(1-2)]
unfolding $I H[$ symmetric]
by (rule index-add-vec, insert $i$, insert finsum-vec-closed $[$ OF vs(1)], auto) qed (insert $i$, auto simp: finsum-vec-empty)

Definition of pointwise ordering on vectors for non-strict part, and strict version is defined in a way such that the order constraints are satisfied.

```
instantiation vec :: (ord) ord
begin
definition less-eq-vec :: 'a vec }=>\mathrm{ ' 'a vec }=>\mathrm{ bool where
    less-eq-vec v w=(dim-vec v=dim-vec w\wedge(\foralli<dim-vec w.v$ i\leqw$i))
definition less-vec :: 'a vec }=>\mathrm{ ' 'a vec }=>\mathrm{ bool where
    less-vec v w=(v\leqw^\neg(w\leqv))
instance ..
end
```

instantiation vec :: (preorder) preorder
begin
instance
by (standard, auto simp: less-vec-def less-eq-vec-def order-trans)
end
instantiation vec :: (order) order
begin
instance
by (standard, intro eq-vecI, auto simp: less-eq-vec-def order.antisym)
end

### 4.2 Matrices

Similarly as for vectors, we specify which value should be returned in case an index is out of bounds. It is defined in a way that only few index comparisons have to be performed in the implementation.

```
definition undef-mat \(::\) nat \(\Rightarrow\) nat \(\Rightarrow(n a t \times n a t \Rightarrow ' a) \Rightarrow n a t \times n a t \Rightarrow{ }^{\prime} a\) where
    undef-mat \(n r n c f \equiv \lambda(i, j) .[[f(i, j) . j<-[0 . .<n c]] . i<-[0 . .<n r]]!i!j\)
lemma undef-cong-mat: assumes \(\bigwedge i j . i<n r \Longrightarrow j<n c \Longrightarrow f(i, j)=f^{\prime}(i, j)\)
    shows undef-mat nr nc f \(x=\) undef-mat \(n r n c f^{\prime} x\)
proof (cases \(x\) )
    case (Pair \(i j\) )
    have nth-map-ge: \(\bigwedge i x s . \neg i<\) length \(x s \Longrightarrow x s!i=[]!(i-\) length \(x s)\)
        by (metis append-Nil2 nth-append)
    note \([\) simp \(]=\) Pair undef-mat-def nth-map-ge[of i] nth-map-ge[of \(j]\)
    show ?thesis
        by (cases \(i<n r\), simp, cases \(j<n c\), insert assms, auto)
qed
```

definition $m k$-mat $::$ nat $\Rightarrow n a t \Rightarrow\left(n a t \times n a t \Rightarrow{ }^{\prime} a\right) \Rightarrow\left(n a t \times n a t \Rightarrow{ }^{\prime} a\right)$ where mk-mat nr nc $f \equiv \lambda(i, j)$. if $i<n r \wedge j<n c$ then $f(i, j)$ else undef-mat $n r n c f$ $(i, j)$
lemma cong-mk-mat: assumes $\bigwedge i j . i<n r \Longrightarrow j<n c \Longrightarrow f(i, j)=f^{\prime}(i, j)$
shows mk-mat nr nc $f=m k$-mat nr nc $f^{\prime}$
using undef-cong-mat[of nr nc $f f^{\prime}$, OF assms]
using assms unfolding mk-mat-def
by auto
typedef 'a mat $=\left\{(n r, n c, m k\right.$-mat $n r n c f) \mid n r n c f:: n a t \times n a t \Rightarrow{ }^{\prime} a$. True $\}$ by auto
setup-lifting type-definition-mat
lift-definition dim-row :: 'a mat $\Rightarrow$ nat is $f s t$.
lift-definition dim-col :: 'a mat $\Rightarrow$ nat is fst o snd .
lift-definition index-mat :: 'a mat $\Rightarrow\left(n a t \times n a t \Rightarrow{ }^{\prime} a\right)(\operatorname{infixl} \$ \$ 100)$ is snd o snd .
lift-definition mat :: nat $\Rightarrow$ nat $\Rightarrow\left(\right.$ nat $\left.\times n a t \Rightarrow{ }^{\prime} a\right) \Rightarrow^{\prime}$ a mat
is $\lambda n r n c f .(n r, n c, m k$-mat $n r n c f)$ by auto
lift-definition mat-of-row-fun :: nat $\Rightarrow$ nat $\Rightarrow\left(\right.$ nat $\Rightarrow{ }^{\prime}$ a vec $) \Rightarrow{ }^{\prime}$ a mat $\left(\right.$ mat $\left._{r}\right)$
is $\lambda n r n c f .(n r, n c, m k$-mat $n r n c(\lambda(i, j) . f i \$ j))$ by auto
definition mat-to-list :: 'a mat $\Rightarrow$ 'a list list where
mat-to-list $A=[[A \$ \$(i, j) . j<-[0 . .<\operatorname{dim}-c o l ~ A]] . i<-[0 . .<\operatorname{dim}$-row $A]]$
fun square-mat $::$ ' $a$ mat $\Rightarrow$ bool where square-mat $A=(\operatorname{dim}-c o l ~ A=d i m$-row $A)$
definition upper-triangular :: 'a::zero mat $\Rightarrow$ bool
where upper-triangular $A \equiv$
$\forall i<$ dim-row $A . \forall j<i . A \$ \$(i, j)=0$
lemma upper-triangular $D[$ elim $]$ :
upper-triangular $A \Longrightarrow j<i \Longrightarrow i<$ dim-row $A \Longrightarrow A \$ \$(i, j)=0$
unfolding upper-triangular-def by auto
lemma upper-triangularI[intro]:
$(\bigwedge i j . j<i \Longrightarrow i<$ dim-row $A \Longrightarrow A \$ \$(i, j)=0) \Longrightarrow$ upper-triangular $A$
unfolding upper-triangular-def by auto
lemma dim-row-mat $[\operatorname{simp}]$ : dim-row (mat nr nc $f)=n r$ dim-row $\left(\right.$ mat $\left._{r} n r n c g\right)$ $=n r$
by (transfer, simp) +
lemma dim-col-mat[simp]: dim-col (mat nr nc f) $=$ nc dim-col $\left(\right.$ mat $\left._{r} n r n c g\right)=$ $n c$

$$
\text { by }(\text { transfer }, \operatorname{simp})+
$$

definition carrier-mat :: nat $\Rightarrow$ nat $\Rightarrow{ }^{\prime}$ a mat set
where carrier-mat $n r n c=\{m$. dim-row $m=n r \wedge$ dim-col $m=n c\}$
lemma carrier-mat-triv[simp]: $m \in$ carrier-mat (dim-row m) (dim-col m) unfolding carrier-mat-def by auto
lemma mat-carrier [simp]: mat nr nc $f \in$ carrier-mat $n r n c$ unfolding carrier-mat-def by auto
definition elements-mat :: 'a mat $\Rightarrow$ 'a set
where elements-mat $A=$ set $[A \$ \$(i, j) . i<-[0 . .<$ dim-row $A], j<-[0 . .<$
dim-col A]]
lemma elements-matD [dest]
$a \in$ elements-mat $A \Longrightarrow \exists i j . i<\operatorname{dim}$-row $A \wedge j<\operatorname{dim}-c o l A \wedge a=A \$ \$(i, j)$
unfolding elements-mat-def by force
lemma elements-matI [intro]:
$A \in$ carrier-mat $n r n c \Longrightarrow i<n r \Longrightarrow j<n c \Longrightarrow a=A \$ \$(i, j) \Longrightarrow a \in$ elements-mat $A$
unfolding elements-mat-def carrier-mat-def by force
lemma index-mat[simp]: $i<n r \Longrightarrow j<n c \Longrightarrow$ mat nr nc $f \$ \$(i, j)=f(i, j)$ $i<n r \Longrightarrow j<n c \Longrightarrow m^{2} t_{r} n r n c g \$ \$(i, j)=g i \$ j$ by (transfer ${ }^{\prime}$, simp add: mk-mat-def)+
lemma eq-matI[intro]: ( $\bigwedge i j . i<$ dim-row $B \Longrightarrow j<\operatorname{dim}$-col $B \Longrightarrow A \$ \$(i, j)$ $=B \$ \$(i, j))$
$\Longrightarrow$ dim-row $A=$ dim-row $B$
$\Longrightarrow d i m-\operatorname{col} A=\operatorname{dim}-\operatorname{col} B$
$\Longrightarrow A=B$
by (transfer, auto intro!: cong-mk-mat, auto simp: mk-mat-def)
lemma carrier-matI[intro]:
assumes dim-row $A=n r \operatorname{dim}-\operatorname{col} A=n c$ shows $A \in$ carrier-mat nr nc using assms unfolding carrier-mat-def by auto
lemma carrier-matD[dest,simp]: assumes $A \in$ carrier-mat nr nc shows dim-row $A=n r$ dim-col $A=n c$ using assms unfolding carrier-mat-def by auto
lemma cong-mat: assumes $n r=n r^{\prime} n c=n c^{\prime} \bigwedge i j . i<n r \Longrightarrow j<n c \Longrightarrow$ $f(i, j)=f^{\prime}(i, j)$ shows mat $n r n c f=$ mat $n r^{\prime} n c^{\prime} f^{\prime}$ by (rule eq-matI, insert assms, auto)
definition row :: 'a mat $\Rightarrow$ nat $\Rightarrow{ }^{\prime} a$ vec where row $A i=\operatorname{vec}($ dim-col $A)(\lambda j$. $A \$ \$(i, j))$
definition rows :: 'a mat $\Rightarrow{ }^{\prime} a$ vec list where

$$
\text { rows } A=\operatorname{map}(\text { row } A)[0 . .<\text { dim-row } A]
$$

lemma row-carrier $[$ simp $]$ : row $A i \in$ carrier-vec (dim-col $A$ ) unfolding row-def by auto
lemma rows-carrier $[$ simp $]$ : set (rows $A$ ) $\subseteq$ carrier-vec (dim-col A) unfolding rows-def by auto
lemma length-rows $[$ simp $]$ : length (rows $A)=$ dim-row $A$ unfolding rows-def by auto
lemma nth-rows $[$ simp $]: i<$ dim-row $A \Longrightarrow$ rows $A!i=$ row $A i$ unfolding rows-def by auto
lemma row-mat-of-row-fun $[$ simp $]: i<n r \Longrightarrow \operatorname{dim-vec}(f i)=n c \Longrightarrow r o w\left(\right.$ mat $_{r}$ $n r n c f) i=f i$
by (rule eq-vecI, auto simp: row-def)
lemma set-rows-carrier:
assumes $A \in$ carrier-mat $m n$ and $v \in$ set (rows $A$ ) shows $v \in$ carrier-vec $n$ using assms by (auto simp: rows-def row-def)
definition mat-of-rows :: nat $\Rightarrow$ 'a vec list $\Rightarrow$ 'a mat where mat-of-rows $n$ rs $=$ mat (length rs) $n(\lambda(i, j)$.rs ! $i \$ j)$
definition mat-of-rows-list :: nat $\Rightarrow$ 'a list list $\Rightarrow$ ' $a$ mat where mat-of-rows-list nc rs $=$ mat (length rs) nc $(\lambda(i, j) . r s!i!j)$
lemma mat-of-rows-carrier[simp]:
mat-of-rows $n$ vs $\in$ carrier-mat (length vs) $n$
dim-row (mat-of-rows $n$ vs) $=$ length vs
dim-col (mat-of-rows $n$ vs) $=n$
unfolding mat-of-rows-def by auto
lemma mat-of-rows-row[simp]:
assumes $i: i<$ length vs and $n:$ vs $!i \in$ carrier-vec $n$
shows row (mat-of-rows $n$ vs) $i=v s!i$
unfolding mat-of-rows-def row-def using $n i$ by auto
lemma rows-mat-of-rows [simp]:
assumes set vs $\subseteq$ carrier-vec $n$ shows rows (mat-of-rows $n$ vs) $=$ vs
unfolding rows-def apply (rule nth-equalityI)
using assms unfolding subset-code(1) by auto
lemma mat-of-rows-rows [simp]:
mat-of-rows (dim-col $A)($ rows $A)=A$
unfolding mat-of-rows-def by (rule, auto simp: row-def)

```
definition col :: 'a mat }=>\mathrm{ nat }=>\mp@subsup{|}{}{\prime}a\mathrm{ vec where
    col A j = vec(dim-row A) (\lambda i. A$$(i,j))
definition cols :: 'a mat }=>\mathrm{ 'a vec list where
    cols A = map (col A) [0..<dim-col A]
definition mat-of-cols :: nat }=>\mp@subsup{}{}{\prime}\mathrm{ 'a vec list }=>\mathrm{ ' 'a mat
    where mat-of-cols n cs = mat n (length cs) ( }\lambda(i,j).cs!j$ i
definition mat-of-cols-list :: nat => 'a list list => 'a mat where
    mat-of-cols-list nr cs = mat nr (length cs) ( }\lambda(i,j).cs!j!i
lemma col-dim[simp]: col A i \in carrier-vec (dim-row A) unfolding col-def by
auto
lemma dim-col[simp]: dim-vec (col A i)= dim-row A by auto
lemma cols-dim[simp]: set (cols A)\subseteqcarrier-vec (dim-row A) unfolding cols-def
by auto
lemma cols-length[simp]: length (cols A) = dim-col A unfolding cols-def by auto
lemma cols-nth[simp]:i<dim-col A\Longrightarrow cols A!i= col A i
    unfolding cols-def by auto
lemma mat-of-cols-carrier[simp]:
    mat-of-cols n vs \in carrier-mat n (length vs)
    dim-row (mat-of-cols n vs) = n
    dim-col (mat-of-cols n vs) = length vs
    unfolding mat-of-cols-def by auto
lemma col-mat-of-cols[simp]:
    assumes j:j< length vs and n: vs ! j carrier-vec n
    shows col (mat-of-cols n vs) j=vs!j
    unfolding mat-of-cols-def col-def using j n by auto
lemma cols-mat-of-cols[simp]:
    assumes set vs \subseteqcarrier-vec n shows cols (mat-of-cols n vs)=vs
    unfolding cols-def apply(rule nth-equalityI)
    using assms unfolding subset-code(1) by auto
lemma mat-of-cols-cols[simp]:
    mat-of-cols (dim-row A) (cols A)=A
    unfolding mat-of-cols-def by (rule, auto simp: col-def)
instantiation mat :: (ord) ord
begin
```

```
definition less-eq-mat :: 'a mat }=>\mathrm{ 'a mat }=>\mathrm{ bool where
    less-eq-mat A B = (dim-row }A=\mathrm{ dim-row }B\wedge\operatorname{dim-col }A=\operatorname{dim}-col B
        (}\foralli<\mathrm{ dim-row B.}\forallj<\mathrm{ dim-col B. A $$ (i,j) < B$$(i,j)))
definition less-mat :: 'a mat }=>\mathrm{ 'a mat }=>\mathrm{ bool where
    less-mat A B = (A\leqB\wedge\neg (B\leqA))
instance ..
end
instantiation mat :: (preorder) preorder
begin
instance
proof (standard, auto simp: less-mat-def less-eq-mat-def, goal-cases)
    case (1 A B Cij)
    thus ?case using order-trans[of A $$ (i,j) B $$ (i,j) C $$ (i,j)] by auto
qed
end
instantiation mat :: (order) order
begin
instance
    by (standard, intro eq-matI, auto simp: less-eq-mat-def order.antisym)
end
instantiation mat :: (plus) plus
begin
definition plus-mat :: ('a :: plus) mat }=>\mp@subsup{}{}{\prime}'a mat => 'a mat where
    A + B\equivmat (dim-row B) (dim-col B) (\lambda ij. A $$ ij + B $$ ij)
instance ..
end
definition map-mat :: (' }a=>\mathrm{ 'b) 吘'a mat }=>\mathrm{ 'b mat where
    map-mat f A \equivmat (dim-row A) (dim-col A) (\lambda ij.f(A$$ ij))
definition smult-mat :: 'a :: times }=>\mathrm{ ' 'a mat }=>\mathrm{ 'a mat (infixl }\mp@subsup{}{m}{\prime}70
    where }a\cdotmA\equivmap-mat (\lambdab.a*b)
definition zero-mat :: nat }=>\mathrm{ nat }=>\mp@subsup{}{}{\prime}\a:: zero mat ( ( ) m ) where
    0m nr nc \equiv mat nr nc (\lambdaij.0)
lemma elements-0-mat [simp]: elements-mat ( }0mnn\mp@code{nc})\subseteq{0
    unfolding elements-mat-def zero-mat-def by auto
definition transpose-mat :: 'a mat => 'a mat where
    transpose-mat A =mat (dim-col A) (dim-row A) (\lambda (i,j). A $$ (j,i))
definition one-mat :: nat }=>\mp@subsup{}{}{\prime}a::{\mathrm{ zero,one } mat (1m) where
    1m n =mat n n ( }\lambda(i,j). if i=j then 1 else 0)
```

```
instantiation mat :: (uminus) uminus
begin
definition uminus-mat :: ' \(a\) :: uminus mat \(\Rightarrow\) 'a mat where
    \(-A \equiv \operatorname{mat}(\) dim-row \(A)(\) dim-col \(A)(\lambda i j\). - (A\$\$ij))
instance ..
end
instantiation mat :: (minus) minus
begin
definition minus-mat \(::\left({ }^{\prime} a\right.\) :: minus \() ~ m a t ~ \Rightarrow ' a ~ m a t ~ \Rightarrow ' a ~ m a t ~ w h e r e ~\)
    \(A-B \equiv\) mat (dim-row \(B)(\) dim-col \(B)(\lambda i j . A \$ \$ i j-B \$ \$ i j)\)
instance ..
end
instantiation mat :: (semiring-0) times
begin
definition times-mat \(::\) ' \(a\) :: semiring-0 mat \(\Rightarrow\) 'a mat \(\Rightarrow\) 'a mat
    where \(A * B \equiv \operatorname{mat}(\) dim-row \(A)(\) dim-col \(B)(\lambda(i, j)\). row \(A i \cdot \operatorname{col} B j)\)
instance ..
end
definition mult-mat-vec \(::\) ' \(a\) :: semiring-0 mat \(\Rightarrow{ }^{\prime} a\) vec \(\Rightarrow\) ' \(a\) vec (infixl \(*_{v}\) 70)
    where \(A *_{v} v \equiv \operatorname{vec}(\) dim-row \(A)(\lambda i\). row \(A i \cdot v)\)
definition inverts-mat :: ' \(a\) :: semiring-1 mat \(\Rightarrow\) 'a mat \(\Rightarrow\) bool where
    inverts-mat \(A B A * B=1_{m}(\) dim-row \(A)\)
definition invertible-mat :: ' \(a\) :: semiring-1 mat \(\Rightarrow\) bool
    where invertible-mat \(A \equiv\) square-mat \(A \wedge(\exists B\). inverts-mat \(A B \wedge\) inverts-mat
\(B A\) )
definition monoid-mat \(::\) ' \(a\) :: monoid-add itself \(\Rightarrow\) nat \(\Rightarrow\) nat \(\Rightarrow\) 'a mat monoid
where
    monoid-mat ty nr nc \(\equiv\) (
    carrier \(=\) carrier-mat \(n r n c\),
    mult \(=(+)\),
    one \(=0_{m} n r n c()\)
definition ring-mat \(::\) ' \(a::\) semiring-1 itself \(\Rightarrow n a t \Rightarrow{ }^{\prime} b \Rightarrow\left({ }^{\prime} a\right.\) mat, \(\left.{ }^{\prime} b\right)\) ring-scheme where
    ring-mat ty \(n b \equiv 0\)
    carrier \(=\) carrier-mat \(n n\),
    mult \(=(*)\),
    one \(=1_{m} n\),
    zero \(=0_{m} n\),
    \(a d d=(+)\),
    \(\ldots=b\) D
```

definition module-mat $::$ ' $a$ :: semiring-1 itself $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow\left({ }^{\prime} a,^{\prime}\right.$ a mat) module

```
where
    module-mat ty nr nc \equiv (
    carrier = carrier-mat nr nc,
    mult = (*),
    one = 1m nr,
    zero = 0 m nr nc,
    add = (+),
    smult = (.m))
lemma ring-mat-simps:
    mult (ring-mat ty n b) =(*)
    add (ring-mat ty n b) = (+)
    one (ring-mat ty nb)}=1m
    zero (ring-mat ty n b) = 0 0m n n
    carrier (ring-mat ty n b) = carrier-mat n n
    unfolding ring-mat-def by auto
lemma module-mat-simps:
    mult (module-mat ty nr nc) = (*)
    add (module-mat ty nr nc) = (+)
    one (module-mat ty nr nc)=1m nr
    zero (module-mat ty nr nc) = 0 m nr nc
    carrier (module-mat ty nr nc) = carrier-mat nr nc
    smult (module-mat ty nr nc) = ( }\mp@subsup{m}{m}{\prime
    unfolding module-mat-def by auto
lemma index-zero-mat[simp]: i<nr\Longrightarrowj<nc\Longrightarrow \Longrightarrow 0m nr nc $$ (i,j)=0
    dim-row (0m nr nc) = nr dim-col ( (0m nr nc) =nc
    unfolding zero-mat-def by auto
lemma index-one-mat[simp]: i<n\Longrightarrowj<n\Longrightarrow 1m n$$(i,j)=(if i=j then
1 else 0)
    dim-row (1m n) = n dim-col (1m n)=n
    unfolding one-mat-def by auto
lemma index-add-mat[simp]:
    i<dim-row B\Longrightarrowj<dim-col B\Longrightarrow(A+B)$$ (i,j)=A$$ (i,j)+B$$ (i,j)
    dim-row }(A+B)=\mathrm{ dim-row }B\mathrm{ dim-col }(A+B)=\mathrm{ dim-col }
    unfolding plus-mat-def by auto
lemma index-minus-mat[simp]:
    i<dim-row B\Longrightarrowj<dim-col B\Longrightarrow(A-B)$$ (i,j)=A$$ (i,j) - B$$ (i,j)
    dim-row (A-B) = dim-row B dim-col ( }A-B)=\mathrm{ dim-col B
    unfolding minus-mat-def by auto
lemma index-map-mat[simp]:
    i<dim-row }A\Longrightarrowj<\mathrm{ dim-col }A\Longrightarrow\mathrm{ map-mat f A $$ (i,j) = f (A $$ (i,j))
    dim-row (map-mat f A) = dim-row A dim-col (map-mat f A) = dim-col A
    unfolding map-mat-def by auto
```

```
lemma index-smult-mat[simp]:
    i<dim-row A\Longrightarrowj< dim-col A\Longrightarrow(a\cdotmA)$$ (i,j)=a*A$$ (i,j)
    dim-row ( }a\cdotm,\mp@code{A) = dim-row A dim-col ( }a\cdotmA)=\operatorname{dim}-col 
    unfolding smult-mat-def by auto
lemma index-uminus-mat[simp]:
    i<dim-row }A\Longrightarrowj<\mathrm{ dim-col }A\Longrightarrow(-A)$$(i,j)=-(A$$(i,j)
    dim-row }(-A)=\mathrm{ dim-row A dim-col ( }-A)=\mathrm{ dim-col A
    unfolding uminus-mat-def by auto
lemma index-transpose-mat[simp]:
    i<dim-col A\Longrightarrowj< dim-row A \Longrightarrow transpose-mat A $$(i,j)=A$$(j,i)
    dim-row (transpose-mat A) = dim-col A dim-col (transpose-mat A) = dim-row A
    unfolding transpose-mat-def by auto
lemma index-mult-mat[simp]:
    i<dim-row A\Longrightarrowj<dim-col B\Longrightarrow(A*B)$$ (i,j) = row A i . col Bj
    dim-row }(A*B)=\mathrm{ dim-row A dim-col }(A*B)=\operatorname{dim
    by (auto simp: times-mat-def)
lemma dim-mult-mat-vec[simp]: dim-vec (A*vv) = dim-row }
    by (auto simp: mult-mat-vec-def)
lemma index-mult-mat-vec[simp]:i<dim-row A\Longrightarrow(A*vv)$i=row A i
    by (auto simp: mult-mat-vec-def)
lemma index-row[simp]:
    i<dim-row A\Longrightarrowj<dim-col A \Longrightarrow row A i$j=A$$ (i,j)
    dim-vec (row A i) = dim-col A
    by (auto simp: row-def)
lemma index-col[simp]: i< dim-row }A\Longrightarrowj<\operatorname{dim}-\operatorname{col}A\Longrightarrow\operatorname{col}Aj$i=A$
(i,j)
    by (auto simp: col-def)
lemma upper-triangular-one[simp]: upper-triangular (1m n)
    by (rule, auto)
lemma upper-triangular-zero[simp]:upper-triangular ( ( }\mp@subsup{0}{m}{
    by (rule, auto)
lemma mat-row-carrierI[intro,simp]: mat n nr nc r c carrier-mat nr nc
    by (unfold carrier-mat-def carrier-vec-def, auto)
lemma eq-rowI: assumes rows: \bigwedge i. i< dim-row B\Longrightarrow row A i= row B i
    and dims: dim-row }A=\operatorname{dim}\mathrm{ -row }B\mathrm{ dim-col }A=\operatorname{dim}-col 
    shows }A=
proof (rule eq-matI[OF - dims])
```

fix $i j$
assume $i: i<d i m$-row $B$ and $j: j<d i m-c o l B$
from rows $[O F i]$ have $i d$ : row $A i \$ j=$ row $B i \$ j$ by simp
show $A \$ \$(i, j)=B \$ \$(i, j)$
using index-row(1)[OF ij, folded id] index-row(1)[of iA $j] i j d i m s$ by auto
qed
lemma elements-mat-map[simp]: elements-mat (map-mat $f A)=f$ 'elements-mat A
by fastforce
lemma row-mat $[$ simp $]: i<n r \Longrightarrow$ row $($ mat nr nc $f) i=\operatorname{vec} n c(\lambda j . f(i, j))$
by auto
lemma col-mat[simp]: $j<n c \Longrightarrow \operatorname{col}($ mat $n r n c f) j=\operatorname{vec} n r(\lambda i . f(i, j))$ by auto
lemma zero-carrier-mat[simp]: $0_{m} n r n c \in$ carrier-mat nr nc unfolding carrier-mat-def by auto
lemma smult-carrier-mat[simp]:
$A \in$ carrier-mat $n r n c \Longrightarrow k \cdot m A \in$ carrier-mat $n r n c$ unfolding carrier-mat-def by auto
lemma add-carrier-mat[simp]:
$B \in$ carrier-mat $n r n c \Longrightarrow A+B \in$ carrier-mat $n r n c$ unfolding carrier-mat-def by force
lemma one-carrier-mat $[$ simp $]: 1_{m} n \in$ carrier-mat $n n$ unfolding carrier-mat-def by auto
lemma uminus-carrier-mat:
$A \in$ carrier-mat $n r n c \Longrightarrow(-A \in$ carrier-mat $n r n c)$
unfolding carrier-mat-def by auto
lemma uminus-carrier-iff-mat $[$ simp $]$ :
$(-A \in$ carrier-mat $n r n c)=(A \in$ carrier-mat $n r n c)$
unfolding carrier-mat-def by auto
lemma minus-carrier-mat:
$B \in$ carrier-mat $n r n c \Longrightarrow(A-B \in$ carrier-mat $n r n c)$
unfolding carrier-mat-def by auto
lemma transpose-carrier-mat $[$ simp $]:($ transpose-mat $A \in$ carrier-mat nc $n r)=(A$ $\in$ carrier-mat $n r n c$ )
unfolding carrier-mat-def by auto
lemma row-carrier-vec $[\operatorname{simp}]: i<n r \Longrightarrow A \in \operatorname{carrier-mat} n r n c \Longrightarrow$ row $A i \in$
carrier-vec nc
unfolding carrier-vec-def by auto
lemma col-carrier-vec[simp]: $j<n c \Longrightarrow A \in$ carrier-mat $n r n c \Longrightarrow \operatorname{col} A j \in$ carrier-vec $n r$
unfolding carrier-vec-def by auto
lemma mult-carrier-mat[simp]:
$A \in$ carrier-mat $n r n \Longrightarrow B \in$ carrier-mat $n n c \Longrightarrow A * B \in$ carrier-mat $n r n c$ unfolding carrier-mat-def by auto
lemma mult-mat-vec-carrier[simp]:
$A \in$ carrier-mat $n r n \Longrightarrow v \in$ carrier-vec $n \Longrightarrow A *_{v} v \in$ carrier-vec $n r$
unfolding carrier-mat-def carrier-vec-def by auto
lemma comm-add-mat[ac-simps]:
( $A$ :: ' $a$ :: comm-monoid-add mat $) \in$ carrier-mat $n r n c \Longrightarrow B \in$ carrier-mat $n r$ $n c \Longrightarrow A+B=B+A$
by (intro eq-matI, auto simp: ac-simps)
lemma minus-r-inv-mat[simp]:
$\left(A::{ }^{\prime} a::\right.$ group-add mat $) \in$ carrier-mat $n r n c \Longrightarrow(A-A)=0_{m} n r n c$ by (intro eq-matI, auto)
lemma uminus-l-inv-mat[simp]:
( $A::^{\prime}$ 'a $::$ group-add mat $) \in$ carrier-mat $n r n c \Longrightarrow(-A+A)=0_{m} n r n c$ by (intro eq-matI, auto)
lemma add-inv-exists-mat:
( $A$ :: 'a :: group-add mat $) \in$ carrier-mat $n r n c \Longrightarrow \exists B \in$ carrier-mat nr nc. $B$ $+A=0_{m} n r n c \wedge A+B=0_{m} n r n c$
by (intro bexI[of $--A]$, auto)
lemma assoc-add-mat[simp]:
( $A$ :: 'a :: monoid-add mat $) \in$ carrier-mat $n r n c \Longrightarrow B \in$ carrier-mat $n r n c \Longrightarrow$ $C \in$ carrier-mat $n r n c$
$\Longrightarrow(A+B)+C=A+(B+C)$
by (intro eq-matI, auto simp: ac-simps)
lemma uminus-add-mat: fixes $A$ :: ' $a$ :: group-add mat assumes $A \in$ carrier-mat nr nc
and $B \in$ carrier-mat $n r n c$
shows $-(A+B)=-B+-A$
by (intro eq-matI, insert assms, auto simp: minus-add)
lemma transpose-transpose[simp]:
transpose-mat (transpose-mat $A)=A$

```
    by (intro eq-matI, auto)
lemma transpose-one[simp]: transpose-mat (1m n) = (1m n)
    by auto
lemma row-transpose[simp]:
    j<dim-col A > row (transpose-mat A) j= col A j
    unfolding row-def col-def
    by (intro eq-vecI, auto)
lemma col-transpose[simp]:
    i< dim-row A \Longrightarrowcol (transpose-mat A) i = row A i
    unfolding row-def col-def
    by (intro eq-vecI, auto)
lemma row-zero[simp]:
    i<nr\Longrightarrow row (0m nr nc) i= 0v nc
    by (intro eq-vecI, auto)
lemma col-zero[simp]:
    j<nc\Longrightarrowcol (0m nr nc) j= 0vv nr
        by (intro eq-vecI, auto)
lemma row-one[simp]:
    i<n\Longrightarrow row (1m n) i= unit-vec n i
    by (intro eq-vecI, auto)
lemma col-one[simp]:
    j<n\Longrightarrowcol (1m n) j=unit-vec n j
    by (intro eq-vecI, auto)
lemma transpose-add: A carrier-mat nr nc \LongrightarrowB\incarrier-mat nr nc
    \Longrightarrow \text { transpose-mat (A+B) = transpose-mat A + transpose-mat B}
    by (intro eq-matI, auto)
lemma transpose-minus: A \in carrier-mat nr nc \LongrightarrowB E carrier-mat nr nc
    \Longrightarrow ~ t r a n s p o s e - m a t ~ ( A - B ) = t r a n s p o s e - m a t ~ A ~ - ~ t r a n s p o s e - m a t ~ B ~
    by (intro eq-matI, auto)
lemma transpose-uminus: transpose-mat (-A) = - (transpose-mat A)
    by (intro eq-matI, auto)
lemma row-add[simp]:
    A carrier-mat nr nc \LongrightarrowB\in carrier-mat nr nc \Longrightarrowi<nr
    \Longrightarrow \text { row } ( A + B ) i = \text { row } A
    i<dim-row }A\Longrightarrow\mathrm{ dim-row }B=\operatorname{dim}\mathrm{ -row }A\Longrightarrow\operatorname{dim}-\operatorname{col}B=\operatorname{dim}-\operatorname{col}A\Longrightarrow\mathrm{ row
(A+B)i= row A i + row B i
    by (rule eq-vecI, auto)
```

lemma col-add[simp]:
$A \in$ carrier-mat $n r n c \Longrightarrow B \in$ carrier-mat $n r n c \Longrightarrow j<n c$
$\Longrightarrow \operatorname{col}(A+B) j=\operatorname{col} A j+\operatorname{col} B j$
by (rule eq-vecI, auto)
lemma row-mult $[$ simp $]$ : assumes $m: A \in$ carrier-mat nr $n B \in$ carrier-mat n nc and $i: i<n r$
shows row $(A * B) i=$ vec $n c(\lambda j$. row $A i \cdot \operatorname{col} B j)$
by (rule eq-vecI, insert $m i$, auto)
lemma col-mult[simp]: assumes $m: A \in$ carrier-mat $n r n B \in$ carrier-mat $n$ nc and $j: j<n c$
shows $\operatorname{col}(A * B) j=\operatorname{vec} n r(\lambda i$ row $A i \cdot \operatorname{col} B j)$
by (rule eq-vecI, insert mj, auto)
lemma transpose-mult:
( $A$ :: 'a :: comm-semiring-0 mat $) \in$ carrier-mat $n r n \Longrightarrow B \in$ carrier-mat $n$ nc $\Longrightarrow$ transpose-mat $(A * B)=$ transpose-mat $B *$ transpose-mat $A$
by (intro eq-matI, auto simp: comm-scalar-prod $[$ of $-n]$ )
lemma left-add-zero-mat[simp]:
( $A$ :: 'a :: monoid-add mat $) \in$ carrier-mat $n r n c \Longrightarrow 0_{m} n r n c+A=A$
by (intro eq-matI, auto)
lemma add-uminus-minus-mat: $A \in$ carrier-mat nr nc $\Longrightarrow B \in$ carrier-mat nr nc $\Longrightarrow$ $A+(-B)=A-(B::$ 'a :: group-add mat $)$ by (intro eq-matI, auto)
lemma right-add-zero-mat $[$ simp $]: A \in$ carrier-mat $n r n c \Longrightarrow$ $A+0_{m} n r n c=\left(A::{ }^{\prime} a::\right.$ monoid-add mat $)$ by (intro eq-matI, auto)
lemma left-mult-zero-mat:
$A \in$ carrier-mat $n n c \Longrightarrow 0_{m} n r n * A=0_{m} n r n c$ by (intro eq-matI, auto)
 (dim-col A)
by (rule left-mult-zero-mat, unfold carrier-mat-def, simp)
lemma right-mult-zero-mat:
$A \in$ carrier-mat $n r n \Longrightarrow A * 0_{m} n n c=0_{m} n r n c$
by (intro eq-matI, auto)
lemma right-mult-zero-mat' $[$ simp $]: \operatorname{dim}-c o l ~ A=n \Longrightarrow A * O_{m} n n c=0_{m}$ (dim-row A) $n c$
by (rule right-mult-zero-mat, unfold carrier-mat-def, simp)

## lemma left-mult-one-mat:

( $A$ :: 'a :: semiring-1 mat) $\in$ carrier-mat $n r n c \Longrightarrow 1_{m} n r * A=A$
by (intro eq-matI, auto)
lemma left-mult-one-mat' $[$ simp $]$ : dim-row $\left(A::{ }^{\prime} a\right.$ :: semiring-1 mat $)=n \Longrightarrow 1_{m}$ $n * A=A$
by (rule left-mult-one-mat, unfold carrier-mat-def, simp)
lemma right-mult-one-mat:
( $A::$ ' $a::$ semiring-1 mat $) \in$ carrier-mat $n r n c \Longrightarrow A * 1_{m} n c=A$
by (intro eq-matI, auto)
lemma right-mult-one-mat'[simp]:dim-col ( $A$ :: ' $a$ :: semiring-1 mat $)=n \Longrightarrow A$ * $1{ }_{m} n=A$
by (rule right-mult-one-mat, unfold carrier-mat-def, simp)
lemma one-mult-mat-vec[simp]:
( $v::$ ' $a::$ semiring-1 vec) $\in$ carrier-vec $n \Longrightarrow 1_{m} n *_{v} v=v$ by (intro eq-vecI, auto)
lemma minus-add-uminus-mat: fixes $A$ :: ' $a$ :: group-add mat shows $A \in$ carrier-mat $n r n c \Longrightarrow B \in$ carrier-mat $n r n c \Longrightarrow$
$A-B=A+(-B)$
by (intro eq-matI, auto)
lemma add-mult-distrib-mat[algebra-simps]: assumes $m$ : $A \in$ carrier-mat $n r n$ $B \in$ carrier-mat nr $n C \in$ carrier-mat $n$ nc
shows $(A+B) * C=A * C+B * C$
using $m$ by (intro eq-matI, auto simp: add-scalar-prod-distrib[of - n])
lemma mult-add-distrib-mat[algebra-simps]: assumes m: $A \in$ carrier-mat nr $n$ $B \in$ carrier-mat $n$ nc $C \in$ carrier-mat $n$ nc shows $A *(B+C)=A * B+A * C$
using $m$ by (intro eq-matI, auto simp: scalar-prod-add-distrib[of - n])
lemma add-mult-distrib-mat-vec[algebra-simps]: assumes m: $A \in$ carrier-mat $n r$ nc
$B \in$ carrier-mat nr nc $v \in$ carrier-vec nc
shows $(A+B) *_{v} v=A *_{v} v+B *_{v} v$
using $m$ by (intro eq-vecI, auto intro!: add-scalar-prod-distrib)
lemma mult-add-distrib-mat-vec[algebra-simps]: assumes m: $A \in$ carrier-mat $n r$ $n c$
$v_{1} \in$ carrier-vec nc $v_{2} \in$ carrier-vec nc
shows $A *_{v}\left(v_{1}+v_{2}\right)=A *_{v} v_{1}+A *_{v} v_{2}$
using $m$ by (intro eq-vecI, auto simp: scalar-prod-add-distrib[of -nc])
lemma mult-mat-vec:
assumes $m:\left(A::{ }^{\prime} a::\right.$ field mat $) \in$ carrier-mat $n r n c$ and $v: v \in$ carrier-vec nc

```
    shows}A\mp@subsup{*}{v}{}(k\cdotvv)=k\cdotv(A*vv)(is ?l = ?r)
proof
    have nr: dim-vec ?l = nr using mv by auto
    also have ... = dim-vec ?r using m v by auto
    finally show dim-vec ?l = dim-vec ?r.
    show \i.i<dim-vec ?r \Longrightarrow?l $i=?r $ i
    proof -
        fix i assume i<dim-vec ?r
        hence i: i< dim-row A using nr m by auto
    hence i2: i< dim-vec ( }A\mp@subsup{*}{v}{}v\mathrm{ ) using m by auto
    show ?l $ i= ?r $ i
    apply (subst (1) mult-mat-vec-def)
    apply (subst (2) smult-vec-def)
    unfolding index-vec[OF i] index-vec[OF i2]
    unfolding mult-mat-vec-def smult-vec-def
    unfolding scalar-prod-def index-vec[OF i]
    by (simp add: mult.left-commute sum-distrib-left)
    qed
qed
```

lemma assoc-scalar-prod: assumes $*: v_{1} \in$ carrier-vec $n r A \in$ carrier-mat nr nc $v_{2} \in$ carrier-vec nc
shows vec nc $\left(\lambda j . v_{1} \cdot \operatorname{col} A j\right) \cdot v_{2}=v_{1} \cdot \operatorname{vec} n r\left(\lambda i\right.$. row $\left.A i \cdot v_{2}\right)$
proof -
have vec $n c\left(\lambda j . v_{1} \cdot \operatorname{col} A j\right) \cdot v_{2}=\left(\sum i \in\{0 . .<n c\}\right.$. vec $n c\left(\lambda j . \sum k \in\{0 . .<n r\}\right.$.
$\left.\left.v_{1} \$ k * \operatorname{col} A j \$ k\right) \$ i * v_{2} \$ i\right)$
unfolding scalar-prod-def using $*$ by auto
also have $\ldots=\left(\sum i \in\{0 . .<n c\} .\left(\sum k \in\{0 . .<n r\} . v_{1} \$ k * \operatorname{col} A i \$ k\right) * v_{2} \$ i\right)$
by (rule sum.cong, auto)
also have $\ldots=\left(\sum i \in\{0 . .<n c\} .\left(\sum k \in\{0 . .<n r\} . v_{1} \$ k * \operatorname{col} A i \$ k * v_{2} \$ i\right)\right)$
unfolding sum-distrib-right ..
also have $\ldots=\left(\sum k \in\{0 . .<n r\} .\left(\sum i \in\{0 . .<n c\} . v_{1} \$ k * \operatorname{col} A i \$ k * v_{2} \$ i\right)\right)$
by (rule sum.swap)
also have $\ldots=\left(\sum k \in\{0 . .<n r\} .\left(\sum i \in\{0 . .<n c\} . v_{1} \$ k *\left(\operatorname{col} A i \$ k * v_{2} \$\right.\right.\right.$
i)))
by (simp add: ac-simps)
also have $\ldots=\left(\sum k \in\{0 . .<n r\} . v_{1} \$ k *\left(\sum i \in\{0 . .<n c\}\right.\right.$. col $\left.\left.A i \$ k * v_{2} \$ i\right)\right)$
unfolding sum-distrib-left ..
also have $\ldots=\left(\sum k \in\{0 . .<n r\} . v_{1} \$ k * \operatorname{vec} n r\left(\lambda k . \sum i \in\{0 . .<n c\}\right.\right.$. row $A k \$$
$\left.i * v_{2} \$ i\right) \$ k$ )
using * by auto
also have $\ldots=v_{1} \cdot \operatorname{vec} n r\left(\lambda i\right.$. row $\left.A i \cdot v_{2}\right)$ unfolding scalar-prod-def using

* by $\operatorname{simp}$
finally show ?thesis.
qed
lemma transpose-vec-mult-scalar:
fixes $A$ :: ' $a$ :: comm-semiring-0 mat

```
    assumes A:A\in carrier-mat nr nc
    and x:x\in carrier-vec nc
    and y: y\in carrier-vec nr
    shows (transpose-mat }A\mp@subsup{*}{v}{}y)\cdotx=y\cdot(A*v x
proof -
    have (transpose-mat A*v y)=vec nc (\lambdai.col A i . y)
        unfolding mult-mat-vec-def using A by auto
    also have ... = vec nc (\lambdai.y.\operatorname{col A i})
    by (intro eq-vecI, simp, rule comm-scalar-prod [OF - y], insert A, auto)
    also have ... 戊=y vec nr (\lambdai. row A i | x)
    by (rule assoc-scalar-prod[OF y A x])
    also have vec nr (\lambdai. row A i | x)=A*v}
    unfolding mult-mat-vec-def using A by auto
    finally show ?thesis .
qed
lemma assoc-mult-mat[simp]:
    A\in carrier-mat n}\mp@subsup{n}{1}{}\mp@subsup{n}{2}{}\LongrightarrowB\in\mathrm{ carrier-mat }\mp@subsup{n}{2}{}\mp@subsup{n}{3}{}\LongrightarrowC\in\mathrm{ carrier-mat n}\mp@subsup{n}{3}{}\mp@subsup{n}{4}{
    \Longrightarrow ( A * B ) * C = A * ( B * C )
    by (intro eq-matI, auto simp: assoc-scalar-prod)
lemma assoc-mult-mat-vec[simp]:
    A\incarrier-mat n}\mp@subsup{n}{1}{}\mp@subsup{n}{2}{}\LongrightarrowB\in\mathrm{ carrier-mat }\mp@subsup{n}{2}{}\mp@subsup{n}{3}{}\Longrightarrowv\in\mathrm{ carrier-vec n}\mp@subsup{n}{3}{
    \Longrightarrow(A*B)*vv=A*v(B*vv)
    by (intro eq-vecI, auto simp add: mult-mat-vec-def assoc-scalar-prod)
lemma comm-monoid-mat: comm-monoid (monoid-mat TYPE('a :: comm-monoid-add)
nr nc)
    by (unfold-locales, auto simp: monoid-mat-def ac-simps)
lemma comm-group-mat: comm-group (monoid-mat TYPE('a :: ab-group-add) nr
nc)
    by (unfold-locales, insert add-inv-exists-mat, auto simp: monoid-mat-def ac-simps
Units-def)
lemma semiring-mat: semiring (ring-mat TYPE('a :: semiring-1) n b)
    by (unfold-locales, auto simp: ring-mat-def algebra-simps)
lemma ring-mat: ring (ring-mat TYPE('a :: comm-ring-1) n b)
    by (unfold-locales, insert add-inv-exists-mat, auto simp: ring-mat-def algebra-simps
Units-def)
lemma abelian-group-mat: abelian-group (module-mat TYPE('a :: comm-ring-1)
nr nc)
    by (unfold-locales, insert add-inv-exists-mat, auto simp: module-mat-def Units-def)
lemma row-smult[simp]: assumes i: i< dim-row A
    shows row (k\cdotmA)i=k\cdotv(row A i)
    by (rule eq-vecI, insert i, auto)
```

```
lemma col-smult[simp]: assumes i: i<dim-col A
```

    shows \(\operatorname{col}(k \cdot m A) i=k \cdot v(\operatorname{col} A i)\)
    by (rule eq-vecI, insert \(i\), auto)
    lemma row-uminus[simp]: assumes $i$ : $i<$ dim-row $A$
shows row $(-A) i=-($ row $A i)$
by (rule eq-vecI, insert $i$, auto)
lemma scalar-prod-uminus-left[simp]: assumes dim: dim-vec $v=\operatorname{dim-vec~(w~::~'a~}$
:: ring vec)
shows $-v \cdot w=-(v \cdot w)$
unfolding scalar-prod-def dim[symmetric]
by (subst sum-negf [symmetric], rule sum.cong, auto)
lemma col-uminus[simp]: assumes $i: i<\operatorname{dim}-c o l ~ A$
shows $\operatorname{col}(-A) i=-(\operatorname{col} A i)$
by (rule eq-vecI, insert $i$, auto)
lemma scalar-prod-uminus-right $[\operatorname{simp}]$ : assumes dim: dim-vec $v=\operatorname{dim-vec}(w::$ ' $a::$ ring vec)
shows $v \cdot-w=-(v \cdot w)$
unfolding scalar-prod-def dim
by (subst sum-negf $[$ symmetric], rule sum.cong, auto)
context fixes $A B::{ }^{\prime} a$ :: ring mat
assumes dim: dim-col $A=$ dim-row $B$
begin
lemma uminus-mult-left-mat[simp]: $(-A * B)=-(A * B)$
by (intro eq-matI, insert dim, auto)
lemma uminus-mult-right-mat $[\operatorname{simp}]:(A *-B)=-(A * B)$
by (intro eq-matI, insert dim, auto)
end
lemma minus-mult-distrib-mat[algebra-simps]: fixes $A$ :: 'a :: ring mat
assumes $m$ : $A \in$ carrier-mat nr $n B \in$ carrier-mat $n r n C \in$ carrier-mat $n$ nc
shows $(A-B) * C=A * C-B * C$
unfolding minus-add-uminus-mat $[$ OF $m(1,2)]$
add-mult-distrib-mat[OF $m$ (1) uminus-carrier-mat[OF m(2)] $m$ (3)]
by (subst uminus-mult-left-mat, insert m, auto)
lemma minus-mult-distrib-mat-vec[algebra-simps]: assumes $A:(A$ :: 'a :: ring mat) $\in$ carrier-mat nr nc
and $B: B \in$ carrier-mat $n r n c$
and $v: v \in$ carrier-vec $n c$
shows $(A-B) *_{v} v=A *_{v} v-B *_{v} v$
unfolding minus-add-uminus-mat $[O F A B]$
by (subst add-mult-distrib-mat-vec $[O F A-v]$, insert $A B$, auto)
lemma mult-minus-distrib-mat-vec[algebra-simps]: assumes $A:(A::$ 'a :: ring mat) $\in$ carrier-mat nr nc
and $v: v \in$ carrier-vec $n c$
and $w: w \in$ carrier-vec $n c$
shows $A *_{v}(v-w)=A *_{v} v-A *_{v} w$
unfolding minus-add-uminus-vec $[O F \quad v \quad w]$
by (subst mult-add-distrib-mat-vec $[O F A]$, insert $A v w$, auto)
lemma mult-minus-distrib-mat[algebra-simps]: fixes $A$ :: ' $a$ :: ring mat
assumes m: $A \in$ carrier-mat nr $n B \in$ carrier-mat $n$ nc $C \in$ carrier-mat $n$ nc shows $A *(B-C)=A * B-A * C$
unfolding minus-add-uminus-mat[OF m(2,3)]
mult-add-distrib-mat[OF $m$ (1) $m$ (2) uminus-carrier-mat[OF $m$ (3)]]
by (subst uminus-mult-right-mat, insert $m$, auto)
lemma uminus-mult-mat-vec $[\operatorname{simp}]$ : assumes $v: \operatorname{dim-vec} v=\operatorname{dim}-\operatorname{col}\left(A::{ }^{\prime} a \quad::\right.$ ring mat)
shows $-A *_{v} v=-\left(A *_{v} v\right)$
using $v$ by (intro eq-vecI, auto)
lemma uminus-zero-vec-eq: assumes $v:(v::$ ' $a::$ group-add vec $) \in$ carrier-vec $n$ shows $\left(-v=0_{v} n\right)=\left(v=0_{v} n\right)$
proof assume $z:-v=0_{v} n$
\{
fix $i$
assume $i: i<n$
have $v \$ i=-(-(v \$ i))$ by simp
also have $-(v \$ i)=0$ using $\arg -\operatorname{cong}[O F z$, of $\lambda v . v \$ i] i v$ by auto also have $-0=\left(0::{ }^{\prime} a\right)$ by $\operatorname{simp}$ finally have $v \$ i=0$.
\}
thus $v=O_{v} n$ using $v$
by (intro eq-vecI, auto)
qed auto
lemma map-carrier-mat[simp]:
(map-mat $f A \in$ carrier-mat $n r n c)=(A \in$ carrier-mat $n r n c)$
unfolding carrier-mat-def by auto
lemma col-map-mat[simp]:
assumes $j<\operatorname{dim}-\operatorname{col} A$ shows $\operatorname{col}($ map-mat $f A) j=\operatorname{map-vec} f(\operatorname{col} A j)$
unfolding map-mat-def map-vec-def using assms by auto
lemma scalar-vec-one[simp]: $1 \cdot v(v:: ' a$ :: semiring- 1 vec $)=v$
by (rule eq-vecI, auto)
lemma scalar-prod-smult-right[simp]:
dim-vec $w=$ dim-vec $v \Longrightarrow w \cdot(k \cdot v v)=\left(k::{ }^{\prime} a::\right.$ comm-semiring- 0$) *(w \cdot v)$ unfolding scalar-prod-def sum-distrib-left
by (auto intro: sum.cong simp: ac-simps)
lemma scalar-prod-smult-left[simp]:
dim-vec $w=$ dim-vec $v \Longrightarrow(k \cdot v w) \cdot v=(k:: ' a::$ comm-semiring- 0$) *(w \cdot v)$
unfolding scalar-prod-def sum-distrib-left
by (auto intro: sum.cong simp: ac-simps)
lemma mult-smult-distrib: assumes $A: A \in$ carrier-mat $n r n$ and $B: B \in c a r$ -rier-mat $n n c$
shows $A *(k \cdot m B)=(k::$ ' $a::$ comm-semiring- 0$) \cdot m(A * B)$
by (rule eq-matI, insert $A B$, auto)
lemma add-smult-distrib-left-mat: assumes $A \in$ carrier-mat nr nc $B \in$ carrier-mat $n r n c$
shows $k \cdot{ }_{m}(A+B)=\left(k::{ }^{\prime} a::\right.$ semiring $) \cdot{ }_{m} A+k \cdot m B$
by (rule eq-matI, insert assms, auto simp: field-simps)
lemma add-smult-distrib-right-mat: assumes $A \in$ carrier-mat nr nc
shows $(k+l) \cdot m A=\left(k::{ }^{\prime} a::\right.$ semiring $) \cdot m A+l \cdot m A$
by (rule eq-matI, insert assms, auto simp: field-simps)
lemma mult-smult-assoc-mat: assumes $A: A \in$ carrier-mat $n r n$ and $B: B \in$ carrier-mat $n n c$
shows $(k \cdot m A) * B=\left(k::{ }^{\prime} a::\right.$ comm-semiring- 0$) \cdot m(A * B)$
by (rule eq-matI, insert $A B$, auto)
definition similar-mat-wit :: 'a :: semiring-1 mat $\Rightarrow{ }^{\prime} a$ mat $\Rightarrow$ 'a mat $\Rightarrow{ }^{\prime} a$ mat $\Rightarrow$ bool where
similar-mat-wit $A B P Q=$ (let $n=$ dim-row $A$ in $\{A, B, P, Q\} \subseteq$ carrier-mat $n$ $n \wedge P * Q=1_{m} n \wedge Q * P=1_{m} n \wedge$

$$
A=P * B * Q)
$$

definition similar-mat :: 'a :: semiring-1 mat $\Rightarrow{ }^{\prime} a$ mat $\Rightarrow$ bool where similar-mat $A B=(\exists P Q$. similar-mat-wit $A B P Q)$
lemma similar-matD: assumes similar-mat $A B$
shows $\exists n P Q .\{A, B, P, Q\} \subseteq$ carrier-mat $n n \wedge P * Q=1_{m} n \wedge Q * P=$ $1_{m} n \wedge A=P * B * Q$
using assms unfolding similar-mat-def similar-mat-wit-def[abs-def] Let-def by blast
lemma similar-matI: assumes $\{A, B, P, Q\} \subseteq$ carrier-mat $n n P * Q=1_{m} n Q$ * $P=1_{m} n A=P * B * Q$ shows similar-mat $A B$ unfolding similar-mat-def
by (rule ex $[$ of - $P]$, rule ex $[$ of $-Q]$, unfold similar-mat-wit-def Let-def, insert assms, auto)
fun pow-mat :: ' $a$ :: semiring-1 mat $\Rightarrow$ nat $\Rightarrow{ }^{\prime} a$ mat (infixr ${ }_{m}$ 75) where
$A \widehat{m}_{m} 0=1_{m}($ dim-row $A)$
$\mid A \widehat{m}_{m}($ Suc $k)=A \widehat{m}_{m} k A$
lemma pow-mat-dim[simp]:
dim-row $\left(A \widehat{m}_{m} k\right)=$ dim-row $A$
dim-col $\left(A \widehat{m}_{m} k\right)=($ if $k=0$ then dim-row $A$ else dim-col $A)$
by (induct $k$, auto)
lemma pow-mat-dim-square [simp]:
$A \in$ carrier-mat $n n \Longrightarrow$ dim-row $\left(A{ }_{m} k\right)=n$
$A \in$ carrier-mat $n n \Longrightarrow \operatorname{dim}-\operatorname{col}\left(A \widehat{m}_{m} k\right)=n$
by auto
lemma pow-carrier-mat[simp]: $A \in$ carrier-mat $n n \Longrightarrow A{ }_{m} k \in$ carrier-mat $n$ $n$
unfolding carrier-mat-def by auto
definition diag-mat $::$ ' $a$ mat $\Rightarrow$ 'a list where
diag-mat $A=\operatorname{map}(\lambda i . A \$ \$(i, i))[0 . .<$ dim-row $A]$
lemma prod-list-diag-prod: prod-list $(\operatorname{diag-mat} A)=\left(\prod i=0 . .<\right.$ dim-row $A . A$ $\$ \$(i, i))$
unfolding diag-mat-def
by (subst prod.distinct-set-conv-list[symmetric], auto)
lemma diag-mat-transpose $[$ simp $]$ : dim-row $A=\operatorname{dim}$-col $A \Longrightarrow$ diag-mat (transpose-mat $A)=\operatorname{diag-mat} A$ unfolding diag-mat-def by auto
lemma diag-mat-zero[simp]: diag-mat $\left(0_{m} n n\right)=$ replicate $n 0$ unfolding diag-mat-def
by (rule nth-equalityI, auto)
lemma diag-mat-one[simp]: diag-mat $\left(1_{m} n\right)=$ replicate $n 1$ unfolding diag-mat-def
by (rule nth-equalityI, auto)
lemma pow-mat-ring-pow: assumes $A:\left(A::\left({ }^{\prime} a::\right.\right.$ semiring- 1$)$ mat $) \in$ carrier-mat $n$ n
shows $A \widehat{m}_{m} k=A[]_{\text {ring-mat } \operatorname{TYPE}\left({ }^{\prime} a\right) n b} k$
$\left(\right.$ is - $\left.=A[]_{? C} k\right)$
proof -
interpret semiring ?C by (rule semiring-mat)
show ?thesis
by (induct $k$, insert $A$, auto simp: ring-mat-def nat-pow-def)
qed
definition diagonal-mat :: 'a::zero mat $\Rightarrow$ bool where
diagonal-mat $A \equiv \forall i<$ dim-row $A . \forall j<\operatorname{dim}$-col $A . i \neq j \longrightarrow A \$ \$(i, j)=0$
definition (in comm-monoid-add) sum-mat :: 'a mat $\Rightarrow$ ' $a$ where sum-mat $A=\operatorname{sum}(\lambda i j . A \$ \$ i j)(\{0 . .<$ dim-row $A\} \times\{0 . .<\operatorname{dim}-c o l A\})$
lemma sum-mat- $0[$ simp $]$ : sum-mat $\left(0_{m} n r n c\right)=(0::$ 'a :: comm-monoid-add $)$ unfolding sum-mat-def
by (rule sum.neutral, auto)
lemma sum-mat-add: assumes $A:\left(A::{ }^{\prime} a::\right.$ comm-monoid-add mat $) \in$ car-rier-mat $n r n c$ and $B: B \in$ carrier-mat $n r n c$
shows sum-mat $(A+B)=$ sum-mat $A+$ sum-mat $B$
proof -
from $A B$ have $i d$ : dim-row $A=n r d i m$-row $B=n r \operatorname{dim}-c o l A=n c \operatorname{dim}$-col $B$ $=n c$
by auto
show ?thesis unfolding sum-mat-def id
by (subst sum.distrib[symmetric], rule sum.cong, insert $A B$, auto)
qed

### 4.3 Update Operators

definition update-vec $::$ 'a vec $\Rightarrow$ nat $\Rightarrow{ }^{\prime} a \Rightarrow$ 'a vec $\left(-\left.\right|_{v}-\mapsto-[60,61,62] 60\right)$ where $\left.v\right|_{v} i \mapsto a=\operatorname{vec}($ dim-vec $v)\left(\lambda i^{\prime}\right.$. if $i^{\prime}=i$ then a else $\left.v \$ i^{\prime}\right)$
definition update-mat $::$ 'a mat $\Rightarrow$ nat $\times n a t \Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a$ mat $\left(-\left.\right|_{m}-\mapsto-[60,61,62]\right.$ 60)
where $\left.A\right|_{m} i j \mapsto a=$ mat $($ dim-row $A)(\operatorname{dim-col} A)\left(\lambda i j^{\prime}\right.$. if $i j^{\prime}=i j$ then a else $\left.A \$ \$ i j^{\prime}\right)$
lemma dim-update-vec[simp]:
dim-vec $\left(\left.v\right|_{v} i \mapsto a\right)=$ dim-vec $v$ unfolding update-vec-def by simp
lemma index-update-vec1[simp]:
assumes $i<d i m$-vec $v$ shows $\left(\left.v\right|_{v} i \mapsto a\right) \$ i=a$
unfolding update-vec-def using assms by simp
lemma index-update-vec2 [simp]:
assumes $i^{\prime} \neq i$ shows $\left(\left.v\right|_{v} i \mapsto a\right) \$ i^{\prime}=v \$ i^{\prime}$
unfolding update-vec-def
using assms apply transfer unfolding $m k$-vec-def by auto
lemma dim-update-mat[simp]:
dim-row $\left(\left.A\right|_{m} i j \mapsto a\right)=$ dim-row $A$
$\operatorname{dim}-c o l\left(\left.A\right|_{m}\right.$ ij $\left.\mapsto a\right)=\operatorname{dim}-c o l A$ unfolding update-mat-def by $\operatorname{simp}+$
lemma index-update-mat1 [simp]:
assumes $i<$ dim-row $A j<\operatorname{dim}$-col $A$ shows $\left(\left.A\right|_{m}(i, j) \mapsto a\right) \$ \$(i, j)=a$
unfolding update-mat-def using assms by simp
lemma index-update-mat2[simp]:
assumes $i^{\prime}: i^{\prime}<\operatorname{dim}$-row $A$ and $j^{\prime}: j^{\prime}<\operatorname{dim}-c o l A$ and neq: $\left(i^{\prime}, j^{\prime}\right) \neq i j$
shows $\left(\left.A\right|_{m} i j \mapsto a\right) \$ \$\left(i^{\prime}, j^{\prime}\right)=A \$ \$\left(i^{\prime}, j^{\prime}\right)$
unfolding update-mat-def using assms by auto

### 4.4 Block Vectors and Matrices

definition append-vec :: 'a vec $\Rightarrow$ 'a vec $\Rightarrow$ 'a vec (infixr @ ${ }_{v} 65$ ) where $v @_{v} w \equiv$ let $n=\operatorname{dim-vec} v ; m=\operatorname{dim-vec} w$ in vec $(n+m)(\lambda i$. if $i<n$ then $v \$ i$ else $w \$(i-n))$
lemma index-append-vec [simp]: $i<\operatorname{dim}$-vec $v+$ dim-vec $w$ $\Longrightarrow\left(v @_{v} w\right) \$ i=($ if $i<$ dim-vec $v$ then $v \$ i$ else $w \$(i-$ dim-vec $v))$ $\operatorname{dim}$-vec $\left(v @_{v} w\right)=\operatorname{dim}-v e c \quad v+\operatorname{dim}-v e c ~ w$ unfolding append-vec-def Let-def by auto
lemma append-carrier-vec[simp,intro]: $v \in$ carrier-vec $n 1 \Longrightarrow w \in$ carrier-vec $n 2 \Longrightarrow v @_{v} w \in \operatorname{carrier-vec}(n 1+n \mathbb{2})$ unfolding carrier-vec-def by auto
lemma scalar-prod-append: assumes v1 $\in$ carrier-vec n1 v2 $\in$ carrier-vec n2 $w 1 \in$ carrier-vec $n 1 w 2 \in$ carrier-vec n2
shows $\left(v 1 @_{v} v 2\right) \cdot\left(w 1 @_{v} w 2\right)=v 1 \cdot w 1+v 2 \cdot w 2$
proof -
from assms have dim: dim-vec v1 = n1 dim-vec v2 = n2 dim-vec $w 1=n 1$
dim-vec $w 2=$ n2 by auto
have $i d:\{0 . .<n 1+n 2\}=\{0 . .<n 1\} \cup\{n 1 . .<n 1+n 2\}$ by auto
have id2: $\{n 1 . .<n 1+n 2\}=($ plus n1 $) \cdot\{0 . .<n 2\}$
by (simp add: ac-simps)
have $\left(v 1 @_{v} v 2\right) \cdot\left(w 1 @_{v} w 2\right)=\left(\sum i=0 . .<n 1 . v 1 \$ i * w 1 \$ i\right)+$ $\left(\sum i=n 1 . .<n 1+n 2 . v 2 \$(i-n 1) *\right.$ w2 $\left.\$(i-n 1)\right)$
unfolding scalar-prod-def
by (auto simp: dim id, subst sum.union-disjoint, insert assms, force+)
also have $\left(\sum i=n 1 . .<n 1+n 2 . v 2 \$(i-n 1) * w 2 \$(i-n 1)\right)$
$=\left(\sum i=0 . .<n 2 . v 2 \$ i * w 2 \$ i\right)$
by (rule sum.reindex-cong [OF - id2]) simp-all
finally show ?thesis by (simp, insert assms, auto simp: scalar-prod-def)
qed
definition vec-first v $n \equiv \operatorname{vec} n(\lambda i . v \$ i)$
definition vec-last $v n \equiv$ vec $n(\lambda i . v \$(\operatorname{dim}-v e c v-n+i))$
lemma dim-vec-first[simp]: dim-vec (vec-first $v n$ ) $=n$ unfolding vec-first-def by auto
lemma dim-vec-last $[s i m p]$ : dim-vec (vec-last $v n)=n$ unfolding vec-last-def by auto
lemma vec-first-carrier[simp]: vec-first $v n \in$ carrier-vec $n$ by (rule carrier-vecI, auto)
lemma vec-last-carrier [simp]: vec-last $v n \in$ carrier-vec $n$ by (rule carrier-vecI, auto)
lemma vec-first-last-append[simp]:
assumes $v \in$ carrier-vec $(n+m)$ shows vec-first $v n @_{v}$ vec-last $v m=v$
apply (rule) unfolding vec-first-def vec-last-def using assms by auto
lemma append-vec-le: assumes $v \in$ carrier-vec $n$ and $w: w \in$ carrier-vec $n$
shows $v @_{v} v^{\prime} \leq w @_{v} w^{\prime} \longleftrightarrow v \leq w \wedge v^{\prime} \leq w^{\prime}$
proof -
\{ fix $i$ assume $*: \forall i .\left(\neg i<n \longrightarrow i<n+\operatorname{dim}-v e c w^{\prime} \longrightarrow v^{\prime} \$(i-n) \leq w^{\prime} \$(i-\right.$ n))
and $i: i<d i m-v e c w^{\prime}$
have $v^{\prime} \$ i \leq w^{\prime} \$ i$ using $*[$ rule-format, of $n+i] i$ by auto \}
thus ?thesis using assms unfolding less-eq-vec-def by auto qed
lemma all-vec-append: $(\forall x \in$ carrier-vec $(n+m) . P x) \longleftrightarrow(\forall x 1 \in$ carrier-vec n. $\forall x 2 \in$ carrier-vec m. P (x1 @ $x_{v}$ ) $)$
proof (standard, force, intro ballI, goal-cases)
case (1 $x$ )
have $x=\operatorname{vec} n(\lambda i . x \$ i) @_{v} \operatorname{vec} m(\lambda i . x \$(n+i))$
by (rule eq-vecI, insert 1 (2), auto)
hence $P x=P\left(\operatorname{vec} n(\lambda i . x \$ i) @_{v} \operatorname{vec} m(\lambda i . x \$(n+i))\right)$ by $\operatorname{simp}$
also have ... using 1 by auto
finally show ?case .
qed
definition four-block-mat :: 'a mat $\Rightarrow$ 'a mat $\Rightarrow{ }^{\prime}$ 'a mat $\Rightarrow{ }^{\prime} a$ mat $\Rightarrow{ }^{\prime} a$ mat where four-block-mat A B C D=
(let nra $=$ dim-row $A ; n r d=$ dim-row $D$; $n c a=d i m-c o l A ; n c d=d i m-\operatorname{col} D$
in
mat $(n r a+n r d)(n c a+n c d)(\lambda(i, j)$. if $i<n r a$ then
if $j<n$ ca then $A \$ \$(i, j)$ else $B \$ \$(i, j-n c a)$
else if $j<n c a$ then $C \$(i-n r a, j)$ else $D \$(i-n r a, j-n c a)))$
lemma index-mat-four-block[simp]:
$i<$ dim-row $A+$ dim-row $D \Longrightarrow j<$ dim-col $A+$ dim-col $D \Longrightarrow$ four-block-mat $A B C D \$ \$(i, j)$
$=($ if $i<$ dim-row $A$ then
if $j<\operatorname{dim}$-col $A$ then $A \$ \$(i, j)$ else $B \$ \$(i, j-\operatorname{dim}$-col $A)$
else if $j<$ dim-col $A$ then $C \$(i-$ dim-row $A, j)$ else $D \$ \$(i-d i m-r o w$ $A, j-\operatorname{dim}-\operatorname{col} A))$

$$
\text { dim-row }(\text { four-block-mat } A B C D)=\text { dim-row } A+\text { dim-row } D
$$ dim-col (four-block-mat A B C D) $=$ dim-col $A+\operatorname{dim}-c o l D$

unfolding four-block-mat-def Let-def by auto
lemma four-block-carrier-mat[simp]:
A $\in$ carrier-mat nr1 nc1 $\Longrightarrow D \in$ carrier-mat nr2 nc2 $\Longrightarrow$
four-block-mat A B C D carrier-mat $(n r 1+n r 2)(n c 1+n c 2)$
unfolding carrier-mat-def by auto
lemma cong-four-block-mat: $A 1=B 1 \Longrightarrow A 2=B 2 \Longrightarrow A 3=B 3 \Longrightarrow A 4=$ B4 $\Longrightarrow$
four-block-mat A1 A2 A3 A4 $=$ four-block-mat B1 B2 B3 B4 by auto
lemma four-block-one-mat[simp]:
four-block-mat $\left(1_{m} n 1\right)\left(0_{m} n 1 n 2\right)\left(0_{m} n 2 n 1\right)\left(1_{m} n 2\right)=1_{m}(n 1+n 2)$
by (rule eq-matI, auto)
lemma four-block-zero-mat[simp]:
four-block-mat ( $\left.0_{m} n r 1 n c 1\right)\left(0_{m} n r 1 n c 2\right)\left(0_{m} n r 2 n c 1\right)\left(0_{m} n r 2 n c 2\right)=0_{m}$ $(n r 1+n r 2)(n c 1+n c 2)$
by (rule eq-matI, auto)
lemma row-four-block-mat:
assumes $c: A \in$ carrier-mat nr1 nc1 $B \in$ carrier-mat nr1 nc2
$C \in$ carrier-mat nr2 nc1 $D \in$ carrier-mat nr2 nc2
shows
$i<n r 1 \Longrightarrow$ row (four-block-mat $A B C D) i=$ row $A i @_{v}$ row $B i(\mathbf{i s}-\Longrightarrow$ ? $A B$ )
$\neg i<n r 1 \Longrightarrow i<n r 1+n r 2 \Longrightarrow$ row (four-block-mat A B C D) $i=$ row $C(i$
$-n r 1) @_{v}$ row $D(i-n r 1)$
(is $-\Longrightarrow-\Longrightarrow$ ? $C D$ )
proof -
assume $i: i<n r 1$
show ?AB by (rule eq-vecI, insert ic, auto)
next
assume $i: \neg i<n r 1 i<n r 1+n r 2$
show ?CD by (rule eq-vecI, insert $i c$, auto)
qed
lemma col-four-block-mat:
assumes $c: A \in$ carrier-mat nr1 nc1 $B \in$ carrier-mat nr1 ncZ
$C \in$ carrier-mat nr2 nc1 $D \in$ carrier-mat nr2 nc2
shows
$j<n c 1 \Longrightarrow \operatorname{col}($ four-block-mat $A B C D) j=\operatorname{col} A j @_{v} \operatorname{col} C j($ is $-\Longrightarrow ? A C)$
$\neg j<n c 1 \Longrightarrow j<n c 1+n c 2 \Longrightarrow \operatorname{col}(f o u r-b l o c k-m a t ~ A B C D) j=\operatorname{col} B(j-$ $n c 1) @_{v} \operatorname{col} D(j-n c 1)$
(is $-\Longrightarrow-\Longrightarrow$ ? $B D$ )
proof -
assume $j: j<n c 1$

```
    show ?AC by (rule eq-vecI, insert j c, auto)
next
    assume j:\negj<nc1 j<nc1 + nc2
    show ?BD by (rule eq-vecI, insert j c, auto)
qed
lemma mult-four-block-mat: assumes
    c1: A1 \in carrier-mat nr1 n1 B1 \in carrier-mat nr1 n2 C1 \incarrier-mat nr2 n1
D1 \in carrier-mat nr2 n2 and
    c2: A2 \in carrier-mat n1 nc1 B2 \in carrier-mat n1 nc2 C2 \in carrier-mat n2 nc1
D2 \in carrier-mat n2 nc2
    shows four-block-mat A1 B1 C1 D1 * four-block-mat A2 B2 C2 D2
    = four-block-mat (A1*A2 + B1*C2) (A1*B2 + B1*D2)
    (C1*A2 + D1*C2) (C1*B2 + D1*D2) (is ?M1 * ?M2 = -)
proof -
    note row = row-four-block-mat[OF c1]
    note col = col-four-block-mat[OF c2]
    {
    fix ij
    assume i:i<nr1 and j: j<nc1
    have row ?M1 i c col ?M2 j = row A1 i c col A2 j + row B1 i c col C2 j
        unfolding row(1)[OF i] col(1)[OF j]
        by (rule scalar-prod-append[of-n1-n2], insert c1 c2 i j, auto)
    }
    moreover
    {
    fix ij
    assume i:\negi<nr1 i<nr1 + nr2 and j:j<nc1
    hence }\mp@subsup{i}{}{\prime}:i-nr1<nr2 by aut
    have row ?M1 i col ?M2 j = row C1 (i-nr1) • col A2 j + row D1 (i-
nr1) • col C2 j
            unfolding row(2)[OF i] col(1)[OF j]
            by (rule scalar-prod-append[of-n1-n2], insert c1 c2 i i' j, auto)
}
moreover
{
    fix ij
    assume i:i<nr1 and j:\negj<nc1j<nc1 + nc2
    hence j': j - nc1 < nc2 by auto
    have row ?M1 i col ?M2 j = row A1 i . col B2 (j - nc1) + row B1 i c col
D2 (j - nc1)
            unfolding row(1)[OF i] col(2)[OF j]
            by (rule scalar-prod-append[of-n1-n2], insert c1 c2 i j' j, auto)
}
moreover
{
    fix ij
    assume i:\negi<nr1 i<nr1 + nr2 and j:\negj<nc1j<nc1 + nc2
    hence i':i-nr1<nr2 and j':j-nc1<nc2 by auto
```

```
    have row ?M1 \(i \cdot\) col ?M2 \(j=\) row \(C 1(i-n r 1) \cdot \operatorname{col} B 2(j-n c 1)+\) row D1
\((i-n r 1) \cdot \operatorname{col}\) D2 \((j-n c 1)\)
    unfolding \(\operatorname{row}(2)[O F i] \operatorname{col}(2)[O F j]\)
    by (rule scalar-prod-append[of-n1-n2], insert c1 c2 \(i i^{\prime} j^{\prime} j\), auto)
    \}
    ultimately show ?thesis
    by (intro eq-matI, insert c1 c2, auto)
qed
definition append-rows :: 'a :: zero mat \(\Rightarrow{ }^{\prime} a\) mat \(\Rightarrow{ }^{\prime} a\) mat (infixr \(@_{r} 65\) )where
    \(A @_{r} B=\) four-block-mat \(A\left(0_{m}(\right.\) dim-row \(\left.A) 0\right) B\left(0_{m}(\right.\) dim-row \(\left.B) 0\right)\)
lemma carrier-append-rows[simp,intro]: \(A \in\) carrier-mat nr1 nc \(\Longrightarrow B \in\) car-
rier-mat nr2 \(n c \Longrightarrow\)
    \(A @_{r} B \in\) carrier-mat (nr1 + nr2) nc
    unfolding append-rows-def by auto
lemma col-mult2[simp]:
    assumes \(A\) : A: carrier-mat \(n r n\)
        and \(B: B\) : carrier-mat \(n n c\)
        and \(j: j<n c\)
    shows \(\operatorname{col}(A * B) j=A *_{v} \operatorname{col} B j\)
proof
    have \(A B: A * B\) : carrier-mat \(n r n c\) using \(A B\) by auto
    fix \(i\) assume \(i: i<\operatorname{dim}\)-vec \(\left(A *_{v} \operatorname{col} B j\right)\)
    show \(\operatorname{col}(A * B) j \$ i=\left(A *_{v} \operatorname{col} B j\right) \$ i\)
        using \(A B A B j i\) by simp
qed auto
lemma mat-vec-as-mat-mat-mult: assumes \(A: A \in\) carrier-mat nr nc and \(v: v \in\) carrier-vec \(n c\)
shows \(A *_{v} v=\operatorname{col}(A *\) mat-of-cols \(n c[v]) 0\)
by (subst col-mult2[OF A], insert v, auto)
lemma mat-mult-append: assumes \(A: A \in\) carrier-mat nr1 nc
and B: B carrier-mat nr2 nc
and \(v: v \in\) carrier-vec nc
shows \(\left(A @_{r} B\right) *_{v} v=\left(A *_{v} v\right) @_{v}\left(B *_{v} v\right)\)
proof -
let ?Fb1 = four-block-mat \(A\left(0_{m} n r 10\right) B\left(0_{m} n r 20\right)\)
let ? Fb2 \(=\) four-block-mat \((\) mat-of-cols nc \([v])\left(\begin{array}{lll}O_{m} n c & 0\end{array}\right)\left(\begin{array}{lll}0_{m} & 0 & 1\end{array}\right)\left(\begin{array}{lll}O_{m} & 0 & 0\end{array}\right)\)
have \(i d\) : ?Fb2 = mat-of-cols nc [v]
using \(v\) by auto
have \(\left(A @_{r} B\right) *_{v} v=\operatorname{col}(? F b 1 * ? F b 2) 0\) unfolding \(i d\)
by (subst mat-vec-as-mat-mat-mult \([O F-v]\), insert A B, auto simp: append-rows-def)
also have ?Fb1 * ? Fb2 = four-block-mat \(\left(A *\right.\) mat-of-cols nc \([v]+0_{m} n r 10 *\)
\(\left.0_{m} 01\right)\left(A * 0_{m} n c 0+0_{m} n r 10 * 0_{m} 00\right)\)
( \(B *\) mat-of-cols nc \(\left.[v]+0_{m} n r 20 * 0_{m} 01\right)\left(B * 0_{m} n c 0+0_{m} n r 20 *\right.\) \(\left.0_{m} 00\right)\)
```

by (rule mult-four-block-mat $[O F A-B]$, auto)
also have $\left(A *\right.$ mat-of-cols nc $\left.[v]+0_{m} n r 10 * O_{m} 01\right)=A *$ mat-of-cols nc [v]
using $A v$ by auto
also have $\left(B *\right.$ mat-of-cols nc $\left.[v]+0_{m} n r 20 * 0_{m} 01\right)=B *$ mat-of-cols nc [ $v$ ]
using $B v$ by auto
also have $\left(A * 0_{m} n c 0+0_{m} n r 10 * 0_{m} 00\right)=0_{m} n r 10$ using $A$ by auto also have $\left(B * O_{m} n c 0+O_{m} n r 20 * O_{m} 00\right)=0_{m} n r 20$ using $B$ by auto finally have $\left(A @_{r} B\right) *_{v} v=\operatorname{col}$ (four-block-mat $(A *$ mat-of-cols nc $[v])\left(0_{m}\right.$ $n r 10)(B *$ mat-of-cols nc [v] $\left.)\left(0_{m} n r 20\right)\right) 0$.
also have $\ldots=\operatorname{col}(A *$ mat-of-cols nc $[v]) 0 @_{v} \operatorname{col}(B *$ mat-of-cols nc $[v]) 0$
by (rule col-four-block-mat, insert A B v, auto)
also have col $(A *$ mat-of-cols nc $[v]) 0=A * v v$
by (rule mat-vec-as-mat-mat-mult $[$ symmetric, OF A v])
also have $\operatorname{col}(B *$ mat-of-cols nc $[v]) 0=B *_{v} v$
by (rule mat-vec-as-mat-mat-mult $[$ symmetric, OF Br $\quad$ ])
finally show ?thesis.
qed
lemma append-rows-le: assumes $A: A \in$ carrier-mat nr1 nc and $B: B \in$ carrier-mat nr2 nc
and $a: a \in$ carrier-vec nr1
and $v: v \in$ carrier-vec $n c$
shows $\left(A @_{r} B\right) *_{v} v \leq\left(a @_{v} b\right) \longleftrightarrow A *_{v} v \leq a \wedge B *_{v} v \leq b$
unfolding mat-mult-append $[O F A B v]$
by (rule append-vec-le[OF-a], insert A v, auto)
lemma elements-four-block-mat:
assumes $c: A \in$ carrier-mat nr1 nc1 $B \in$ carrier-mat nr1nc2
$C \in$ carrier-mat nr2 nc1 $D \in$ carrier-mat nr2 nc2

## shows

elements-mat (four-block-mat A B C D) $\subseteq$
elements-mat $A \cup$ elements-mat $B \cup$ elements-mat $C \cup$ elements-mat $D$
(is elements-mat ?four $\subseteq$-)
proof rule
fix $a$ assume $a \in$ elements-mat ?four
then obtain $i j$
where $i_{4}: i<$ dim-row?four and $j 4: j<$ dim-col?four and $a: a=$ ?four $\$ \$$ $(i, j)$
by auto
show $a \in$ elements-mat $A \cup$ elements-mat $B \cup$ elements-mat $C \cup$ elements-mat D
proof (cases $i<n r 1$ )
case True note $i 1=$ this
show ?thesis
proof (cases $j<n c 1$ )
case True

```
        then have }a=A$$(i,j) using c i1 a by sim
        thus ?thesis using c i1 True by auto next
        case False
        then have }a=B$$(i,j-nc1) using c i1 a j4 by sim
        moreover have j - nc1<nc2 using c j4 False by auto
        ultimately show ?thesis using c i1 by auto
    qed next
    case False note i1 = this
    have i2: i - nr1<nr2 using c i1 i4 by auto
    show ?thesis
    proof (cases j<nc1)
        case True
        then have }a=C$$(i-nr1,j) using c i2 a i1 by sim
        thus ?thesis using c i2 True by auto next
        case False
        then have a=D $$ (i-nr1,j-nc1) using c i2 a i1 j4 by simp
        moreover have j-nc1<nc2 using c j4 False by auto
        ultimately show ?thesis using c i2 by auto
    qed
    qed
qed
lemma assoc-four-block-mat: fixes FB :: 'a mat = ' a mat 知 'a :: zero mat
    defines FB:FB\equiv\lambdaBb Cc.four-block-mat Bb (0m (dim-row Bb) (dim-col Cc))
(0m (dim-row Cc) (dim-col Bb)) Cc
    shows FB A (FB B C) = FB (FBAB)C (is ?L = ?R)
proof -
    let ?ar = dim-row A let ?ac = dim-col A
    let ?br = dim-row }B\mathrm{ let ?bc = dim-col B
    let ?cr = dim-row C let ?cc= dim-col C
    let ?r = ?ar + ?br + ?cr let ?c = ?ac + ?bc + ?cc
    let ? BC=FB B C let ?AB=FBAB
    have dL: dim-row ?L = ?r dim-col ?L = ?c unfolding FB by auto
    have dR: dim-row ?R = ?ar + ?br + ?cr dim-col ?R = ?ac + ?bc + ?cc
unfolding FB by auto
    have dBC: dim-row ?BC=?br + ?cr dim-col ? BC = ?bc+?cc unfolding FB
by auto
    have dAB: dim-row ?AB = ?ar + ?br dim-col ?AB = ?ac + ?bc unfolding FB
by auto
    show ?thesis
    proof (intro eq-matI[of ?R ?L, unfolded dL dR,OF - refl refl])
        fix ij
        assume i: i< ?r and j:j<?c
    show ?L $$ (i,j) = ?R $$ (i,j)
    proof (cases i< ?ar)
        case True note i= this
        thus ?thesis using j
            by (cases j< ?ac, auto simp: FB)
    next
```

```
        case False note ii= this
        show ?thesis
        proof (cases j< ?ac)
            case True
            with i ii show ?thesis unfolding FB by auto
        next
            case False note jj = this
            from j jj i ii have L: ? L $$ (i,j) = ?BC $$ (i- ?ar, j - ?ac) unfolding
FB by auto
            have R:?R $$ (i,j) =? BC $$ (i - ?ar, j - ?ac) using ii jj ij
                by (cases i<?ar + ?br; cases j<?ac + ?bc, auto simp: FB)
            show ?thesis unfolding L R ..
        qed
    qed
    qed
qed
definition split-block :: 'a mat => nat => nat => ('a mat }\times\mp@subsup{}{}{\prime}amat \times 'a mat > 'a
mat)
    where split-block A sr sc = (let
        nr = dim-row A; nc = dim-col A;
        nr2 = nr - sr; nc\mathcal{L = nc - sc;};
        A1 = mat sr sc ( }\lambdaij. A$$ ij)
        A2 = mat sr nc\mathcal{2}}(\lambda(i,j).A $$ (i,j+sc))
        A3 = mat nr2 sc (\lambda (i,j). A $$ (i+sr,j));
        A4 = mat nr2 nc2 ( }\lambda(i,j). A $$ (i+sr,j+sc)
    in (A1,A2,A3,A4))
lemma split-block: assumes res: split-block A sr1 sc1 = (A1,A2,A3,A4)
```



```
    shows A1 \in carrier-mat sr1 sc1 A2 \in carrier-mat sr1 sc2
        A3 \in carrier-mat sr2 sc1 A4 \in carrier-mat sr2 sc2
        A = four-block-mat A1 A2 A3 A4
    using res unfolding split-block-def Let-def
    by (auto simp: dims)
Using four-block-mat we define block-diagonal matrices.
fun diag-block-mat :: ' \(a\) :: zero mat list \(\Rightarrow\) ' \(a\) mat where
    diag-block-mat [] = Om 0 0
| diag-block-mat (A # As) = (let
            B= diag-block-mat As
            in four-block-mat A (Om (dim-row A) (dim-col B)) (0m (dim-row B) (dim-col
A)) B)
lemma dim-diag-block-mat:
dim-row (diag-block-mat \(A s)=\) sum-list (map dim-row As) (is ?row)
dim-col \((\) diag-block-mat As \()=\) sum-list \((\) map dim-col As) (is ?col)
proof -
have ? row \(\wedge\) ?col
```

```
    by (induct As, auto simp: Let-def)
    thus ?row and ?col by auto
qed
lemma diag-block-mat-singleton[simp]:diag-block-mat [A] = A
    by auto
lemma diag-block-mat-append: diag-block-mat (As@ Bs)=
    (let A = diag-block-mat As; B = diag-block-mat Bs
    in four-block-mat A ( }0m(\mathrm{ dim-row A) (dim-col B)) (0m (dim-row B) (dim-col A))
B)
    unfolding Let-def
proof (induct As)
    case (Cons A As)
    show ?case
        unfolding append.simps
        unfolding diag-block-mat.simps Let-def
    unfolding Cons
    by (rule assoc-four-block-mat)
qed auto
lemma diag-block-mat-last: diag-block-mat (As @ [B])=
    (let A = diag-block-mat As
    in four-block-mat A ( Om (dim-row A) (dim-col B)) (Om (dim-row B) (dim-col A))
B)
    unfolding diag-block-mat-append diag-block-mat-singleton by auto
lemma diag-block-mat-square:
    Ball (set As) square-mat \Longrightarrow square-mat (diag-block-mat As)
by (induct As, auto simp:Let-def)
lemma diag-block-one-mat[simp]:
    diag-block-mat (map (\lambdaA. 1m (dim-row A)) As) = (1m (sum-list (map dim-row
As)))
    by (induct As, auto simp: Let-def)
lemma elements-diag-block-mat:
    elements-mat (diag-block-mat As)\subseteq{0}\cup\bigcup(set (map elements-mat As))
proof (induct As)
    case Nil then show ?case using dim-diag-block-mat[of Nil] by auto next
    case (Cons A As)
        let ?D = diag-block-mat As
        let ?B = Om (dim-row A) (dim-col ?D)
    let ?C = Om (dim-row ?D) (dim-col A)
    have A:A\incarrier-mat (dim-row A) (dim-col A) by auto
    have B: ?B \in carrier-mat (dim-row A) (dim-col ?D) by auto
    have C:?C \in carrier-mat (dim-row ?D) (dim-col A) by auto
    have D:?D \in carrier-mat (dim-row ?D) (dim-col ?D) by auto
```

```
    have
        elements-mat (diag-block-mat (A#As))\subseteq
        elements-mat A \cup elements-mat ? B \cup elements-mat ?C \cup elements-mat ?D
        unfolding diag-block-mat.simps Let-def
        using elements-four-block-mat[OF A B C D] elements-0-mat
        by auto
    also have ...\subseteq{0}\cup elements-mat A\cup elements-mat ? D
        using elements-0-mat by auto
    finally show ?case using Cons by auto
qed
lemma diag-block-pow-mat: assumes sq: Ball (set As) square-mat
    shows diag-block-mat As \widehat{m}n= diag-block-mat (map (\lambdaA.A \widehat{m}n) As) (is
?As 㐌 - = -)
proof (induct n)
    case 0
    have ?As \widehat{m}}0=1\mp@subsup{|}{m}{(dim-row ?As) by simp
    also have dim-row ?As = sum-list (map dim-row As)
    using diag-block-mat-square[OF sq] unfolding dim-diag-block-mat by auto
    also have 1m .. = diag-block-mat (map ( }\lambdaA.\mp@subsup{1}{m}{\prime}(\mathrm{ dim-row A)) As) by simp
    also have ... = diag-block-mat (map ( }\lambda,A.A\mp@subsup{\widehat{m}}{m}{0})\mathrm{ As) by simp
    finally show ?case .
next
    case (Suc n)
    let ?An = \lambda As. diag-block-mat (map ( }\lambdaA.A\mp@subsup{\widehat{~}}{m}{}n)As
    let ?Asn = \lambda As. diag-block-mat (map ( }\lambdaA.A\mp@subsup{\widehat{m}}{m}{}n*A)As
    from Suc have ?case = (?An As * diag-block-mat As = ?Asn As) by simp
    also have ... using sq
    proof (induct As)
        case (Cons A As)
    hence IH: ?An As * diag-block-mat As = ?Asn As
        and sq: Ball (set As) square-mat and A: dim-col A = dim-row A by auto
    have sq2: Ball (set (List.map ( }\lambdaA.A\mp@subsup{\widehat{m}}{m}{}n) As)) square-ma
        and sq3: Ball (set (List.map ( }\lambdaA.A\widehat{m}n*A)As)) square-ma
        using sq by auto
    define n1 where n1 = dim-row }
    define n2 where n2 = sum-list (map dim-row As)
    from A have A:A\incarrier-mat n1 n1 unfolding n1-def carrier-mat-def by
simp
    have [simp]: dim-col (?An As) = n2 dim-row (?An As) = n2
        unfolding n2-def
        using diag-block-mat-square[OF sq2,unfolded square-mat.simps]
        unfolding dim-diag-block-mat map-map by (auto simp:o-def)
    have [simp]: dim-col (?Asn As) = n2 dim-row (?Asn As) = n2
            unfolding n2-def
            using diag-block-mat-square[OF sq3,unfolded square-mat.simps]
            unfolding dim-diag-block-mat map-map by (auto simp:o-def)
    have [simp]:
                dim-row (diag-block-mat As) = n2
```

```
dim-col (diag-block-mat As) = n2
```

unfolding n2-def
using diag-block-mat-square[OF sq,unfolded square-mat.simps]
unfolding dim-diag-block-mat by auto
have [simp]: diag-block-mat As $\in$ carrier-mat n2 n2 unfolding carrier-mat-def by $\operatorname{simp}$
have [simp]: ?An As $\in$ carrier-mat n2 n2 unfolding carrier-mat-def by simp show ?case unfolding diag-block-mat.simps Let-def list.simps
by (subst mult-four-block-mat[of-n1 n1-n2-n2--n1-n2],
insert $A$, auto simp: $I H$ )
qed auto
finally show? case by simp
qed
lemma diag-block-upper-triangular: assumes
$\bigwedge A i j . A \in$ set $A s \Longrightarrow j<i \Longrightarrow i<\operatorname{dim}$-row $A \Longrightarrow A \$ \$(i, j)=0$
and Ball (set As) square-mat
and $j<i i<$ dim-row (diag-block-mat As)
shows diag-block-mat As $\$ \$(i, j)=0$
using assms
proof (induct As arbitrary: $i j$ )
case (Cons A As ij)
let ? $n 1=$ dim-row $A$
let ? $n 2=$ sum-list (map dim-row As)
from Cons have $[\operatorname{simp}]$ : dim-col $A=? n 1$ by simp
from Cons have Ball (set As) square-mat by auto
note $[$ simp $]=$ diag-block-mat-square $[$ OF this,unfolded square-mat.simps $]$
note $[$ simp $]=$ dim-diag-block-mat(1)
from Cons(5) have $i: i<? n 1+$ ?n2 by $\operatorname{simp}$
show ?case
proof (cases $i<? n 1$ )
case True
with $\operatorname{Cons}(4)$ have $j: j<? n 1$ by auto
with True Cons(2)[of A, OF - Cons(4)] show ?thesis
by (simp add: Let-def)
next
case False note $i A s=$ this
show ?thesis
proof (cases $j<?$ ? 1 )
case True
with $i$ iAs show ?thesis by (simp add: Let-def)
next
case False note $j A s=$ this
from Cons(4) $i$ have $j: j<? n 1+$ ?n2 by auto
show ?thesis using iAs jAs ij
by (simp add: Let-def, subst Cons(1), insert Cons(2-4), auto)
qed
qed
qed $\operatorname{simp}$
lemma smult-four-block-mat: assumes $c: A \in$ carrier-mat nr1 nc1 $B \in$ car-rier-mat nr1 nc2
$C \in$ carrier-mat nr2 nc1 $D \in$ carrier-mat nr2 nc2
shows $a \cdot m$ four-block-mat A B C D = four-block-mat $\left(a \cdot{ }_{m} A\right)\left(a \cdot{ }_{m} B\right)\left(a \cdot{ }_{m}\right.$ C) $\left(a \cdot{ }_{m} D\right)$
by (rule eq-matI, insert $c$, auto)
lemma map-four-block-mat: assumes $c: A \in$ carrier-mat nr1 nc1 $B \in$ carrier-mat $n r 1 n c 2$
$C \in$ carrier-mat nr2 nc1 $D \in$ carrier-mat nr2 nc2
shows map-mat $f$ (four-block-mat $A B C D)=$ four-block-mat (map-mat $f A$ ) (map-mat $f B)($ map-mat $f C)($ map-mat $f D)$
by (rule eq-matI, insert $c$, auto)
lemma add-four-block-mat: assumes
c1: A1 $\in$ carrier-mat nr1 nc1 B1 $\in$ carrier-mat nr1 nc2 C1 $\in$ carrier-mat nr2 nc1 D1 $\in$ carrier-mat nr2 nc2 and
c2: A2 $\in$ carrier-mat nr1 nc1 B2 $\in$ carrier-mat nr1 nc2 C2 $\in$ carrier-mat nr2 nc1 D2 $\in$ carrier-mat nr2 nc2
shows four-block-mat A1 B1 C1 D1 + four-block-mat A2 B2 C2 D2
= four-block-mat $(A 1+A 2)(B 1+B 2)(C 1+C 2)(D 1+D 2)$
by (rule eq-matI, insert assms, auto)
lemma diag-four-block-mat: assumes $c: A \in$ carrier-mat n1 n1
$D \in$ carrier-mat n2 n2
shows diag-mat (four-block-mat $A B C D)=\operatorname{diag}-m a t A @ \operatorname{diag}-m a t ~ D$ by (rule nth-equalityI, insert $c$, auto simp: diag-mat-def nth-append)
definition $m k$-diagonal $::$ ' $a::$ zero list $\Rightarrow$ 'a mat where $m k$-diagonal as $=$ diag-block-mat $(\operatorname{map}(\lambda a . \operatorname{mat}($ Suc 0) $($ Suc 0) $(\lambda$-. a) $)$
as)
lemma mk-diagonal-dim: dim-row $(m k$-diagonal as $)=$ length as dim-col (mk-diagonal as $)=$ length as unfolding $m k$-diagonal-def by (induct as, auto simp: Let-def)
lemma $m k$-diagonal-diagonal: diagonal-mat ( $m k$-diagonal as) unfolding mk-diagonal-def
proof (induct as)
case Nil show? ?case unfolding mk-diagonal-def diagonal-mat-def by simp next
case (Cons a as)
let $? n=$ length $(a \# a s)$
let ? $A=\operatorname{mat}($ Suc 0) $($ Suc 0) $(\lambda-. a)$
let ?f $=\operatorname{map}(\lambda a . \operatorname{mat}(S u c 0)(S u c 0)(\lambda-. a))$
let ?AS = diag-block-mat (?f as)
let ?AAS = diag-block-mat $($ ?f $(a \# a s))$

```
    show ?case
    unfolding diagonal-mat-def
    proof(intro allI impI)
    fix ij assume ir: i< dim-row ?AAS and jc: j<dim-col ?AAS and ij:i\not=
j
    hence ir2: i<1 + dim-row ?AS and jc2: j<1 + dim-col ?AS
        unfolding dim-row-mat list.map diag-block-mat.simps Let-def
        by auto
    show ?AAS $$ (i,j)=0
    proof (cases i=0)
        case True
            then show ?thesis using jc ij by (auto simp: Let-def) next
        case False note i0 = this
            show ?thesis
            proof (cases j=0)
                case True
                    then show ?thesis using ir ij by (auto simp: Let-def) next
                    case False
                    have ir3: i-1<dim-row?AS and jc3: j-1<dim-col ?AS
                    using ir2 jc2 i0 False by auto
                    have IH: \bigwedgeij.i<dim-row ?AS \Longrightarrowj<dim-col ?AS \Longrightarrow \ \ = j\Longrightarrow
                        ?AS $$ (i,j) = 0
                            using Cons unfolding diagonal-mat-def by auto
                    have ?AS $$ (i-1,j-1)=0
                    using IH[OF ir3 jc3] i0 False ij by auto
                    thus ?thesis using ir jc ij by (simp add: Let-def)
                qed
        qed
    qed
qed
definition orthogonal-mat :: 'a::semiring-0 mat => bool
    where orthogonal-mat A}
    let B = transpose-mat A*A in
    diagonal-mat }B\wedge(\foralli<dim-col A.B$$(i,i)\not=0
lemma orthogonal-matD[elim]:
    orthogonal-mat A\Longrightarrow
    i<dim-col A\Longrightarrowj< dim-col A\Longrightarrow(\operatorname{col A i col A j=0) = (i\not=j)})
unfolding orthogonal-mat-def diagonal-mat-def by auto
lemma orthogonal-matI[intro]:
    (\ij. i<dim-col A\Longrightarrowj<dim-col A\Longrightarrow(col A i . col A j=0) = (i\not=j))
    orthogonal-mat A
    unfolding orthogonal-mat-def diagonal-mat-def by auto
definition orthogonal :: 'a::semiring-0 vec list }=>\mathrm{ bool
    where orthogonal vs \equiv
```

```
    \(\forall i j . i<\) length vs \(\longrightarrow j<\) length vs \(\longrightarrow\)
        \((v s!i \cdot v s!j=0)=(i \neq j)\)
lemma orthogonalD[elim]:
    orthogonal vs \(\Longrightarrow i<\) length \(v s \Longrightarrow j<\) length \(v s \Longrightarrow\)
    (nth vs \(i \cdot\) nth vs \(j=0)=(i \neq j)\)
    unfolding orthogonal-def by auto
lemma orthogonalI[intro]:
    \((\bigwedge i j . i<\) length \(v s \Longrightarrow j<\) length \(v s \Longrightarrow(n t h\) vs \(i \cdot n t h\) vs \(j=0)=(i \neq j))\)
\(\Longrightarrow\)
    orthogonal vs
    unfolding orthogonal-def by auto
lemma transpose-four-block-mat: assumes \(*: A \in\) carrier-mat nr1 nc1 \(B \in\) car-
rier-mat nr1 nc2
    \(C \in\) carrier-mat nr2 nc1 \(D \in\) carrier-mat nr2 nc2
    shows transpose-mat (four-block-mat \(A B C D)=\)
    four-block-mat (transpose-mat \(A\) ) (transpose-mat \(C\) ) (transpose-mat \(B\) ) (transpose-mat
D)
    by (rule eq-matI, insert *, auto)
lemma zero-transpose-mat [simp]: transpose-mat \(\left(0_{m} n m\right)=\left(\begin{array}{lll}0_{m} & m & n\end{array}\right)\)
    by (rule eq-matI, auto)
lemma upper-triangular-four-block: assumes \(A D: A \in\) carrier-mat \(n\) n \(D \in\) car-
rier-mat \(m m\)
    and ut: upper-triangular \(A\) upper-triangular \(D\)
    shows upper-triangular (four-block-mat \(\left.A B\left(\begin{array}{lll}m & m & n\end{array}\right) D\right)\)
proof -
    let ? \(C=\) four-block-mat \(A B\left(0_{m} m n\right) D\)
    from \(A D\) have dim: dim-row ? \(C=n+m\) dim-col ? \(C=n+m\) dim-row \(A=\)
\(n\) by auto
    show ?thesis
    proof (rule upper-triangularI, unfold dim)
        fix \(i j\)
        assume \(*: j<i i<n+m\)
        show ? \(C \$ \$(i, j)=0\)
        proof (cases \(i<n\) )
            case True
            with upper-triangular \(D[O F\) ut (1) \(*(1)] * A D\) show ?thesis by auto
        next
            case False note \(i=\) this
                show ?thesis by (cases \(j<n\), insert upper-triangular \(D[O F\) ut(2)] \(* i A D\),
auto)
        qed
    qed
qed
```

```
lemma pow-four-block-mat: assumes A: A \in carrier-mat n n
    and B:B\incarrier-mat m m
    shows (four-block-mat A (0 ( n n m) (0m m n) B) \widehat{m}
    four-block-mat ( }A\mp@subsup{\widehat{m}}{m}{*
proof (induct k)
    case (Suc k)
    let ?FB = \lambda A B. four-block-mat A (0m n m) (0mmn)B
    let ?A = ?FB A B
    let ?B = ?FB (A 㐌k) (B\mp@subsup{\widehat{m}}{m}{}k)
```



```
mm}\mathrm{ by auto
    have ?A \widehat{m}}\mathrm{ Suc k=?A ^}\mp@subsup{m}{k}{**?A by simp
    also have ?A \widehat{ }
```



```
    by (subst mult-four-block-mat[OF Ak--Bk A - B], insert A B, auto)
    finally show ?case.
qed (insert A B,auto)
lemma uminus-scalar-prod:
    assumes [simp]:v:carrier-vec n w:carrier-vec n
    shows - ((v::'a::field vec) • w) = (-v) • w
    unfolding scalar-prod-def uminus-vec-def
    apply (subst sum-negf[symmetric])
proof (rule sum.cong[OF refl])
    fix i assume i:i:{0 ..<dim-vec w}
    have [simp]: dim-vec v=n dim-vec w=n by auto
    show - (v$i*w$i)=vec (dim-vec v) (\lambdai. - v$i)$i*w$i
        unfolding minus-mult-left using i by auto
qed
lemma append-vec-eq:
    assumes [simp]:v:carrier-vec n v':carrier-vec n
    shows [simp]:v @ vw= v' @ v w'}\longleftrightarrow\longleftrightarrowv=\mp@subsup{v}{}{\prime}\wedgew=\mp@subsup{w}{}{\prime}(\mathbf{is}?L\longleftrightarrow?R
proof
    have [simp]: dim-vec v=n dim-vec v'}=n\mathrm{ by auto
    { assume L: ?L
        have }v\mp@subsup{v}{}{\prime}:v=\mp@subsup{v}{}{\prime
        proof
            fix i assume i:i<dim-vec v'
            have (v @ vw)$ i=( v}\mp@subsup{v}{}{\prime}\mp@subsup{@}{v}{}\mp@subsup{w}{}{\prime})$i\mathrm{ using L by auto
            thus v$i= v}
        qed auto
        moreover have w= w'
        proof
            show dim-vec w= dim-vec w' using vv'L
            by (metis add-diff-cancel-left' index-append-vec(2))
```

```
        moreover fix }i\mathrm{ assume i:i<dim-vec w'
        have (v @ 
        ultimately show w$i=w'$i using i by simp
        qed
        ultimately show ?R by simp
    }
qed auto
lemma append-vec-add:
    assumes [simp]:v:carrier-vec n v':carrier-vec n
        and [simp]:w:carrier-vec m w' : carrier-vec m
    shows (v @ 
proof
    have [simp]: dim-vec v=n dim-vec v}\mp@subsup{v}{}{\prime}=n\mathrm{ by auto
    have [simp]: dim-vec w=m dim-vec w' = m by auto
    fix i assume i: i<dim-vec ?R
    thus ?L $ i=?R $ i by (cases i<n,auto)
qed auto
lemma four-block-mat-mult-vec:
    assumes A: A : carrier-mat nr1 nc1
        and B: B: carrier-mat nr1 nc2
        and C:C : carrier-mat nr2 nc1
        and D:D : carrier-mat nr2 nc2
        and a: a:carrier-vec nc1
        and d:d : carrier-vec nc2
    shows four-block-mat A B C D *v (a @ d d ) = (A *va+B 汭 d) @ v (C *va
+ D *v d)
        (is ?ABCD *v - = ?r)
proof
    have ABCD: ?ABCD : carrier-mat (nr1+nr2) (nc1+nc2) using four-block-carrier-mat[OF
A D].
    fix i assume i: i< dim-vec ?r
    show (?ABCD *v (a@@ d)) $i=?r $ i (is ?li= -)
    proof (cases i<nr1)
        case True
        have ?li = (row A i @ row B i) • ( a @ v d)
            using A row-four-block-mat[OF A B C D] True by simp
        also have ... = row A i . a + row B i . d
            apply (rule scalar-prod-append) using A B D a d True by auto
        finally show ?thesis using A B True by auto
        next case False
            let ?i=i - nr1
        have ?li = (row C ? i @ row D ? i) • ( a @ v d )
            using i row-four-block-mat[OF A B C D] False A B C D by simp
        also have ... = row C?i . a + row D ? i . d
            apply (rule scalar-prod-append) using A B C D a d False by auto
        finally show ?thesis using A B C D False i by auto
    qed
```

```
qed (insert A B,auto)
```


## lemma mult-mat-vec-split:

assumes $A$ : A : carrier-mat $n n$
and $D: D:$ carrier-mat $m m$
and $a: a$ : carrier-vec $n$
and $d: d:$ carrier-vec $m$
shows four-block-mat $A\left(0_{m} n m\right)\left(0_{m} m n\right) D *_{v}\left(a @_{v} d\right)=A *_{v} a @_{v} D *_{v}$ $d$
by (subst four-block-mat-mult-vec[OF A-D a d], insert A D a d, auto)
lemma similar-mat-witI: assumes $P * Q=1_{m} n Q * P=1_{m} n A=P * B *$ $Q$

A $\in$ carrier-mat $n n B \in$ carrier-mat $n n P \in$ carrier-mat $n n Q \in$ carrier-mat $n n$
shows similar-mat-wit $A B P Q$ using assms unfolding similar-mat-wit-def Let-def by auto
lemma similar-mat-witD: assumes $n=$ dim-row A similar-mat-wit $A B P Q$
shows $P * Q=1_{m} n Q * P=1_{m} n A=P * B * Q$
A carrier-mat $n$ n $B \in$ carrier-mat $n n P \in$ carrier-mat $n n Q \in$ carrier-mat n $n$
using assms(2) unfolding similar-mat-wit-def Let-def assms(1)[symmetric] by auto
lemma similar-mat-witD2: assumes $A \in$ carrier-mat $n$ m similar-mat-wit $A B P$ Q
shows $P * Q=1_{m} n Q * P=1_{m} n A=P * B * Q$
A carrier-mat n n $B \in$ carrier-mat $n n P \in$ carrier-mat $n n Q \in$ carrier-mat $n$ n
using similar-mat-witD[OF - assms(2), of n] assms(1)[unfolded carrier-mat-def] by auto
lemma similar-mat-wit-sym: assumes sim: similar-mat-wit ABPQ shows similar-mat-wit $B A Q P$
proof -
from similar-mat-witD[OF refl sim] obtain $n$ where
$A B:\{A, B, P, Q\} \subseteq$ carrier-mat $n n P * Q=1_{m} n Q * P=1_{m} n$ and $A$ : $A=P * B * Q$ by blast
hence $*:\{B, A, Q, P\} \subseteq$ carrier-mat $n n Q * P=1_{m} n P * Q=1_{m} n$ by auto
let $? c=\lambda A . A \in$ carrier-mat $n n$
from * have Carr: ?c $B$ ?c $P$ ?c $Q$ by auto
note $[$ simp $]=$ assoc-mult-mat $[o f-n n-n-n]$
show ?thesis
proof (rule similar-mat-witI[of - $n$ ])
have $Q * A * P=(Q * P) * B *(Q * P)$
using Carr unfolding $A$ by simp
also have $\ldots=B$ using Carr unfolding $A B$ by simp

$$
\text { finally show } B=Q * A * P \text { by simp }
$$ qed (insert $* A B$, auto)

qed
lemma similar-mat-wit-refl: assumes $A: A \in$ carrier-mat $n n$
shows similar-mat-wit $A A\left(1_{m} n\right)\left(1_{m} n\right)$
by (rule similar-mat-witI[OF - - A], insert A, auto)
lemma similar-mat-wit-trans: assumes $A B$ : similar-mat-wit $A B P Q$
and $B C$ : similar-mat-wit $B C P^{\prime} Q^{\prime}$
shows similar-mat-wit $A C\left(P * P^{\prime}\right)\left(Q^{\prime} * Q\right)$
proof -
from similar-mat-witD $[$ OF refl $A B]$ obtain $n$ where
$A B:\{A, B, P, Q\} \subseteq$ carrier-mat $n n P * Q=1_{m} n Q * P=1_{m} n A=P *$
$B * Q$ by blast
hence $B: B \in$ carrier-mat $n n$ by auto
from similar-mat-witD2 $[O F B B C]$ have
$B C:\left\{C, P^{\prime}, Q^{\prime}\right\} \subseteq$ carrier-mat $n n P^{\prime} * Q^{\prime}=1_{m} n Q^{\prime} * P^{\prime}=1_{m} n B=P^{\prime}$

* $C * Q^{\prime}$ by auto
let $? c=\lambda A . A \in$ carrier-mat $n n$
let ? $P=P * P^{\prime}$
let ? $Q=Q^{\prime} * Q$
from $A B B C$ have carr: ?c $A$ ?c $B$ ?c $C$ ?c $P$ ?c $P^{\prime}$ ?c $Q$ ?c $Q^{\prime}$
and Carr: $\{A, C, ? P, ? Q\} \subseteq$ carrier-mat $n n$ by auto
note $[$ simp $]=$ assoc-mult-mat $[o f-n n-n-n]$
have $i d: A=? P * C * ? Q$ unfolding $A B(4)[$ unfolded $B C(4)]$ using carr
by $\operatorname{simp}$
have ? $P * ? Q=P *\left(P^{\prime} * Q^{\prime}\right) * Q$ using carr by simp
also have $\ldots=1_{m} n$ unfolding $B C$ using carr $A B$ by simp
finally have $P Q: ? P * ? Q=1_{m} n$.
have ? $Q * ? P=Q^{\prime} *(Q * P) * P^{\prime}$ using carr by simp
also have $\ldots=1_{m} n$ unfolding $A B$ using carr $B C$ by simp
finally have $Q P: ? Q * ? P=1_{m} n$.
show ?thesis
by (rule similar-mat-witI[OF PQ QP id], insert Carr, auto)
qed
lemma similar-mat-refl: $A \in$ carrier-mat $n n \Longrightarrow$ similar-mat $A A$ using similar-mat-wit-refl unfolding similar-mat-def by blast
lemma similar-mat-trans: similar-mat $A B \Longrightarrow$ similar-mat $B C$ similar-mat $A C$
using similar-mat-wit-trans unfolding similar-mat-def by blast
lemma similar-mat-sym: similar-mat $A \quad B \Longrightarrow$ similar-mat $B A$ using similar-mat-wit-sym unfolding similar-mat-def by blast
lemma similar-mat-wit-four-block: assumes
1: similar-mat-wit A1 B1 P1 Q1
and 2: similar-mat-wit A2 B2 P2 Q2
and URA: URA $=(P 1 * U R * Q 2)$
and $L L A: L L A=(P 2 * L L * Q 1)$
and A1: A1 $\in$ carrier-mat $n n$
and A2: A2 $\in$ carrier-mat $m m$
and $L L: L L \in$ carrier-mat $m n$
and $U R: U R \in$ carrier-mat $n m$
shows similar-mat-wit (four-block-mat A1 URA LLA A2) (four-block-mat B1 UR LL B2)
(four-block-mat P1 ( $\left.0_{m} n m\right)\left(0_{m} m n\right)$ P2) (four-block-mat Q1 ( $\left.0_{m} n m\right)\left(0_{m}\right.$ $m n) Q 2)$
(is similar-mat-wit ?A ?B ?P ?Q)
proof -
let ? $n=n+m$
let ? $O 1=1 m$ let ? O2 $=1 m m$ let ? $O=1_{m}$ ? $n$
from similar-mat-witD2[OF A1 1] have 11: P1 * Q1 = ?O1 Q1 * P1 = ?O1
and P1: P1 $\in$ carrier-mat $n n$ and Q1: Q1 $\in$ carrier-mat $n n$
and B1: B1 $\in$ carrier-mat $n n$ and $1: A 1=P 1 * B 1 * Q 1$ by auto
from similar-mat-witD2[OF A2 2] have 21: P2 * Q2 = ? O2 $Q 2 * P 2=? O 2$
and P2: P2 $\in$ carrier-mat $m m$ and Q2: Q2 $\in$ carrier-mat $m m$
and B2: B2 $\in$ carrier-mat $m m$ and 2: A2 $=P 2 * B 2 * Q 2$ by auto
have $P Q 1$ : ? $P * ? Q=$ ? $O$
by (subst mult-four-block-mat[OF P1--P2 Q1-- Q2], unfold 11 21, insert P1 P2 Q1 Q2, auto intro!: eq-matI)
have $Q P 1$ : ? $Q * ? P=? O$
by (subst mult-four-block-mat[OF Q1--Q2 P1--P2], unfold 11 21, insert
P1 P2 Q1 Q2, auto intro!: eq-matI)
let $? P B=? P * ? B$
have $P:$ ? P $\in$ carrier-mat ?n ?n using P1 P2 by auto
have $Q:$ ? $Q \in$ carrier-mat ?n ?n using $Q 1$ Q2 by auto
have $B: ? B \in$ carrier-mat ?n ?n using $B 1 U R L L B 2$ by auto
have $P B: ? P B \in$ carrier-mat ?n ?n using $P B$ by auto
have $P B 1: P 1 * B 1 \in$ carrier-mat $n n$ using $P 1 B 1$ by auto
have PB2: P2 * B2 $\in$ carrier-mat $m m$ using P2 B2 by auto
have P1UR: P1 * UR $\in$ carrier-mat $n m$ using P1 UR by auto
have P2LL: P2 * LL $\in$ carrier-mat $m n$ using P2 LL by auto
have $i d:$ ? PB $=$ four-block-mat $(P 1 * B 1)(P 1 * U R)(P 2 * L L)(P 2 * B 2)$
by (subst mult-four-block-mat[OF P1--P2 B1 UR LL B2], insert P1 P2 B1 B2 LL UR, auto)
have $i d: ? P B * ? Q=$ four-block-mat $(P 1 * B 1 * Q 1)(P 1 * U R * Q 2)$
$(P 2 * L L * Q 1)(P 2 * B 2 * Q 2)$ unfolding $i d$
by (subst mult-four-block-mat[OF PB1 P1UR P2LL PB2 Q1 - - Q2],
insert P1 P2 B1 B2 Q1 Q2 UR LL, auto)
have $i d: ? A=? P * ? B * ? Q$ unfolding $i d 12 U R A L L A .$.
show ?thesis
by (rule similar-mat-witI[OF PQ1 QP1 id], insert A1 A2 B1 B2 Q1 Q2 P1 P2, auto)


## qed

lemma similar-mat-four-block-0-ex: assumes
1: similar-mat A1 B1
and 2: similar-mat A2 B2
and A0: A0 $\in$ carrier-mat $n \mathrm{~m}$
and A1: A1 $\in$ carrier-mat $n n$
and A2: A2 $\in$ carrier-mat $m m$
shows $\exists B 0 . B 0 \in$ carrier-mat $n m \wedge$ similar-mat (four-block-mat A1 A0 $\left(0_{m}\right.$ $m n)$ A2)
(four-block-mat B1 B0 ( $0_{m} m n$ ) B2)
proof -
from 1[unfolded similar-mat-def] obtain P1 Q1 where 1: similar-mat-wit A1 B1 P1 Q1 by auto
note $w 1$ = similar-mat-witD2[OF A1 1]
from 2[unfolded similar-mat-def] obtain P2 Q2 where 2: similar-mat-wit A2
B2 P2 Q2 by auto
note $w 2$ = similar-mat-witD2[OF A2 2]
from $w 1$ w2 have $C: B 1 \in$ carrier-mat $n n$ B2 $\in$ carrier-mat $m m$ by auto
from w1 w2 have $i d: 0_{m} m n=Q 2 * O_{m} m n * P 1$ by simp
let ?wit $=Q 1 * A 0 * P 2$
from $w 1$ w2 A0 have wit: ? wit $\in$ carrier-mat $n m$ by auto
from similar-mat-wit-sym[OF similar-mat-wit-four-block[OF similar-mat-wit-sym[OF
1] similar-mat-wit-sym[OF 2]
refl id C zero-carrier-mat A0]]
have similar-mat (four-block-mat A1 A0 ( $0_{m} m$ n) A2) (four-block-mat B1 (Q1

* $A 0$ * P2 $\left.)\left(0_{m} m n\right) B 2\right)$
unfolding similar-mat-def by auto
thus ?thesis using wit by auto
qed
lemma similar-mat-four-block-0-0: assumes
1: similar-mat A1 B1
and 2: similar-mat A2 B2
and A1: A1 $\in$ carrier-mat $n n$
and A2: A2 $\in$ carrier-mat $m$
shows similar-mat (four-block-mat A1 ( $\left.0_{m} n m\right)\left(0_{m} m n\right)$ A2)
(four-block-mat B1 ( $0_{m} n m$ ) ( $0_{m} m n$ ) B2)
proof -
from 1[unfolded similar-mat-def] obtain P1 Q1 where 1: similar-mat-wit A1 B1 P1 Q1 by auto
note $w 1=$ similar-mat-witD2[OF A1 1]
from 2[unfolded similar-mat-def] obtain P2 Q2 where 2: similar-mat-wit A2
B2 P2 Q2 by auto
note $w 2$ = similar-mat-witD2[OF A2 2]
from $w 1$ w2 have $C: B 1 \in$ carrier-mat $n n B 2 \in$ carrier-mat $m m$ by auto
from $w 1$ w2 have $i d: 0_{m} m n=Q 2 * 0_{m} m n * P 1$ by simp
from $w 1$ w2 have $i d 2: 0_{m} n m=Q 1 * 0_{m} n m * P 2$ by $\operatorname{simp}$
from similar-mat-wit-sym[OF similar-mat-wit-four-block[OF similar-mat-wit-sym[OF 1] similar-mat-wit-sym[OF 2]
id2 id $C$ zero-carrier-mat zero-carrier-mat]]
show ?thesis unfolding similar-mat-def by blast
qed
lemma similar-diag-mat-block-mat: assumes $\bigwedge A B .(A, B) \in$ set $M s \Longrightarrow$ simi-lar-mat A B
shows similar-mat (diag-block-mat (map fst Ms)) (diag-block-mat (map snd Ms))
using assms
proof (induct Ms)
case Nil
show ?case by (auto intro!: similar-mat-refl[of - 0])
next
case (Cons AB Ms)
obtain $A B$ where $A B: A B=(A, B)$ by force
from $\operatorname{Cons}(2)[$ of $A B]$ have $\operatorname{sim} A B$ : similar-mat $A B$ unfolding $A B$ by auto
from similar-matD[OF this] obtain $n$ where $A: A \in$ carrier-mat $n n$ and $B: B$ $\in$ carrier-mat $n n$ by auto
hence $[$ simp $]$ : dim-row $A=n$ dim-col $A=n$ dim-row $B=n \operatorname{dim}$-col $B=n$ by auto
let $? C=$ diag-block-mat (map fst $M s$ ) let ? $D=$ diag-block-mat (map snd Ms)
from Cons (1) [OF Cons(2)] have simRec: similar-mat ?C ?D by auto
from similar-matD[OF this] obtain $m$ where $C: ? C \in$ carrier-mat $m m$ and $D: ? D \in$ carrier-mat $m m$ by auto
hence [simp]: dim-row? $C=m$ dim-col ? $C=m$ dim-row ? $D=m$ dim-col $? D=$ $m$ by auto
have similar-mat (diag-block-mat (map fst (AB \# Ms))) (diag-block-mat (map snd $(A B \# M s)))$
$=$ similar-mat (four-block-mat $A\left(O_{m} \quad n \quad m\right)\left(O_{m} m n\right)$ ? $\left.C\right)$ (four-block-mat $B$ $\left(0_{m} n m\right)\left(0_{m} m n\right)$ ? $\left.D\right)$
unfolding $A B$ by (simp add: Let-def)
also have ...
by (rule similar-mat-four-block- $0-0[$ OF $\operatorname{simAB} \operatorname{simRec} A C]$ )
finally show ?case .
qed
lemma similar-mat-wit-pow: assumes wit: similar-mat-wit A B P $Q$
shows similar-mat-wit $\left(A \widehat{m}_{m} k\right)\left(B \widehat{m}_{m} k\right) P Q$
proof -
define $n$ where $n=$ dim-row $A$
let $? C=$ carrier-mat $n n$
from similar-mat-witD $[$ OF refl wit, folded $n$-def $]$ have
$A: A \in ? C$ and $B: B \in ? C$ and $P: P \in ? C$ and $Q: Q \in ? C$
and $P Q: P * Q=1_{m} n$ and $Q P: Q * P=1_{m} n$
and $A B: A=P * B * Q$
by auto
from $A B$ have $*:\left(A \widehat{m}_{m} k\right) \in$ carrier-mat $n n B \widehat{m}_{m} k \in$ carrier-mat $n n$ by auto
note carr $=A B P Q$
have $i d: A \widehat{~}_{m} k=P * B \widehat{m}_{m} k * Q$ unfolding $A B$
proof (induct $k$ )
case 0
thus ?case using carr by (simp add: $P Q$ )
next
case (Suc k)
define $B k$ where $B k=B^{{ }_{m}}{ }_{k}$
have $B k: B k \in$ carrier-mat $n n$ unfolding $B k$-def using carr by simp
have $(P * B * Q){ }_{m}$ Suc $k=(P * B k * Q) *(P * B * Q)$ by (simp add:
Suc Bk-def)
also have $\ldots=P *(B k *(Q * P) * B) * Q$
using carr $B k$ by (simp add: assoc-mult-mat $[o f-n n-n-n])$
also have $B k *(Q * P)=B k$ unfolding $Q P$ using $B k$ by simp
finally show ?case unfolding $B k$-def by simp
qed
show ?thesis
by (rule similar-mat-witI[OF PQ QP id * P $Q]$ )
qed
lemma similar-mat-wit-pow-id: similar-mat-wit $A B P Q \Longrightarrow A{ }_{m} k=P * B$ $\widehat{m}_{m} k Q$
using similar-mat-wit-pow[of A B P $Q k$ ] unfolding similar-mat-wit-def Let-def by blast


### 4.5 Homomorphism properties

```
context semiring-hom
begin
abbreviation mat-hom :: 'a mat => 'b mat (math)
    where math \equiv map-mat hom
abbreviation vec-hom :: 'a vec }=>\mathrm{ 'b vec (vech)
    where vec}\mp@subsup{h}{h}{}\equiv\mathrm{ map-vec hom
lemma vec-hom-zero: vec}\mp@subsup{h}{}{\prime}(\mp@subsup{0}{v}{}n)=\mp@subsup{0}{v}{}
    by (rule eq-vecI, auto)
lemma mat-hom-one: math}(\mp@subsup{1}{m}{}n)=1m 
    by (rule eq-matI, auto)
```

lemma mat-hom-mult: assumes $A: A \in$ carrier-mat $n r n$ and $B: B \in$ carrier-mat $n n c$
shows $\operatorname{mat}_{h}(A * B)=\operatorname{mat}_{h} A * \operatorname{mat}_{h} B$
proof -
let $? L=$ mat $_{h}(A * B)$
let ? $R=$ mat $_{h} A *$ mat $_{h} B$
let $? A=$ mat $_{h} A$
let $? B=\operatorname{mat}_{h} B$

```
    from A B have id:
    dim-row ?L = nr dim-row ?R = nr
    dim-col ?L = nc dim-col ?R = nc by auto
    show ?thesis
    proof (rule eq-matI, unfold id)
    fix ij
    assume *: i<nr j<nc
    define I where I={0 ..<n}
    have id:{0 ..< dim-vec (col ?B j)} = I {0 ..<dim-vec (col B j)} = I
        unfolding I-def using * B by auto
    have finite: finite I unfolding I-def by auto
    have I:I\subseteq{0..< n} unfolding I-def by auto
    have ?L $$ (i,j) = hom (row A i col B j) using A B* by auto
    also have ... = row ?A i c col ?B j unfolding scalar-prod-def id using finite
I
    proof (induct I)
        case (insert kI)
        show ?case unfolding sum.insert[OF insert(1-2)] hom-add hom-mult
        using insert(3-) * A B by auto
    qed simp
    also have \ldots.. =?R $$ (i,j) using A B * by auto
    finally
    show ?L $$ (i,j)=?R $$ (i,j).
    qed auto
qed
lemma mult-mat-vec-hom: assumes \(A: A \in\) carrier-mat \(n r n\) and \(v: v \in\) car-
rier-vec n
    shows vech}(A\mp@subsup{*}{v}{}v)=\mp@subsup{math}{h}{}A\mp@subsup{*}{v}{}ve\mp@subsup{c}{h}{}
proof -
    let ?L = vech
    let ?R = math A *v vech}
    let ?A = math }
    let ?v = vech v
    from Av have id:
            dim-vec ?L = nr dim-vec ?R = nr
            by auto
    show ?thesis
    proof (rule eq-vecI, unfold id)
            fix }
            assume *: i<nr
            define }I\mathrm{ where I={0 ..< n}
            have id:{0 ..< dim-vec v}=I{0 ..< dim-vec (vech v)}=I
            unfolding I-def using *v by auto
    have finite: finite I unfolding I-def by auto
    have I:I\subseteq{0..<n} unfolding I-def by auto
    have ?L $ i= hom (row A i v v) using A v* by auto
    also have ... = row ?A i . ?v unfolding scalar-prod-def id using finite I
    proof (induct I)
```

```
            case (insert kI)
            show ?case unfolding sum.insert[OF insert(1-2)] hom-add hom-mult
            using insert(3-)*A v by auto
        qed simp
        also have ... = ?R $ i using Av* by auto
        finally
        show ?L $ i=?R $ i.
        qed auto
qed
end
lemma vec-eq-iff: (x=y)=(dim-vec x = dim-vec y ^(\foralli<dim-vec y. x $i=
y$i))(is ?l = ?r)
proof
    assume ?r
    show ?l
        by (rule eq-vecI, insert <?r\rangle, auto)
qed simp
lemma mat-eq-iff: (x=y)=(dim-row x = dim-row y ^dim-col x = dim-col y ^
    (\forall ij. i<dim-row y \longrightarrowj<dim-col y \longrightarrowx $$ (i,j)=y$$(i,j)))(is ?l = ?r)
proof
    assume ?r
    show ?l
        by (rule eq-matI, insert <?r>, auto)
qed simp
lemma (in inj-semiring-hom) vec-hom-zero-iff[simp]:(vech}x=\mp@subsup{0}{v}{}n)=(x=\mp@subsup{0}{v}{
n)
proof -
    {
        fix }
        assume i: i<n dim-vec x=n
        hence vech}\mp@subsup{\mp@code{C}}{}{x}$i=0\longleftrightarrowx$i=
            using index-map-vec(1)[of i x] by simp
    } note main = this
    show ?thesis unfolding vec-eq-iff by (simp, insert main, auto)
qed
lemma (in inj-semiring-hom) mat-hom-inj: math }A=\mp@subsup{math}{h}{}B\LongrightarrowA=
    unfolding mat-eq-iff by auto
lemma (in inj-semiring-hom) vec-hom-inj: vech}v=vech w\Longrightarrowv=
    unfolding vec-eq-iff by auto
lemma (in semiring-hom) mat-hom-pow: assumes A: A \in carrier-mat n n
    shows math (A ^}\mp@subsup{}{m}{}k)=(\mp@subsup{math}{h}{}A)\mp@subsup{\widehat{m}}{m}{}
proof (induct k)
    case (Suc k)
```

thus ?case using mat-hom-mult $[$ OF pow-carrier-mat $[$ OF $A$, of $k] A]$ by simp qed (simp add: mat-hom-one)
lemma (in semiring-hom) hom-sum-mat: hom (sum-mat $A)=\operatorname{sum-mat}\left(\right.$ mat $\left._{h} A\right)$ proof -
obtain $B$ where $i d:$ ?thesis $=\left(\operatorname{hom}(\operatorname{sum}((\$ \$) A) B)=\operatorname{sum}\left((\$ \$)\left(\right.\right.\right.$ mat $\left.\left._{h} A\right)\right)$ B)
and $B: B \subseteq\{0 . .<$ dim-row $A\} \times\{0 . .<\operatorname{dim}-\operatorname{col} A\}$
unfolding sum-mat-def by auto
from $B$ have finite $B$
using finite-subset by blast
thus ?thesis unfolding id using $B$
proof (induct $B$ )
case (insert $x F$ )
show ?case unfolding sum.insert[OF insert(1-2)] hom-add
using insert(3-) by auto
qed $\operatorname{simp}$
qed
lemma (in semiring-hom) vec-hom-smult: vec $h_{h}\left(e v \cdot{ }_{v} v\right)=h o m e v \cdot{ }_{v} v e c_{h} v$ by (rule eq-vecI, auto simp: hom-distribs)
lemma minus-scalar-prod-distrib: fixes $v_{1}::$ ' $a$ :: ring vec assumes $v: v_{1} \in$ carrier-vec $n v_{2} \in$ carrier-vec $n v_{3} \in$ carrier-vec $n$ shows $\left(v_{1}-v_{2}\right) \cdot v_{3}=v_{1} \cdot v_{3}-v_{2} \cdot v_{3}$
unfolding minus-add-uminus-vec[OF v(1-2)]
by (subst add-scalar-prod-distrib[OF v(1)], insert v, auto)
lemma scalar-prod-minus-distrib: fixes $v_{1}::{ }^{\prime} a$ :: ring vec assumes $v: v_{1} \in$ carrier-vec $n v_{2} \in$ carrier-vec $n v_{3} \in$ carrier-vec $n$ shows $v_{1} \cdot\left(v_{2}-v_{3}\right)=v_{1} \cdot v_{2}-v_{1} \cdot v_{3}$
unfolding minus-add-uminus-vec[ $[O F \quad v(2-3)]$
by (subst scalar-prod-add-distrib[OF v(1)], insert v, auto)
lemma uminus-add-minus-vec:
assumes $l \in$ carrier-vec $n r \in$ carrier-vec $n$
shows $-((l:: ' a::$ ab-group-add vec $)+r)=(-l-r)$
using assms by auto
lemma minus-add-minus-vec: fixes $u$ :: ' $a$ :: ab-group-add vec assumes $u \in$ carrier-vec $n v \in$ carrier-vec $n w \in$ carrier-vec $n$
shows $u-(v+w)=u-v-w$
using assms by auto
lemma uminus-add-minus-mat:
assumes $l \in$ carrier-mat nr nc $r \in$ carrier-mat nr nc
shows $-((l:: ' a$ :: ab-group-add mat $)+r)=(-l-r)$
using assms by auto
lemma minus-add-minus-mat: fixes $u$ :: ' $a$ :: ab-group-add mat assumes $u \in$ carrier-mat $n r n c v \in$ carrier-mat $n r n c w \in$ carrier-mat $n r n c$ shows $u-(v+w)=u-v-w$ using assms by auto
lemma uminus-uminus-vec $[\operatorname{simp}]:-(-(v:: ' a::$ group-add vec $))=v$ by auto
lemma uminus-eq-vec[simp]: - (v::'a:: group-add vec) $=-w \longleftrightarrow v=w$ by (metis uminus-uminus-vec)
lemma uminus-uminus-mat $[$ simp $]:-(-(A:: ' a::$ group-add mat $))=A$ by auto
lemma uminus-eq-mat[simp]: $-\left(A::^{\prime} a::\right.$ group-add mat $)=-B \longleftrightarrow A=B$ by (metis uminus-uminus-mat)
lemma smult-zero-mat[simp]: ( $k::{ }^{\prime} a$ :: mult-zero $) \cdot{ }_{m} 0_{m} n r n c=0_{m} n r n c$ by (intro eq-matI, auto)
lemma similar-mat-wit-smult: fixes $A$ :: ' $a$ :: comm-ring-1 mat
assumes similar-mat-wit $A B P Q$
shows similar-mat-wit $(k \cdot m A)\left(k \cdot_{m} B\right) P Q$
proof -
define $n$ where $n=$ dim-row $A$
note main $=$ similar-mat-witD $[$ OF $n$-def assms $]$
show ?thesis
by (rule similar-mat-witI[OF main(1-2) -- main( $6-7$ )], insert main(3-), auto simp: mult-smult-distrib mult-smult-assoc-mat $[$ of $-n n-n]$ )
qed
lemma similar-mat-smult: fixes $A$ :: ' $a$ :: comm-ring-1 mat
assumes similar-mat $A B$
shows similar-mat $(k \cdot m A)(k \cdot m B)$
using similar-mat-wit-smult assms unfolding similar-mat-def by blast
definition mat-diag :: nat $\Rightarrow\left(n a t \Rightarrow{ }^{\prime} a::\right.$ zero $) \Rightarrow{ }^{\prime} a$ mat where mat-diag $n f=$ Matrix.mat $n n(\lambda(i, j)$. if $i=j$ then $f j$ else 0$)$
lemma mat-diag-dim[simp]: mat-diag $n f \in$ carrier-mat $n n$
unfolding mat-diag-def by auto
lemma mat-diag-mult-left: assumes $A: A \in$ carrier-mat $n n r$
shows mat-diag $n f * A=$ Matrix.mat $n \operatorname{nr}(\lambda(i, j)$. fi*A $\$ \$(i, j))$
proof (rule eq-matI, insert $A$, auto simp: mat-diag-def scalar-prod-def, goal-cases) case ( $1 i j$ )
thus ?case by (subst sum.remove[of $-i]$, auto)
qed

```
lemma mat-diag-mult-right: assumes A:A\incarrier-mat nr n
    shows A* mat-diag n f = Matrix.mat nr n (\lambda (i,j). A $$ (i,j)*fj)
proof (rule eq-matI, insert A, auto simp: mat-diag-def scalar-prod-def, goal-cases)
    case (1 ij)
    thus ?case by (subst sum.remove[of - j], auto)
qed
lemma mat-diag-diag[simp]:mat-diag nf* mat-diag ng=mat-diag n (\lambdai.fi*
g i)
    by (subst mat-diag-mult-left[of-n n], auto simp: mat-diag-def)
lemma mat-diag-one[simp]:mat-diag n (\lambdax.1) = 1m n unfolding mat-diag-def
by auto
Interpret vector as row-matrix
definition mat-of-row y = mat 1 (dim-vec y) (\lambda ij. y $(snd ij))
lemma mat-of-row-carrier[simp,intro]:
    y}\in\mathrm{ carrier-vec }n\Longrightarrow\mathrm{ mat-of-row y carrier-mat 1 n
    y\incarrier-vec n \Longrightarrowmat-of-row y carrier-mat (Suc 0) n
    unfolding mat-of-row-def by auto
lemma mat-of-row-dim[simp]: dim-row (mat-of-row y) = 1
    dim-col (mat-of-row y) = dim-vec y
    unfolding mat-of-row-def by auto
lemma mat-of-row-index[simp]: x < dim-vec y \Longrightarrow mat-of-row y $$ (0,x)=y$x
    unfolding mat-of-row-def by auto
lemma row-mat-of-row[simp]: row (mat-of-row y) 0 = y
    by auto
lemma mat-of-row-mult-append-rows: assumes y1: y1 \in carrier-vec nr1
    and y2: y2 \in carrier-vec nr2
    and A1:A1 \in carrier-mat nr1 nc
    and A2: A2 \in carrier-mat nr2 nc
shows mat-of-row (y1 @ y2)*(A1 @ }\mp@subsup{}{v}{}\mathrm{ A2) =
    mat-of-row y1 * A1 + mat-of-row y2 * A2
proof -
    from A1 A2 have dim: dim-row A1 = nr1 dim-row A2 = nr2 by auto
    let ?M1 = mat-of-row y1
    have M1: ?M1 \in carrier-mat 1 nr1 using y1 by auto
    let ?M2 = mat-of-row y2
    have M2: ?M2 \in carrier-mat 1 nr2 using y2 by auto
    let ?M3 = 0 m 0 nr1
    let ?M4 = Om 0 nr2
    note z = zero-carrier-mat
```

```
    have id:mat-of-row (y1 @ y2) = four-block-mat
    ?M1 ?M2 ?M3 ?M4 using y1 y2
    by (intro eq-matI, auto simp: mat-of-rows-def)
    show ?thesis
    unfolding id append-rows-def dim
    by (subst mult-four-block-mat[OF M1 M2 z z A1 z A2 z], insert A1 A2, auto)
qed
lemma mat-of-row-uminus: mat-of-row (-v)=- mat-of-row v
    by auto
```

    Allowing to construct and deconstruct vectors like lists
    abbreviation $v N i l$ where $v N i l \equiv$ vec 0 ((!) [])
definition $v$ Cons where $v$ Cons $a v \equiv \operatorname{vec}(S u c(\operatorname{dim}-v e c ~ v))(\lambda i$. case $i$ of $0 \Rightarrow a$
$\mid$ Suc $i \Rightarrow v \$ i)$
lemma vec-index-vCons-0 [simp]: vCons a v \$ $0=a$
by (simp add: vCons-def)
lemma vec-index-vCons-Suc [simp]:
fixes $v::$ 'a vec
shows $v$ Cons a $v \$$ Suc $n=v \$ n$
proof-
have 1: vec $\left(\right.$ Suc d) $f \$$ Suc $n=\operatorname{vec} d(f \circ S u c) \$ n$ for $d$ and $f:: n a t \Rightarrow{ }^{\prime} a$
by (transfer, auto simp: mk-vec-def)
show ?thesis
apply (auto simp: 1 vCons-def o-def) apply transfer apply (auto simp:
$m k-v e c-d e f)$
done
qed
lemma vec-index-vCons: vCons a $v \$ n=($ if $n=0$ then a else $v \$(n-1))$
by (cases $n$, auto)
lemma dim-vec-vCons [simp]: dim-vec (vCons a v) $=$ Suc (dim-vec $v$ )
by (simp add: vCons-def)
lemma vCons-carrier-vec[simp]: vCons a $v \in$ carrier-vec (Suc n) $\longleftrightarrow v \in$ car-
rier-vec $n$
by (auto dest!: carrier-vecD intro: carrier-vecI)
lemma vec-Suc: vec (Suc n) $f=v$ Cons $(f 0)($ vec $n(f \circ S u c))($ is ?l $=? r)$
proof (unfold vec-eq-iff, intro conjI allI impI)
fix $i$ assume $i<$ dim-vec ?r
then show ?l $\$ i=? r \$ i$ by (cases $i$, auto)
qed $\operatorname{simp}$
declare Abs-vec-cases[cases del]

```
lemma vec-cases [case-names vNil vCons, cases type: vec]:
```



```
    shows thesis
proof (cases dim-vec v)
    case 0 then show thesis by (intro assms(1), auto)
next
    case (Suc n)
    show thesis
    proof (rule assms(2))
        show v:v=vCons (v$0) (vec n (\lambdai.v$ Suc i)) (is v=?r)
        proof (rule eq-vecI, unfold dim-vec-vCons dim-vec Suc)
            fix i
            assume i<Suc n
            then show v$i=?r $ i by (cases i, auto simp: vCons-def)
        qed simp
    qed
qed
lemma vec-induct [case-names vNil vCons, induct type: vec]:
    assumes PvNil and \av. Pv\LongrightarrowP(vCons a v)
    shows P v
proof (induct dim-vec v arbitrary:v)
    case 0 then show ?case by (cases v, auto intro: assms(1))
next
    case (Suc n) then show ?case by (cases v, auto intro: assms(2))
qed
lemma carrier-vec-induct [consumes 1, case-names 0 Suc, induct set:carrier-vec]:
    assumes v: v\incarrier-vec n
        and 1:P0vNil and 2: \nav.v\in carrier-vec n\LongrightarrowPnv\LongrightarrowP(Suc n)
(vCons a v)
    shows Pnv
proof (insert v, induct n arbitrary: v)
    case 0 then have v=\operatorname{vec}0((!) []) by auto
    with 1 show ?case by auto
next
    case (Suc n) then show ?case by (cases v, auto dest!: carrier-vecD intro:2)
qed
lemma vec-of-list-Cons[simp]: vec-of-list (a#as) = vCons a (vec-of-list as)
    by (unfold vCons-def, transfer, auto simp:mk-vec-def split:nat.split)
lemma vec-of-list-Nil[simp]: vec-of-list [] = vNil
    by (transfer', auto)
lemma scalar-prod-vCons[simp]:
    vCons a v v vCons b w=a*b+v \cdotw
    apply (unfold scalar-prod-def atLeast0-lessThan-Suc-eq-insert-0 dim-vec-vCons)
    apply (subst sum.insert) apply (simp,simp)
```

```
    apply (subst sum.reindex) apply force
    apply simp
    done
lemma zero-vec-Suc: 0v (Suc n)=vCons 0 (0v n)
    by (auto simp:zero-vec-def vec-Suc o-def)
lemma zero-vec-zero[simp]: 0 v 0 = vNil by auto
lemma vCons-eq-vCons[simp]: vCons a v=vCons b w\longleftrightarrowa=b^v=w (is ?l
\longleftrightarrow?r)
proof
    assume ?l
    note arg-cong[OF this]
    from this[of dim-vec] this[of \lambdax. x$0] this[of \lambdax. x$Suc -]
    show ?r by (auto simp: vec-eq-iff)
qed simp
lemma vec-carrier-vec[simp]: vec n f\incarrier-vec m}\longleftrightarrown=
    unfolding carrier-vec-def by auto
notation transpose-mat ((-T) [1000])
lemma map-mat-transpose: (map-mat f A)
lemma cols-transpose[simp]: cols }\mp@subsup{A}{}{T}=\mathrm{ rows A unfolding cols-def rows-def by
auto
lemma rows-transpose[simp]: rows }\mp@subsup{A}{}{T}=\mathrm{ cols A unfolding cols-def rows-def by
auto
lemma list-of-vec-vec [simp]:list-of-vec (vec nf)=\operatorname{map}f[0..<n]
    by (transfer, auto simp: mk-vec-def)
lemma list-of-vec-0 [simp]: list-of-vec ( }0vn)=\mathrm{ replicate n 0
    by (simp add:zero-vec-def map-replicate-trivial)
lemma diag-mat-map:
    assumes M-carrier: M \in carrier-mat n n
    shows diag-mat (map-mat f M) = map f (diag-mat M)
proof -
    have dim-eq: dim-row M = dim-col M using M-carrier by auto
    have m: map-mat f M $$ (i,i)=f(M$$(i,i)) if i:i<dim-row M for i
        using dim-eq i by auto
    show ?thesis
        by (rule nth-equalityI, insert m, auto simp add: diag-mat-def M-carrier)
qed
lemma mat-of-rows-map [simp]:
    assumes x: set vs \subseteqcarrier-vec n
    shows mat-of-rows n (map (map-vec f)vs) = map-mat f (mat-of-rows n vs)
```

```
proof-
    have }\forallx\in\mathrm{ set vs. dim-vec }x=n\mathrm{ using }x\mathrm{ by auto
    then show ?thesis by (auto simp add: mat-eq-iff map-vec-def mat-of-rows-def)
qed
lemma mat-of-cols-map [simp]:
    assumes x: set vs \subseteqcarrier-vec n
    shows mat-of-cols n (map (map-vec f)vs)= map-mat f (mat-of-cols n vs)
proof-
    have }\forallx\inset vs. dim-vec x=n using x by aut
    then show ?thesis by (auto simp add: mat-eq-iff map-vec-def mat-of-cols-def)
qed
lemma vec-of-list-map [simp]: vec-of-list (map fxs) = map-vec f (vec-of-list xs)
    unfolding map-vec-def by (transfer, auto simp add:mk-vec-def)
lemma map-vec: map-vec f (vec n g) = vec n (fog) by auto
lemma mat-of-cols-Cons-index-0:i<n\Longrightarrow mat-of-cols n (w# ws) $$(i,0)=
w$ i
    by (unfold mat-of-cols-def, transfer', auto simp: mk-mat-def)
lemma nth-map-out-of-bound: i\geq length xs \Longrightarrowmap fxs!i=[]!(i - length xs)
    by (induct xs arbitrary:i, auto)
lemma mat-of-cols-Cons-index-Suc:
    i<n\Longrightarrow mat-of-cols n (w # ws) $$ (i, Suc j) = mat-of-cols n ws $$ (i,j)
    by (unfold mat-of-cols-def, transfer, auto simp: mk-mat-def undef-mat-def nth-append
nth-map-out-of-bound)
lemma mat-of-cols-index: i<n\Longrightarrowj< length ws \Longrightarrow mat-of-cols n ws $$ (i,j)
= ws ! j$ i
    by (unfold mat-of-cols-def, auto)
lemma mat-of-rows-index: i< length rs \Longrightarrow < < n\Longrightarrow mat-of-rows n rs $$ (i,j)
=rs!i$j
    by (unfold mat-of-rows-def, auto)
lemma transpose-mat-of-rows: (mat-of-rows n vs)}\mp@subsup{)}{}{T}=mat-of-cols n v
    by (auto intro!: eq-matI simp: mat-of-rows-index mat-of-cols-index)
lemma transpose-mat-of-cols: (mat-of-cols n vs)}\mp@subsup{)}{}{T}=mat-of-rows n v
    by (auto intro!: eq-matI simp: mat-of-rows-index mat-of-cols-index)
lemma nth-list-of-vec [simp]:
    assumes }i<dim-vec v shows list-of-vec v!i=v$
    using assms by (transfer, auto)
lemma length-list-of-vec [simp]:
```

```
    length (list-of-vec \(v\) ) \(=\) dim-vec \(v\) by (transfer, auto)
lemma vec-eq-0-iff:
    \(v=0_{v} n \longleftrightarrow n=\operatorname{dim}-v e c \quad v \wedge(n=0 \vee\) set (list-of-vec \(\left.v)=\{0\}\right)(\) is \(? l \longleftrightarrow\)
? \(r\) )
proof
    show ?l \(\Longrightarrow\) ?r by auto
    show ?r \(\Longrightarrow\) ?l by (intro iffI eq-vecI, force simp: set-conv-nth, force)
qed
lemma list-of-vec-vCons[simp]: list-of-vec (vCons a v)=a\# list-of-vec v (is ?l =
? \(r\) )
proof (intro nth-equalityI)
    fix \(i\)
    assume \(i<\) length ?l
    then show ?l ! \(i=? r!i\) by (cases \(i\), auto)
qed simp
lemma append-vec-vCons[simp]:vCons a \(v @_{v} w=v C o n s ~ a\left(v @_{v} w\right)(i s ? l=\)
? \(r\) )
proof (unfold vec-eq-iff, intro conjI allI impI)
    fix \(i\) assume \(i<\) dim-vec ?r
    then show ?l \(\$ i=? r \$ i\) by (cases \(i\); subst index-append-vec, auto)
qed \(\operatorname{simp}\)
lemma append-vec-vNil[simp]: vNil @ \({ }_{v} v=v\)
    by (unfold vec-eq-iff, auto)
lemma list-of-vec-append \([\) simp \(]\) : list-of-vec \(\left(v @{ }_{v} w\right)=\) list-of-vec \(v @ l i s t-o f-v e c ~ w ~\)
    by (induct \(v\), auto)
lemma transpose-mat-eq[simp]: \(A^{T}=B^{T} \longleftrightarrow A=B\)
    using transpose-transpose by metis
lemma mat-col-eqI: assumes cols: \(\bigwedge i . i<d i m-\operatorname{col} B \Longrightarrow \operatorname{col} A i=\operatorname{col} B i\)
    and dims: dim-row \(A=\) dim-row \(B\) dim-col \(A=\operatorname{dim}\)-col \(B\)
shows \(A=B\)
    by(subst transpose-mat-eq[symmetric], rule eq-rowI,insert assms,auto)
lemma upper-triangular-imp-distinct:
    assumes \(A: A \in\) carrier-mat \(n n\)
        and tri: upper-triangular \(A\)
        and diag: \(0 \notin \operatorname{set}(\operatorname{diag}-m a t A)\)
    shows distinct (rows A)
proof-
    \(\{\) fix \(i\) and \(j\)
        assume eq: rows \(A!i=\) rows \(A!j\) and \(i j: i<j\) and \(j n: j<n\)
        from tri \(A\) ij jn have rows \(A!j \$ i=0\) by (auto dest!:upper-triangularD)
        with eq have rows \(A!i \$ i=0\) by auto
```

with $\operatorname{diag}$ ij $j n A$ have False by (auto simp: diag-mat-def) \}
with $A$ show ?thesis by (force simp: distinct-conv-nth nat-neq-iff)
qed
lemma dim-vec-of-list $[s i m p]$ :dim-vec (vec-of-list as) $=$ length as by transfer auto
lemma list-vec: list-of-vec (vec-of-list xs) $=x s$
by (transfer, metis (mono-tags, lifting) atLeastLessThan-iff map-eq-conv map-nth $m k$-vec-def old.prod.case set-upt)
lemma vec-list: vec-of-list (list-of-vec v) $=v$ apply transfer unfolding $m k$-vec-def by auto
lemma index-vec-of-list: $i<$ length $x s \Longrightarrow($ vec-of-list xs) $\$ i=x s!i$
by (metis vec.abs-eq index-vec vec-of-list.abs-eq)
lemma vec-of-list-index: vec-of-list xs $\$ j=x s!j$ apply transfer unfolding $m k$-vec-def unfolding undef-vec-def by (simp, metis append-Nil2 nth-append)
lemma list-of-vec-index: list-of-vec $v!j=v \$ j$
by (metis vec-list vec-of-list-index)
lemma list-of-vec-map: list-of-vec $x s=\operatorname{map}((\$) x s)[0 . .<$ dim-vec $x s]$ by transfer auto
definition component-mult $v w=\operatorname{vec}(\min (\operatorname{dim}-v e c v)($ dim-vec $w))(\lambda i . v \$ i *$ $w \$ i)$
definition vec-set::'a vec $\Rightarrow$ 'a set (setv)
where vec-set $v=$ vec-index $v '\{. .<\operatorname{dim}$-vec $v\}$
lemma vec-set-map[simp]: set $($ map-vec $f v)=f$ 'set $v$ unfolding vec-set-def by auto
lemma index-component-mult:
assumes $i<\operatorname{dim}$-vec $v i<\operatorname{dim}-v e c ~ w$
shows component-mult $v w \$ i=v \$ i * w \$ i$
unfolding component-mult-def using assms index-vec by auto
lemma dim-component-mult:
dim-vec $($ component-mult $v w)=\min ($ dim-vec $v)($ dim-vec $w)$
unfolding component-mult-def using index-vec by auto
lemma vec-setE:
assumes $a \in \operatorname{set}_{v} v$
 blast

## lemma vec-setI

assumes $v \$ i=a i<d i m-v e c v$
shows $a \in$ set $_{v} v$ using assms unfolding vec-set-def using image-eqI lessThan-iff by blast
lemma set-list-of-vec: set (list-of-vec $v)=\operatorname{set}_{v} v$ unfolding vec-set-def by transfer auto
instantiation vec :: (conjugate) conjugate
begin
definition conjugate-vec :: 'a :: conjugate vec $\Rightarrow$ ' $a$ vec
where conjugate $v=$ vec (dim-vec $v$ ) ( $\lambda i$. conjugate ( $v \$ i$ )
lemma conjugate-vCons [simp]:
conjugate (vCons a $v$ ) $=v$ Cons (conjugate a) (conjugate $v$ )
by (auto simp: vec-Suc conjugate-vec-def)
lemma dim-vec-conjugate [simp]: dim-vec (conjugate $v$ ) $=\operatorname{dim-vec} v$ unfolding conjugate-vec-def by auto
lemma carrier-vec-conjugate $[$ simp $]: v \in$ carrier-vec $n \Longrightarrow$ conjugate $v \in$ carrier-vec $n$ by (auto intro!: carrier-vecI)
lemma vec-index-conjugate[simp]:
shows $i<$ dim-vec $v \Longrightarrow$ conjugate $v \$ i=$ conjugate $(v \$ i)$
unfolding conjugate-vec-def by auto
instance
proof
fix $v w::{ }^{\prime} a$ vec
show conjugate $($ conjugate $v)=v$ by $($ induct $v$, auto simp: conjugate-vec-def)
let $? v=$ conjugate $v$
let $? w=$ conjugate $w$
show conjugate $v=$ conjugate $w \longleftrightarrow v=w$
proof(rule iffI)
assume $c v w: ~ ? v=? w$ show $v=w$
proof (rule)
have dim-vec ?v = dim-vec ?w using cvw by auto
then show dim: dim-vec $v=\operatorname{dim}-v e c ~ w$ by simp
fix $i$ assume $i: i<d i m$-vec $w$
then have conjugate $v \$ i=$ conjugate $w \$ i$ using cvw by auto
then have conjugate $(v \$ i)=$ conjugate ( $w \$ i$ ) using $i$ dim by auto
then show $v \$ i=w \$ i$ by auto
qed
qed auto

```
qed
end
lemma conjugate-add-vec:
    fixes v w :: 'a :: conjugatable-ring vec
    assumes dim: v:carrier-vec n w : carrier-vec n
    shows conjugate (v+w)= conjugate v+ conjugate w
    by (rule, insert dim, auto simp: conjugate-dist-add)
lemma uminus-conjugate-vec:
    fixes v w :: 'a :: conjugatable-ring vec
    shows - (conjugate v)= conjugate ( }-v
    by (rule, auto simp:conjugate-neg)
lemma conjugate-zero-vec[simp]:
    conjugate ( }0v n :: ' a :: conjugatable-ring vec) = 0 v n by aut
lemma conjugate-vec- O[simp]:
    conjugate (vec 0f) = vec 0f by auto
lemma sprod-vec- O[simp]:v - vec 0f = 0
    by(auto simp: scalar-prod-def)
lemma conjugate-zero-iff-vec[simp]:
    fixes v :: 'a :: conjugatable-ring vec
    shows conjugate v=\mp@subsup{0}{v}{}n\longleftrightarrowv=\mp@subsup{0}{v}{}n
    using conjugate-cancel-iff[of-0v n :: 'a vec] by auto
lemma conjugate-smult-vec:
    fixes k :: 'a :: conjugatable-ring
    shows conjugate ( }k\cdotvv)=\mathrm{ conjugate }k\cdotv\mathrm{ conjugate v
    using conjugate-dist-mul by (intro eq-vecI, auto)
lemma conjugate-sprod-vec:
    fixes v w :: 'a :: conjugatable-ring vec
    assumes v:v:carrier-vec n and w: w:carrier-vec n
    shows conjugate (v • w) = conjugate v conjugate w
proof (insert w v, induct w arbitrary: v rule:carrier-vec-induct)
    case 0 then show ?case by (cases v, auto)
next
    case (Suc n b w) then show ?case
    by (cases v, auto dest: carrier-vecD simp:conjugate-dist-add conjugate-dist-mul)
qed
abbreviation cscalar-prod :: 'a vec }=>\mp@subsup{}{}{\prime}'a vec = ' 'a :: conjugatable-ring (infix •c
70)
    where }(\cdotc)\equiv\lambdavw.v\cdotconjugate 
```

```
lemma conjugate-conjugate-sprod[simp]:
    assumes v[simp]:v:carrier-vec n and w[simp]:w:carrier-vec n
    shows conjugate (conjugate v • w) =v c c w
    apply (subst conjugate-sprod-vec[of - n]) by auto
lemma conjugate-vec-sprod-comm:
    fixes v w ::' 'a :: {conjugatable-ring,comm-ring} vec
    assumes v:carrier-vec n and w:carrier-vec n
    shows v}
    unfolding scalar-prod-def using assms by(subst sum.ivl-cong, auto simp: ac-simps)
lemma conjugate-square-ge-0-vec[intro!]:
    fixes v :: 'a :: conjugatable-ordered-ring vec
    shows v•c v\geq0
proof (induct v)
    case vNil
    then show ?case by auto
next
    case (vCons a v)
    then show ?case using conjugate-square-positive[of a] by auto
qed
lemma conjugate-square-eq-0-vec[simp]:
    fixes v :: 'a :: {conjugatable-ordered-ring,semiring-no-zero-divisors} vec
    assumes v\incarrier-vec n
    shows v\cdotc v=0\longleftrightarrowv=0v}
proof (insert assms, induct rule: carrier-vec-induct)
    case 0
    then show ?case by auto
next
    case (Suc n a v)
    then show ?case
        using conjugate-square-positive[of a] conjugate-square-ge-0-vec[of v]
        by (auto simp: le-less add-nonneg-eq-0-iff zero-vec-Suc)
qed
lemma conjugate-square-greater-0-vec[simp]:
    fixes v :: 'a :: {conjugatable-ordered-ring,semiring-no-zero-divisors} vec
    assumes v\in carrier-vec n
    shows v}ccv>0\longleftrightarrowv\not=\mp@subsup{0}{v}{}
    using assms by (auto simp: less-le)
lemma vec-conjugate-rat[simp]:(conjugate :: rat vec }=>\mathrm{ rat vec) = ( }\lambdax.x)\mathrm{ by force
lemma vec-conjugate-real[simp]:(conjugate :: real vec }=>\mathrm{ real vec) = ( }\lambdax.x)\mathrm{ by
force
```

end

## 5 Code Generation for Basic Matrix Operations

In this theory we implement matrices as arrays of arrays. Due to the target language serialization, access to matrix entries should be constant time. Hence operations like matrix addition, multiplication, etc. should all have their standard complexity.

There might be room for optimizations.
To implement the infinite carrier set, we use A. Lochbihler's container framework [4].
theory Matrix-IArray-Impl
imports
Matrix
HOL-Library.IArray
Containers.Set-Impl

## begin

typedef 'a vec-impl $=\{(n, v::$ 'a iarray). IArray.length $v=n\}$ by auto
typedef 'a mat-impl $=\{(n r, n c, m::$ 'a iarray iarray $)$.
IArray.length $m=n r \wedge$ IArray.all $(\lambda r$. IArray.length $r=n c) m\}$
by (rule exI[of - (0,0,IArray [])], auto)
setup-lifting type-definition-vec-impl
setup-lifting type-definition-mat-impl
lift-definition vec-impl :: 'a vec-impl $\Rightarrow$ ' $a$ vec is
$\lambda(n, v) .(n, m k$-vec $n($ IArray.sub $v))$ by auto
lift-definition vec-add-impl :: 'a::plus vec-impl $\Rightarrow{ }^{\prime} a$ vec-impl $\Rightarrow{ }^{\prime} a$ vec-impl is $\lambda(n, v)(m, w)$.
( $n$, IArray.of-fun ( $\lambda i$. IArray.sub $v i+$ IArray.sub $w i) n$ )
by auto
lift-definition mat-impl :: 'a mat-impl $\Rightarrow$ 'a mat is
$\lambda(n r, n c, m) .(n r, n c, m k$-mat $n r n c(\lambda(i, j)$. IArray.sub (IArray.sub mi) j)) by auto
lift-definition vec-of-list-impl :: 'a list $\Rightarrow$ 'a vec-impl is
$\lambda v$. (length $v$, IArray $v$ ) by auto
lift-definition list-of-vec-impl :: 'a vec-impl $\Rightarrow$ 'a list is
$\lambda(n, v)$. IArray.list-of $v$.
lift-definition vec-of-fun :: nat $\Rightarrow\left(n a t \Rightarrow{ }^{\prime} a\right) \Rightarrow{ }^{\prime} a$ vec-impl is
$\lambda n f$. ( $n$, IArray.of-fun $f n$ ) by auto
lift-definition mat-of-fun :: nat $\Rightarrow$ nat $\Rightarrow\left(\right.$ nat $\left.\times n a t \Rightarrow{ }^{\prime} a\right) \Rightarrow{ }^{\prime} a$ mat-impl is $\lambda n r n c f$. (nr, nc, IArray.of-fun ( $\lambda$ i.IArray.of-fun $(\lambda j . f(i, j)) n c) n r)$ by auto

```
lift-definition vec-index-impl :: 'a vec-impl => nat = 'a
    is \lambda(n,v). IArray.sub v .
lift-definition index-mat-impl :: 'a mat-impl => nat }\times\mathrm{ nat => 'a
    is \lambda(nr,nc,m) (i,j). if i<nr then IArray.sub (IArray.sub m i) j
        else IArray.sub (IArray ([]!(i - nr))) j .
lift-definition vec-equal-impl :: 'a vec-impl => 'a vec-impl }=>\mathrm{ bool
    is }\lambda(n1,v1)(n2,v2).n1=n2 \wedgev1=v2.
lift-definition mat-equal-impl :: 'a mat-impl => 'a mat-impl => bool
    is \lambda (nr1,nc1,m1) (nr2,nc2,m2).nr1 = nr2 ^ nc1 = nc2 ^ m1 = m2 .
lift-definition dim-vec-impl :: 'a vec-impl => nat is fst .
lift-definition dim-row-impl :: 'a mat-impl => nat is fst .
lift-definition dim-col-impl :: 'a mat-impl }=>\mathrm{ nat is fst o snd.
code-datatype vec-impl
code-datatype mat-impl
lemma vec-code[code]: vec nf=vec-impl (vec-of-fun nf)
    by (transfer, auto simp: mk-vec-def)
lemma mat-code[code]: mat nr nc f= mat-impl (mat-of-fun nr nc f)
    by (transfer, auto simp: mk-mat-def, intro ext, clarsimp,
    auto intro: undef-cong-mat)
lemma vec-of-list[code]:vec-of-list v = vec-impl (vec-of-list-impl v)
    by (transfer, auto simp: mk-vec-def)
lemma list-of-vec-code[code]: list-of-vec (vec-impl v) = list-of-vec-impl v
    by (transfer, auto simp: mk-vec-def, case-tac b, auto intro: nth-equalityI)
lemma empty-nth: ᄀi< length }x\Longrightarrowx!i=[]!(i- length x)
    by (metis append-Nil2 nth-append)
lemma undef-vec: \negi< length }x\Longrightarrow\mathrm{ undef-vec ( }i=\mathrm{ length }x)=x!
    unfolding undef-vec-def by (rule empty-nth[symmetric])
lemma vec-index-code[code]: (vec-impl v) $ i= vec-index-impl vi
    by (transfer, auto simp: mk-vec-def, case-tac b, auto simp: undef-vec)
lemma index-mat-code[code]:(mat-impl m) $$ ij = (index-mat-impl m ij :: 'a)
proof (transfer, unfold o-def, clarify)
    fix m :: 'a iarray iarray and i j nc
    assume all: IArray.all ( }\lambdar\mathrm{ . IArray.length r =nc) m
    obtain mm where m: m= IArray mm by (cases m)
    with all have all: \v.v\in set mm\Longrightarrow IArray.length v=nc by auto
```

show snd (snd (IArray.length $m, n c$, mk-mat (IArray.length m) nc $(\lambda(i, y) . m$ !! $i!!y))(i, j)=$
(if $i<$ IArray.length $m$ then $m!!i!!j$
else IArray $([]!(i-$ IArray.length $m))!!j)($ is $? l=? r)$
proof (cases $i<$ length mm )

## case False

hence $\wedge f . \neg i<$ length (map $f[0 . .<$ length mm]) by simp note $[$ simp $]=$ empty-nth $[O F$ this]
have ?l $=[]!(i-$ length $m m)!j$ using False unfolding $m m k$-mat-def undef-mat-def by simp
also have $\ldots=$ ? $r$ unfolding $m$ by (simp add: False empty-nth[OF False])
finally show ?thesis .
next
case True
obtain $v$ where $m m: m m!i=$ IArray $v$ by (cases $m m!i$ )
with True all $[$ of $m m!i]$ have len: length $v=n c$ unfolding set-conv-nth by force
from $m m$ True have $? l=\operatorname{map}((!) v)[0 . .<n c]!j($ is $-=? m)$ unfolding $m$ mk-mat-def undef-mat-def by simp
also have $? m=m!!i!!j$
proof (cases $j<$ length $v$ )
case True
thus ?thesis unfolding $m$ using $m m$ len by auto
next
case False
hence $j: \neg j<$ length (map ((!) v) $[0 . .<$ length $v]$ ) by simp
show ?thesis unfolding $m$ using $m m$ len by (auto simp: empty-nth[OF $j$ ] empty-nth[OF False])
qed
also have $\ldots=$ ? $r$ using True $m$ by simp
finally show ?thesis.
qed
qed
lift-definition (code-dt) mat-of-rows-list-impl :: nat $\Rightarrow$ 'a list list $\Rightarrow{ }^{\prime}$ a mat-impl option is
$\lambda n$ rows. if list-all $(\lambda r$. length $r=n$ ) rows then Some (length rows, n, IArray (map IArray rows))

## else None

by (auto split: if-splits simp: list-all-iff)
lemma mat-of-rows-list-impl: mat-of-rows-list-impl n rs $=$ Some $A \Longrightarrow$ mat-impl A $=$ mat-of-rows-list $n$ rs
unfolding mat-of-rows-list-def
by (transfer, auto split: if-splits simp: list-all-iff intro!: cong-mk-mat)
lemma mat-of-rows-list-code[code]: mat-of-rows-list nc vs $=$ (case mat-of-rows-list-impl nc vs of Some $A \Rightarrow$ mat-impl $A$
$\mid$ None $\Rightarrow$ mat-of-rows nc (map ( $\lambda$ v. vec nc (nth v)) vs))

```
proof (cases mat-of-rows-list-impl nc vs)
    case (Some A)
    from mat-of-rows-list-impl[OF this] show ?thesis unfolding Some by simp
next
    case None
    show ?thesis unfolding None unfolding mat-of-rows-list-def mat-of-rows-def
        by (intro eq-matI, auto)
qed
lemma dim-vec-code[code]: dim-vec (vec-impl v) = dim-vec-impl v
    by (transfer, auto)
lemma dim-row-code[code]: dim-row (mat-impl m) = dim-row-impl m
    by (transfer, auto)
lemma dim-col-code[code]:dim-col (mat-impl m) = dim-col-impl m
    by (transfer, auto)
instantiation vec :: (type)equal
begin
    definition (equal-vec :: ('a vec 歽'a vec => bool)) = (=)
instance
    by (intro-classes, auto simp: equal-vec-def)
end
instantiation mat :: (type)equal
begin
    definition (equal-mat :: ('a mat }=>\mp@subsup{}{}{\prime}\mathrm{ 'a mat }=>\mathrm{ bool)) = (=)
instance
    by (intro-classes, auto simp: equal-mat-def)
end
lemma veq-equal-code[code]: HOL.equal (vec-impl (v1 :: 'a vec-impl)) (vec-impl
v2) = vec-equal-impl v1 v2
proof -
    {
        fix x1 x2 :: 'a list
        assume len: length x1 = length x2
            and index: (\lambdai. if i < length x2 then IArray x1 !! i else undef-vec ( i - length
(IArray.list-of (IArray x1)))) =
                            (\lambdai. if i< length x2 then IArray x2 !! i else undef-vec ( }i-l=l\mp@code{length
(IArray.list-of (IArray x2))))
    have x1 = x2
    proof (intro nth-equalityI[OF len])
        fix }
        assume i< length x1
        with fun-cong[OF index, of i] len show x1 ! i=x2! i by simp
    qed
} note * = this
```

```
    show ?thesis unfolding equal-vec-def
    by (transfer, insert *, auto simp: mk-vec-def, case-tac b, case-tac ba, auto)
qed
lemma mat-equal-code[code]: HOL.equal (mat-impl (m1 :: 'a mat-impl)) (mat-impl
m2) = mat-equal-impl m1 m2
proof -
    show ?thesis unfolding equal-mat-def
    proof (transfer, auto, case-tac b, case-tac ba, auto)
    fix x1 x2 :: 'a iarray list and nc
    assume len: }\forallr\inset x1. length (IArray.list-of r) = n
            \forallr\inset x2.length (IArray.list-of r) = nc
            length x1 = length x2
    and index: mk-mat (length x2) nc ( }\lambda(i,j).x1!i!! j) = mk-mat (length x2)
nc ( }\lambda(i,j).x2!i!! j
    show x1 = x2
    proof (rule nth-equalityI[OF len(3)])
            fix }
            assume i: i< length x1
            obtain ia1 where 1:x1!i=IArray ia1 by (cases x1 ! i)
            obtain ia2 where 2: x2 ! i = IArray ia2 by (cases x2 ! i)
            from i }1\mathrm{ len(1) have l1: length ia1 = nc using nth-mem by fastforce
            from i 2 len(2-3) have l2: length ia2 = nc using nth-mem by fastforce
            from l1 l2 have l: length ia1 = length ia2 by simp
            show x1! i = x2! i unfolding 12
            proof (simp, rule nth-equalityI[OF l])
                fix }
                assume j: j < length ia1
                with fun-cong[OF index, of (i,j)] i len(3)
                have x1!i !! j=x2! ! !! j
                    by (simp add: mk-mat-def l1)
                thus ia1!j = ia2! j unfolding 1 2 by simp
            qed
    qed
    qed
qed
declare prod.set-conv-list[code del, code-unfold]
derive (eq) ceq mat vec
derive (no) ccompare mat vec
derive (dlist) set-impl mat vec
derive (no) cenum mat vec
lemma carrier-mat-code[code]: carrier-mat \(n r n c=\operatorname{Collect-set}(\lambda A\). dim-row \(A\) \(=n r \wedge \operatorname{dim}-c o l ~ A=n c\) ) by auto
lemma carrier-vec-code[code]: carrier-vec \(n=\operatorname{Collect-set}(\lambda v\). dim-vec \(v=n)\)
unfolding carrier-vec-def by auto
```

end

## 6 Gauss-Jordan Algorithm

We define the elementary row operations and use them to implement the Gauss-Jordan algorithm to transform matrices into row-echelon-form. This algorithm is used to implement the inverse of a matrix and to derive certain results on determinants, as well as determine a basis of the kernel of a matrix.
theory Gauss-Jordan-Elimination
imports Matrix
begin

### 6.1 Row Operations

definition mat-multrow-gen $::\left({ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a\right) \Rightarrow$ nat $\Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a$ mat $\Rightarrow{ }^{\prime} a$ mat where
mat-multrow-gen mul $k$ a $A=\operatorname{mat}(\operatorname{dim}-r o w A)(\operatorname{dim}-c o l A)$

$$
(\lambda(i, j) . \text { if } k=i \text { then mul a }(A \$ \$(i, j)) \text { else } A \$ \$(i, j))
$$

abbreviation mat-multrow $::$ nat $\Rightarrow{ }^{\prime} a::$ semiring- $1 \Rightarrow^{\prime}$ 'a mat $\Rightarrow^{\prime}$ 'a mat (multrow) where

$$
\text { multrow } \equiv \text { mat-multrow-gen }((*))
$$

lemmas mat-multrow-def $=$ mat-multrow-gen-def
definition multrow-mat $::$ nat $\Rightarrow$ nat $\Rightarrow$ ' $a::$ semiring- $1 \Rightarrow{ }^{\prime} a$ mat where
multrow-mat $n k a=$ mat $n n$ $(\lambda(i, j)$. if $k=i \wedge k=j$ then a else if $i=j$ then 1 else 0$)$
definition mat-swaprows :: nat $\Rightarrow$ nat $\Rightarrow{ }^{\prime} a$ mat $\Rightarrow{ }^{\prime} a$ mat (swaprows)where swaprows kl $A=$ mat (dim-row $A)(\operatorname{dim}-c o l ~ A)$

$$
(\lambda(i, j) \text {. if } k=i \text { then } A \$ \$(l, j) \text { else if } l=i \text { then } A \$ \$(k, j) \text { else } A \$ \$(i, j))
$$

definition swaprows-mat $::$ nat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow{ }^{\prime} a$ :: semiring-1 mat where swaprows-mat $n k l=$ mat $n n$

$$
(\lambda(i, j) . \text { if } k=i \wedge l=j \vee k=j \wedge l=i \vee i=j \wedge i \neq k \wedge i \neq l \text { then } 1 \text { else } 0)
$$

definition mat-addrow-gen :: $\left({ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a\right) \Rightarrow\left({ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a\right) \Rightarrow{ }^{\prime} a \Rightarrow n a t \Rightarrow$ nat $\Rightarrow$ 'a mat $\Rightarrow$ 'a mat where
mat-addrow-gen ad mul a $k l A=$ mat (dim-row $A)(\operatorname{dim-col} A)$

$$
(\lambda(i, j) . \text { if } k=i \text { then ad }(\text { mul } a(A \$ \$(l, j)))(A \$ \$(i, j)) \text { else } A \$ \$(i, j))
$$

abbreviation mat-addrow :: 'a :: semiring-1 $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ 'a mat $\Rightarrow$ 'a mat (addrow) where

$$
\text { addrow } \equiv \text { mat-addrow-gen }(+)((*))
$$

lemmas mat-addrow-def $=$ mat-addrow-gen-def
definition addrow-mat :: nat $\Rightarrow$ ' $a$ :: semiring-1 $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ 'a mat where addrow-mat nakl=mat n $n(\lambda(i, j)$.
$($ if $k=i \wedge l=j$ then $(+) a$ else id) $($ if $i=j$ then 1 else 0$))$
lemma index-mat-multrow[simp]:
$i<$ dim-row $A \Longrightarrow j<$ dim-col $A \Longrightarrow$ mat-multrow-gen mul $k$ a $A \$ \$(i, j)=($ if $k=i$ then mul a $(A \$ \$(i, j))$ else $A \$ \$(i, j))$
$i<$ dim-row $A \Longrightarrow j<$ dim-col $A \Longrightarrow$ mat-multrow-gen mul i a $A \$ \$(i, j)=$ mul $a(A \$ \$(i, j))$
$i<$ dim-row $A \Longrightarrow j<\operatorname{dim}$-col $A \Longrightarrow k \neq i \Longrightarrow$ mat-multrow-gen mul $k$ a $A \$ \$$ $(i, j)=A \$ \$(i, j)$
dim-row (mat-multrow-gen mul $k$ a $A$ ) $=$ dim-row $A$ dim-col (mat-multrow-gen mul $k$ a $A$ ) $=\operatorname{dim}-\operatorname{col} A$
unfolding mat-multrow-def by auto
lemma index-mat-multrow-mat[simp]:
$i<n \Longrightarrow j<n \Longrightarrow$ multrow-mat $n k a \$ \$(i, j)=($ if $k=i \wedge k=j$ then a else if $i=j$
then 1 else 0)
dim-row (multrow-mat $n k a)=n$ dim-col $($ multrow-mat $n k a)=n$
unfolding multrow-mat-def by auto
lemma index-mat-swaprows [simp]:
$i<\operatorname{dim}$-row $A \Longrightarrow j<$ dim-col $A \Longrightarrow$ swaprows $k l A \$ \$(i, j)=($ if $k=i$ then A $\$ \$(l, j)$ else
if $l=i$ then $A \$ \$(k, j)$ else $A \$ \$(i, j))$
dim-row (swaprows kl $A$ ) $=$ dim-row $A$ dim-col (swaprows $k l A)=\operatorname{dim}-c o l A$
unfolding mat-swaprows-def by auto
lemma index-mat-swaprows-mat $[$ simp $]$ :
$i<n \Longrightarrow j<n \Longrightarrow$ swaprows-mat $n k l \$ \$(i, j)=$ (if $k=i \wedge l=j \vee k=j \wedge l=i \vee i=j \wedge i \neq k \wedge i \neq l$ then 1 else 0$)$ dim-row (swaprows-mat $n k l)=n$ dim-col (swaprows-mat $n k l)=n$
unfolding swaprows-mat-def by auto
lemma index-mat-addrow[simp]:
$i<$ dim-row $A \Longrightarrow j<$ dim-col $A \Longrightarrow$ mat-addrow-gen ad mul a kl $A \$ \$(i, j)=$ (if $k=i$ then
ad (mul a $(A \$ \$(l, j)))(A \$ \$(i, j))$ else $A \$ \$(i, j))$
$i<$ dim-row $A \Longrightarrow j<$ dim-col $A \Longrightarrow$ mat-addrow-gen ad mul a i l $A \$ \$(i, j)=$ ad $($ mul $a(A \$ \$(l, j)))(A \$ \$(i, j))$
$i<\operatorname{dim}-$ row $A \Longrightarrow j<\operatorname{dim}-\operatorname{col} A \Longrightarrow k \neq i \Longrightarrow$ mat-addrow-gen ad mul a kl $A$ $\$ \$(i, j)=A \$ \$(i, j)$
dim-row (mat-addrow-gen ad mul a kl A) = dim-row A dim-col (mat-addrow-gen ad mul a $k l A)=\operatorname{dim}-\operatorname{col} A$
unfolding mat-addrow-def by auto
lemma index-mat-addrow-mat[simp]:

$$
i<n \Longrightarrow j<n \Longrightarrow \text { addrow-mat } n \text { a } k l \$ \$(i, j)=
$$

$$
\begin{aligned}
& \quad(\text { if } k=i \wedge l=j \text { then }(+) \text { a else id })(\text { if } i=j \text { then } 1 \text { else } 0) \\
& \text { dim-row }(\text { addrow-mat } n \text { a } k l)=n \text { dim-col }(\text { addrow-mat } n \text { a } k l)=n \\
& \text { unfolding addrow-mat-def by auto }
\end{aligned}
$$

lemma multrow-carrier[simp]: (mat-multrow-gen mul $k$ a $A \in$ carrier-mat $n$ nc) $=(A \in$ carrier-mat $n$ nc $)$ unfolding carrier-mat-def by fastforce
lemma multrow-mat-carrier[simp]: multrow-mat $n k a \in$ carrier-mat $n n$ unfolding carrier-mat-def by auto
lemma addrow-mat-carrier [simp]: addrow-mat naklecarrier-mat $n n$ unfolding carrier-mat-def by auto
lemma swaprows-mat-carrier $[$ simp $]$ : swaprows-mat $n k l \in$ carrier-mat $n n$ unfolding carrier-mat-def by auto
lemma swaprows-carrier $[$ simp $]:($ swaprows $k l A \in$ carrier-mat $n n c)=(A \in$ car-rier-mat $n n c$ )
unfolding carrier-mat-def by fastforce
lemma addrow-carrier[simp]: (mat-addrow-gen ad mul a kl $A \in$ carrier-mat n nc) $=(A \in$ carrier-mat $n$ nc $)$ unfolding carrier-mat-def by fastforce
lemma row-multrow: $k \neq i \Longrightarrow i<n \Longrightarrow$ row (multrow-mat $n k$ a) $i=$ unit-vec $n i$

$$
k<n \Longrightarrow \text { row (multrow-mat } n k a) k=a \cdot v \text { unit-vec } n k
$$

by (rule eq-vecI, auto)
lemma multrow-mat: assumes $A: A \in$ carrier-mat $n$ nc shows multrow $k$ a $A=$ multrow-mat $n k a * A$
by (rule eq-matI, insert $A$, auto simp: row-multrow smult-scalar-prod-distrib[of $n]$ )
lemma row-addrow:
$k \neq i \Longrightarrow i<n \Longrightarrow$ row (addrow-mat $n$ a $k l$ ) $i=$ unit-vec $n i$
$k<n \Longrightarrow l<n \Longrightarrow$ row (addrow-mat $n a k l$ ) $k=a \cdot v$ unit-vec $n l+$ unit-vec
$n k$
by (rule eq-vecI, auto)
lemma addrow-mat: assumes $A: A \in$ carrier-mat $n n c$ and $l: l<n$
shows addrow a $k l A=$ addrow-mat $n$ a $k l * A$
by (rule eq-matI, insert $l$, auto simp: row-addrow add-scalar-prod-distrib[of-n] smult-scalar-prod-distrib[of - $n$ ])
lemma row-swaprows:
$l<n \Longrightarrow$ row (swaprows-mat $n l l$ ) $l=$ unit-vec $n l$

```
\(i \neq k \Longrightarrow i \neq l \Longrightarrow i<n \Longrightarrow\) row (swaprows-mat \(n k l\) ) \(i=\) unit-vec \(n i\)
\(k<n \Longrightarrow l<n \Longrightarrow\) row (swaprows-mat \(n k l\) ) \(l=\) unit-vec \(n k\)
\(k<n \Longrightarrow l<n \Longrightarrow\) row (swaprows-mat \(n k l\) ) \(k=\) unit-vec \(n l\)
by (rule eq-vecI, auto)
```

lemma swaprows-mat: assumes $A$ : $A \in$ carrier-mat $n n c$ and $k: k<n l<n$ shows swaprows $k l A=$ swaprows-mat $n k l * A$
by (rule eq-matI, insert $A k$, auto simp: row-swaprows)
lemma swaprows-mat-inv: assumes $k: k<n$ and $l: l<n$
shows swaprows-mat $n k l *$ swaprows-mat $n k l=1_{m} n$
proof -
have swaprows-mat $n k l *$ swaprows-mat $n k l=$
swaprows-mat $n k l *\left(\right.$ swaprows-mat $\left.n k l * 1_{m} n\right)$
by (simp add: right-mult-one-mat $[o f-n]$ )
also have swaprows-mat $n k l * 1_{m} n=$ swaprows $k l\left(1_{m} n\right)$
by (rule swaprows-mat $[$ symmetric, $O F-k l$, of $-n]$, simp)
also have swaprows-mat $n k l * \ldots=$ swaprows $k l \ldots$
by (rule swaprows-mat $[$ symmetric, of $--n]$, insert $k l$, auto)
also have $\ldots=1_{m} n$
by (rule eq-matI, insert $k l$, auto)
finally show? ?thesis.
qed
lemma swaprows-mat-Unit: assumes $k: k<n$ and $l: l<n$
shows swaprows-mat $n k l \in$ Units (ring-mat TYPE ('a :: semiring-1) n b)
proof -
interpret $m$ : semiring ring-mat $T Y P E\left({ }^{\prime} a\right) n b$ by (rule semiring-mat)
show ?thesis unfolding Units-def
by (rule, rule conjI[OF - bexI[of - swaprows-mat $n k l]]$,
auto simp: ring-mat-def swaprows-mat-inv[OF $k l]$ swaprows-mat-inv $[O F l k])$
qed
lemma addrow-mat-inv: assumes $k: k<n$ and $l: l<n$ and $n e q: k \neq l$
shows addrow-mat $n$ a $k l *$ addrow-mat $n(-(a:: ' a::$ comm-ring-1)) $k l=$
$1_{m} n$
proof -
have addrow-mat nakl* addrow-mat $n(-a) k l=$
addrow-mat $n$ a $k l *\left(\right.$ addrow-mat $\left.n(-a) k l * 1_{m} n\right)$
by (simp add: right-mult-one-mat $[o f-n]$ )
also have addrow-mat $n(-a) k l * 1_{m} n=$ addrow $(-a) k l\left(1_{m} n\right)$
by (rule addrow-mat $[$ symmetric, of $-n]$, insert $k l$, auto)
also have addrow-mat $n$ a $k l * \ldots=$ addrow a $k l \ldots$
by (rule addrow-mat[symmetric, of $--n$ ], insert $k l$, auto)
also have $\ldots=1_{m} n$
by (rule eq-matI, insert $k l$ neq, auto, algebra)
finally show? ?thesis.
qed
lemma addrow-mat-Unit: assumes $k: k<n$ and $l: l<n$ and neq: $k \neq l$ shows addrow-mat $n$ a $k l \in$ Units (ring-mat TYPE('a :: comm-ring-1) n b) proof -
interpret $m$ : semiring ring-mat $T Y P E\left({ }^{\prime} a\right) n b$ by (rule semiring-mat)
show ?thesis unfolding Units-def
by (rule, rule conjI[OF - bexI[of - addrow-mat $n(-a) k l]]$, insert neq, auto simp: ring-mat-def addrow-mat-inv[OF $k l n e q]$, rule trans $[O F-a d d r o w-m a t-i n v[O F k l n e q$, of $-a]]$, auto)
qed
lemma multrow-mat-inv: assumes $k: k<n$ and $a:\left(a::{ }^{\prime} a::\right.$ division-ring $) \neq 0$
shows multrow-mat $n k a *$ multrow-mat $n k($ inverse $a)=1_{m} n$
proof -
have multrow-mat $n k a *$ multrow-mat $n k$ (inverse $a)=$ multrow-mat $n k a *$ (multrow-mat $n k$ (inverse $a) * 1_{m} n$ ) using $k$ by (simp add: right-mult-one-mat $[o f-n]$ )
also have multrow-mat $n k$ (inverse $a) * 1_{m} n=$ multrow $k$ (inverse $\left.a\right)\left(1_{m} n\right)$
by (rule multrow-mat [symmetric, of - $n$ ], insert $k$, auto)
also have multrow-mat $n k a * \ldots=$ multrow $k a \ldots$
by (rule multrow-mat $[$ symmetric, of $-n]$, insert $k$, auto)
also have $\ldots=1_{m} n$
by (rule eq-matI, insert a $k$ a, auto)
finally show? ?thesis.
qed
lemma multrow-mat-Unit: assumes $k: k<n$ and $a:\left(a::{ }^{\prime} a::\right.$ division-ring $) \neq 0$ shows multrow-mat $n k a \in$ Units (ring-mat $\operatorname{TYPE}\left({ }^{\prime} a\right) n b$ )
proof -
from $a$ have ia: inverse $a \neq 0$ by auto
interpret $m$ : semiring ring-mat $T Y P E\left({ }^{\prime} a\right) n b$ by (rule semiring-mat)
show ?thesis unfolding Units-def
by (rule, rule conjI[OF - bexI[of - multrow-mat $n k$ (inverse a) $]$, insert a, auto simp: ring-mat-def multrow-mat-inv $[O F k]$,
rule trans $[$ OF - multrow-mat-inv $[$ OF $k$ ia] ], insert a, auto)
qed

### 6.2 Gauss-Jordan Elimination

fun eliminate-entries-rec where
eliminate-entries-rec $B i[]=B$
| eliminate-entries-rec $B i\left(\left(a i^{\prime} j, i^{\prime}\right) \# i s\right)=($
eliminate-entries-rec (mat-addrow-gen $\left((+):: ' b:: r i n g-1 \Rightarrow{ }^{\prime} b \Rightarrow{ }^{\prime} b\right)(*) a i^{\prime} j i^{\prime}$ i B) $i$ is)

```
context
    fixes minus :: ' }a=>\mp@subsup{}{}{\prime}a=>\mp@subsup{}{}{\prime}
    and times :: ' }a=>\mp@subsup{}{}{\prime}a=>\mp@subsup{}{}{\prime}
begin
```

definition eliminate-entries-gen $::\left(\right.$ nat $\left.\Rightarrow{ }^{\prime} a\right) \Rightarrow{ }^{\prime} a$ mat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow{ }^{\prime}$ a mat where
eliminate-entries-gen v AIJ=mat (dim-row $A)(\operatorname{dim-col} A)(\lambda(i, j)$. if $i \neq I$ then minus $(A \$ \$(i, j))$ (times $(v i)(A \$ \$(I, j)))$ else $A \$ \$(i, j))$
lemma dim-eliminate-entries-gen[simp]: dim-row (eliminate-entries-gen v B i as)
= dim-row $B$
dim-col (eliminate-entries-gen vBias) $=$ dim-col $B$
unfolding eliminate-entries-gen-def by auto
lemma dimc-eliminate-entries-rec $[$ simp $]$ : dim-col (eliminate-entries-rec B i as) $=$ dim-col B
by (induct as arbitrary: B, auto simp: Let-def)
lemma dimr-eliminate-entries-rec[simp]: dim-row (eliminate-entries-rec Bias)= dim-row $B$
by (induct as arbitrary: B, auto simp: Let-def)
lemma carrier-eliminate-entries: $A \in$ carrier-mat $n r n c \Longrightarrow$ eliminate-entries-gen $v A i b s \in$ carrier-mat nr nc
$B \in$ carrier-mat $n r n c \Longrightarrow$ eliminate-entries-rec $B$ i as $\in$ carrier-mat $n r n c$ unfolding carrier-mat-def by auto
end
abbreviation eliminate-entries $\equiv$ eliminate-entries-gen $(-)((*):: ~ ' a ~:: ~ r i n g-1 \Rightarrow$ ${ }^{\prime} a \Rightarrow{ }^{\prime} a$ )
lemma eliminate-entries-convert:
assumes $j A: J<\operatorname{dim}$-col $A$ and $*: I<d i m$-row $A$ dim-row $B=d i m$-row $A$ shows eliminate-entries $(\lambda i . A \$ \$(i, J)) B I J=$
eliminate-entries-rec BI(map $(\lambda i$. $(-A \$ \$(i, J), i))(f i l t e r(\lambda i . i \neq I)[0$
$. .<\operatorname{dim}-$ row $A]$ ))
proof -
let ?ais $=\lambda$ is. map $(\lambda i .(-A \$ \$(i, J), i))($ filter $(\lambda i . i \neq I) i s)$
define one-go where one-go $=(\lambda B$ is. mat (dim-row $B)(d i m-c o l B)(\lambda(i, j)$. if $i \neq I \wedge i \in$ set is then $B \$(i, j)-(A \$ \$(i, J)) * B \$ \$(I, j)$ else $B \$ \$$ $(i, j))$ )
\{
fix is :: nat list
assume distinct is
from $*$ this have eliminate-entries-rec $B I$ (?ais is) $=$ one-go $B$ is
proof (induct is arbitrary: B)
case Nil
show ?case unfolding one-go-def
by (rule eq-matI, auto)
next
case (Cons $i$ is)
note $I=\operatorname{Cons}(2)$ note $\operatorname{dim}=\operatorname{Cons(3)}$
note $I I=\operatorname{Cons}(2)[$ folded $\operatorname{dim}]$

```
    let ?B = addrow (-A $$ (i,J)) i I B
    from Cons(4) I dim have I < dim-row A dim-row ?B = dim-row A and
dist: distinct is by auto
    note IH=Cons(1)[OF this]
    from Cons(4) have i:i\not\in set is by auto
    show ?case
    proof (cases i=I)
    case False
    hence id: ?ais ( }i#\mathrm{ # is) = (-A $$ (i, J), i) # ?ais is by simp
    show ?thesis unfolding id eliminate-entries-rec.simps IH
        unfolding one-go-def index-mat-addrow
    proof (rule eq-matI, goal-cases)
        case (1 ii jj)
        hence ii: ii < dim-row B and jj: jj < dim-col B and iiA: ii < dim-row
A using dim by auto
            show ?case unfolding index-mat[OF ii jj] split
                index-mat-addrow(1)[OF ii jj] index-mat-addrow(1)[OF II jj]
                using i False by auto
        qed auto
    next
        case True
        hence id:?ais (i# is) = ?ais is by simp
        show ?thesis unfolding id Cons(1)[OF I dim dist]
            unfolding one-go-def True by auto
        qed
    qed
    } note main = this
    show ?thesis
    by (subst main, force, unfold one-go-def eliminate-entries-gen-def, rule eq-matI,
    insert *, auto)
qed
lemma Unit-prod-eliminate-entries: i<nr\Longrightarrow(\bigwedgea i'.(a,i') \in set is \Longrightarrow < i'<
nr\wedge 汶\not=i)
    \Longrightarrow \exists P \in U n i t s ~ ( r i n g - m a t ~ T Y P E ( ' a ~ : : ~ c o m m - r i n g - 1 ) ~ n r ~ b ) . \forall ~ B ~ n c . ~ B \in
carrier-mat nr nc\longrightarrow eliminate-entries-rec B i is=P*B
proof (induct is)
    case Nil
    thus ?case by (intro bexI[of-1m nr], auto simp: Units-def ring-mat-def)
next
    case (Cons ai' is)
    obtain a i' where ai': a\mp@subsup{i}{}{\prime}=(a,\mp@subsup{i}{}{\prime})\mathrm{ by force}
    let ?U = Units (ring-mat TYPE('a) nr b)
    interpret m: ring ring-mat TYPE ('a) nr b by (rule ring-mat)
    from Cons(1)[OF Cons(2-3)]
    obtain P where P:P\in?U and id: \bigwedgeBnc.B\incarrier-mat nr nc \Longrightarrow
    eliminate-entries-rec B i is =P*B by force
    let ?Add=addrow-mat nr a i'}
```

```
    have Add:?Add \in?U
    by (rule addrow-mat-Unit, insert Cons ai', auto)
from m.Units-m-closed[OF P Add] have PI: P*?Add \in?U unfolding ring-mat-def
by simp
    from m.Units-closed[OF P] have P:P\incarrier-mat nr nr unfolding ring-mat-def
by simp
    show ?case
    proof (rule bexI[OF - PI], intro allI impI)
    fix B :: 'a mat and nc
    assume BB: B\incarrier-mat nr nc
    let ? B = addrow a i' i B
    from }BB\mathrm{ have B:?B G carrier-mat nr nc by simp
    from id [OF B] have id: eliminate-entries-rec ? B i is = P* ?B.
    have id2: eliminate-entries-rec B i (ai' # is)= eliminate-entries-rec ?B i is
unfolding ai' by simp
    show eliminate-entries-rec B i (ai' # is) = P * ?Add * B
        unfolding id2 id unfolding addrow-mat[OF BB Cons(2)]
        by (rule assoc-mult-mat[symmetric, OF P - BB], auto)
    qed
qed
function gauss-jordan-main :: 'a :: field mat }=>\mp@subsup{|}{}{\prime}a\mathrm{ mat }=>\mathrm{ nat }=>\mathrm{ nat }=>\mp@subsup{}{}{\prime}'a mat ×
'a mat where
    gauss-jordan-main A Bij=(let nr = dim-row A;nc=dim-col A in
    if i<nr\wedgej<nc then let aij =A $$(i,j) in if aij=0 then
        (case [ }\mp@subsup{i}{}{\prime}.\mp@subsup{i}{}{\prime}<-[\mathrm{ Suc i ..<nr], A $$ (i',j) =0]
            of [] => gauss-jordan-main A B i (Suc j)
            | (i'# -) = gauss-jordan-main (swaprows i i' A)(swaprows i i' B) i j)
    else if aij = 1 then let
            v=(\lambdai.A$$(i,j)) in
            gauss-jordan-main
            (eliminate-entries v A i j) (eliminate-entries v B i j) (Suc i) (Suc j)
            else let iaij = inverse aij in gauss-jordan-main (multrow i iaij A) (multrow i
iaij B) i j
        else (A,B))
    by pat-completeness auto
termination
proof -
    let ?R = measures [\lambda ( }A:: ' a :: field mat,B,i,j). dim-col A-j
    \lambda(A,B,i,j). if A $$ (i,j)=0 then 2 else if A $$ (i,j)=1 then 0 else 1]
    show ?thesis
    proof
        show wf ?R by auto
    next
    fix A B :: 'a mat and ijnrnc a i' is
    assume *:nr = dim-row A nc = dim-col A i<nr ^j<nc a=A$$ (i,j) a
= 0
    and ne: [ [ ' . . i'<- [Suc i..<nr], A $$ (i',j)\not=0]=\mp@subsup{i}{}{\prime}# is
```

from ne have $i^{\prime} \in \operatorname{set}\left(\left[i^{\prime} . i^{\prime}<-[S u c i . .<n r], A \$ \$\left(i^{\prime}, j\right) \neq 0\right]\right)$ by auto with *
show ((swaprows $i i^{\prime} A$, swaprows $\left.\left.i i^{\prime} B, i, j\right), A, B, i, j\right) \in ? R$ by auto qed auto
qed
declare gauss-jordan-main.simps[simp del]
definition gauss-jordan $A B \equiv$ gauss-jordan-main $A B 00$
lemma gauss-jordan-transform: assumes $A: A \in$ carrier-mat nr nc and $B: B \in$ carrier-mat $n r n c^{\prime}$
and res: gauss-jordan ( $A::$ 'a :: field mat) $B=\left(A^{\prime}, B^{\prime}\right)$
shows $\exists P \in$ Units (ring-mat TYPE('a)nr b). $A^{\prime}=P * A \wedge B^{\prime}=P * B$
proof -
let ? $U=$ Units (ring-mat TYPE ('a) nr b)
interpret $m$ : ring ring-mat TYPE ('a) nr bly (rule ring-mat)
\{
fix $i j$ :: nat
assume gauss-jordan-main $A B i j=\left(A^{\prime}, B^{\prime}\right)$
with $A B$
have $\exists P \in ? U . A^{\prime}=P * A \wedge B^{\prime}=P * B$
proof (induction A B ij rule: gauss-jordan-main.induct)
case ( $1 A B i j$ )
note $A=1$ (5)
hence dim: dim-row $A=n r$ dim-col $A=n c$ by auto
note $B=1(6)$
hence $\operatorname{dim} B$ : dim-row $B=n r$ dim-col $B=n c^{\prime}$ by auto
note $I H=1(1-4)[O F \operatorname{dim}[$ symmetric $]]$
note res $=1(7)$
note $\operatorname{simp}=$ gauss-jordan-main.simps[of $A B i j]$ Let-def
let ? $g=$ gauss-jordan-main $A B i j$
show ?case
proof (cases $i<n r \wedge j<n c$ )
case False
with res have res: $A^{\prime}=A B^{\prime}=B$ unfolding simp dim by auto show ?thesis unfolding res
by (rule bexI[of-1m nr], insert A B, auto simp: Units-def ring-mat-def) next
case True note valid $=$ this
note $I H=I H[$ OF valid refl $]$
show ?thesis
proof (cases A $\$ \$(i, j)=0)$
case False note $n Z=$ this
note $I H=I H(3-4)[O F n Z]$
show ?thesis
proof (cases A $\$ \$(i, j)=1$ )
case False note $n O=$ this
let ? inv $=$ inverse $(A \$ \$(i, j))$
from $n O n Z$ valid res
have gauss-jordan-main (multrow $i$ ? inv A) (multrow $i$ ? inv $B$ ) $i j=$ $\left(A^{\prime}, B^{\prime}\right)$
unfolding simp dim by simp
note $I H=I H$ (2)[OF nO refl, unfolded multrow-carrier, OF A B this]
from $I H$ obtain $P$ where $P: P \in ? U$ and
id: $A^{\prime}=P *$ multrow $i$ ? inv $A B^{\prime}=P *$ multrow $i$ ? inv $B$ by blast
let ? Inv $=$ multrow-mat nr $i$ ? inv
from $n Z$ valid have $i<n r$ ? inv $\neq 0$ by auto
from multrow-mat-Unit[OF this]
have Inv: ?Inv $\in$ ? U .
from m.Units-m-closed $[$ OF P Inv] have PI: P *?Inv $\in$ ? U unfolding ring-mat-def by simp
from m.Units-closed $[O F P]$ have $P: P \in$ carrier-mat nr nr unfolding ring-mat-def by simp
show ?thesis unfolding id unfolding multrow-mat $[O F A]$ mul-trow-mat $[$ OF B]
by (rule bexI $[O F-P I]$, intro conjI,
rule assoc-mult-mat $[$ symmetric, OF P - A], simp,
rule assoc-mult-mat $[s y m m e t r i c, ~ O F ~ P ~-~ B], ~ s i m p) ~(~) ~$
next
case True note $O=$ this
let ? is $=$ filter $\left(\lambda i^{\prime} . i^{\prime} \neq i\right)[0 \quad . .<n r]$
let ?ais $=\operatorname{map}\left(\lambda i^{\prime} .\left(-A \$ \mathbb{S}\left(i^{\prime}, j\right), i^{\prime}\right)\right)$ ? is
let ? $E=\lambda$ B. eliminate-entries $(\lambda i . A \$ \$(i, j)) B i j$
let ? $E E=\lambda B$. eliminate-entries-rec $B i$ ?ais
let $? A=$ ? $E A$
let $? B=$ ? $E B$
let ? $A A=$ ? $E E A$
let ? $B B=$ ? $E E B$
from $O n Z$ valid res have gauss-jordan-main ?A ?B $($ Suc $i)($ Suc $j)=$ $\left(A^{\prime}, B^{\prime}\right)$
unfolding $\operatorname{simp} \operatorname{dim}$ by $\operatorname{simp}$
note $I H=I H(1)[O F O$ refl carrier-eliminate-entries $(1)[O F A]$ car-rier-eliminate-entries(1)[OF B] this]
from $I H$ obtain $P$ where $P: P \in ? U$ and $i d: A^{\prime}=P * ? A B^{\prime}=P *$ ?B by blast
have $*: j<\operatorname{dim}$-col $A i<d i m-r o w A$ by (auto simp add: dim valid)
have $\exists P \in$ ? U. $\forall B$ nc. $B \in$ carrier-mat $n r n c \longrightarrow$ ? $E E B=P * B$
by (rule Unit-prod-eliminate-entries, insert valid, auto)
then obtain $Q$ where $Q: Q \in ? U$ and $Q Q: \bigwedge B$ nc. $B \in$ carrier-mat $n r n c \Longrightarrow$ ? $E E B=Q * B$ by auto
\{
fix $B$ :: 'a mat and $n c$
assume $B: B \in$ carrier-mat nr nc with $\operatorname{dim}$ have dim-row $B=\operatorname{dim}$-row $A$ by auto
from eliminate-entries-convert[OF * this]
have ? $E B=$ ? EE $B$ using $\operatorname{dim}$ by $\operatorname{simp}$
also have $\ldots=Q * B$ using $Q Q[O F B]$ by simp
finally have ? $E B=Q * B$.
\} note $Q Q=$ this
have $i d 3: ? A=Q * A$ by (rule $Q Q[O F A]$ )
have $i d_{4}: ? B=Q * B$ by $($ rule $Q Q[$ OF $B])$
from m.Units-closed $[O F P]$ have $P c: P \in$ carrier-mat nr nr unfolding ring-mat-def by simp
from m.Units-closed $[O F Q]$ have $Q c: Q \in$ carrier-mat $n r$ nr unfolding ring-mat-def by simp
from m.Units-m-closed[ $O F P \quad Q$ ] have $P Q: P * Q \in$ ? $U$ unfolding ring-mat-def by simp
show ?thesis unfolding id unfolding id3 id4
by (rule bex $[$ $O F-P Q]$, rule conjI,
rule assoc-mult-mat[symmetric, OF Pc Qc A],
rule assoc-mult-mat $[$ symmetric, OF Pc $Q c \quad B])$
qed
next
case True note $Z=$ this
note $I H=I H(1-2)[O F Z]$
let ? is $=\left[i^{\prime} . i^{\prime}<-[\right.$ Suc $\left.i . .<n r], A \$ \$\left(i^{\prime}, j\right) \neq 0\right]$
show ?thesis
proof (cases ?is)
case Nil
from $Z$ valid res have $i d$ : gauss-jordan-main $A B i(S u c j)=\left(A^{\prime}, B^{\prime}\right)$ unfolding simp dim Nil by simp
from $I H(1)[O F$ Nil $A B$ this] show ?thesis unfolding id.
next
case (Cons $i^{\prime}$ iis)
from $Z$ valid res have gauss-jordan-main (swaprows $i i^{\prime} A$ ) (swaprows $i$ $\left.i^{\prime} B\right) i j=\left(A^{\prime}, B^{\prime}\right)$
unfolding simp dim Cons by simp
from $I H$ (2) $[$ OF Cons, unfolded swaprows-carrier, OF A B this]
obtain $P$ where $P: P \in ? U$ and
$i d: A^{\prime}=P *$ swaprows $i i^{\prime} A B^{\prime}=P *$ swaprows $i i^{\prime} B$ by blast
let ? Swap = swaprows-mat nr $i i^{\prime}$
from Cons have $i^{\prime} \in$ set ? is by auto
with valid have $i^{\prime}: i<n r i^{\prime}<n r$ by auto
from swaprows-mat-Unit[OF this] have Swap: ?Swap $\in$ ? U .
from m.Units-m-closed $[$ OF P Swap] have PI: P*?Swap $\in$ ? U unfolding ring-mat-def by simp
from m.Units-closed $[O F P]$ have $P: P \in$ carrier-mat $n r n r$ unfolding ring-mat-def by simp
show ?thesis unfolding id swaprows-mat [OF A i ] swaprows-mat $[O F B$ $\left.{ }^{\dagger}\right]$
by (rule bexI $[O F-P I]$, rule conjI,
rule assoc-mult-mat $[$ symmetric, $O F P-A]$, simp,
rule assoc-mult-mat $[$ symmetric, OF $P-B]$, simp)
qed
qed
qed
qed
\}
from this[of 00 , folded gauss-jordan-def, OF res] show ?thesis.
qed
lemma gauss-jordan-carrier: assumes $A:\left(A::{ }^{\prime} a\right.$ :: field mat $) \in$ carrier-mat $n r$ $n c$
and $B: B \in$ carrier-mat $n r n c^{\prime}$
and res: gauss-jordan $A B=\left(A^{\prime}, B^{\prime}\right)$
shows $A^{\prime} \in$ carrier-mat nr nc $B^{\prime} \in$ carrier-mat $n r n c^{\prime}$
proof -
from gauss-jordan-transform[OF A B res, of undefined]
obtain $P$ where $P: P \in$ Units (ring-mat TYPE $\left(^{\prime} a\right)$ nr undefined)
and $i d: A^{\prime}=P * A B^{\prime}=P * B$ by auto
from $P$ have $P: P \in$ carrier-mat $n r n r$ unfolding Units-def ring-mat-def by auto
show $A^{\prime} \in$ carrier-mat $n r n c B^{\prime} \in$ carrier-mat $n r n c^{\prime}$ unfolding id
using $P A B$ by auto
qed
definition pivot-fun $::$ ' $a::\{$ zero,one $\}$ mat $\Rightarrow(n a t \Rightarrow n a t) \Rightarrow$ nat $\Rightarrow$ bool where pivot-fun $A f n c \equiv$ let $n r=$ dim-row $A$ in
$(\forall i<n r . f i \leq n c \wedge$
$\left(f i<n c \longrightarrow A \$ \$(i, f i)=1 \wedge\left(\forall i^{\prime}<n r . i^{\prime} \neq i \longrightarrow A \$ \$\left(i^{\prime}, f i\right)=0\right)\right) \wedge$ $(\forall j<f i . A \$ \$(i, j)=0) \wedge$
$($ Suc $i<n r \longrightarrow f($ Suc $i)>f i \vee f($ Suc $i)=n c))$
lemma pivot-funI: assumes $d$ : dim-row $A=n r$
and $*: \bigwedge i . i<n r \Longrightarrow f i \leq n c$
^ij. $i<n r \Longrightarrow j<f i \Longrightarrow A \$ \$(i, j)=0$
$\bigwedge i . i<n r \Longrightarrow S u c i<n r \Longrightarrow f($ Suc $i)>f i \vee f($ Suc $i)=n c$
ヘi. $i<n r \Longrightarrow f i<n c \Longrightarrow A \$ \$(i, f i)=1$
$\wedge i i^{\prime} . i<n r \Longrightarrow f i<n c \Longrightarrow i^{\prime}<n r \Longrightarrow i^{\prime} \neq i \Longrightarrow A \$ \$\left(i^{\prime}, f i\right)=0$
shows pivot-fun $A f n c$
unfolding pivot-fun-def Let-def $d$ using $*$ by blast
lemma pivot-funD: assumes $d$ : dim-row $A=n r$
and $p$ : pivot-fun $A f n c$
shows $\bigwedge i . i<n r \Longrightarrow f i \leq n c$
$\bigwedge i j . i<n r \Longrightarrow j<f i \Longrightarrow A \$ \$(i, j)=0$
$\bigwedge i . i<n r \Longrightarrow S u c i<n r \Longrightarrow f($ Suc $i)>f i \vee f($ Suc $i)=n c$
ヘi. $i<n r \Longrightarrow f i<n c \Longrightarrow A \$ \$(i, f i)=1$
$\bigwedge i i^{\prime} . i<n r \Longrightarrow f i<n c \Longrightarrow i^{\prime}<n r \Longrightarrow i^{\prime} \neq i \Longrightarrow A \$ \$\left(i^{\prime}, f i\right)=0$
using $p$ unfolding pivot-fun-def Let-def $d$ by blast+
lemma pivot-fun-multrow: assumes p: pivot-fun $A f j j$
and $d$ : dim-row $A=n r \operatorname{dim}-c o l A=n c$
and $f: f i 0=j j$

```
    and jj: jj\leqnc
    shows pivot-fun (multrow i0 a A) fjj
proof -
    note p = pivot-funD[OF d(1) p]
    let ?A = multrow i0 a A
    have dim-row ?A = nr using d by simp
    thus ?thesis
    proof (rule pivot-funI)
        fix i
        assume i:i<nr
        note p=p[OFi]
        show fi\leqjj by fact
        show Suc i<nr\Longrightarrowfi<f(Suc i)\veef(Suc i)=jj by fact
        {
        fix }\mp@subsup{i}{}{\prime
        assume *: fi<jj i'<nr i'\not=i
        from p(5)[OF this]
        show ?A $$ (i', fi)=0
            by (subst index-mat-multrow(1), insert *d jj, auto)
    }
        assume *: fi<jj
        from p(4)[OF this] have A: A $$ (i,fi)=1 by auto
        show ?A $$ (i,fi)=1
            by (subst index-mat-multrow(1), insert * di A jj fi,auto)
        }
        {
            fix }
        assume j: j<fi
        from p(2)[OF j]
        show ?A $$ (i,j)=0
            by (subst index-mat-multrow(1), insert j d i p jj fi, auto)
        }
    qed
qed
lemma pivot-fun-swaprows: assumes p: pivot-fun A f jj
    and d: dim-row }A=nr\mathrm{ dim-col }A=n
    and flk:fl=jjfk=jj
    and nr:l<nrk<nr
    and jj: jj\leqnc
    shows pivot-fun (swaprows l k A)fjj
proof -
    note pivot = pivot-funD[OF d(1) p]
    let ?A = swaprows l k A
    have dim-row ?A = nr using d by simp
    thus ?thesis
    proof (rule pivot-funI)
    fix }
```

```
        assume i: i<nr
        note p= pivot[OF i]
        show fi\leqjj by fact
        show Suc i<nr\Longrightarrowfi<f(Suc i)\veef(Suc i)=jj by fact
        {
        fix }\mp@subsup{i}{}{\prime
        assume *: fi<jj i'<nr i'\not=i
        from *(1) flk have diff: l}\not=ik\not=i\mathrm{ by auto
        from p(5)[OF *] p(5)[OF *(1) nr(1) diff(1)] p(5)[OF *(1) nr(2) diff(2)]
        show ?A $$ (i', fi)=0
            by (subst index-mat-swaprows(1), insert *d jj, auto)
    }
    {
        assume *: fi<jj
        from p(4)[OF this] have A:A $$ (i,fi)=1 by auto
        show ?A $$ (i,fi)=1
            by (subst index-mat-swaprows(1), insert *d i A jj flk, auto)
        }
        {
        fix }
        assume j:j<fi
        with p(1) flk have le: j<flj<fk by auto
        from p(2)[OF j] pivot(2)[OF nr(1) le(1)] pivot(2)[OF nr(2) le(2)]
        show ?A $$ (i,j)=0
        by (subst index-mat-swaprows(1), insert j d i p jj, auto)
    }
    qed
qed
lemma pivot-fun-eliminate-entries: assumes p: pivot-fun A fjj
    and d:dim-row }A=nr\operatorname{dim}-col A=n
    and fl: fl=jj
    and nr:l<nr
    and jj: jj \leqnc
shows pivot-fun (eliminate-entries vs A lj) fjj
proof -
    note pD = pivot-funD[OF d(1) p}
    {
        fix i j
        assume *: i<nr j<fi
        from pD(1)[OF this(1)] this(2) jj have j: j<nc by auto
        from pDnrfl*j have A$$(l,j)=0 by (meson less-le-trans)
        note j this
    } note hint = this
    show ?thesis by (rule pivot-funI, insert fl nr jj pD, auto simp: eliminate-entries-gen-def
d hint)
qed
definition row-echelon-form :: ' a :: {zero,one} mat => bool where
```

```
    row-echelon-form A \equiv\existsf.pivot-fun A f(dim-col A)
lemma pivot-fun-init: pivot-fun A (\lambda -. 0) 0
    by (rule pivot-funI, auto)
lemma gauss-jordan-main-row-echelon:
    assumes
    A \in carrier-mat nr nc
    gauss-jordan-main A B i j = ( }\mp@subsup{A}{}{\prime},\mp@subsup{B}{}{\prime}
    pivot-fun A fj
    \ i'. i'< i\Longrightarrowfi'<j\bigwedge i'. i'\geq i\Longrightarrowfi'=j
    i\leqnrj\leqnc
    shows row-echelon-form A'
proof -
    fix b
    interpret m: ring ring-mat TYPE('a) nr b by (rule ring-mat)
    show ?thesis
        using assms
    proof (induct A B i j arbitrary: f rule: gauss-jordan-main.induct)
        case (1 A Bijf)
        note }A=1(5
        hence dim: dim-row A = nr dim-col A = nc by auto
        note res = 1(6)
        note pivot = 1(7)
        note f=1(8-9)
        note ij = 1(10-11)
        note IH=1(1-4)[OF dim[symmetric]]
        note simp = gauss-jordan-main.simps[of A B i j] Let-def
        let ?g = gauss-jordan-main A B ij
        show ?case
        proof (cases i<nr ^j<nc)
        case False note nij = this
        with res have id: A' = A unfolding simp dim by auto
        have pivot-fun A fnc
        proof (cases j\geqnc)
            case True
            with ij have j:j=nc by auto
            with pivot show pivot-fun A f nc by simp
        next
            case False
            hence j: j<nc by simp
            from False nij ij have i:i=nr by auto
            note }f=f[\mathrm{ unfolded i]
            note p = pivot-funD[OF dim(1) pivot]
            show ?thesis
            proof (rule pivot-funI[OF dim(1)])
                    fix }
                    assume i:i<nr
                    note p=p[OFi
```

```
            from p(1) j show fi\leqnc by simp
            from f(1)[OF i] have fij: fi<j.
            from p(4)[OF fij] show A $$ (i,fi)=1.
            from p(5)[OF fij] show }\wedge\mp@subsup{i}{}{\prime}.\mp@subsup{i}{}{\prime}<nr\Longrightarrow\mp@subsup{i}{}{\prime}\not=i\LongrightarrowA$$(\mp@subsup{i}{}{\prime},fi)=0
            show }\j.j<fi\LongrightarrowA$$(i,j)=0 by (rule p(2))
            assume Suc i<nr
            with p(3)[OF this] f
            show fi<f(Suc i)\veef(Suc i)=nc by auto
        qed
        qed
        thus ?thesis using pivot unfolding id row-echelon-form-def dim by blast
    next
    case True note valid = this
    hence sij: Suc i\leqnr Suc j\leqnc by auto
    note IH = IH[OF valid refl]
    show ?thesis
    proof (cases A $$ (i,j)=0)
            case False note nZ = this
            note IH=IH(3-4)[OF nZ]
            show ?thesis
            proof (cases A $$ (i,j)=1)
            case False note nO= this
            let ?inv = inverse ( }A$$(i,j)
            let ?A = multrow i ?inv A
                    from nO nZ valid res have id: gauss-jordan-main (multrow i ?inv A)
(multrow i ?inv B) ij = ( }\mp@subsup{A}{}{\prime},\mp@subsup{B}{}{\prime}
            unfolding simp dim by simp
            have pivot-fun ?A fj
                    by (rule pivot-fun-multrow[OF pivot \operatorname{dim}f(2) ij(2)], auto)
            note IH =IH(2)[OF nO refl, unfolded multrow-carrier,OF A id this f ij]
            show ?thesis unfolding id using IH .
            next
                case True note O= this
            let ?E = \ B. eliminate-entries ( }\lambdai.A$$(i,j))Bi
            let ?A = ?E A
            let ?B=?E B
            define E where E=?A
            let ?f = \lambda i'. if i'}=i\mathrm{ then }j\mathrm{ else if }\mp@subsup{i}{}{\prime}>> i then Suc j else f i'
            have pivot: pivot-fun E fj unfolding E-def
                    by (rule pivot-fun-eliminate-entries[OF pivot \operatorname{dim}f(2)], insert valid,
auto)
            {
                fix }\mp@subsup{i}{}{\prime
                assume i': i'<nr
                have E$$(\mp@subsup{i}{}{\prime},j)=(if \mp@subsup{i}{}{\prime}=i then 1 else 0)
                    unfolding E-def eliminate-entries-gen-def using dim O i' valid by
auto
            } note Ej = this
            have E: E\in carrier-mat nr nc unfolding E-def by (rule carrier-eliminate-entries[OF
```

A])

```
hence dimE: dim-row E = nr dim-col E=nc by auto
note pivot = pivot-funD[OF dimE(1) pivot]
have pivot-fun E ?f (Suc j)
proof (rule pivot-funI[OF dimE(1)])
    fix ii
    assume ii: ii < nr
    note p = pivot[OF ii]
    show ?f ii \leqSuc j using p(1) by simp
    {
    fix jj
    assume jj: jj < ?f ii
    show E $$ (ii,jj) = 0
    proof (cases ii < i)
        case True
        with jj have jj<f ii by auto
        from p(2)[OF this] show ?thesis.
    next
        case False note ge = this
        with f have fiij: fii=j by simp
        show ?thesis
        proof (cases i<ii)
            case True
            with jj have jj: jj \leq j by auto
            show ?thesis
            proof (cases jj<j)
            case True
            with p(2)[of jj] fij show ?thesis by auto
            next
            case False
            with jj have jj: jj= j by auto
            with Ej[OF ii] True show ?thesis by auto
        qed
        next
            case False
            with ge have ii: ii=i by simp
            with jj have jj: jj< < by simp
            from p(2)[of jj] ii jj fij show ?thesis by auto
        qed
    qed
}
    assume Suc ii < nr
    from p(3)[OF this]f
    show ?f (Suc ii)> ?f ii \vee ?f (Suc ii)=Suc j by auto
}
{
    assume fii: ?f ii < Suc j
    show E $$(ii,?f ii) = 1
```

```
            proof (cases \(i i=i\) )
            case True
            with Ej[of i] valid show ?thesis by auto
            next
                    case False
                            with fii have \(i i: i i<i\) by (auto split: if-splits)
            from \(f(1)[O F\) this \(]\) have \(f i i<j\) by auto
            from \(p(4)[O F\) this \(] i i\) show ?thesis by simp
        qed
    \}
    \{
            fix \(i^{\prime}\)
            assume \(*\) : ?f \(i i<S u c j i^{\prime}<n r i^{\prime} \neq i i\)
            show \(E \$ \$\left(i^{\prime}\right.\), ?f \(\left.i i\right)=0\)
            proof (cases \(i i=i\) )
            case False
            with \(*(1)\) have \(i i i: i i<i\) by (auto split: if-splits)
            from \(f(1)[O F\) this \(]\) have \(f i i<j\) by auto
            from \(p(5)[O F\) this \(*(2-3)]\) show ?thesis using iii by simp
            next
            case True
            with \(*(2-3)\) Ej[of \(i]\) show ?thesis by auto
            qed
        \}
    qed
    note \(I H=I H(1)[O F O\) refl, folded E-def, OF E-this - sij]
    from \(O n Z\) valid res have gauss-jordan-main \(E\) ? B (Suc \(i)(\) Suc \(j)=\left(A^{\prime}\right.\),
        unfolding \(E\)-def simp dim by simp
        note \(I H=I H[O F\) this \(]\)
        show ?thesis
        proof (rule IH)
            fix \(i^{\prime}\)
            assume \(i^{\prime}<\) Suc \(i\)
            thus ?f \(i^{\prime}<\) Suc \(j\) using \(f[\) of \(i]\) by (cases \(i^{\prime}<i\), auto)
        qed auto
    qed
next
case True note \(Z=\) this
note \(I H=I H(1-2)[O F Z]\)
let ? is \(=\left[i^{\prime} . i^{\prime}<-[\right.\) Suc \(\left.i . .<n r], A \$ \$\left(i^{\prime}, j\right) \neq 0\right]\)
show ?thesis
proof (cases? is)
    case Nil
    \{
        fix \(i^{\prime}\)
        assume \(i \leq i^{\prime}\) and \(i^{\prime}<n r\)
        hence \(i^{\prime}=i \vee i^{\prime} \in\{\) Suc \(i . .<n r\}\) by auto
        from this arg-cong[OF Nil, of set] \(Z\) have \(A \$ \$\left(i^{\prime}, j\right)=0\) by auto
```

$\left.B^{\prime}\right)$

```
} note zero = this
let ?f = \lambda i'. if i'< i then f }\mp@subsup{i}{}{\prime}\mathrm{ else Suc j
note p= pivot-funD[OF \operatorname{dim}(1) pivot]
have pivot-fun A ?f (Suc j)
proof (rule pivot-funI[OF dim(1)])
    fix ii
    assume ii: ii<nr
    note p=p[OF this]
    show ?f ii \leqSuc j using p(1) by simp
    {
        fix jj
        assume jj: jj < ?f ii
        show A $$ (ii,jj)=0
        proof (cases ii < i)
            case True
            with jj have jj<f ii by auto
            from p(2)[OF this] show ?thesis.
        next
            case False
            with jj have }i\mp@subsup{i}{}{\prime}:ii\geqi\mathrm{ and jjj: jj 
            from zero[OF ii' ii] have Az:A $$ (ii,j)=0.
            show ?thesis
            proof (cases jj<j)
                case False
                with jjj have jj = j by auto
                with Az show ?thesis by simp
            next
                case True
            show ?thesis
                by (rule p(2), insert True False f, auto)
        qed
        qed
    }
    {
        assume sii: Suc ii < nr
        show ?f ii < ?f (Suc ii)\vee ?f (Suc ii)=Suc j
            using p(3)[OF sii] f by auto
    }
    assume fi: ?f ii< Suc j
        thus A $$ (ii, ?f ii)=1
            using p(4)f by (cases ii < i, auto)
        fix }\mp@subsup{i}{}{\prime
        assume i'<nr i' }=i
        from p(5)[OF - this]f fi
        show A$$(i', ?f ii)=0
            by (cases ii < i, auto)
    }
qed
```

note $I H=I H(1)[O F$ Nil $A-$ this $-i j(1) \operatorname{sij}(2)]$
from $Z$ valid res have gauss-jordan-main $A B i(S u c j)=\left(A^{\prime}, B^{\prime}\right)$ unfolding simp dim Nil by simp
note $I H=I H[O F$ this]
show ?thesis
by (rule IH, insert f, force + )

## next

case (Cons $i^{\prime}$ iis)
from arg-cong[OF this, of set] have $i^{\prime}: i^{\prime} \geq$ Suc $i i^{\prime}<n r$ by auto
from $f[$ of $i] f\left[\right.$ of $\left.i^{\prime}\right] i^{\prime}$ have $f i j: f i=j f i^{\prime}=j$ by auto
let ? $A=$ swaprows $i i^{\prime} A$
let $? B=$ swaprows $i i^{\prime} B$
have pivot-fun ?A $f j$
by (rule pivot-fun-swaprows[OF pivot dim fij], insert $i^{\prime} i j$, auto)
note $I H=I H(2)[$ OF Cons, unfolded swaprows-carrier, OF A-this fij]
from $Z$ valid res have id: gauss-jordan-main ?A ?B $i j=\left(A^{\prime}, B^{\prime}\right)$
unfolding simp dim Cons by simp
note $I H=I H[O F$ this $]$
show ?thesis using $I H$.
qed
qed
qed
qed
qed
lemma gauss-jordan-row-echelon: assumes $A: A \in$ carrier-mat $n r n c$
and res: gauss-jordan $A B=\left(A^{\prime}, B^{\prime}\right)$
shows row-echelon-form $A^{\prime}$
by (rule gauss-jordan-main-row-echelon[OF A res[unfolded gauss-jordan-def] pivot-fun-init], auto)
lemma pivot-bound: assumes dim: dim-row $A=n r$
and pivot: pivot-fun A fn
shows $i+j<n r \Longrightarrow f(i+j)=n \vee f(i+j) \geq j+f i$
proof (induct $j$ )
case (Suc j)
hence IH: $f(i+j)=n \vee j+f i \leq f(i+j)$
and $l t: i+j<n r$ Suc $(i+j)<n r$ by auto
note $p=$ pivot-fun $D[$ OF dim pivot $]$
from $p(3)[$ OF $l t]$ IH $p(1)[$ OF $l t(2)]$ show ?case by auto
qed $\operatorname{simp}$
context
fixes zero :: ' $a$
and $A$ :: 'a mat
and $n r n c::$ nat
begin
function pivot-positions-main-gen $::$ nat $\Rightarrow$ nat $\Rightarrow$ (nat $\times$ nat) list where

```
pivot-positions-main-gen \(i j=(\)
    if \(i<n r\) then
        if \(j<n c\) then
            if \(A \$ \$(i, j)=\) zero then
                    pivot-positions-main-gen \(i\) (Suc j)
            else \((i, j)\) \# pivot-positions-main-gen (Suc i) (Suc j)
        else []
    else []) by pat-completeness auto
```

termination by (relation measures $[(\lambda(i, j)$. Suc $n r-i),(\lambda(i, j)$. Suc nc $-j)]$, auto)
declare pivot-positions-main-gen.simps[simp del]
end

## context

fixes $A$ :: ' $a$ :: semiring-1 mat
and $n r n c::$ nat

## begin

abbreviation pivot-positions-main $\equiv$ pivot-positions-main-gen (0 :: 'a) Anr nc
lemma pivot-positions-main: assumes $A: A \in$ carrier-mat nr nc
and pivot: pivot-fun $A f n c$
shows $j \leq f i \vee i \geq n r \Longrightarrow$
set (pivot-positions-main $i j)=\left\{\left(i^{\prime}, f i^{\prime}\right) \mid i^{\prime} . i \leq i^{\prime} \wedge i^{\prime}<n r\right\}-U N I V \times$ $\{n c\}$
$\wedge$ distinct (map snd (pivot-positions-main ij))
$\wedge$ distinct (map fst (pivot-positions-main ij))
proof (induct ij rule: pivot-positions-main-gen.induct[of nr nc A 0 ])
case ( $1 i j$ )
let $? a=A \$ \$(i, j)$
let ?pivot $=\lambda i j$. pivot-positions-main $i j$
let ?set $=\lambda i .\left\{\left(i^{\prime}, f i^{\prime}\right) \mid i^{\prime} . i \leq i^{\prime} \wedge i^{\prime}<n r\right\}$
let ?s $=$ ? set $i$
let ?set $=\lambda i .\left\{\left(i^{\prime}, f i^{\prime}\right) \mid i^{\prime} . i \leq i^{\prime} \wedge i^{\prime}<n r\right\}$
let ?s $=$ ? set $i$
let ?p $=$ ? pivot $i j$
from $A$ have $d A$ : dim-row $A=n r$ by $\operatorname{simp}$
note $[$ simp $]=$ pivot-positions-main-gen.simps[of 0 A nr nc $i j$ ]
show ?case
proof (cases $i<n r$ )
case True note $i=$ this
note $I H=1(1-2)[$ OF True $]$
have $j f: j \leq f i$ using $1(3) i$ by auto
note pivotB $=$ pivot-bound $[O F d A$ pivot $]$
note pivot $^{\prime}=$ pivot-fun $D[$ OF dA pivot $]$
note pivot $=$ pivot $^{\prime}[$ OF True $]$
have id1: $[i . .<n r]=i \#[$ Suc $i . .<n r]$ using $i$ by (rule upt-conv-Cons)

```
    show ?thesis
    proof (cases j<nc)
        case True note j= this
        note IH=IH(1-2)[OF True]
        show ?thesis
        proof (cases ?a=0)
            case True note a= this
        from i j a have p: ?p = ?pivot i (Suc j) by simp
        {
            assume fi=j
            with pivot(4) j have ?a=1 by simp
            with a have False by simp
        }
        with jfi have Suc j\leqfi\veei\geqnr by fastforce
        note IH=IH(1)[OF True this]
        thus ?thesis unfolding p.
        next
            case False note a=this
            from i j a have p: ?p = (i,j) # ?pivot (Suc i) (Suc j) by simp
            from pivot(2)[of j] jfi a have jfi: j=fi}\mathrm{ by force
            from pivotB[of i Suc 0] jfi have Suc j\leqf(Suc i)\vee nr \leqSuc i
                using Suc-le-eq j leI by auto
            note IH = IH(2)[OF False this]
            {
                fix }\mp@subsup{i}{}{\prime
                assume *: fi=fi'Suc i\leq i' i'< nr
                hence }i+(\mp@subsup{i}{}{\prime}-i)=\mp@subsup{i}{}{\prime}\mathrm{ by auto
                from pivotB[of i i' - i, unfolded this] * jf j have False by auto
            } note distinct = this
            have id2: ?s = insert (i,j) (?set (Suc i)) using i jfinot-less-eq-eq
                by fastforce
            show ?thesis using IH j jfi i unfolding p id1 id2 by (auto intro: distinct)
        qed
    next
        case False note j= this
        from pivot(1) j jf have *: fi=nc nc=j by auto
        from ij have p: ?p = [] by simp
        from pivotB[of i Suc 0]* have j\leqf(Suc i)\veenr\leqSuc i by auto
        {
            fix }\mp@subsup{i}{}{\prime
            assume **: i\leq i' i' < nr
            hence }i+(\mp@subsup{i}{}{\prime}-i)=\mp@subsup{i}{}{\prime}\mathrm{ by auto
            from pivotB[of i i' - i, unfolded this] *** have nc \leqf fi' by auto
            with pivot'(1)[OF< <i'<nr`] have f i'}=nc\mathrm{ by auto
        }
            thus ?thesis using IH unfolding p id1 by auto
        qed
    qed auto
qed
```


## end

lemma pivot-fun-zero-row-iff: assumes pivot: pivot-fun ( $A$ :: ' ${ }^{\prime}$ :: semiring-1 mat) $f n c$
and $A: A \in$ carrier-mat nr nc
and $i: i<n r$
shows $f i=n c \longleftrightarrow$ row $A i=O_{v} n c$
proof -
from $A$ have dim: dim-row $A=n r$ by auto
note pivot $=$ pivot-fun $D[O F$ dim pivot $i]$
\{
assume $f i=n c$
from pivot(2)[unfolded this]
have row $A i=O_{v} n c$
by (intro eq-vecI, insert $A$, auto simp: row-def)
\}
moreover
\{
assume row: row $A i=O_{v} n c$
assume $f i \neq n c$
with $\operatorname{pivot}(1)$ have $f i<n c$ by auto
with pivot(4)[OF this] i A arg-cong[OF row, of $\lambda v . v \$ f i]$ have False by auto
\}
ultimately show ?thesis by auto
qed
definition pivot-positions-gen $::$ ' $a \Rightarrow$ 'a mat $\Rightarrow(n a t \times n a t)$ list where
pivot-positions-gen zer $A \equiv$ pivot-positions-main-gen zer $A$ (dim-row $A$ ) (dim-col A) 00
abbreviation pivot-positions :: 'a :: semiring-1 mat $\Rightarrow$ (nat $\times$ nat) list where pivot-positions $\equiv$ pivot-positions-gen 0
lemmas pivot-positions-def $=$ pivot-positions-gen-def
lemma pivot-positions: assumes $A: A \in$ carrier-mat $n r n c$
and pivot: pivot-fun $A f n c$
shows
set $($ pivot-positions $A)=\{(i, f i) \mid i . i<n r \wedge f i \neq n c\}$
distinct (map fst (pivot-positions A))
distinct (map snd (pivot-positions A))
length $($ pivot-positions $A)=\operatorname{card}\left\{i . i<n r \wedge\right.$ row $\left.A i \neq 0_{v} n c\right\}$
proof -
from $A$ have $\operatorname{dim}$ : dim-row $A=n r$ by auto
let $? p p=$ pivot-positions $A$
show id: set ? $p p=\{(i, f i) \mid i . i<n r \wedge f i \neq n c\}$
and dist: distinct (map fst ?pp)
and distinct (map snd ?pp)

```
    using pivot-positions-main[OF A pivot, of 0 0] A
    unfolding pivot-positions-def by auto
    have length ?pp = length (map fst ?pp) by simp
    also have ... = card (fst'set ?pp) using distinct-card[OF dist] by simp
    also have fst'set ? pp ={ i.i<nr\wedgefi\not=nc} unfolding id by force
    also have ...={i.i<nr ^ row A i\not= 0v nc}
    using pivot-fun-zero-row-iff[OF pivot A] by auto
    finally show length ?pp = card {i. i<nr ^ row A i\not= Ov nc} .
qed
context
    fixes uminus :: ' }a>>'\mp@code{'
    and zero :: 'a
    and one :: 'a
begin
definition non-pivot-base-gen :: 'a mat }=>\mathrm{ (nat }\timesnat)list => nat => 'a vec where
    non-pivot-base-gen A pivots \equivlet nr = dim-row A;nc=dim-col A;
    invers = map-of (map prod.swap pivots)
    in (\lambda qj. vec nc ( }\lambdai\mathrm{ .
    if i=qj then one else (case invers i of Some j=> uminus (A$$(j,qj))| None
=> zero)))
definition find-base-vectors-gen :: 'a mat => 'a vec list where
    find-base-vectors-gen }A
        let
            pp = pivot-positions-gen zero A;
            cands = filter ( }\lambdaj.j\not\in\operatorname{set}(\mathrm{ map snd pp)) [0 ..< dim-col A]
        in map (non-pivot-base-gen A pp) cands
end
abbreviation non-pivot-base \(\equiv\) non-pivot-base-gen uminus 0 (1 :: 'a :: comm-ring-1)
abbreviation find-base-vectors \equiv find-base-vectors-gen uminus 0 (1 :: 'a :: comm-ring-1)
lemmas non-pivot-base-def \(=\) non-pivot-base-gen-def
lemmas find-base-vectors-def \(=\) find-base-vectors-gen-def
The soundness of find-base-vectors is proven in theory Matrix-Kern, where it is shown that find-base-vectors is a basis of the kern of \(A\).
```

```
definition find-base-vector :: 'a :: comm-ring-1 mat \(\Rightarrow\) 'a vec where
```

definition find-base-vector :: 'a :: comm-ring-1 mat $\Rightarrow$ 'a vec where
find-base-vector $A \equiv$
find-base-vector $A \equiv$
let
let
pp $=$ pivot-positions $A$;
pp $=$ pivot-positions $A$;
cands $=$ filter $(\lambda j . j \notin \operatorname{set}($ map snd $p p))[0 . .<\operatorname{dim}-c o l A]$
cands $=$ filter $(\lambda j . j \notin \operatorname{set}($ map snd $p p))[0 . .<\operatorname{dim}-c o l A]$
in non-pivot-base A pp (hd cands)
in non-pivot-base A pp (hd cands)
context
context
fixes $A$ :: ' $a$ :: field mat and $n r n c::$ nat and $p::$ nat $\Rightarrow$ nat
fixes $A$ :: ' $a$ :: field mat and $n r n c::$ nat and $p::$ nat $\Rightarrow$ nat
assumes ref: row-echelon-form $A$
assumes ref: row-echelon-form $A$
and $A: A \in$ carrier-mat $n r n c$

```
    and \(A: A \in\) carrier-mat \(n r n c\)
```


## begin

lemma non-pivot-base:
defines $p p: p p \equiv$ pivot-positions $A$
assumes $q j: q j<n c q j \notin$ snd ' set $p p$
shows non-pivot-base A pp qj $\in$ carrier-vec nc
non-pivot-base $A$ pp qj $\$ q j=1$
$A *_{v}$ non-pivot-base A pp $q j=0_{v} n r$
$\wedge q j^{\prime} \cdot q j^{\prime}<n c \Longrightarrow q j^{\prime} \notin$ snd'set $p p \Longrightarrow q j \neq q j^{\prime} \Longrightarrow$ non-pivot-base $A p p q j$
$\$ q j^{\prime}=0$
proof -
from $A$ have dim: dim-row $A=n r \operatorname{dim}-c o l ~ A=n c$ by auto
from ref[unfolded row-echelon-form-def] obtain $p$
where pivot: pivot-fun $A p n c$ using $\operatorname{dim}$ by auto
note pivot $^{\prime}=$ pivot-fun $D[O F \operatorname{dim}(1)$ pivot $]$
note $p p=$ pivot-positions[OF A pivot, folded pp]
let $? p=\lambda i . i<n r \wedge p i=n c \vee i=n r$
let ? $s p p=$ map prod.swap $p p$
let ? map $=$ map-of ? spp
define $I$ where $I=(\lambda i$. case map-of (map prod.swap pp) $i$ of Some $j \Rightarrow-A$
$\$ \$(j, q j) \mid N o n e \Rightarrow 0)$
have $d$ : non-pivot-base $A$ pp $q j=v e c n c(\lambda i$. if $i=q j$ then 1 else $I i)$
unfolding non-pivot-base-def Let-def $\operatorname{dim}$ I-def ..
from $p p$ have dist: distinct (map fst ?spp)
unfolding map-map o-def prod.swap-def by auto
let $? r=s e t(m a p$ snd $p p)$
have $r$ : ? $r=p$ ' $\{0 . .<n r\}-\{n c\}$ unfolding set-map $p p$ by force
let $? l=\operatorname{set}($ map fst $p p)$
from $q j$ have $q j^{\prime}: q j \notin p$ ' $\{0 \quad . .<n r\}$ using $r$ by auto
let $? v=$ non-pivot-base $A p p q j$
let ? $P=p$ ' $\{0 . .<n r\}$
have dimv: dim-vec ? $v=n c$ unfolding $d$ by simp
thus ? $v \in$ carrier-vec nc unfolding carrier-vec-def by auto
show $v q j$ : ? $v \$ q j=1$ unfolding $d$ using $q j$ by auto
\{
fix $q j^{\prime}$
assume $*: q j^{\prime}<n c q j \neq q j^{\prime} q j^{\prime} \notin$ snd ' set $p p$
hence ?map $q j^{\prime}=$ None unfolding map-of-eq-None-iff by force
hence $I q j^{\prime}=0$ unfolding $I$-def by simp
with $*$ show non-pivot-base $A$ pp qj $\$ q j^{\prime}=0$
unfolding $d$ by simp
\}
\{
fix $i$
assume $i: i<n r$
let ? $I=\{j$. ? map $j=$ Some $i\}$
have row $A$ i ? $v=0$
proof -
have $i d:(\{0 . .<n c\} \cap ? P) \cup(\{0 . .<n c\}-? P)=\{0 . .<n c\}$ by auto

```
let ?e = \ j. row A i$j*?v$j
```

let ? $e^{\prime}=\lambda j$. (if ? map $j=$ Some $i$ then $-A \$ \$(i, q j)$ else 0$)$
\{
fix $j$
assume $j: j<n c j \in ? P$
then obtain $i i$ where $i i: i i<n r$ and $j p i: j=p i i$ and $p i i: p i i<n c$ by
auto
hence mem: $(i i, j) \in$ set $p p$ and $(j, i i) \in$ set ?spp by (auto simp: pp)
from map-of-is-SomeI[OF dist this(2)]
have map: ?map $j=$ Some ii by auto
from mem $j q j$ have $j q j: j \neq q j$ by force
note $p=\operatorname{pivot}^{\prime}(4-5)[$ OF ii pii]
define start where start $=? e j$
have start $=A \$ \$(i, j) * ? v \$ j$ using $j i A$ by (auto simp: start-def)
also have $A \$ \$(i, j)=A \$ \$(i, p i i)$ unfolding $j p i$..
also have $\ldots=($ if $i=$ ii then 1 else 0$)$ using $p(1) p(2)[$ OF $i]$ by auto
also have $\ldots *$ ?v $\$ j=($ if $i=i i$ then ?v $\$ j$ else 0$)$ by simp
also have ?v $\$ j=I j$ unfolding $d$
using $j$ jqj $A$ by auto
also have $I j=-A \$ \$(i i, q j)$ unfolding $I$-def map by simp
finally have ? $e j=? e^{\prime} j$
unfolding start-def map by auto
$\}$ note piv $=$ this
have row $A$ i $\cdot$ ?v $=\left(\sum j=0 . .<n c . ? e j\right)$ unfolding row-def scalar-prod-def dimv ..

$$
\text { also have } \ldots=\operatorname{sum} ? e(\{0 . .<n c\} \cap ? P)+\operatorname{sum} ? e(\{0 . .<n c\}-? P)
$$

by (subst sum.union-disjoint[symmetric], auto simp: id)
also have sum ?e $(\{0 . .<n c\}-? P)=? e q j+\operatorname{sum} ? e(\{0 . .<n c\}-? P-$ $\{q j\})$
by (rule sum.remove, insert qj $q j^{\prime}$, auto)
also have ?e $q j=$ row $A i \$ q j$ unfolding $v q j$ by simp
also have row $A i \$ q j=A \$ \$(i, q j)$ using $i A q j$ by auto
also have sum ? e $(\{0 . .<n c\}-? P-\{q j\})=0$
proof (rule sum.neutral, intro ballI)
fix $j$
assume $j \in\{0 \quad . .<n c\}-? P-\{q j\}$
hence $j: j<n c j \notin ? P j \neq q j j \notin ? r$ unfolding $r$ by auto
hence id: map-of ? spp $j=$ None unfolding map-of-eq-None-iff by force
have ? $v \$ j=I j$ unfolding $d$ using $j$ by simp
also have $\ldots=0$ unfolding $I$-def id by simp
finally show row $A i \$ j * ? v \$ j=0$ by simp
qed
also have $A \$ \$(i, q j)+0=A \$ \$(i, q j)$ by $\operatorname{simp}$
also have sum ?e $(\{0 . .<n c\} \cap ? P)=\operatorname{sum} ? e^{\prime}(\{0 . .<n c\} \cap ? P)$
by (rule sum.cong, insert piv, auto)
also have $\{0 . .<n c\} \cap ? P=\{0 . .<n c\} \cap ? I \cap ? P \cup(\{0 . .<n c\}-? I) \cap ? P$ by auto
also have sum? $e^{\prime}(\{0 . .<n c\} \cap ? I \cap ? P \cup(\{0 . .<n c\}-? I) \cap ? P)$
$=\operatorname{sum} ? e^{\prime}(\{0 . .<n c\} \cap ? I \cap ? P)+\operatorname{sum} ? e^{\prime}((\{0 . .<n c\}-? I) \cap ? P)$

```
        by (rule sum.union-disjoint, auto)
    also have sum? ? ( }(({0..<nc}-?I)\cap?P)=
        by (rule sum.neutral, auto)
    also have sum? ? e' ({0..<nc} \cap?I \cap?P) =
        sum (\lambda -. - A $$ (i,qj)) ({0..<nc} \cap?I \cap?P)
        by (rule sum.cong, auto)
    also have \ldots.+0=\ldots. by simp
    also have sum (\lambda -. - A$$ (i,qj)) ({0..<nc}\cap?I\cap?P)+A$$ (i,qj)=
0
    proof (cases i }\in\mathrm{ ?l)
        case False
        with pp(1) i have pi=nc by force
        from pivot'(2)[OF i, unfolded this, OF qj(1)] have z:A $$ (i,qj) = 0.
        show ?thesis
            by (subst sum.neutral, auto simp: z)
    next
    case True
    then obtain j where mem: (i,j)\in set pp and id: (j,i)\in set ?spp by auto
    from map-of-is-SomeI[OF dist this(2)] have map: ?map j=Some i.
    from pivot'(1)[OF i] have pi: pi\leqnc.
    with mem[unfolded pp] have j:j=pij<nc by auto
    {
            fix j'
            assume j'\in?I
            hence ?map j' = Some i by auto
            from map-of-SomeD[OF this] have (i, j')\in set pp by auto
            with mem pp(2) have j'=j using map-of-is-SomeI by fastforce
            }
            with map have II:?I = {j} by blast
            have II:{0..<nc}\cap?I\cap?P={j} unfolding II using mem[unfolded pp]
ij by auto
            show ?thesis unfolding II by simp
            qed
            finally show row A i \cdot ?v=0 .
    qed
} note main = this
show }A\mp@subsup{*}{v}{}\mathrm{ ?v = 0 v nr
    by (rule eq-vecI, auto simp: dim main)
qed
lemma find-base-vector: assumes snd'set (pivot-positions A)}\not={0..<nc
    shows
    find-base-vector A \in carrier-vec nc
    find-base-vector A = 0vv nc
    A *v find-base-vector }A=\mp@subsup{0}{v}{}n
proof -
    define cands where cands = filter ( }\lambdaj.j\not\in\mathrm{ snd'set (pivot-positions A)) [0 ..<
nc]
    from A have dim: dim-row A = nr dim-col A = nc by auto
```

from ref[unfolded row-echelon-form-def] obtain $p$ where pivot: pivot-fun $A p n c$ using $\operatorname{dim}$ by auto note piv $=$ pivot-fun $D[O F \operatorname{dim}(1)$ pivot $]$
have set cands $\neq\{ \}$ using assms piv unfolding cands-def pivot-positions[OF A pivot]
by (auto simp: le-neq-implies-less)
then obtain $c$ cs where cands: cands $=c \# c s$ by (cases cands, auto)
hence res: find-base-vector $A=$ non-pivot-base $A($ pivot-positions $A) c$
unfolding find-base-vector-def Let-def cands-def dim by auto
from cands have $c \in$ set cands by auto
hence $c: c<n c c \notin$ snd' set (pivot-positions A)
unfolding cands-def by auto
from non-pivot-base[OF this, folded res] c show
find-base-vector $A \in$ carrier-vec nc
find-base-vector $A \neq 0_{v} n c$
$A *_{v}$ find-base-vector $A=0_{v} n r$
by auto
qed
end
lemma row-echelon-form-imp-1-or-0-row: assumes $A$ : $A \in$ carrier-mat $n n$ and row: row-echelon-form $A$
shows $A=1_{m} n \vee\left(n>0 \wedge\right.$ row $\left.A(n-1)=0_{v} n\right)$
proof -
from $A$ have dim: dim-row $A=n \operatorname{dim}-\operatorname{col} A=n$ by auto
from row[unfolded row-echelon-form-def] $A$
obtain $f$ where pivot: pivot-fun $A f n$ by auto
note $p=$ pivot-funD $[O F \operatorname{dim}(1)$ this]
show ?thesis
proof (cases $\exists i<n . f i \neq i$ )
case True
then obtain $i$ where $i: i<n$ and $f: f i \neq i$ by auto
note $p b=$ pivot-bound $[O F \operatorname{dim}(1)$ pivot $]$
from $p b[$ of $0 i] i$ have $f i=n \vee i \leq f i$ by auto
with $f i$ have $f: f i=n \vee i<f i$ by auto
from $i$ have $n$ : $n-1=i+(n-i-1)$ by auto
from $p b[$ of $i n-i-1$, folded n] fi $i p(1)[$ of $n-1]$
have $f n: f(n-1)=n$ by auto
from $i$ have $n 0: n>0$ and $n 1: n-1<n$ by auto
from $p(2)[O F n 1$, unfolded fn] have zero: $\wedge j . j<n \Longrightarrow A \$ \$(n-1, j)=$
0 by auto
show ?thesis
by (rule disjI2[OF conjI[OF n0]], rule eq-vecI, insert zero $A$, auto)
next
case False
\{
fix $j$
assume $j: j<n$
with False have id: $f j=j$ by auto

```
        note pj=p[OF j, unfolded id]
        from pj(5)[OF j] pj(4)[OF j]
        have }\i.i<n\LongrightarrowA$$(i,j)=(\mathrm{ if }i=j\mathrm{ then 1 else 0) by auto
    } note id = this
    show ?thesis
        by (rule disjI1, rule eq-matI, subst id, insert A, auto)
        qed
qed
context
    fixes A :: ' }a\mathrm{ :: field mat and n :: nat and p :: nat }=>\mathrm{ nat
    assumes ref:row-echelon-form A
    and A:A\incarrier-mat n n
    and 1:A\not=1m}
begin
lemma find-base-vector-not-1-pivot-positions: snd' set (pivot-positions A) }={
..<n}
proof
    let ?pp = pivot-positions A
    assume id: snd' set ?pp = {0 ..< n}
    from }A\mathrm{ have dim: dim-row }A=n\operatorname{dim}-\operatorname{col}A=n\mathrm{ by auto
    let ? }n=n-
    from row-echelon-form-imp-1-or-0-row[OF A ref] 1
    have *: 0<n and row: row }A\mathrm{ ? n = O n n by auto
    from ref[unfolded row-echelon-form-def] obtain p
            where pivot: pivot-fun A p n using dim by auto
    note pp= pivot-positions[OF A pivot]
    note piv = pivot-funD[OF dim(1) pivot]
    from * have n: ?n<n by auto
    {
            assume p ?n<n
            with piv(4)[OF n this] row n A have False
            by (metis dim index-row(1) index-zero-vec(1) zero-neq-one)
    }
    with piv(1)[OF n] have pn: p ?n = n by fastforce
    hence ? n }\not\infst'set ? pp unfolding pp by aut
    hence fst' set ?pp\subseteq{0..< n}-{?n} unfolding pp by force
    also have ...\subseteq{0..<n-1} by auto
    finally have card (fst'set ?pp) \leqcard {0..<n-1} using card-mono by blast
    also have ... =n-1 by auto
    also have card (fst ' set ?pp) = card (snd ' set ?pp)
            unfolding set-map[symmetric] distinct-card[OF pp(2)] distinct-card[OF pp(3)]
by simp
    also have ... = n unfolding id by simp
    finally show False using n by simp
qed
```

```
lemma find-base-vector-not-1:
    find-base-vector A \in carrier-vec n
    find-base-vector }A\not=
    A *v find-base-vector }A=0=0
    using find-base-vector[OF ref A find-base-vector-not-1-pivot-positions].
end
lemma gauss-jordan: assumes A:A\incarrier-mat nr nc
    and B:B\in carrier-mat nr nc2
    and gauss: gauss-jordan A B = (C,D)
    shows }x\in\mathrm{ carrier-vec nc # (A*v x = 0 v nr) = (C *v x = 0 v nr) (is - "
?l = ?r)
    X\in carrier-mat nc nc2 \Longrightarrow (A*X=B)=(C*X=D)(is - \Longrightarrow?12 =
?r2)
    C \in carrier-mat nr nc
    D\in carrier-mat nr nc*
proof -
    from gauss-jordan-transform[OF A B gauss, unfolded ring-mat-def Units-def,
simplified]
    obtain P Q where P:P\incarrier-mat nr nr and Q:Q\incarrier-mat nr nr
        and inv: Q*P=1m nr
        and CPA:C = P*A
        and DPB: D=P*B by auto
    from CPA P A show }C:C\in\mathrm{ carrier-mat nr nc by auto
    from DPB P B show D: D\incarrier-mat nr nc2 by auto
    have}A=\mp@subsup{1}{m}{}nr*A\mathrm{ using }A\mathrm{ by simp
    also have \ldots=Q*C unfolding inv[symmetric] CPA using Q P A by simp
    finally have AQC:A=Q*C.
    have }B=\mp@subsup{1}{m}{}nr*B\mathrm{ using }B\mathrm{ by simp
    also have \ldots=Q*D unfolding inv[symmetric] DPB using Q P B by simp
    finally have BQD: B=Q*D .
    {
    assume x: x \in carrier-vec nc
    {
            assume ?l
            from arg-cong[OF this, of \lambdav. P**v] PA x have ?r unfolding CPA by
auto
    }
        moreover
        {
            assume ?r
            from arg-cong[OF this, of \lambda v. Q*vv] Q C x have ?l unfolding AQC by
auto
    }
    ultimately show ?l = ?r by auto
    }
    {
        assume X:X \in carrier-mat nc nc2
        {
```

```
        assume ?12
            from arg-cong[OF this, of \lambda X.P*X]PAX have ?r2 unfolding CPA
DPB by simp
    }
    moreover
    {
        assume ?r2
            from arg-cong[OF this, of \lambdaX.Q*X] Q C X have ?l2 unfolding AQC
BQD by simp
    }
    ultimately show ?l2 = ?r2 by auto
    }
qed
definition gauss-jordan-single :: 'a :: field mat => 'a mat where
    gauss-jordan-single A = fst (gauss-jordan A (0m (dim-row A) 0))
lemma gauss-jordan-single: assumes A:A carrier-mat nr nc
    and gauss: gauss-jordan-single }A=
    shows }x\in\mathrm{ carrier-vec nc \(A*v x = Ov nr ) = (C*vv 
            C\in carrier-mat nr nc
            row-echelon-form C
                            \exists PQ.C = P*A\wedgeP\incarrier-mat nr nr ^Q Q carrier-mat nr nr ^ P*
Q=1m}nr\wedgeQ*P=1mnr (is ?ex)
proof -
    from A gauss[unfolded gauss-jordan-single-def] obtain D where gauss: gauss-jordan
A (0m nr 0) = (C,D)
            by (cases gauss-jordan A (0m nr 0), auto)
    from gauss-jordan[OF A zero-carrier-mat gauss] gauss-jordan-row-echelon[OF A
gauss]
    gauss-jordan-transform[OF A zero-carrier-mat gauss, of ()]
    show }x\in\mathrm{ carrier-vec nc # (A*v x = Ov nr) = (C *v x= Ov nr)
    C\in carrier-mat nr nc row-echelon-form C ?ex unfolding Units-def ring-mat-def
by auto
qed
lemma gauss-jordan-inverse-one-direction:
assumes \(A: A \in\) carrier-mat \(n n\) and \(B: B \in \operatorname{carrier-mat} n n c\) and res: gauss-jordan \(A B=\left(1_{m} n, B^{\prime}\right)\)
    shows A U Units (ring-mat TYPE(' }a:: field) n b
    B=1m n\LongrightarrowA* B'=1m n^ 列*A=1m}
proof -
    let ?R = ring-mat TYPE('a) n b
    let ?U = Units ?R
    interpret m: ring ?R by (rule ring-mat)
    from gauss-jordan-transform[OF A B res, of b]
    obtain P where P:P\in?U and id: P*A=1m n and B': B'}=P*B\mathrm{ by
```

```
auto
    from P have Pc: P carrier-mat n n unfolding Units-def ring-mat-def by
auto
    from m.Units-one-side-I(1)[of A P] A P id show Au: A \in?U unfolding
ring-mat-def by auto
    assume B: B=1 m}
    from }\mp@subsup{B}{}{\prime}[\mathrm{ unfolded this] Pc have }\mp@subsup{B}{}{\prime}:\mp@subsup{B}{}{\prime}=P\mathrm{ by auto
    show }A*\mp@subsup{B}{}{\prime}=\mp@subsup{1}{m}{}n\wedge\mp@subsup{B}{}{\prime}*A=\mp@subsup{1}{m}{}n\mathrm{ unfolding B'
    using m.Units-inv-comm[OF - P Au] id by (auto simp: ring-mat-def)
qed
lemma gauss-jordan-inverse-other-direction:
    assumes AU:A\inUnits (ring-mat TYPE(' }a:: field) nb) and B:B\incarrier-mat
n nc
    shows fst (gauss-jordan A B)=1m n
proof -
    let ?R = ring-mat TYPE('a) n b
    let ?U = Units ?R
    interpret m: ring?R by (rule ring-mat)
    from AU have A:A \in carrier-mat n n unfolding Units-def ring-mat-def by
auto
    obtain }\mp@subsup{A}{}{\prime}\mp@subsup{B}{}{\prime}\mathrm{ where res: gauss-jordan A B = ( }\mp@subsup{A}{}{\prime},\mp@subsup{B}{}{\prime})\mathrm{ by force
    from gauss-jordan-transform[OF A B res, of b]
    obtain P where P:P\in?U and id: A' = P*A by auto
    from m.Units-m-closed[OF P AU] have }\mp@subsup{A}{}{\prime}:\mp@subsup{A}{}{\prime}\in?U\mathrm{ unfolding id ring-mat-def
by auto
    hence }\mp@subsup{A}{}{\prime}c:\mp@subsup{A}{}{\prime}\in\mathrm{ carrier-mat n n unfolding Units-def ring-mat-def by auto
    from }\mp@subsup{A}{}{\prime}[\mathrm{ unfolded Units-def ring-mat-def] obtain IA' where IA':IA'}\in\mathrm{ car-
rier-mat n n
            and IA: A'}*I\mp@subsup{A}{}{\prime}=1m n by aut
    from row-echelon-form-imp-1-or-0-row[OF gauss-jordan-carrier(1)[OF A B res]
gauss-jordan-row-echelon[OF A res]]
    have choice: }\mp@subsup{A}{}{\prime}=\mp@subsup{1}{m}{}n\vee0<n\wedge row A' (n-1)= 0 v n. 
    hence }\mp@subsup{A}{}{\prime}=\mp@subsup{1}{m}{\prime}
    proof
        let ? n = n-1
        assume 0<n^ row A' ? n = Ov n
        hence n:?n}<n\mathrm{ and row: row A' ?n = 0 v n by auto
        have 1= 1m n$$ (?n,?n) using n by simp
        also have 1m n= A' * IA' unfolding IA ..
        also have ( }\mp@subsup{A}{}{\prime}*I\mp@subsup{A}{}{\prime})$$(?n,?n)=row A'?n \cdot col IA' ?n
            using nI\mp@subsup{A}{}{\prime}}\mp@subsup{A}{}{\prime}c\mathrm{ by simp
        also have row A' ?n = Ov n unfolding row ..
        also have }\mp@subsup{0}{v}{}n\cdot\operatorname{col IA' ?n = 0 using IA' n by simp
        finally have 1 = (0:: 'a) by simp
        thus ?thesis by simp
    qed
    with res show ?thesis by simp
qed
```

```
lemma gauss-jordan-compute-inverse:
    assumes \(A: A \in\) carrier-mat \(n n\)
    and res: gauss-jordan \(A\left(1_{m} n\right)=\left(1_{m} n, B^{\prime}\right)\)
    shows \(A * B^{\prime}=1_{m} n B^{\prime} * A=1_{m} n B^{\prime} \in\) carrier-mat \(n n\)
proof -
    from gauss-jordan-inverse-one-direction(2)[OF A - res refl, of \(n]\)
    show \(A * B^{\prime}=1_{m} n B^{\prime} * A=1_{m} n\) by auto
    from gauss-jordan-carrier(2)[OF \(A-\) res, of \(n]\) show \(B^{\prime} \in\) carrier-mat \(n n\) by
auto
qed
lemma gauss-jordan-check-invertable: assumes \(A: A \in \operatorname{carrier-mat} n n\) and \(B\) :
\(B \in\) carrier-mat \(n n c\)
    shows \(\left(A \in\right.\) Units (ring-mat TYPE \(\left({ }^{\prime} a::\right.\) field) \(\left.\left.n b\right)\right) \longleftrightarrow f s t(\) gauss-jordan \(A B)\)
\(=1 m n\)
    (is \(? l=? r\) )
proof
    assume ?l
    show ?r
        by (rule gauss-jordan-inverse-other-direction \([O F\langle ? l\rangle B])\)
next
    let \(? g=\) gauss-jordan \(A B\)
    assume ?r
    then obtain \(B^{\prime}\) where \(? g=\left(1_{m} n, B^{\prime}\right)\) by (cases \(? g\), auto \()\)
    from gauss-jordan-inverse-one-direction(1)[OF A B this]
    show ?l.
qed
definition mat-inverse \(::\) ' \(a\) :: field mat \(\Rightarrow{ }^{\prime} a\) mat option where
    mat-inverse \(A=\) (if dim-row \(A=\operatorname{dim}\)-col \(A\) then
        let one \(=1_{m}(\) dim-row \(A)\) in
        (case gauss-jordan A one of
            \((B, C) \Rightarrow\) if \(B=\) one then Some \(C\) else None) else None)
lemma mat-inverse: assumes \(A: A \in\) carrier-mat \(n n\)
    shows mat-inverse \(A=\) None \(\Longrightarrow A \notin\) Units (ring-mat TYPE('a \(::\) field) \(n b\) )
    mat-inverse \(A=\) Some \(B \Longrightarrow A * B=1_{m} n \wedge B * A=1_{m} n \wedge B \in\) carrier-mat
\(n\) n
proof -
    let ?one \(=1 m n\)
    obtain \(B B C\) where res: gauss-jordan \(A\) ?one \(=(B B, C)\) by force
    \{
        assume mat-inverse \(A=\) None
        with res have \(B B \neq\) ? one unfolding mat-inverse-def using \(A\) by auto
        thus \(A \notin\) Units (ring-mat TYPE ('a :: field) \(n b\) )
            using gauss-jordan-check-invertable[OF A, of ?one n] res by force
    \}
    \{
```

```
    assume mat-inverse A = Some B
    with res A have BB=?one C=B unfolding mat-inverse-def
    by (auto split: if-splits option.splits)
    from gauss-jordan-compute-inverse[OF A res[unfolded this]]
    show }A*B=1mn\wedgeB*A=1mn\wedgeB\incarrier-mat n n by aut
}
qed
end
```


## 7 Code Generation for Basic Matrix Operations

In this theory we provide efficient implementations for the elementary rowtransformations. These are necessary since the default implementations would construct a whole new matrix in every step.

```
theory Gauss-Jordan-IArray-Impl
imports
    Polynomial-Interpolation.Missing-Unsorted
    Matrix-IArray-Impl
    Gauss-Jordan-Elimination
begin
lift-definition mat-swaprows-impl :: nat }=>\mathrm{ nat }=>\mp@subsup{}{}{\prime}\mp@subsup{}{}{\prime}a\mathrm{ mat-impl }=>\mp@subsup{}{}{\prime
    \lambdaij (nr,nc,A). if i<nr\wedgej<nr then
    let Ai=IArray.sub A i;
        Aj = IArray.sub A j;
        Arows = IArray.list-of A;
        A'=IArray.IArray (Arows [i:=Aj, j:=Ai])
        in ( nr,nc, A')
        else (nr,nc,A)
    by (auto split: if-splits)
lemma [code]: mat-swaprows kl (mat-impl A) = (let nr = dim-row-impl A in
    if l<nr\wedgek<nr then
    mat-impl (mat-swaprows-impl k l A) else Code.abort (STR "index out of bounds
in mat-swaprows')
    (\lambda -. mat-swaprows kl (mat-impl A))) (is ?l = ?r)
proof (cases l<dim-row-impl A ^k<dim-row-impl A)
    case True
    hence id: ?r = mat-impl (mat-swaprows-impl kl A) by simp
    show ?thesis unfolding id unfolding mat-swaprows-def
    proof (rule eq-matI, goal-cases)
        case (1 i j)
        thus ?case using True
        proof (transfer, goal-cases)
            case (1iklAj)
            obtain nr nc rows where A:A=(nr,nc,rows) by (cases A,auto)
            from 1[unfolded A]
            have nr:length (IArray.list-of rows) = nr
```

and nc: IArray.all $(\lambda r$. length (IArray.list-of $r)=n c)$ rows
and $i j: i<n r j<n c$ and $i j^{\prime}:(i<n r \wedge j<n c)=$ True
and $l: l<n r k<n r$ by auto
show ?case unfolding A prod.simps fst-conv o-def snd-conv Let-def mk-mat-def $i j^{\prime}$ if-True
using ij nr nc $l$
by (cases $k=i$; cases $l=i$, auto)
qed
qed ((transfer, auto)+)
qed auto
lift-definition mat-multrow-gen-impl $::\left({ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a\right) \Rightarrow$ nat $\Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a$ mat-impl $\Rightarrow{ }^{\prime}$ a mat-impl is
$\lambda$ mul $k a(n r, n c, A)$. let $A k=$ IArray.sub $A k ;$ Arows $=$ IArray.list-of $A$;
$A k^{\prime}=$ IArray.IArray (map (mul a) (IArray.list-of Ak));
$A^{\prime}=\operatorname{IArray} \cdot I A r r a y\left(\right.$ Arows $\left.\left[k:=A k^{\prime}\right]\right)$
in ( $n r, n c, A^{\prime}$ )
proof (auto, goal-cases)
case (1 mul kanc b row)
show ? case
proof (cases b)
case (IArray rows)
with 1 have row $\in$ set rows $\vee k<$ length rows $\wedge$ row $=$ IArray ( $\operatorname{map}(\operatorname{mul} a)$
(IArray.list-of (rows!k)))
by (cases $k<$ length rows, auto simp: set-list-update dest: in-set-takeD
in-set-drop $D$ )
with 1 IArray show ?thesis by (cases, auto)
qed
qed
lemma [code]: mat-multrow-gen mul ka(mat-impl $A$ ) $=$ mat-impl (mat-multrow-gen-impl mul $k$ a A)
unfolding mat-multrow-gen-def
proof (rule eq-matI, goal-cases)
case ( $1 i j$ )
thus ?case
proof (transfer, goal-cases)
case ( 1 i mul k a $A$ j)
obtain $n r n c$ rows where $A$ : $A=(n r, n c$, rows $)$ by (cases $A$, auto)
from 1 [unfolded $A$ ]
have $n r$ : length (IArray.list-of rows) $=n r$
and nc: IArray.all ( $\lambda r$. length (IArray.list-of $r)=n c$ ) rows
and $i j: i<n r j<n c$ and $i j^{\prime}:(i<n r \wedge j<n c)=$ True by auto
have len: $j<$ length (IArray.list-of (IArray.list-of rows $!i)$ )
using ij nc nr by (cases rows, auto)
show ? case unfolding A prod.simps fst-conv o-def snd-conv Let-def mk-mat-def $i j^{\prime}$ if-True
using ij $n r n c$

```
    by (cases k = i, auto simp: len)
    qed
qed ((transfer, auto)+)
lift-definition mat-addrow-gen-impl
    ::('a=>'}a=>\mp@subsup{'}{}{\prime}a)=>('a=>\mp@subsup{'}{}{\prime}a=>\mp@subsup{'}{}{\prime}a)=>'a=>nat => nat => 'a mat-impl => 'a
mat-impl is
    \lambda ad mul a kl (nr,nc,A). if l<nr then let Ak=IArray.sub A k;Al=IArray.sub
A l;
    Ak' = IArray.of-fun (\lambda i. ad (mul a (Al !! i)) (Ak !! i)) (min (IArray.length
Ak) (IArray.length Al));
    A' = IArray.of-fun ( }\lambda\mathrm{ i. if }i=k\mathrm{ then Ak' else A !! i) (IArray.length A)
    in (nr,nc,A') else (nr,nc,A)
proof (goal-cases)
    case (1 ad mul a k l pp)
    obtain nr nc A where pp: pp=(nr,nc,A) by (cases pp)
    obtain rows where A: A = IArray rows by (cases A)
    from 1[unfolded pp A, simplified]
    have nr: length rows = nr and nc: \r.r\inset rows \Longrightarrowlength (IArray.list-of r)
= nc by auto
    show ?case
    proof (cases l < nr)
        case False
        thus ?thesis unfolding pp A prod.simps using nr nc by auto
    next
        case True
        thus ?thesis unfolding pp A prod.simps Let-def using nr nc
            by (auto simp: set-list-update dest: in-set-takeD in-set-dropD)
    qed
qed
lemma mat-addrow-gen-impl[code]: mat-addrow-gen ad mul a kl(mat-impl A)= (if \(l<\) dim-row-impl \(A\) then
    mat-impl (mat-addrow-gen-impl ad mul a k l A) else Code.abort (STR ''index out
of bounds in mat-addrow')
    (\lambda -. mat-addrow-gen ad mul a kl (mat-impl A))) (is ?l = ?r)
proof (cases l < dim-row-impl A)
    case True
    hence id: ?r = mat-impl (mat-addrow-gen-impl ad mul a kl A) by simp
    show ?thesis unfolding id unfolding mat-addrow-gen-def
    proof (rule eq-matI, goal-cases)
    case (1 i j)
    thus ?case using True
    proof (transfer, goal-cases)
            case (1 i ad mul a kl A j)
            obtain nr nc rows where A:A=(nr,nc,rows) by (cases A,auto)
            from 1[unfolded A Let-def]
            have nr: length (IArray.list-of rows) = nr
            and nc: IArray.all (\lambdar. length (IArray.list-of r) = nc) rows
```

and $i j: i<n r j<n c$ and $i j^{\prime}:(i<n r \wedge j<n c)=$ True
and $l: l<n r$ by auto
have len: $j<$ length (IArray.list-of (IArray.list-of rows ! i))
$j<$ length (IArray.list-of (IArray.list-of rows ! l))
using ij nc nr $l$ by (cases rows, auto) +
show ?case unfolding A prod.simps fst-conv o-def snd-conv Let-def mk-mat-def $i j^{\prime}$ if-True
using ijnr nc $l$
by (cases $k=i$, auto simp: len)
qed next
qed ((transfer, auto simp:Let-def $)+$ )
qed $\operatorname{simp}$
lemma gauss-jordan-main-code[code]:
gauss-jordan-main $A B i j=($ let $n r=\operatorname{dim}$-row $A ; n c=\operatorname{dim}$-col $A$ in if $i<n r \wedge j<n c$ then let aij $=A \$ \$(i, j)$ in if aij $=0$ then
(case $\left[i^{\prime} . i^{\prime}<-[\right.$ Suc $\left.i . .<n r], A \$ \$\left(i^{\prime}, j\right) \neq 0\right]$
of []$\Rightarrow$ gauss-jordan-main A B $i(S u c j)$
$\mid\left(i^{\prime} \#-\right) \Rightarrow$ gauss-jordan-main (swaprows $\left.i i^{\prime} A\right)\left(\right.$ swaprows $\left.i i^{\prime} B\right) i j$ )
else if aij $=1$ then let $v=(\lambda i . A \$ \$(i, j))$ in
gauss-jordan-main
(eliminate-entries v A ij) (eliminate-entries v Bij) (Suc i) (Suc j)
else let iaij $=$ inverse aij; $A^{\prime}=$ multrow $i$ iaij $A ; B^{\prime}=$ multrow $i$ iaij $B$;
$v=\left(\lambda i . A^{\prime} \$ \$(i, j)\right)$ in gauss-jordan-main
(eliminate-entries $\left.v A^{\prime} i j\right)$ (eliminate-entries v $B^{\prime} i j$ ) (Suc $i$ ) (Suc $j$ )
else $(A, B))($ is ?l $=? r)$
proof -
note simps = gauss-jordan-main.simps[of A B ij] Let-def
let $? n r=$ dim-row $A$
let $? n c=\operatorname{dim}-\operatorname{col} A$
let ? $A^{\prime}=$ multrow $i($ inverse $(A \$ \$(i, j))) A$
let $? B^{\prime}=$ multrow $i($ inverse $(A \$ \$(i, j))) B$
show ?thesis
proof (cases $i<? n r \wedge j<? n c \wedge A \$ \$(i, j) \neq 0 \wedge A \$ \$(i, j) \neq 1)$
case False
thus ?thesis unfolding simps by (auto split: if-splits)
next
case True
from True have $i d$ : ? $A^{\prime} \$ \$(i, j)=1$ by auto
from True have ?l = gauss-jordan-main ? $A^{\prime} ? B^{\prime}$ ij unfolding simps by (simp add: Let-def)
also have $\ldots=$ ? $r$ unfolding Let-def gauss-jordan-main.simps $\left[o f\right.$ ? $A^{\prime}$ ? $B^{\prime}$ i $j$ ] id
using True by simp
finally show ?thesis
qed
qed

## 8 Elementary Column Operations

We define elementary column operations and also combine them with elementary row operations. These combined operations are the basis to perform operations which preserve similarity of matrices. They are applied later on to convert upper triangular matrices into Jordan normal form.

## theory Column-Operations imports

Gauss-Jordan-Elimination
begin
definition mat-multcol :: nat $\Rightarrow{ }^{\prime} a$ :: semiring- $1 \Rightarrow{ }^{\prime} a$ mat $\Rightarrow{ }^{\prime} a$ mat (multcol) where
multcol $k$ a $A=$ mat (dim-row $A)($ dim-col $A)$ $(\lambda(i, j)$. if $k=j$ then $a * A \$ \$(i, j)$ else $A \$ \$(i, j))$
definition mat-swapcols :: nat $\Rightarrow$ nat $\Rightarrow$ 'a mat $\Rightarrow{ }^{\prime} a$ mat (swapcols)where swapcols kl $A=$ mat (dim-row $A)($ dim-col $A)$ $(\lambda(i, j)$. if $k=j$ then $A \$ \$(i, l)$ else if $l=j$ then $A \$ \$(i, k)$ else $A \$ \$(i, j))$
definition mat-addcol-vec :: nat $\Rightarrow{ }^{\prime} a$ :: plus vec $\Rightarrow{ }^{\prime} a$ mat $\Rightarrow$ 'a mat where mat-addcol-vec $k v A=$ mat (dim-row $A)($ dim-col $A)$

$$
(\lambda(i, j) . \text { if } k=j \text { then } v \$ i+A \$ \$(i, j) \text { else } A \$ \$(i, j))
$$

definition mat-addcol $::$ ' $a::$ semiring-1 $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow{ }^{\prime} a$ mat $\Rightarrow$ 'a mat (addcol) where

$$
\begin{aligned}
& \text { addcol a } k l A=\text { mat }(\text { dim-row } A)(\text { dim-col } A) \\
& \qquad(\lambda(i, j) \text {. if } k=j \text { then } a * A \$ \$(i, l)+A \$ \$(i, j) \text { else } A \$ \$(i, j))
\end{aligned}
$$

lemma index-mat-multcol[simp]:
$i<$ dim-row $A \Longrightarrow j<\operatorname{dim}$-col $A \Longrightarrow$ multcol $k$ a $A \$ \$(i, j)=($ if $k=j$ then a * $A \$ \$(i, j)$ else $A \$ \$(i, j))$
$i<$ dim-row $A \Longrightarrow j<$ dim-col $A \Longrightarrow$ multcol $j$ a $A \$ \$(i, j)=a * A \$ \$(i, j)$
$i<$ dim-row $A \Longrightarrow j<\operatorname{dim}$-col $A \Longrightarrow k \neq j \Longrightarrow$ multcol $k$ a $A \$ \$(i, j)=A \$ \$$ $(i, j)$
dim-row $($ multcol $k$ a $A)=$ dim-row $A$ dim-col $($ multcol $k$ a $A)=\operatorname{dim-col~} A$ unfolding mat-multcol-def by auto
lemma index-mat-swapcols[simp]:
$i<$ dim-row $A \Longrightarrow j<$ dim-col $A \Longrightarrow$ swapcols $k l A \$ \$(i, j)=($ if $k=j$ then $A$ \$\$ $(i, l)$ else
if $l=j$ then $A \$ \$(i, k)$ else $A \$ \$(i, j))$
dim-row (swapcols klA) $=$ dim-row $A$ dim-col (swapcols $k l A)=\operatorname{dim-col} A$
unfolding mat-swapcols-def by auto
lemma index-mat-addcol[simp]:

```
    \(i<\) dim-row \(A \Longrightarrow j<\) dim-col \(A \Longrightarrow\) addcol a \(k l A \$ \$(i, j)=(\) if \(k=j\) then
    \(a * A \$ \$(i, l)+A \$ \$(i, j)\) else \(A \$ \$(i, j))\)
    \(i<\) dim-row \(A \Longrightarrow j<\operatorname{dim}-\) col \(A \Longrightarrow\) addcol a \(j l A \$ \$(i, j)=a * A \$ \$(i, l)+\)
A\$\$(i,j)
    \(i<\operatorname{dim}-\) row \(A \Longrightarrow j<\operatorname{dim}-\operatorname{col} A \Longrightarrow k \neq j \Longrightarrow\) addcol a \(k l A \$ \$(i, j)=A\)
\$\$(i,j)
    dim-row \((\) addcol a \(k l A)=\) dim-row \(A\) dim-col \((\) addcol a \(k l A)=\operatorname{dim}-c o l A\)
    unfolding mat-addcol-def by auto
```

Each column-operation can be seen as a multiplication of an elementary matrix from the right

## lemma col-addrow:

```
    \(l \neq i \Longrightarrow i<n \Longrightarrow\) col (addrow-mat nakl) \(i=\) unit-vec \(n i\)
    \(k<n \Longrightarrow l<n \Longrightarrow \operatorname{col}(\) addrow-mat \(n a k l) l=a \cdot v\) unit-vec \(n k+\) unit-vec
\(n l\)
    by (rule eq-vecI, auto)
lemma col-addcol[simp]:
    \(k<\operatorname{dim}-\operatorname{col} A \Longrightarrow l<\operatorname{dim}-\operatorname{col} A \Longrightarrow \operatorname{col}(a d d \operatorname{col} \operatorname{akl} A) k=a \cdot{ }_{v} \operatorname{col} A l+\operatorname{col}\)
A \(k\)
    by (rule eq-vecI;simp)
```

lemma addcol-mat: assumes $A: A \in$ carrier-mat $n r n$
and $k$ : $k<n$
shows addcol ( $a$ :: 'a :: comm-semiring-1) $l k A=A *$ addrow-mat $n$ a $k l$
by (rule eq-matI, insert $A k$, auto simp: col-addrow
scalar-prod-add-distrib[of-n] scalar-prod-smult-distrib[of-n])
lemma col-multrow: $k \neq i \Longrightarrow i<n \Longrightarrow \operatorname{col}$ (multrow-mat $n k a) i=$ unit-vec $n$
$i$
$k<n \Longrightarrow$ col (multrow-mat $n k a) k=a \cdot v$ unit-vec $n k$
by (rule eq-vecI, auto)
lemma multcol-mat: assumes $A:\left(A::^{\prime} a::\right.$ comm-ring-1 mat $) \in$ carrier-mat $n r n$ shows multcol $k$ a $A=A *$ multrow-mat $n k$ a
by (rule eq-matI, insert $A$, auto simp: col-multrow smult-scalar-prod-distrib[of $n]$ )
lemma col-swaprows:
$l<n \Longrightarrow$ col (swaprows-mat n l $l$ ) $l=$ unit-vec $n l$
$i \neq k \Longrightarrow i \neq l \Longrightarrow i<n \Longrightarrow$ col (swaprows-mat $n k l$ ) $i=$ unit-vec $n i$
$k<n \Longrightarrow l<n \Longrightarrow \operatorname{col}$ (swaprows-mat $n k l$ ) $l=$ unit-vec $n k$
$k<n \Longrightarrow l<n \Longrightarrow$ col (swaprows-mat $n k l$ ) $k=$ unit-vec $n l$
by (rule eq-vecI, auto)
lemma swapcols-mat: assumes $A: A \in$ carrier-mat $n r n$ and $k: k<n l<n$ shows swapcols $k l A=A *$ swaprows-mat $n k l$
by (rule eq-matI, insert $A k$, auto simp: col-swaprows)
Combining row and column-operations yields similarity transformations.
definition add-col-sub-row :: 'a $::$ ring-1 $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ 'a mat $\Rightarrow$ ' $a$ mat where add-col-sub-row a $k l A=$ addrow $(-a) k l(a d d c o l a l k A)$
definition mult-col-div-row :: 'a $::$ field $\Rightarrow$ nat $\Rightarrow$ ' $a$ mat $\Rightarrow$ 'a mat where mult-col-div-row a $k A=$ multrow $k$ (inverse a) (multcol $k$ a $A$ )
definition swap-cols-rows :: nat $\Rightarrow$ nat $\Rightarrow$ 'a mat $\Rightarrow{ }^{\prime}$ 'a mat where swap-cols-rows $k l A=$ swaprows $k l$ (swapcols $k l A$ )
lemma add-col-sub-row-carrier[simp]:
dim-row (add-col-sub-row a kl A) $=$ dim-row $A$
dim-col (add-col-sub-row a $k l A)=\operatorname{dim}-c o l ~ A$
$A \in$ carrier-mat $n n \Longrightarrow$ add-col-sub-row a $k l A \in$ carrier-mat $n n$
unfolding add-col-sub-row-def carrier-mat-def by auto
lemma add-col-sub-index-row[simp]:
$i<$ dim-row $A \Longrightarrow i<\operatorname{dim}$-col $A \Longrightarrow j<\operatorname{dim}$-row $A \Longrightarrow j<\operatorname{dim}$-col $A \Longrightarrow l$
$<$ dim-row $A$
$\Longrightarrow$ add-col-sub-row a kl A $\$ \$(i, j)=($ if
$i=k \wedge j=l$ then $A \$ \$(i, j)+a * A \$ \$(i, i)-a * a * A \$ \$(j, i)-a *$
A $\$ \$(j, j)$ else if
$i=k \wedge j \neq l$ then $A \$ \$(i, j)-a * A \$ \$(l, j)$ else if
$i \neq k \wedge j=l$ then $A \$ \$(i, j)+a * A \$ \$(i, k)$ else $A \$ \$(i, j))$
unfolding add-col-sub-row-def by (auto simp: field-simps)
lemma mult-col-div-index-row[simp]:
$i<$ dim-row $A \Longrightarrow i<\operatorname{dim}-c o l A \Longrightarrow j<d i m-r o w ~ A \Longrightarrow j<d i m-c o l ~ A \Longrightarrow a$ $\neq 0$
$\Longrightarrow$ mult-col-div-row a $k A \$ \$(i, j)=(i f$
$i=k \wedge j \neq i$ then inverse $a * A \$ \$(i, j)$ else if
$j=k \wedge j \neq i$ then $a * A \$ \$(i, j)$ else $A \$ \$(i, j))$
unfolding mult-col-div-row-def by auto
lemma mult-col-div-row-carrier[simp]:
dim-row (mult-col-div-row a $k A$ ) $=$ dim-row $A$
dim-col (mult-col-div-row a $k A$ ) $=\operatorname{dim}-c o l ~ A$
A $\in$ carrier-mat $n n \Longrightarrow$ mult-col-div-row a $k A \in$ carrier-mat $n n$
unfolding mult-col-div-row-def carrier-mat-def by auto
lemma swap-cols-rows-carrier [simp]:
dim-row (swap-cols-rows klA) $=$ dim-row $A$
dim-col (swap-cols-rows klA) $=\operatorname{dim}-\operatorname{col} A$
$A \in$ carrier-mat $n n \Longrightarrow$ swap-cols-rows $k l A \in$ carrier-mat $n n$
unfolding swap-cols-rows-def carrier-mat-def by auto
lemma swap-cols-rows-index[simp]:
$i<\operatorname{dim}$-row $A \Longrightarrow i<\operatorname{dim}$-col $A \Longrightarrow j<\operatorname{dim}$-row $A \Longrightarrow j<\operatorname{dim}$-col $A \Longrightarrow a$
$<$ dim-row $A \Longrightarrow b<$ dim-row $A$
$\Longrightarrow$ swap-cols-rows a $b A \$(i, j)=A \$ \$$ (if $i=a$ then $b$ else if $i=b$ then $a$ else $i$,
if $j=a$ then $b$ else if $j=b$ then $a$ else $j$ )
unfolding swap-cols-rows-def
by auto
lemma add-col-sub-row-similar: assumes $A: A \in \operatorname{carrier-mat~} n n$ and $k l: k<n$ $l<n k \neq l$
shows similar-mat (add-col-sub-row a $k l A$ ) ( $A$ :: 'a :: comm-ring-1 mat)
proof (rule similar-matI)
let $? P=$ addrow-mat $n(-a) k l$
let ? $Q=$ addrow-mat $n$ a $k l$
let $? B=a d d$-col-sub-row a $k l A$
show carr: $\{? B, A, ? P, ? Q\} \subseteq$ carrier-mat $n n$ using $A$ by auto
show ? $Q * ? P=1_{m} n$ by (rule addrow-mat-inv $[$ OF kl])
show ? $P * ? Q=1_{m} n$ using addrow-mat-inv $[O F k l$, of $-a]$ by simp
have col: addcol a $l k A=A * ? Q$
by (rule addcol-mat[OF A kl(1)])
have ? $B=? P *(A * ? Q)$ unfolding add-col-sub-row-def col
by (rule addrow-mat $[O F-k l(2)$, of $-n]$, insert $A$, simp)
thus $? B=? P * A * ? Q$ using carr by (simp add: assoc-mult-mat $[o f-n n-n$ - $n]$ )
qed
lemma mult-col-div-row-similar: assumes $A: A \in$ carrier-mat $n n$ and $a k: k<$ $n a \neq 0$
shows similar-mat (mult-col-div-row a $k$ A) A
proof (rule similar-matI)
let ? $P=$ multrow-mat $n k$ (inverse a)
let $? Q=$ multrow-mat $n k a$
let $? B=$ mult-col-div-row a $k A$
show carr: $\{? B, A, ? P, ? Q\} \subseteq$ carrier-mat $n n$ using $A$ by auto
show ? $Q * ? P=1_{m} n$ by (rule multrow-mat-inv $[O F a k]$ )
show ? $P * ? Q=1_{m} n$ using multrow-mat-inv[OF ak(1), of inverse a] ak(2)
by $\operatorname{simp}$
have col: multcol $k$ a $A=A * ? Q$
by (rule multcol-mat[OF A])
have ? $B=? P *(A * ? Q)$ unfolding mult-col-div-row-def col
by (rule multrow-mat $[o f-n n]$, insert $A$, simp)
thus $? B=? P * A * ? Q$ using carr by (simp add: assoc-mult-mat[of-n $n-n$ $-n]$ )
qed
lemma swap-cols-rows-similar: assumes $A: A \in c a r r i e r-m a t ~ n n$ and $k l: k<n l$ $<n$
shows similar-mat (swap-cols-rows $k l A$ ) $A$
proof (rule similar-matI)
let $? P=$ swaprows-mat $n k l$
let $? B=$ swap-cols-rows $k l A$
show carr: $\{? B, A, ? P, ? P\} \subseteq$ carrier-mat $n n$ using $A$ by auto
show ? $P * ? P=1_{m} n$ by (rule swaprows-mat-inv[OF kl])
show ? $P * ? P=1_{m} n$ by fact
have col: swapcols kl $A=A * ? P$
by (rule swapcols-mat $[$ OF A kl])
have ? $B=? P *(A * ? P)$ unfolding swap-cols-rows-def col
by (rule swaprows-mat $[o f-n n]$, insert $A k l$, auto)
thus $? B=? P * A * ? P$ using carr by (simp add: assoc-mult-mat $[o f-n n-n$ $-n]$ )
qed
lemma swapcols-carrier $[$ simp $]:($ swapcols $l k A \in$ carrier-mat $n m)=(A \in$ car-rier-mat $n m$ )
unfolding mat-swapcols-def carrier-mat-def by auto
fun swap-row-to-front :: 'a mat $\Rightarrow n a t \Rightarrow{ }^{\prime} a$ mat where
swap-row-to-front $A \quad 0=A$
| swap-row-to-front $A($ Suc $I)=$ swap-row-to-front (swaprows I (Suc I) A) I
fun swap-col-to-front $::$ 'a mat $\Rightarrow$ nat $\Rightarrow$ 'a mat where
swap-col-to-front A $0=A$
| swap-col-to-front $A($ Suc I $)=$ swap-col-to-front (swapcols I (Suc I) A) I
lemma swap-row-to-front-result: $A \in$ carrier-mat $n m \Longrightarrow I<n \Longrightarrow$ swap-row-to-front

$$
A I=
$$

mat $n m(\lambda(i, j)$. if $i=0$ then $A \$ \$(I, j)$
else if $i \leq I$ then $A \$ \$(i-1, j)$ else $A \$ \$(i, j))$
proof (induct I arbitrary: A)
case 0
thus?case
by (intro eq-matI, auto)
next
case (Suc I A)
from $S u c(3)$ have $I: I<n$ by auto
let $? I=$ Suc $I$
let ? $A=$ swaprows $I$ ?I $A$
have AA: ?A $\in$ carrier-mat $n m$ using Suc(2) by simp
have swap-row-to-front $A($ Suc $I)=$ swap-row-to-front ?A I by simp
also have $\ldots=$ mat $n \mathrm{~m}$
$(\lambda(i, j)$. if $i=0$ then ? A $\$ \$(I, j)$
else if $i \leq I$ then ?A $\$ \$(i-1, j)$ else ? $A \$ \$(i, j))$
using $\operatorname{Suc}(1)[O F A A I]$ by simp
also have $\ldots=$ mat $n \mathrm{~m}$
$(\lambda(i, j)$. if $i=0$ then $A \$ \$(? I, j)$
else if $i \leq$ ? I then $A \$ \$(i-1, j)$ else $A \$ \$(i, j))$
by (rule eq-matI, insert I Suc(2), auto)
finally show ?case.
lemma swap-col-to-front-result: $A \in$ carrier-mat $n m \Longrightarrow J<m \Longrightarrow$ swap-col-to-front A $J=$
mat $n m(\lambda(i, j)$. if $j=0$ then $A \$ \$(i, J)$
else if $j \leq J$ then $A \$ \$(i, j-1)$ else $A \$ \$(i, j))$
proof (induct $J$ arbitrary: A)
case 0
thus ?case
by (intro eq-matI, auto)
next
case (Suc J A)
from $S u c(3)$ have $J: J<m$ by auto
let ? $J=$ Suc $J$
let ?A $=$ swapcols $J$ ? $J A$
have AA: ?A $\in$ carrier-mat $n m$ using Suc(2) by simp
have swap-col-to-front $A($ Suc $J)=$ swap-col-to-front ?A $J$ by simp
also have $\ldots=$ mat $n \mathrm{~m}$
$(\lambda(i, j)$. if $j=0$ then ? $A \$ \$(i, J)$
else if $j \leq J$ then ? $A \$(i, j-1)$ else ? $A \$ \$(i, j))$
using $\operatorname{Suc}(1)[O F A A J]$ by simp
also have $\ldots=$ mat $n \mathrm{~m}$
$(\lambda(i, j)$. if $j=0$ then $A \$ \$(i$, ? $J)$
else if $j \leq$ ? $J$ then $A \$ \$(i, j-1)$ else $A \$ \$(i, j))$
by (rule eq-matI, insert $J$ Suc(2), auto)
finally show ?case .
qed
lemma swapcols-is-transp-swap-rows: assumes $A: A \in$ carrier-mat $n m k<m l$ $<m$
shows swapcols kl $A=$ transpose-mat (swaprows $k l($ transpose-mat $A))$
using assms by (intro eq-matI, auto)
end

## 9 Determinants

Most of the following definitions and proofs on determinants have been copied and adapted from /src/HOL/Multivariate-Analysis/Determinants.thy.

Exceptions are det-identical-rows.
We further generalized some lemmas, e.g., that the determinant is 0 iff the kernel of a matrix is non-empty is available for integral domains, not just for fields.
theory Determinant
imports

Missing-Misc
Column-Operations
HOL-Computational-Algebra.Polynomial-Factorial
Polynomial-Interpolation.Ring-Hom
Polynomial-Interpolation.Missing-Unsorted

## begin

definition det:: 'a mat $\Rightarrow$ ' $a$ :: comm-ring-1 where
$\operatorname{det} A=\left(\right.$ if dim-row $A=\operatorname{dim}$-col $A$ then $\left(\sum p \in\{p\right.$. p permutes $\{0 . .<$ dim-row $A\}$.
signof $p *\left(\prod i=0 . .<\right.$ dim-row $\left.\left.A . A \$ \$(i, p i)\right)\right)$ else 0$)$
lemma(in ring-hom) hom-signof $[\operatorname{simp}]$ : hom $(\operatorname{signof} p)=\operatorname{signof} p$
by (simp add: hom-uminus sign-def)
lemma(in comm-ring-hom) hom-det[simp]: $\operatorname{det}($ map-mat hom $A)=\operatorname{hom}(\operatorname{det} A)$
unfolding det-def by (auto simp: hom-distribs)
lemma det-def': $A \in$ carrier-mat $n n \Longrightarrow$
$\operatorname{det} A=\left(\sum p \in\{p . p\right.$ permutes $\{0 . .<n\}\}$. signof $\left.p *\left(\prod i=0 . .<n . A \$ \$(i, p i)\right)\right)$ unfolding det-def by auto

```
lemma det-smult \([\operatorname{simp}]: \operatorname{det}\left(a \cdot_{m} A\right)=a^{\wedge} \operatorname{dim}-\operatorname{col} A * \operatorname{det} A\)
proof -
    have \([\operatorname{simp}]:\left(\prod i=0 . .<\operatorname{dim}-c o l\right.\) A. \(\left.a\right)=a^{\wedge}\) dim-col \(A\) by (subst prod-constant;simp \()\)
    show ?thesis
    unfolding det-def
    unfolding index-smult-mat
    by (auto intro: sum.cong simp: sum-distrib-left prod.distrib)
qed
lemma det-transpose: assumes \(A: A \in\) carrier-mat \(n n\)
    shows \(\operatorname{det}(\) transpose-mat \(A)=\operatorname{det} A\)
proof -
    let ? \(d i=\lambda A i j . A \$ \$(i, j)\)
    let \(? U=\{0 . .<n\}\)
    have \(f U\) : finite ? \(U\) by simp
    let ?inv \(=\) Hilbert-Choice.inv
    \{
        fix \(p\)
        assume \(p: p \in\{p . p\) permutes ? \(U\}\)
        from \(p\) have \(p U: p\) permutes ? \(U\)
            by blast
    have sth: signof (?inv \(p\) ) \(=\) signof \(p\)
            by (rule signof-inv \([O F-p U]\), simp)
    from permutes-inj[OF pU]
    have \(p i\) : inj-on \(p\) ? \(U\)
        by (blast intro: subset-inj-on)
    let ?f \(=\lambda i\). transpose-mat \(A \$ \$(i\), ?inv \(p i)\)
```

```
    note pU-U = permutes-image [OF pU]
    note [simp] = permutes-less[OF pU]
    have prod ?f ?U = prod ?f (p'?U)
        using pU-U by simp
    also have ... = prod (?f \circ p) ?U
        by (rule prod.reindex[OF pi])
    also have ... = prod (\lambdai. A $$ (i,pi)) ?U
    by (rule prod.cong, insert A, auto)
    finally have signof (?inv p)* prod ?f ?U =
    signof p* prod (\lambdai. A $$ (i,pi)) ?U
    unfolding sth by simp
}
then show ?thesis
    unfolding det-def using A
    by (simp, subst sum-permutations-inverse, intro sum.cong, auto)
qed
lemma det-col:
    assumes A:A\incarrier-mat n n
    shows det A = (\sum p|p permutes {0..<n}. signof p*(\prodj<n.A $$ (pj,j)))
        (is - = (sum ( }\lambdap.-* ?prod p)?P)
proof -
    let ?i = Hilbert-Choice.inv
    let ?N = {0 ..<n}
    let ?f = \lambdap. signof p * ?prod p
    let ?prod' = \lambdap. \j<n. A $$ (j, ?i p j)
    let ?prod" = \lambdap. \j<n. A $$ (j,pj)
    let ?f'}=\lambdap. signof (?i p)* ?prod' p
    let ?f'\prime}=\lambdap. signof p* ?prod" p
    let ? }\mp@subsup{P}{}{\prime}={\mathrm{ ?i }p|p.p\mathrm{ permutes ?N }
    have [simp]:{0..<n} ={..<n} by auto
    have sum ?f ?P = sum ?f' ?P
    proof (rule sum.cong[OF refl],unfold mem-Collect-eq)
    fix p}\mathrm{ assume p: p permutes ?N
    have [simp]: ?prod p=?prod' p
            using permutes-prod[OF p, of \lambdax y. A $$ (x,y)] by auto
            have [simp]: signof p = signof (?i p)
                apply(rule signof-inv[symmetric]) using p by auto
            show ?f p=?f' p by auto
    qed
    also have ... = sum ?f' ?P'
        by (rule sum.cong[OF image-inverse-permutations[symmetric]],auto)
    also have ... = sum ?f" ?P
        unfolding sum.reindex[OF inv-inj-on-permutes,unfolded image-Collect]
        unfolding o-def
        apply (rule sum.cong[OF refl])
        using inv-inv-eq[OF permutes-bij] by force
    finally show ?thesis unfolding det-def '[OF A] by auto
qed
```

```
lemma mat-det-left-def: assumes A: A \in carrier-mat n n
    shows det A = (\sump\in{p.p permutes {0..<dim-row A}}. signof p* (\prodi=0 ..<
dim-row A. A $$ (pi,i)))
proof -
    have cong: \ abc. b=c\Longrightarrowa*b=a*c by simp
    show ?thesis
    unfolding det-transpose[OF A, symmetric]
    unfolding det-def index-transpose-mat using A by simp
qed
lemma det-upper-triangular:
    assumes ut: upper-triangular A
    and m:A\in carrier-mat n n
    shows }\operatorname{det}A=\mathrm{ prod-list (diag-mat A)
proof -
    note det-def = det-def'[OF m]
    let ?U = {0..<n}
    let ?PU = {p.p permutes ? U}
    let ?pp = \lambdap. signof p*(\Pi i=0 ..<n. A $$(i,pi))
    have fU: finite ?U
    by simp
    from finite-permutations[OF fU] have fPU: finite ?PU .
    have id0: {id}\subseteq?PU
    by (auto simp add: permutes-id)
    {
    fix p
    assume p:p\in?PU - {id}
    from p have pU:p permutes ? U and pid: p\not=id
        by blast+
    from permutes-natset-ge[OF pU] pid obtain i where i:pi< i and i<n
            by fastforce
    from upper-triangularD[OF ut i]<i<n\ranglem
    have ex:\existsi\in?U.A$$(i,pi)=0 by auto
    have (\prod i=0 ..<n. A $$ (i,pi))=0
        by (rule prod-zero[OF fU ex])
    hence ?pp p=0 by simp
    }
    then have p0: \bigwedgep.p\in?PU-{id}\Longrightarrow?pp p=0
    by blast
    from }m\mathrm{ have dim: dim-row }A=n\mathrm{ by simp
    have det A=(\sump\in?PU. ?pp p) unfolding det-def by auto
    also have ... = ?pp id + (\sum p\in?PU - {id}. ?pp p)
    by (rule sum.remove, insert id0 fPU m, auto simp: p0)
    also have ( \sump\in?PU - {id}. ?pp p)=0
    by (rule sum.neutral, insert fPU, auto simp: p0)
    finally show ?thesis using m by (auto simp: prod-list-diag-prod)
qed
```

```
lemma det-single: assumes A\incarrier-mat 1 }
    shows }\operatorname{det}A=A$$(0,0
    by (subst det-upper-triangular[of-1], insert assms, auto simp: diag-mat-def)
lemma det-one[simp]: det (1m n)=1
proof -
    have det (1m n) = prod-list (diag-mat (1m n))
    by (rule det-upper-triangular[of-n], auto)
    also have ...=1 by (induct n, auto)
    finally show ?thesis .
qed
lemma det-zero[simp]: assumes n>0 shows det (0m n n)=0
proof -
    have det ( ( Om n n) = prod-list (diag-mat ( ( Om n n) )
        by (rule det-upper-triangular[of-n], auto)
    also have \ldots=0 using <n>0\rangle by (cases n, auto)
    finally show ?thesis.
qed
lemma det-dim-zero[simp]: A \in carrier-mat 0 0 \Longrightarrow det A=1
    unfolding det-def carrier-mat-def sign-def by auto
lemma det-lower-triangular:
    assumes ld: \ij. i<j\Longrightarrowj<n\LongrightarrowA$$ (i,j)=0
    and m:A\in carrier-mat n n
    shows }\operatorname{det}A=\operatorname{prod}-list (diag-mat A)
proof -
    have }\operatorname{det}A=\operatorname{det}(\mathrm{ transpose-mat A) using det-transpose[OF m] by simp
    also have ... = prod-list (diag-mat (transpose-mat A))
    by (rule det-upper-triangular, insert m ld, auto)
    finally show ?thesis using m by simp
qed
lemma det-permute-rows: assumes A: A\in carrier-mat n n
    and p:p permutes {0..< (n :: nat)}
    shows det (mat n n (\lambda (i,j).A $$ (p i,j))) = signof p*\operatorname{det A}
proof -
    let ?U = {0 ..< (n :: nat)}
    have cong: \abc.b=c\Longrightarrowa*b=a*c by auto
    have det (mat n n (\lambda(i,j). A $$ (pi,j)))=
    (\sumq\in{q.q permutes ?U}. signof q*(\prod i\in?U.A$$(pi,qi)))
    unfolding det-def using A p by auto
    also have ... = (\sumq\in{q.q permutes ?U}. signof (q\circp)* (\prod i\in?U. A
$$(pi,(q\circp)i)))
    by (rule sum-permutations-compose-right[OF p])
    finally have 1: det (mat n n ( }\lambda(i,j). A $$ (pi,j))
    =(\sumq\in{q.q permutes?U}. signof (q\circp)*(\i\in?U.A$$(pi,(q\circ
p) i))).
```

```
    have 2: signof p*\operatorname{det}A=
    (\sumq\in{q. q permutes ? U}. signof p* signof q*(\prodi\in?U. A $$ (i,qi)))
    unfolding det-def'[OF A] sum-distrib-left by (simp add: ac-simps)
    show ?thesis unfolding 12
    proof (rule sum.cong, insert p A, auto)
    fix q
    assume q: q permutes ?U
    let ?inv = Hilbert-Choice.inv
    from permutes-inv[OF p] have ip: ?inv p permutes ?U .
    have prod ( }\lambdai.A$$(pi,(q\circp)i)) ?U
        prod (\lambdai. A $$ ((p\circ?inv p) i, (q\circ(p\circ?inv p)) i)) ?U unfolding o-def
        by (rule trans[OF prod.permute[OF ip] prod.cong], insert A p q,auto)
    also have ... = prod ( }\lambdai.A$$(i,qi)) ?
        by (simp only:o-def permutes-inverses[OF p])
    finally have thp: prod (\lambdai. A $$ (pi,(q\circp) i))?U = prod (\lambdai. A$$(i,qi))
?U .
    show signof (q\circp)*(\prodi\in{0..<n}. A $$ (pi,q(pi)))=
                signof p * signof q * (\prodi\in{0..<n}. A $$ (i,qi))
        unfolding thp[symmetric] signof-compose[OF q p]
        by (simp add: ac-simps)
    qed
qed
lemma det-multrow-mat: assumes }k\mathrm{ : }k<
    shows det (multrow-mat n k a)=a
proof (rule trans[OF det-lower-triangular[of n]], unfold prod-list-diag-prod)
    let ?f = \lambda i. multrow-mat n k a $$ (i,i)
    have (\prodi\in{0..<n}. ?f i)= ?f k* (\prodi\in{0..<n} - {k}. ?f i)
        by (rule prod.remove, insert k, auto)
    also have (\prodi\in{0..<n}-{k}. ?f i)=1
        by (rule prod.neutral, auto)
    finally show (\prodi\in{0..<dim-row (multrow-mat n k a)}. ?f i) = a using k by
simp
qed (insert k, auto)
lemma swap-rows-mat-eq-permute:
    k<n\Longrightarrowl<n\Longrightarrow swaprows-mat nkl=mat n n ( }\lambda(i,j).1mn$$(transpos
kl i,j))
    by (rule eq-matI) (auto simp add: transpose-def)
lemma det-swaprows-mat: assumes }k:k<n\mathrm{ and }l:l<n\mathrm{ and }kl:k\not=
    shows det (swaprows-mat n k l)}=-
proof -
    let ? n = {0 ..< (n :: nat)}
    let ? p = transpose kl
    have p:?p permutes ?n
    by (rule permutes-swap-id, insert kl, auto)
    show ?thesis
            by (rule trans[OF trans[OF - det-permute-rows[OF one-carrier-mat[of n] p]]],
```

subst swap-rows-mat-eq-permute[OF $k l]$, auto simp: sign-swap-id $k l$ )
qed
lemma det-addrow-mat:
assumes $l: k \neq l$
shows det (addrow-mat $n$ a $k l)=1$
proof -
have $\operatorname{det}($ addrow-mat $n$ a $k l)=$ prod-list (diag-mat (addrow-mat $n$ a $k l)$ )
proof (cases $k<l$ )
case True
show ?thesis
by (rule det-upper-triangular[of-n], insert True, auto intro!: upper-triangularI)
next
case False
show ?thesis
by (rule det-lower-triangular[of n], insert False, auto)
qed
also have $\ldots=1$ unfolding prod-list-diag-prod
by (rule prod.neutral, insert l, auto)
finally show? thesis.
qed
The following proof is new, as it does not use $2 \neq 0$ as in MultivariateAnalysis.
lemma det-identical-rows:
assumes $A: A \in$ carrier-mat $n n$
and $i j: i \neq j$
and $i: i<n$ and $j: j<n$
and $r$ : row $A i=$ row $A j$
shows $\operatorname{det} A=0$
proof-
let $? p=$ transpose $i j$
let $? n=\{0 . .<n\}$
have $s p$ : signof ? $p=-1$ sign ? $p=(-1::$ int $)$ using $i j$
by (auto simp add: sign-swap-id)
let ?f $=\lambda p$. signof $p *\left(\prod i \in\right.$ ?n. $\left.A \$ \$(p i, i)\right)$
let ?all $=\{p . p$ permutes ? $n\}$
let ?one $=\{p . p$ permutes ? $n \wedge \operatorname{sign} p=(1::$ int $)\}$
let ?none $=\{p . p$ permutes ? $n \wedge \operatorname{sign} p \neq(1::$ int $)\}$
let ?pone $=(\lambda p$. ?p o $p)$ ' ?one
have split: ? one $\cup$ ?none $=$ ? all by auto
have $p$ : ?p permutes ?n by (rule permutes-swap-id, insert $i j$, auto)
from permutes-inj[OF p] have injp: inj ?p by auto
\{
fix $q$
assume $q$ : $q$ permutes ?n
have $\left(\prod k \in ? n . A \$ \$(? p(q k), k)\right)=\left(\prod k \in ? n . A \$ \$(q k, k)\right)$
proof (rule prod.cong)
fix $k$

```
        assume k: k\in?n
        from r have row: row A i$k= row Aj$k by simp
        hence A$$(i,k)=A$$(j,k) using kij A by auto
        thus A$$(?p (qk),k)=A$$(qk,k)
            by (cases q k=i, auto, cases q k=j, auto)
    qed (insert A q,auto)
} note }*=thi
have pp:\bigwedge q. q permutes ? n \Longrightarrow permutation q unfolding
    permutation-permutes by auto
have }\operatorname{det}A=(\sump\in\mathrm{ ?one }\cup\mathrm{ ?none. ?f }p
    using A unfolding mat-det-left-def[OF A] split by simp
also have }\ldots=(\sump\in\mathrm{ ?one. ?f }p)+(\sump\in\mathrm{ ?none. ?f p}
    by (rule sum.union-disjoint, insert A, auto simp: finite-permutations)
    also have ?none = ?pone
    proof -
    {
        fix q
        assume q\in ?none
        hence q: q permutes ?n and sq: sign q=(-1 :: int) unfolding sign-def by
auto
    from permutes-compose[OF q p] sign-compose[OF pp[OF p] pp[OF q], unfolded
sp sq]
    have ?p o q \in ?one by auto
    hence ?p o (?p o q) \in ?pone by auto
    also have ?p o (?p o q)=q
            by (auto simp: swap-id-eq)
    finally have q\in ?pone.
    }
    moreover
    {
        fix pq
        assume pq\in?pone
        then obtain q}\mathrm{ where q:q ? ?one and pq: pq=?p o q by auto
        from q have q:q permutes ?n and sq: sign q=(1 :: int) by auto
        from sign-compose[OF pp[OF p] pp[OF q], unfolded sq sp]
        have spq: sign pq=(-1 :: int) unfolding pq by auto
        from permutes-compose[OF q p] have pq: pq permutes ?n unfolding pq by
auto
    from pq spq have pq\in?none by auto
    }
    ultimately
    show ?thesis by blast
qed
also have (\sump\in ?pone. ?f p)=(\sump\in ?one. ?f (?p o p))
proof (rule trans[OF sum.reindex])
    show inj-on ((o) ?p) ?one
        using fun.inj-map[OF injp] unfolding inj-on-def by auto
qed simp
also have (\sump\in ?one. ?f p) +(\sump\in ?one. ?f (?p o p))
```

```
    =(\sump\in ?one. ?f p+ ?f (?p o p))
    by (rule sum.distrib[symmetric])
    also have ... = 0
    by (rule sum.neutral, insert A, auto simp:
        sp sign-compose[OF pp[OF p] pp] ij finite-permutations *)
    finally show ?thesis .
qed
lemma det-row-0: assumes k: k<n
    and c:c\in{0 ..<n} }->\mathrm{ carrier-vec n
    shows det (matr n n ( }\lambdai\mathrm{ . if }i=k\mathrm{ then Ov n else c i)) = 0
proof -
    {
        fix p
        assume p:p permutes {0 ..<n}
        have (\prodi\in{0..<n}. matr n n (\lambdai. if i=k then 0v n else c i) $$ (i,pi)) =0
            by (rule prod-zero[OF - bexI[of - k]],
            insert k p c[unfolded carrier-vec-def],auto)
    }
    thus ?thesis unfolding det-def by simp
qed
lemma det-row-add:
    assumes abc: a k\in carrier-vec n b k\in carrier-vec n c { {0..<n} -> carrier-vec
n
    and k:k<n
    shows det (matr n n ( }\lambda\mathrm{ i. if i=k then a i + b i else c i)) =
        det(mat r}nn(\lambdai. if i=k then a i else c i)) +
        det(matr n n ( }\lambda\mathrm{ i. if }i=k\mathrm{ then b i else c i))
    (is ?lhs = ?rhs)
proof -
    let ? }n={0..<n
    let ?m=\lambdaabpi.matr n n ( \lambdai. if i=k then a i else b i)$$(i,pi)
    let ?c = \lambda p i. matr n nc $$(i,p i)
    let ?ab=\lambda i. a i+bi
    note intros=add-carrier-vec[of-n]
    have ?rhs = (\sump\in{p.p permutes ? n}.
        signof p*(\prodi\in?n. ?m a c p i)) +(\sump\in{p. p permutes ? n}. signof p *
(\prodi\in?n. ?m b c pi))
    unfolding det-def by simp
    also have ... =(\sump\in{p.p permutes ? n}. signof p*(\prodi\in?n. ?m a c p i)+
signof p*(\prodi\in? n. ?m b c p i))
    by (rule sum.distrib[symmetric])
    also have ... = (\sump\in{p.p permutes ?n}. signof p*(\prodi\in?n. ?m ?ab c p i))
    proof (rule sum.cong, force)
    fix p
    assume p}\in{p.p\mathrm{ permutes ?n}
    hence p: p permutes ?n by simp
    show signof p*(\i\in?n.?m а с p i)+ signof p*(\i\in?n. ?m b с p i)=
```

```
    signof p * (\prodi\in?n. ?m ?ab c p i)
    unfolding distrib-left[symmetric]
    proof (rule arg-cong[of - - \lambda a. signof p*a])
    from k have f: finite ?n and k':k\in?n by auto
    let ?nk=? n - {k}
    note split = prod.remove[OF f k]
    have id1: (\prodi\in?n. ?m a c pi)=?m acpk*(\i\in?nk. ?m a c pi)
        by (rule split)
    have id2:(\prodi\in?n. ?m b c p i)=?mb c p k*(\i\in?nk. ?m b c p i)
        by (rule split)
    have id3:(\prodi\in?n. ?m ?ab с p i)=?m ?ab с p k*(\prodi\in?nk. ?m ?ab с p i)
        by (rule split)
    have id: \bigwedgea. (\prodi\in?nk. ?m а с p i)=(\prodi\in?nk. ?c p i)
        by (rule prod.cong, insert abc k p, auto intro!: intros)
    have ab: ?ab k carrier-vec n using abc by (auto intro: intros)
    {
        fix f
        assume fk (carrier-vec n :: 'a vec set)
        hence matr n n(\lambdai. if i=k then fi else c i) $$(k,pk)=fk$pk
            by (insert p kabc,auto)
    } note first = this
    note id' = id1 id2 id3
    have dist: (ak+bk)$pk=ak$pk+bk$pk
        by (rule index-add-vec(1), insert p k abc, force)
    show (\prodi\in?n. ?m a c pi)+(\prodi\in?n. ?m b c pi)=(\prodi\in?n. ?m ?ab c pi)
    unfolding id' id first[of a, OF abc(1)] first[of b, OF abc(2)] first[of ?ab, OF
ab] dist
            by (rule distrib-right[symmetric])
        qed
    qed
    also have ... = ?lhs unfolding det-def by simp
    finally show ?thesis by simp
qed
lemma det-linear-row-finsum:
    assumes fS: finite S and c:c\in{0..<n} -> carrier-vec n and k:k<n
    and a: a k}\inS->\mathrm{ carrier-vec n
    shows det (matr n n ( }\lambda\mathrm{ i. if }i=k\mathrm{ then finsum-vec TYPE('a :: comm-ring-1) n
(a i)S else c i))=
        sum ( }\lambdaj\mathrm{ . det (mat r n n ( }\lambda\mathrm{ i. if i=k then a i j else c i))) S
proof -
    let ?sum = finsum-vec TYPE('a) n
    show ?thesis using a
    proof (induct rule: finite-induct[OF fS])
        case 1
        show ?case
            by (simp, unfold finsum-vec-empty, rule det-row-0[OF k c])
    next
```

```
    case (2 x F)
    from 2(4) have ak: a k\inF->carrier-vec n and akx: a k x\incarrier-vec n
by auto
    {
        fix }
        note if-cong[OF refl finsum-vec-insert[OF 2(1-2)],
        of - ain cilci]
    } note }*=\mathrm{ this
    show ?case
    proof (subst *)
        show det (matr nn (\lambdai. if i=k then a i x + ?sum (a i) F else c i)) =
            (\sumj\ininsert x F.det (matr n n (\lambdai. if i=k then a i j else c i)))
        proof (subst det-row-add)
            show det (matr n n ( }\lambdai\mathrm{ . if }i=k\mathrm{ then a i x else c i)) +
                det (matr n n (\lambdai. if i=k then ?sum (a i) F else c i)) =
            (\sumj\ininsert x F.det (matr n n (\lambdai. if i=k then a i j else c i)))
            unfolding 2(3)[OF ak] sum.insert[OF 2(1-2)] by simp
        qed (insert c k ak akx 2(1),
            auto intro!: finsum-vec-closed)
        qed (insert akx ak, force+)
    qed
qed
lemma det-linear-rows-finsum-lemma:
assumes \(f S\) : finite \(S\)
and \(f T\) : finite \(T\) and \(c: c \in\{0 . .<n\} \rightarrow\) carrier-vec \(n\)
and \(T: T \subseteq\{0 . .<n\}\)
and \(a: a \in T \rightarrow S \rightarrow\) carrier-vec \(n\)
shows det \(\left(\right.\) mat \(_{r} n n(\lambda\) i. if \(i \in T\) then finsum-vec TYPE ('a \(::\) comm-ring-1) \(n\) ( \(a\) i) \(S\) else \(c i)\) ) \(=\)
    sum (\lambdaf.det(mat r n n ( }\lambda\mathrm{ i. if }i\inT\mathrm{ then a i (fi) else c i)))
            {f. (\foralli\inT.fi\inS)^(\foralli.i\not\inT\longrightarrow\longrightarrowfi=i)}
proof -
    let ?sum = finsum-vec TYPE('a)n
    show ?thesis using fT c a T
    proof (induct T arbitrary: a c set: finite)
        case empty
        let ?f = (\lambda i. i) :: nat => nat
        have [simp]:{f.\foralli.fi=i}={?f} by auto
        show ?case by simp
    next
        case (insert z T a c)
        hence z:z<n and azS:a z\inS->carrier-vec n by auto
    let ?F=\lambdaT.{f. (\foralli\inT.fi\inS)\wedge(\foralli. i\not\inT\longrightarrow\longrightarrowfi=i)}
    let ?h}=\lambda(y,g) i. if i=z then y else g i
    let ?k = \h. (h(z),(\lambdai. if i=z then i else h i)}
    let ?s = \lambdakacf. det(matr nn (\lambda i. if i\inT then a i (fi) else c i))
    let ?c = \lambdaj i. if i=z then a ij else c i
```

have thif: $\bigwedge a b c d .($ if $a \vee b$ then $c$ else $d)=($ if $a$ then $c$ else if $b$ then $c$ else d)
by $\operatorname{simp}$
have thif2: $\bigwedge a b c d e$. (if a then $b$ else if $c$ then $d$ else $e)=$ (if $c$ then (if $a$ then $b$ else $d$ ) else (if a then $b$ else e))
by simp
from $\langle z \notin T\rangle$ have $n z: \bigwedge i . i \in T \Longrightarrow i=z \longleftrightarrow$ False by auto
from insert have $c: \bigwedge i . i<n \Longrightarrow c i \in$ carrier-vec $n$ by auto
have fin: finite $\{f .(\forall i \in T . f i \in S) \wedge(\forall i . i \notin T \longrightarrow f i=i)\}$ by (rule finite-bounded-functions[OF fS insert(1)])
have $\operatorname{det}\left(\right.$ mat $_{r} n n(\lambda$. if $i \in$ insert $z T$ then ? sum $(a i) S$ else $\left.c i)\right)=$ $\operatorname{det}\left(\right.$ mat $_{r} n n$ ( $\lambda$ i. if $i=z$ then? sum ( $a$ i) $S$ else if $i \in T$ then ?sum (ai) $S$ else $c i$ )) unfolding insert-iff thif ..
also have $\ldots=\left(\sum j \in S\right.$. det $\left(\right.$ mat $_{r} n n$ ( $\lambda$ i. if $i \in T$ then ? sum ( $a$ i) $S$ else if $i=z$ then a $i j$ else $c i)$ )
apply (subst det-linear-row-finsum $[$ OF fS - z] $)$
prefer 3
apply (subst thif2)
using $n z$
apply (simp cong del: if-weak-cong cong add: if-cong)
apply (insert azS c fS insert(5), (force intro!: finsum-vec-closed)+)
done
also have $\ldots=\left(\operatorname{sum}\left(\lambda(j, f)\right.\right.$. det $\left(\right.$ mat $_{r} n n(\lambda i$. if $i \in T$ then a $i(f i)$
else if $i=z$ then a $i j$
else $c i)$ )) $(S \times$ ? $F T))$
unfolding sum.cartesian-product[symmetric]
by (rule sum.cong[OF refl], subst insert.hyps(3), insert $a z S$ c fin $z \operatorname{insert}(5-6)$, auto)
finally have tha:
$\operatorname{det}\left(\right.$ mat $_{r} n n(\lambda$ i. if $i \in$ insert $z T$ then ?sum $(a i) S$ else $\left.c i)\right)=$
$\left(\operatorname{sum}\left(\lambda(j, f) . \operatorname{det}\left(\right.\right.\right.$ mat $_{r} n n(\lambda i$. if $i \in T$ then a $i(f i)$
else if $i=z$ then $a i j$
else $c i))$ ) $(S \times$ ? $F T)$ ).
show ?case unfolding tha
by (rule sum.reindex-bij-witness[where $i=? k$ and $j=? h]$, insert $\langle z \notin T\rangle$
$a z S$ c fS insert $(5-6) z$ fin,
auto intro!: arg-cong[of - det])
qed
qed
lemma det-linear-rows-sum:
assumes $f S$ : finite $S$
and $a: a \in\{0 . .<n\} \rightarrow S \rightarrow$ carrier-vec $n$
shows det ( mat $_{r} n n\left(\lambda\right.$ i. finsum-vec $\operatorname{TYPE}\left({ }^{\prime} a::\right.$ comm-ring-1) $\left.\left.n(a i) S\right)\right)=$ $\operatorname{sum}\left(\lambda f . \operatorname{det}\left(m a t_{r} n n(\lambda\right.\right.$ i. a $\left.\left.i(f i))\right)\right)$
$\{f .(\forall i \in\{0 . .<n\} . f i \in S) \wedge(\forall i . i \notin\{0 . .<n\} \longrightarrow f i=i)\}$
proof -

```
    let ?T = {0..<n}
    have fT: finite ?T by auto
    have th0: \bigwedgex y. matr nn(\lambdai. if i\in?T then x i else y i)= matrr nn (\lambdai.x i)
    by (rule eq-rowI, auto)
    have c:(\lambda -. 䜣 n) \in?T }->\mathrm{ carrier-vec n by auto
    show ?thesis
    by (rule det-linear-rows-finsum-lemma[OF fS fT c subset-refl a, unfolded th0])
qed
lemma det-rows-mul:
    assumes a: a\in{0..<n} -> carrier-vec n
    shows det(mat rr n n (\lambdai.c i f.v a i)) =
    prod c {0..<n}*\operatorname{det}(\mp@subsup{mat}{r}{}nn(\lambdai.ai))
proof -
    have A: matrr n n (\lambda i.ci i \mp@subsup{v}{v}{}ai)\in carrier-mat n n
    and A': matr n n (\lambda i. a i) \in carrier-mat n n using a unfolding carrier-mat-def
by auto
    show ?thesis unfolding det-def'[OF A] det-def'[OF A']
    proof (rule trans[OF sum.cong sum-distrib-left[symmetric]])
        fix p
    assume p:p\in{p.p permutes {0..<n}}
    have id: (\prodia\in{0..<n}. matr n n (\lambdai.c ci vv a i) $$ (ia,p ia))
            =prod c {0..<n}*(\prodia\in{0..<n}. matr n n a $$(ia, p ia))
            unfolding prod.distrib[symmetric]
            by (rule prod.cong, insert p a, force+)
    show signof p*(\prodia\in{0..<n}. mat r n n (\lambdai.ci c .v a i)$$ (ia,pia))=
                prod c {0..<n}*(signof p*(\prodia\in{0..<n}. matr n na $$(ia, p ia)))
            unfolding id by auto
    qed simp
qed
```

lemma mat-mul-finsum-alt:
assumes $A: A \in$ carrier-mat $n r n$ and $B: B \in$ carrier-mat $n n c$
shows $A * B=$ mat $_{r} n r n c(\lambda i$. finsum-vec TYPE ('a :: semiring-0) $n c(\lambda k . A$ $\$ \$(i, k) \cdot v$ row $B k)\{0 . .<n\})$
by (rule eq-matI, insert A B, auto, subst index-finsum-vec, auto simp: scalar-prod-def intro: sum.cong)
lemma det-mult:
assumes $A: A \in$ carrier-mat $n n$ and $B: B \in$ carrier-mat $n n$
shows $\operatorname{det}(A * B)=\operatorname{det} A * \operatorname{det}(B:: ' a::$ comm-ring- 1 mat)
proof -
let $? U=\{0 . .<n\}$
let ?F $=\{f .(\forall i \in ? U . f i \in ? U) \wedge(\forall i . i \notin ? U \longrightarrow f i=i)\}$
let $? P U=\{p . p$ permutes ? $U\}$
have $f U$ : finite ? $U$
by blast

```
    have fF: finite ?F
    by (rule finite-bounded-functions, auto)
    {
    fix p
    assume p: p permutes ?U
    have p}\in\mathrm{ ?F unfolding mem-Collect-eq permutes-in-image[OF p]
        using p[unfolded permutes-def] by simp
    }
    then have PUF:?PU\subseteq?F by blast
    {
    fix f
    assume fPU: f\in?F - ?PU
    have fUU: f'?U\subseteq?U
        using }fPU\mathrm{ by auto
    from fPU have f:\foralli\in?U.fi\in?U\foralli.i\not\in?U\longrightarrow}\longrightarrowfi=i\neg(\forally.\exists!x.f
= y)
    unfolding permutes-def by auto
    let ?A = matr n n (\lambda i. A $$ (i,fi) \cdotv row B (fi))
    let ?B = matr n n (\lambdai. row B (fi))
    have }\mp@subsup{B}{}{\prime}:?B\in\mathrm{ carrier-mat n n
        by (intro mat-row-carrierI)
    {
        assume fi: inj-on f ?U
        from inj-on-nat-permutes[OF fi] f
        have f permutes ? U by auto
        with fPU have False by simp
    }
    hence fni:\neginj-on f ?U by auto
    then obtain ij where ij:fi=fji\not=ji<nj<n
        unfolding inj-on-def by auto
    from ij
    have rth: row ?B i = row ?B j by auto
    have det ?A = 0
        by (subst det-rows-mul, unfold det-identical-rows[OF B' ij(2-4) rth], insert
f A B, auto)
    }
    then have zth: \f.f\in?F-?PU\Longrightarrow det (matr n n (\lambda i. A $$ (i,fi) \cdotv row
B(fi)))=0
    by simp
{
    fix p
    assume pU: p\in?PU
    from pU have p: p permutes ? U
        by blast
    let ?s = \lambdap. (signof p):: ' }
    let ?f = \lambdaq. ?s p*(\prod i\in ?U.A $$ (i,pi))*(?s q*(\prodi\in?U.B$$ (i,qi)))
    have (sum ( \lambdaq. ?s q *
        (\prodi\in?U. matr n n (\lambdai.A$$(i,pi) .v row B (pi))$$ (i,qi))) ?PU)=
        (sum (\lambdaq. ?s p * (\Pi i\in?U. A $$ (i,p i))*(?s q* (\Pi i\in?U.B $$ (i,q
```

```
i)))) ?PU 
unfolding sum-permutations-compose-right[OF permutes-inv[OF p], of ?f]
    proof (rule sum.cong[OF refl])
    fix q
    assume q\in{q.q permutes ?U}
    hence q:q permutes ?U by simp
    from pq}\mathrm{ have pp: permutation }p\mathrm{ and pq: permutation q
            unfolding permutation-permutes by auto
    note sign = signof-compose[OF q permutes-inv[OF p], unfolded signof-inv[OF
fU p]]
    let ?inv = Hilbert-Choice.inv
    have th001: prod (\lambdai. B$$ (i,q(?inv p i)))?U = prod ((\lambdai.B$$ (i,q (?inv
pi))) ○ p) ?U
            by (rule prod.permute[OF p])
    have thp: prod (\lambdai.matr n n (\lambda i. A$$(i,p i) \cdotv row B (p i))$$ (i,q i)) ?U
=
            prod (\lambdai. A$$(i,p i)) ?U*prod (\lambdai. B$$ (i,q(?inv p i))) ?U
            unfolding th001 o-def permutes-inverses[OF p]
            by (subst prod.distrib[symmetric], insert A p q B, auto intro: prod.cong)
    define }AA\mathrm{ where }AA=(\prodi\in?U.A$$(i,pi)
    define BB where BB=(\prodia\in{0..<n}. B $$ (ia,q(?inv p ia)))
    have ?s q* (\prodia\in{0..<n}. matr n n (\lambdai. A $$ (i,p i) \cdotv row B (pi))$$
(ia,q qa)) =
            ?s p * (\prodi\in{0..<n}. A $$ (i,pi))*(?s (q\circ?inv p)* (\prodia\in{0..<n}.B
$$(ia,q(?inv p ia))))
            unfolding sign thp
            unfolding AA-def[symmetric] BB-def[symmetric]
            by (simp add: ac-simps flip: of-int-mult)
                            thus?s q* (\prodi=0..<n.matr n n (\lambdai. A $$ (i,p i) .v row B (pi))$$(i,
qi))=
            ?s p * (\prodi=0..<n. A $$ (i,pi))*
            (?s (q\circ?inv p)*(\prodi=0..<n. B$$(i,(q\circ?inv p) i))) by simp
        qed
    } note * = this
    have th2: sum ( }\lambdaf.\operatorname{det}(\mp@subsup{m}{matr}{}nn(\lambdai.A$$(i,fi)\cdotv row B (f i)))) ?PU = de
A*\operatorname{det}B
    unfolding det-def'[OF A] det-def'[OF B] det-def'[OF mat-row-carrierI]
    unfolding sum-product dim-row-mat
    by (rule sum.cong, insert A, force, subst *, insert A B, auto)
    let ?f = \lambda f. det (mat r n n ( }\lambda\mathrm{ i. A $$ (i,fi) |v row B (fi)))
    have det (A*B)=sum ?f ?F
    unfolding mat-mul-finsum-alt[OF A B]
    by (rule det-linear-rows-sum[OF fU], insert A B, auto)
    also have ... = sum ?f ((?F - ?PU) \cup (?F\cap?PU))
    by (rule arg-cong[where f= sum ?f], auto)
    also have ... = sum ?f (?F - ?PU) + sum ?f (?F \cap?PU)
    by (rule sum.union-disjoint, insert A B finite-bounded-functions[OF fU fU],
auto)
    also have sum ?f (?F - ?PU)=0
```

```
    by (rule sum.neutral, insert zth, auto)
    also have ?F \cap?PU =?PU unfolding permutes-def by fastforce
    also have sum ?f ?PU = det A*\operatorname{det}B
    unfolding th2 ..
    finally show ?thesis by simp
qed
```

lemma unit-imp-det-non-zero: assumes $A \in$ Units (ring-mat TYPE ('a :: comm-ring-1)
$n$ b)
shows $\operatorname{det} A \neq 0$
proof -
from assms[unfolded Units-def ring-mat-def]
obtain $B$ where $A: A \in$ carrier-mat $n n$ and $B: B \in$ carrier-mat $n n$ and $B A$ :
$B * A=1_{m} n$ by auto
from arg-cong[OF BA, of det, unfolded det-mult $[O F B A]$ det-one $]$
show ?thesis by auto
qed

The following proof is based on the Gauss-Jordan algorithm.
lemma det-non-zero-imp-unit: assumes $A: A \in$ carrier-mat $n n$
and $d A: \operatorname{det} A \neq(0:: ' a::$ field $)$
shows $A \in$ Units (ring-mat TYPE ('a) $n b$ )
proof (rule ccontr)
let ? $g=$ gauss-jordan $A\left(\begin{array}{lll}0_{m} & n & 0\end{array}\right)$
let $? B=f s t ? g$
obtain $B C$ where $B: ? g=(B, C)$ by (cases ?g)
assume $\neg$ ?thesis
from this[unfolded gauss-jordan-check-invertable[OF A zero-carrier-mat[of n 0]]] $B]$
have $B \neq 1_{m} n$ by auto
with row-echelon-form-imp-1-or-0-row[OF gauss-jordan-carrier(1)[OF A-B]
gauss-jordan-row-echelon[OF A B], of 0]
have $n: 0<n$ and row: row $B(n-1)=0_{v} n$ by auto
let $? n=n-1$
from $n$ have $n 1: ? n<n$ by auto
from gauss-jordan-transform $[O F A-B$, of $0 b]$ obtain $P$
where $P: P \in$ Units (ring-mat $T Y P E\left({ }^{\prime} a\right) n b$ ) and $P A: B=P * A$ by auto
from unit-imp-det-non-zero $[O F P]$ have $d P: \operatorname{det} P \neq 0$ by auto
from $P$ have $P: P \in$ carrier-mat $n n$ unfolding Units-def ring-mat-def by auto
from det-mult $[O F P A] d P d A$ have $\operatorname{det} B \neq 0$ unfolding $P A$ by simp
also have $\operatorname{det} B=0$
proof -
from gauss-jordan-carrier $[O F A-B$, of 0$]$ have $B: B \in$ carrier-mat $n n$ by auto
\{
fix $j$
assume $j: j<n$
from index-row(1)[symmetric, of ? $n B j$, unfolded row] $B$
have $B \$ \$(? n, j)=0$ using $B n j$ by auto

```
    }
    hence B= matr nn(\lambdai. if i=?? then O}\mp@subsup{O}{v}{}n\mathrm{ else row B i)
        by (intro eq-matI, insert B, auto)
    also have det \ldots. =0
        by (rule det-row-0[OF n1], insert B, auto)
    finally show det B=0.
    qed
    finally show False by simp
qed
```

lemma mat-mult-left-right-inverse: assumes $A$ : ( $A::$ ' $a$ :: field mat $) \in$ carrier-mat $n n$
and $B: B \in$ carrier-mat $n n$ and $A B: A * B=1_{m} n$
shows $B * A=1_{m} n$
proof -
let $? R=$ ring-mat TYPE ('a) $n$ undefined
from det-mult $[O F A B$, unfolded $A B]$ have $\operatorname{det} A \neq 0 \operatorname{det} B \neq 0$ by auto
from det-non-zero-imp-unit[OF A this(1)] det-non-zero-imp-unit[OF B this(2)]
have $U: A \in$ Units $? R B \in$ Units $? R$.
interpret ring ? $R$ by (rule ring-mat)
from Units-inv-comm[unfolded ring-mat-simps, $O F A B U]$ show ?thesis .
qed
lemma det-zero-imp-zero-row: assumes $A:(A:: ' a::$ field mat $) \in$ carrier-mat $n$
$n$
and $\operatorname{det}: \operatorname{det} A=0$
shows $\exists P$. $P \in$ Units (ring-mat TYPE $(' a) n b) \wedge \operatorname{row}(P * A)(n-1)=0_{v}$
$n \wedge 0<n$
$\wedge$ row-echelon-form $(P * A)$
proof -
let $? R=$ ring-mat $T Y P E(' a) n b$
let $? U=$ Units $? R$
interpret $m$ : ring ? $R$ by (rule ring-mat)
let $?$ ? $=$ gauss-jordan $A A$
obtain $A^{\prime} B^{\prime}$ where $g: ? g=\left(A^{\prime}, B^{\prime}\right)$ by (cases ? $g$ )
from det unit-imp-det-non-zero[of $A n b]$ have $A U: A \notin ? U$ by auto
with gauss-jordan-inverse-one-direction(1)[OF A A, of - b]
have $A^{\prime} 1: A^{\prime} \neq 1_{m} n$ using $g$ by auto
from gauss-jordan-carrier (1) [OF A A g] have $A^{\prime}: A^{\prime} \in \operatorname{carrier-mat} n n$ by auto
from gauss-jordan-row-echelon $[O F A g]$ have re: row-echelon-form $A^{\prime}$.
from row-echelon-form-imp-1-or-0-row[OF A' this] $A^{\prime} 1$
have $n: 0<n$ and row: row $A^{\prime}(n-1)=0_{v} n$ by auto
from gauss-jordan-transform[OF A A g, of b] obtain $P$
where $P: P \in ? U$ and $A^{\prime}: A^{\prime}=P * A$ by auto
thus? ?thesis using $n$ row re by auto
qed
lemma det-0-iff-vec-prod-zero-field: assumes $A:(A:: ~ ' a ~:: ~ f i e l d ~ m a t) ~ \in c a r r i e r-m a t ~$

```
n n
    shows det A=0\longleftrightarrow(\existsv.v\incarrier-vec n}\wedgev\not=\mp@subsup{0}{v}{}n\wedgeA*vv=\mp@subsup{0}{v}{}n)(i
?l}=(\existsv.?Pv)
proof -
    let ?R = ring-mat TYPE('a) n ()
    let ?U = Units ?R
    interpret m: ring?R by (rule ring-mat)
    show ?thesis
    proof (cases det A=0)
        case False
        from det-non-zero-imp-unit[OF A this, of ()]
    have }A\in?U\mathrm{ .
    then obtain B where unit: }B*A=1mn\mathrm{ and B: B carrier-mat n n
            unfolding Units-def ring-mat-def by auto
    {
        fix v
        assume ?P v
        hence v:v\in carrier-vec n v}\not=\mp@subsup{0}{v}{}nA*vv=\mp@subsup{0}{v}{}n\mathrm{ by auto
        have v=(B*A) *vv using vB unfolding unit by auto
        also have \ldots= B *v (A*vv) using B A v by simp
        also have \ldots= B *v 林 n unfolding v ..
        also have \ldots= 䜣 n using B by auto
        finally have False using v by simp
    }
    with False show ?thesis by blast
    next
    case True
    let ? n = n-1
    from det-zero-imp-zero-row[OF A True, of ()]
    obtain P where PU:P\in?U and row: row (P*A) ?n = 0v n and n: 0<
n? n < n
            and re: row-echelon-form ( }P*A)\mathrm{ by auto
    define PA where PA=P*A
    note row = row[folded PA-def]
    note re = re[folded PA-def]
    from PU obtain Q where P:P\incarrier-mat n n and Q:Q\incarrier-mat
nn
            and unit: Q*P=1m n P*Q = 1m n unfolding Units-def ring-mat-def
by auto
            from PA have PA: PA \incarrier-mat n n and dimPA: dim-row PA=n
unfolding PA-def by auto
    from re[unfolded row-echelon-form-def] obtain p}\mathrm{ where p: pivot-fun PA p n
using PA by auto
    note piv = pivot-positions[OF PA p]
    note pivot = pivot-funD[OF dimPA p n(2)]
    {
        assume p ?n<n
        with pivot(4)[OF this]n arg-cong[OF row, of \lambdav.v$ p?n] have False using
PA by auto
```

$$
\}
$$

with $\operatorname{pivot}(1)$ have $p n: p ? n=n$ by fastforce
with piv(1) have set (pivot-positions $P A) \subseteq\{(i, p i) \mid i . i<n \wedge p i \neq n\}-$ $\{(? n, p ? n)\}$ by auto
also have $\ldots \subseteq\{(i, p i) \mid i . i<? n\}$ using $n$ by force
finally have card (set (pivot-positions PA)) $\leq \operatorname{card}\{(i, p i) \mid i . i<? n\}$
by (intro card-mono, auto)
also have $\{(i, p i) \mid i . i<? n\}=(\lambda i .(i, p i)) \cdot\{0 . .<? n\}$ by auto
also have card $\ldots=$ card $\{0 \ldots<? n\}$ by (rule card-image, auto simp: inj-on-def)
also have $\ldots<n$ using $n$ by $\operatorname{simp}$
finally have $\operatorname{card}($ set $($ pivot-positions $P A))<n$.
hence card (snd' $($ set $($ pivot-positions PA) $))<n$
using card-image-le[OF finite-set, of snd pivot-positions PA] by auto
hence neq: snd' $($ set $($ pivot-positions $P A)) \neq\{0 . .<n\}$ by auto
from find-base-vector[OF re PA neq] obtain $v$ where $v: v \in$ carrier-vec $n$ and $v 0: v \neq 0_{v} n$ and pav: $P A *_{v} v=0_{v} n$ by auto
have $A *_{v} v=Q * P *_{v}\left(A *_{v} v\right)$ unfolding unit using $A v$ by auto
also have $\ldots=Q *_{v}\left(P A *_{v} v\right)$ unfolding $P A$-def using $Q P A v$ by auto
also have $P A *_{v} v=O_{v} n$ unfolding pav..
also have $Q *_{v} O_{v} n=0_{v} n$ using $Q$ by auto
finally have $A v: A *_{v} v=O_{v} n$ by auto
show ?thesis unfolding True using $A v v 0 v$ by auto
qed
qed
In order to get the result for integral domains, we embed the domain in its fraction field, and then apply the result for fields.
lemma det-0-iff-vec-prod-zero: assumes $A:\left(A::{ }^{\prime} a\right.$ :: idom mat $) \in$ carrier-mat $n$ $n$
shows $\operatorname{det} A=0 \longleftrightarrow\left(\exists v . v \in\right.$ carrier-vec $\left.n \wedge v \neq 0_{v} n \wedge A *_{v} v=0_{v} n\right)$
proof -
let $? h=$ to-fract $::$ ' $a \Rightarrow{ }^{\prime} a$ fract
let $? A=$ map-mat $? h A$
have $A^{\prime}: ? A \in$ carrier-mat $n n$ using $A$ by auto
interpret inj-comm-ring-hom ?h by (unfold-locales, auto)
have $(\operatorname{det} A=0)=(? h(\operatorname{det} A)=? h 0)$ by auto
also have $\ldots=(\operatorname{det} ? A=0)$ unfolding hom-zero hom-det ..
also have $\ldots=\left(\left(\exists v . v \in\right.\right.$ carrier-vec $\left.\left.n \wedge v \neq 0_{v} n \wedge ? A *_{v} v=0_{v} n\right)\right)$
unfolding det-0-iff-vec-prod-zero-field $[O F A]$..
also have $\ldots=\left(\left(\exists v . v \in\right.\right.$ carrier-vec $\left.\left.n \wedge v \neq 0_{v} n \wedge A *_{v} v=0_{v} n\right)\right)$ (is ?l $=? r$ )
proof
assume ?r
then obtain $v$ where $v: v \in$ carrier-vec $n v \neq 0_{v} n A *_{v} v=O_{v} n$ by auto show?l
by (rule exI[of-map-vec ?h v], insert v, auto simp: mult-mat-vec-hom[symmetric, OF Av(1)])
next
assume ?l
then obtain $v$ where $v: v \in$ carrier-vec $n$ and $v 0: v \neq 0_{v} n$ and $A v: ? A *_{v}$ $v=0_{v} n$ by auto
have $\forall i . \exists a b . v \$ i=$ Fraction-Field.Fract $a b \wedge b \neq 0$ using Fract-cases[of $v \$ i$ for $i]$ by metis
from choice[OF this] obtain $a$ where $\forall i . \exists b . v \$ i=$ Fraction-Field.Fract ( $a$ i) $b \wedge b \neq 0$ by metis
from choice $[$ OF this] obtain $b$ where vi: $\bigwedge i . v \$ i=$ Fraction-Field.Fract ( $a$ i) ( $b i$ ) and $b i: \bigwedge i . b i \neq 0$ by auto
define $m$ where $m=$ prod-list (map $b[0 . .<n]$ )
let $? m=$ ? $h m$
have $m 0: m \neq 0$ unfolding $m$-def hom- 0 -iff prod-list-zero-iff using bi by auto
from $v 0$ [unfolded vec-eq-iff] $v$ obtain $i$ where $i: i<n v \$ i \neq 0$ by auto \{
fix $i$
assume $i<n$
hence $b i \in \operatorname{set}(\operatorname{map} b[0 . .<n])$ by auto
from split-list[OF this]
obtain ys zs where map $b[0 . .<n]=y s @ b i \# z s$ by auto
hence $b i$ dvd $m$ unfolding $m$-def by auto
then obtain $c$ where $m=b i * c$..
hence ? $m * v \$ i=? h(a i * c)$ unfolding vi using $b i[o f i]$
by (simp add: eq-fract to-fract-def)
hence $\exists c$. ? $m * v \$ i=? h c$..
\}
hence $\forall i . \exists c . i<n \longrightarrow$ ? $m * v \$ i=? h c$ by auto
from choice[OF this] obtain $c$ where $c: \bigwedge i . i<n \Longrightarrow ? m * v \$ i=? h(c$
i) by auto
define $w$ where $w=$ vec $n c$
have $w: w \in$ carrier-vec $n$ unfolding $w$-def by simp
have mvw: ? $m \cdot{ }_{v} v=$ map-vec ? $h \mathrm{w}$ unfolding $w$-def using $c v$ by (intro eq-vecI, auto)
with m0 $i c[O F i(1)]$ have $w \$ i \neq 0$ unfolding $w$-def by auto
with $i w$ have $w 0: w \neq 0_{v} n$ by auto
from $\arg -c o n g[O F A v$, of $\lambda v . ? m \cdot v v]$
have ? $m \cdot{ }_{v}\left(? A *_{v} v\right)=$ map-vec $? h\left(O_{v} n\right)$ by auto
also have ? $m \cdot v\left(? A *_{v} v\right)=? A *_{v}(? m \cdot v v)$ using $A v$ by auto
also have $\ldots=$ ? $A *_{v}($ map-vec ? $h \mathrm{w})$ unfolding mvw ..
also have $\ldots=$ map-vec $? h\left(A *_{v} w\right)$ unfolding mult-mat-vec-hom[OF $A$ w]
finally have $A *_{v} w=0_{v} n$ by (rule vec-hom-inj)
with $w$ w show ?r by blast
qed
finally show ?thesis .
qed
lemma det-0-negate: assumes $A:\left(A::{ }^{\prime} a::\right.$ field mat $) \in$ carrier-mat $n n$ shows $(\operatorname{det}(-A)=0)=(\operatorname{det} A=0)$
proof -
from $A$ have $m A:-A \in$ carrier-mat $n n$ by auto

```
{
    fix v:: 'a vec
    assume v: v\in carrier-vec n
    hence Av: A *vv v\incarrier-vec n using A by auto
    have id: - A *vv v = - (A *vv) using vA by simp
    have (-A*vv = 价 n)=( (A*vv=\mp@subsup{0}{v}{}n) unfolding id
        unfolding uminus-zero-vec-eq[OF Av] ..
    }
    thus ?thesis unfolding det-0-iff-vec-prod-zero[OF A] det-0-iff-vec-prod-zero[OF
mA] by auto
qed
lemma det-multrow:
    assumes k: k<n and A:A\incarrier-mat n n
    shows det (multrow k a A) =a* det A
proof -
    have multrow k a A = multrow-mat n k a*A
    by (rule multrow-mat[OF A])
```



```
    by (rule det-mult[OF - A],auto)
    also have det (multrow-mat nka)=a
    by (rule det-multrow-mat[OF k])
    finally show ?thesis.
qed
lemma det-multrow-div:
    assumes k:k<n and A:A\incarrier-mat n n and a0:a\not=0
    shows det (multrow k a A :: 'a :: idom-divide mat) div a = det A
proof -
    have det (multrow k a A) div a =a* det A div a using k A
    by (simp add: det-multrow)
    also have ... = det A using a0 by auto
    finally show ?thesis.
qed
lemma det-addrow:
    assumes l:l<n and k:k\not=l and A:A\incarrier-mat n n
    shows det (addrow a klA) = det A
proof -
    have addrow a kl A = addrow-mat n a kl*A
    by (rule addrow-mat[OF A l])
    also have det (addrow-mat n a kl*A)=\operatorname{det (addrow-mat n a kl)* det A}
    by (rule det-mult[OF - A], auto)
    also have det (addrow-mat n a k l)=1
    by (rule det-addrow-mat[OF k])
    finally show ?thesis using A by simp
qed
lemma det-swaprows:
```

```
    assumes *: k<nl<n and k: k\not=l and A:A\incarrier-mat n n
    shows det (swaprows klA) = - det A
proof -
    have swaprows kl A = swaprows-mat n kl*A
    by (rule swaprows-mat[OF A *])
```



```
    by (rule det-mult[OF - A], insert A, auto)
    also have det (swaprows-mat nkl)=-1
    by (rule det-swaprows-mat[OF*k])
    finally show ?thesis using A by simp
qed
lemma det-similar: assumes similar-mat A B
    shows }\operatorname{det}A=\operatorname{det}
proof -
    from similar-matD[OF assms] obtain n P Q where
    carr: {A,B,P,Q}\subseteqcarrier-mat n n (is - \subseteq?C)
    and PQ:P*Q=1 m}
    and AB:A=P*B*Q by blast
    hence A:A\in?C and B:B\in?C and P:P\in?C and Q:Q\in?C by auto
    from det-mult[OF P Q, unfolded PQ] have PQ: det P*\operatorname{det}Q=1 by auto
    from det-mult[OF - Q, of P*B, unfolded det-mult[OF P B] AB[symmetric]] P
B
    have }\operatorname{det}A=\operatorname{det}P*\operatorname{det}B*\operatorname{det}Q\mathrm{ by auto
    also have \ldots=( det P*\operatorname{det}Q)*\operatorname{det}B\mathrm{ by (simp add: ac-simps)}
    also have \ldots= det B unfolding PQ by simp
    finally show ?thesis.
qed
lemma det-four-block-mat-upper-right-zero-col: assumes A1: A1 \(\in\) carrier-mat \(n\) \(n\)
    and A20:A2 = (0m n 1) and A3:A3 \in carrier-mat 1 n
    and A4:A4 \in carrier-mat 11
    shows det (four-block-mat A1 A2 A3 A4) = det A1 * det A4 (is det ?A = -)
proof -
    let ?A = four-block-mat A1 A2 A3 A4
    from A20 have A2: A2 \in carrier-mat n 1 by auto
    define }A\mathrm{ where }A=\mathrm{ ?A
    from four-block-carrier-mat[OF A1 A4] A1
    have A:A\in carrier-mat (Suc n) (Suc n) and dim:dim-row A1 = n unfolding
A-def by auto
    let ?Pn = \lambda p.p permutes {0 ..<n}
    let ?Psn = \lambda p.p permutes {0..< Suc n}
    let ?perm}={p.?Psn p
    let ?permn = {p. ?Pn p}
    let ?prod = \lambda p. signof p*(\prodi=0..<Suc n. A $$ (pi,i))
    let ?prod' = \lambda p. A$$(pn, n)* signof p * (\prodi=0..<n. A $$ (pi,i))
    let ?prod" = 人 p. signof p * (\prodi=0..<n.A$$(pi,i))
    let ?prod"'\prime }=\lambda\mathrm{ p. signof p* (Пi=0..<n. A1 $$ (pi,i))
```

```
let ?p0 = {p.p 0=0}
have [simp]:{0..<Suc n}-{n}={0..<n} by auto
{
    fix }
    assume ?Psn p
    have ?prod p= signof p*(A$$(pn,n)*(\prodi\in{0..<n}.A$$(pi,i)))
        by (subst prod.remove[of - n], auto)
    also have \ldots=A$$(pn,n)* signof p*(\prodi\in{0..<n}.A$$ (pi,i)) by
simp
    finally have ?prod p = ?prod' }p
    } note prod-id = this
    define prod' where prod}\mp@subsup{}{}{\prime}=?\mathrm{ ?prod'
    {
    fix iq
    assume i:i\in{0..<n} q permutes {0 ..< n}
    hence Fun.swap n i id (qn)<n
        unfolding permutes-def by auto
    hence A $$ (Fun.swap n i id (q n), n)=0
        unfolding A-def using A1 A20 A3 A4 by auto
    hence prod'(Fun.swap n i id \circ q) =0
        unfolding prod'-def by simp
    } note zero = this
    have cong: \ a b c. b=c\Longrightarrowa*b=a*c by auto
    have det ?A = sum ?prod ?perm
    unfolding A-def[symmetric] using mat-det-left-def[OF A] A by simp
also have .. = sum prod' ?perm unfolding prod'-def
    by (rule sum.cong[OF refl], insert prod-id, auto)
also have {0 ..< Suc n} = insert n {0 ..< n} by auto
also have sum prod}\mp@subsup{}{}{\prime}{p.p\mathrm{ permutes ...}=
    (\sumi\ininsert n {0..<n}. \sumq\in?permn.prod'(Fun.swap n i id \circq))
    by (subst sum-over-permutations-insert, auto)
also have ... = (\sumq\in?permn. prod' q) +
    (\sumi\in{0..<n}. \sumq\in?permn. prod'(Fun.swap n i id ○ q))
    by (subst sum.insert, auto)
also have (\sumi\in{0..<n}. \sumq\in?permn. prod'(Fun.swap n i id \circ q))=0
    by (rule sum.neutral, intro ballI, rule sum.neutral, intro ballI, rule zero, auto)
also have (\sumq\in ?permn. prod'}q)=A$$(n,n)*(\sumq\in\mathrm{ ?permn. ?prod"/ q)
    unfolding prod'-def
        by (subst sum-distrib-left, rule sum.cong[OF refl], auto simp: permutes-def
ac-simps)
    also have A $$ (n,n)=A4$$(0,0) unfolding A-def using A1 A2 A3 A4 by
auto
    also have (\sumq\in ?permn. ?prod" }q)=(\sumq\in\mathrm{ ?permn. ?prod"'/ q)
        by (rule sum.cong[OF refl], rule cong, rule prod.cong,
        insert A1 A2 A3 A4, auto simp: permutes-def A-def)
    also have ... = det A1
        unfolding mat-det-left-def[OF A1] dim by auto
    also have A4 $$ (0,0) = det A4
    using A4 unfolding det-def[of A4] by (auto simp: sign-def)
```

```
    finally show ?thesis by simp
qed
lemma det-swap-initial-rows: assumes A:A carrier-mat m m
    and lt: k+n\leqm
    shows det A = (-1)^ (k*n)*
    det (mat m m (\lambda(i,j). A $$ (if i<n then i + k else if i<k+n then i-n
else i,j)))
proof -
    define sw where sw = ( }\lambda\mathrm{ ( A :: 'a mat) xs. fold ( }\lambda(i,j). swaprows i j) xs A)
    have dim-sw[simp]: dim-row (sw A xs) = dim-row A dim-col (sw A xs) = dim-col
A for xs A
    unfolding sw-def by (induct xs arbitrary: A, auto)
{
    fix xs and }A\mathrm{ :: 'a mat
    assume dim-row }A=\operatorname{dim}-col A \ij. (i,j)\in set xs \Longrightarrowi<dim-col A ^j<
dim-col }A\wedgei\not=
    hence det (sw A xs) = (-1)^(length xs) * det A
        unfolding sw-def
    proof (induct xs arbitrary: A)
        case (Cons xy xs A)
        obtain x y where xy: xy = (x,y) by force
        from Cons(3)[unfolded xy, of x y] Cons(2)
        have [simp]: det (swaprows x y A) = - det A
            by (intro det-swaprows, auto)
    show ?case unfolding xy by (simp, insert Cons(2-), (subst Cons(1), auto)+)
    qed simp
} note sw = this
define swb where swb = (\lambdaA in.sw A (map (\lambda j. (j,Suc j)) [i..<i+n]))
{
    fix }kn\mathrm{ and }A:: 'a ma
    assume k-n:k+n< dim-row A
    hence swb A k n= mat (dim-row A) (dim-col A) ( }\lambda(i,j). let r =
        (if i<k\veei>k+n then i else if i=k+n then k else Suc i)
        in A $$ (r,j))
    proof (induct n)
        case 0
        show ?case unfolding swb-def sw-def by (rule eq-matI, auto)
    next
        case (Suc n)
        hence dim: }k+n<\mathrm{ dim-row A by auto
            have id: swb A k (Suc n) = swaprows (k+n) (Suc k + n) (swb A k n)
unfolding swb-def sw-def by simp
        show ?case unfolding id Suc(1)[OF dim]
            by (rule eq-matI, insert Suc(2), auto)
    qed
} note swb = this
define swbl where swbl =( }\lambda\mathrm{ A k n. fold ( }\lambdaiA.\operatorname{swb}A\mathrm{ i n) (rev [0 ..<k]) A)
{
```

```
    fix }kn\mathrm{ and A :: 'a mat
    assume k-n:k+n\leqdim-row A
    hence swbl A kn=mat (dim-row A) (dim-col A) ( }\lambda(i,j). let r =
        (if i<n then }i+k\mathrm{ else if }i<k+n\mathrm{ then }i-n\mathrm{ else }i\mathrm{ )
        in A $$ (r,j))
    proof (induct k arbitrary:A)
        case 0
        thus ?case unfolding swbl-def by (intro eq-matI, auto simp: swb)
    next
        case (Suc k)
        hence dim: }k+n<\mathrm{ dim-row A by auto
        have id: swbl A (Suc k) n = swbl (swb A k n) kn unfolding swbl-def by
simp
            show ?case unfolding id swb[OF dim]
            by (subst Suc(1), insert dim, force, intro eq-matI, auto simp: less-Suc-eq-le)
    qed
    } note swbl=this
    {
    fix }kn\mathrm{ and }A:: ' a mat
    assume k-n: k+n\leqdim-col A dim-row A = dim-col A
    hence det (swbl A kn)=(-1)^(k*n)*\operatorname{det}A
    proof (induct k arbitrary: A)
            case 0
            thus ?case unfolding swbl-def by auto
    next
            case (Suc k)
            hence dim: }k+n<\mathrm{ dim-row A by auto
            have id: swbl A (Suc k) n= swbl (swb A k n) kn unfolding swbl-def by
simp
            have det: det (swb A kn)=(-1)^n * det A unfolding swb-def
                by (subst sw, insert Suc(2-), auto)
            show ?case unfolding id
            by (subst Suc(1), insert Suc(2-), auto simp: det, auto simp: swb power-add)
    qed
    } note det-swbl = this
    from assms have dim: dim-row A = dim-col A k + n\leqdim-col A k+n\leq
dim-row A dim-col }A=m\mathrm{ by auto
    from arg-cong[OF det-swbl[OF dim(2,1), unfolded swbl[OF dim(3)], unfolded
Let-def dim],
            of \lambdax.(-1)^(k*n)*x]
    show ?thesis by simp
qed
lemma det-swap-rows: assumes A:A f carrier-mat (k+n) (k+n)
    shows }\operatorname{det}A=(-1)`(k*n)*\operatorname{det}(\mathrm{ mat }(k+n)(k+n)(\lambda(i,j)
        A$$((if i<k then i}+n\mathrm{ else }i-k),j))
proof -
    have le: n+k\leqk+n by simp
```

show ?thesis unfolding det-swap-initial-rows $[$ OF A le]
by (intro arg-cong2[of $-\cdots \lambda x y .((-1) \widehat{x} * \operatorname{det} y)]$, force, intro eq-matI, auto)
qed
lemma det-swap-final-rows: assumes $A: A \in$ carrier-mat $m m$
and $m: m=l+k+n$
shows $\operatorname{det} A=(-1)^{\wedge}(k * n) *$
$\operatorname{det}($ mat $m m(\lambda(i, j) . A \$ \$($ if $i<l$ then $i$ else if $i<l+n$ then $i+k$ else $i$ $-n, j))$ )
(is $-=-* \operatorname{det} ? M)$
proof -
have $m 1: m=(k+n)+l$ using $m$ by simp
have m2: $k+n \leq m$ using $m$ by simp
have $m 3: m=l+(n+k)$ using $m$ by $\operatorname{simp}$
define $M$ where $M=$ ? $M$
let ?M1 $=$ mat $m m(\lambda(i, j)$. A $\$ \$$ (if $i<k+n$ then $i+l$ else $i-(k+n), j))$
let ? M2 $=$ mat $m \mathrm{~m}$
$(\lambda(i, j) . A \$ \$($ if $i<n$ then $i+k+l$ else if $i<k+n$ then $i-n+l$ else $i-(k+n), j))$
have M2: ?M2 $\in$ carrier-mat $m$ by auto
have $\operatorname{det} A=(-1)^{\wedge}((k+n) * l) * \operatorname{det} ? M 1$
unfolding det-swap-rows[OF A[unfolded m1]] m1[symmetric] by simp
also have det ? M1 $=(-1)^{\wedge}(k * n) * \operatorname{det}$ ?M2
by (subst det-swap-initial-rows[OF - m2], force, rule arg-cong[of - $\lambda$. - * det $x]$,
rule eq-matI, auto simp: m)
also have det? ${ }^{\text {M2 }}=(-1) \wedge(l *(n+k)) * \operatorname{det} M$
unfolding $M$-def det-swap-rows[OF M2[unfolded m3], folded m3]
by (rule arg-cong[of $-\lambda x$. $-*$ det $x]$, rule eq-matI, auto simp: m)
finally have $\operatorname{det} A=(-1)^{\wedge}((k+n) * l+(k * n)+l *(n+k)) * \operatorname{det} M$ (is $\left.-=? b{ }^{\wedge}-*-\right)$
by (simp add: power-add)
also have $(k+n) * l+(k * n)+l *(n+k)=2 *(l *(n+k))+k * n$ by simp
also have ?b ^ $\ldots=? b$ ^ $(k * n)$ by (simp add: power-add)
finally show ?thesis unfolding $M$-def .
qed
lemma det-swap-final-cols: assumes $A: A \in$ carrier-mat $m m$
and $m: m=l+k+n$
shows $\operatorname{det} A=(-1) \wedge(k * n) *$
$\operatorname{det}($ mat $m m(\lambda(i, j) . A \$ \$(i$, if $j<l$ then $j$ else if $j<l+n$ then $j+k$ else $j-n)$ )
proof -
have $\operatorname{det} A=\operatorname{det}\left(A^{T}\right)$ unfolding det-transpose[OF A]..
also have $\ldots=(-1)^{\wedge}(k * n) *$
det (mat $m m\left(\lambda(i, j) . A^{T} \$ \$\right.$ (if $i<l$ then $i$ else if $i<l+n$ then $i+k$ else $i$

```
- n,j)))
    (is - = - * det ?M)
    by (rule det-swap-final-rows[OF - m], insert A, auto)
    also have det ?M = det (?M}\mp@subsup{M}{}{T})\mathrm{ by (subst det-transpose, auto)
    also have ? M}\mp@subsup{M}{}{T}=\mathrm{ mat mm( m(i,j).A$$(i, if j<l then j else if j<l+n
then j + k else j - n))
    unfolding transpose-mat-def using A m
    by (intro eq-matI, auto)
    finally show ?thesis.
qed
lemma det-swap-initial-cols: assumes A: A \in carrier-mat m m
    and lt:k+n\leqm
    shows det A = (-1)^ (k*n)*
        det (mat mm (\lambda(i,j). A $$ (i, if j<n then j + k else if j<k+n then j -
n else j)))
proof -
    have }\operatorname{det}A=\operatorname{det}(\mp@subsup{A}{}{T})\mathrm{ unfolding det-transpose[OF A]..
    also have ... = (-1)^ (k*n)*
        det (mat mm ( }\lambda(j,i).\mp@subsup{A}{}{T}$$(\mathrm{ if }j<n\mathrm{ then }j+k\mathrm{ else if j<k+n then j - n
else j,i)))
    (is - = -* det ?M)
    by (rule det-swap-initial-rows[OF-lt], insert A, auto)
    also have det ?M = det (?M }\mp@subsup{M}{}{T})\mathrm{ by (subst det-transpose, auto)
    also have ? M }\mp@subsup{M}{}{T}=\mathrm{ mat mm( 
n then j - n else j))
    unfolding transpose-mat-def using A lt
    by (intro eq-matI, auto)
    finally show ?thesis.
qed
lemma det-swap-cols: assumes A: A G carrier-mat (k+n) (k+n)
    shows det A = (-1)^(k*n)*\operatorname{det}(\operatorname{mat}(k+n)(k+n)(\lambda(i,j).
    A$$(i,(if j<k then j + n else j - k)))) (is - = - * det ?B)
proof -
    have le: n+k\leqk+n by simp
    show ?thesis unfolding det-swap-initial-cols[OF A le]
        by (intro arg-cong2[of - - \lambda x y.((-1)^x * det y)], force, intro eq-matI,
auto)
qed
lemma det-four-block-mat-upper-right-zero: fixes A1 :: 'a :: idom mat
    assumes A1:A1 \in carrier-mat n n
    and A20:A2 = (0m n m) and A3:A3 }\in\mathrm{ carrier-mat m n
    and A4:A4 }\in\mathrm{ carrier-mat m m
shows det (four-block-mat A1 A2 A3 A4) = det A1 * det A4
    using assms(2-)
proof (induct m arbitrary: A2 A3 A4)
    case (0 A2 A3 A4)
```

```
    hence *: four-block-mat A1 A2 A3 A4 = A1 using A1
    by (intro eq-matI, auto)
    from 0 have \(4: A 4=1_{m} 0\) by auto
    show ? case unfolding \(*\) unfolding 4 by simp
next
    case (Suc m A2 A3 A4)
    let \(? m=\) Suc \(m\)
    from Suc have A2: A2 \(\in\) carrier-mat \(n ? m\) by auto
    note \(A 20=S u c(2)\)
    note \(A 34=\operatorname{Suc}(3-4)\)
    let ? A = four-block-mat A1 A2 A3 A4
    let ? \(P=\lambda B 3 B 4 v k . v \neq 0 \wedge v * \operatorname{det} ? A=\operatorname{det}(f o u r-b l o c k-m a t A 1 A 2 B 3 B 4)\)
    \(\wedge v * \operatorname{det} A_{4}=\operatorname{det} B_{4} \wedge B 3 \in\) carrier-mat \(? m n \wedge B 4 \in\) carrier-mat \(? m ? m\)
\(\wedge(\forall i<k . B 4 \$ \$(i, m)=0)\)
    have \(k \leq m \Longrightarrow \exists B 3\) B4 v. ?P B3 B4 \(v k\) for \(k\)
    proof (induct \(k\) )
        case 0
        have ? P A3 A4 10 using A34 by auto
        thus ?case by blast
    next
        case (Suc k)
    then obtain B3 B4 \(v\) where \(v: v \neq 0\) and det: \(v * \operatorname{det} ? A=\)
        \(\operatorname{det}\left(\right.\) four-block-mat A1 A2 B3 B4) \(v * \operatorname{det} A 4=\operatorname{det} B_{4}\)
        and B3: B3 \(\in\) carrier-mat ?m \(n\) and \(B_{4}: B_{4} \in\) carrier-mat \(? m ? m\) and 0 :
\(\forall i<k . B 4 \$ \$(i, m)=0\) by auto
    show ? case
    proof (cases B4 \(\$ \$(k, m)=0)\)
        case True
        with 0 have \(0: \forall i<S u c k\). B4 \(\$ \$(i, m)=0\) using less-Suc-eq by auto
        with \(v\) det B3 B4 have ?P B3 B4 \(v(\) Suc \(k\) ) by auto
        thus ?thesis by blast
    next
        case Bk: False
        let \(? k=\) Suc \(k\)
        from Suc(2) have \(k\) : \(k<\) ? \(m\) Suc \(k<\) ? m \(k \neq\) Suc \(k\) by auto
        show ?thesis
        proof (cases B4 \(\$ \$(? k, m)=0)\)
            case True
            let ? \(B_{4}=\) swaprows \(k(\) Suc \(k) B 4\)
            let ? \(:\) B \(=\) swaprows \(k(\) Suc \(k)\) B3
            let ? \(B=\) four-block-mat A1 A2 ?B3 ?B4
            let ? \(v=-v\)
                from det-swaprows[OF \(k\) B4] det have \(\operatorname{det1:~?~} v * \operatorname{det} A_{4}=\operatorname{det}\) ?B4 by
simp
        from \(v\) have \(v: ? v \neq 0\) by auto
        from \(B 3\) have \(B 3^{\prime}:\) ? B3 \(\in\) carrier-mat ? \(m n\) by auto
        from \(B 4\) have \(B 4^{\prime}: ~ ? B 4 \in\) carrier-mat ? \(m\) ? \(m\) by auto
            have ?v * det ? A = - det (four-block-mat A1 A2 B3 B4) using det by
simp
```

also have $\ldots=\operatorname{det}$ (swaprows $(n+k)(n+? k)$ (four-block-mat A1 A2 B3 B4))
by (rule sym, rule det-swaprows $[$ of $-n+$ ? m], insert A1 A2 B3 B4 $k$, auto)
also have swaprows $(n+k)(n+? k)($ four-block-mat A1 A2 B3 B4) $)=$ ? B
proof (rule eq-matI, unfold index-mat-four-block index-mat-swaprows, goal-cases)
case (1ij)
show ?case
proof (cases $i<n$ )
case True
thus ?thesis using 1(2) A1 A2 B3 B4 by simp
next
case False
hence $i=n+(i-n)$ by simp
then obtain $d$ where $i=n+d$ by blast
thus ?thesis using 1 A1 A2 B3 B4 $k$ (2) by simp
qed
qed auto
finally have $\operatorname{det} 2: ? v * \operatorname{det} ? A=\operatorname{det} ? B$.
from True 0 B4 $k$ (2) have $\forall i<S u c k$. ?B4 $\$ \$(i, m)=0$ unfolding less-Suc-eq by auto
with det1 det2 B3' B4'v have ?P ?B3 ?B4 ?v (Suc k) by auto
thus ?thesis by blast
next
case False
let $? b k=B 4 \$ \$(? k, m)$
let $? b=B 4 \$ \$(k, m)$
let $? v=v * ? b k$
let ?B3 $=$ addrow $(-$ ?b) $k ? k$ (multrow $k$ ? $b k$ B3)
let ? $B_{4}=$ addrow $(-? b) k ? k$ ( multrow $k ? b k B 4$ )
have $*: \operatorname{det}$ ? $B_{4}=? b k * \operatorname{det} B_{4}$
by (subst det-addrow[OF $k(2-3)]$, force simp: B4, rule det-multrow[OF k(1) B4])
with $\operatorname{det}(2)[$ symmetric $]$ have $\operatorname{det2:}: ? v * \operatorname{det} A_{4}=\operatorname{det} ? B 4$ by (auto simp: ac-simps)
from $0 k(2) B 4$ have $0: \forall i<S u c k$. ?B4 $\$ \$(i, m)=0$ unfolding less-Suc-eq by auto
from False $v$ have $v: ? v \neq 0$ by auto
from $B 3$ have $B 3^{\prime}: ~ ? B 3 \in$ carrier-mat $? m n$ by auto
from $B 4$ have $B_{4}^{\prime}$ : ? $B_{4} \in$ carrier-mat ?m ?m by auto
let ? $B^{\prime}=$ multrow $(n+k)$ ?bk (four-block-mat A1 A2 B3 B4)
have $B^{\prime}: ? B^{\prime} \in$ carrier-mat $(n+? m)(n+? m)$ using $A 1 A 2 B 3 B 4 k$ by auto
let ?B $=$ four-block-mat A1 A2 ?B3 ?B4
have ? $v * \operatorname{det} ? A=? b k * \operatorname{det}($ four-block-mat A1 A2 B3 B4) using det by simp
also have $\ldots=\operatorname{det}\left(\right.$ addrow $(-? b)(n+k)(n+? k)$ ? $\left.B^{\prime}\right)$
by (subst det-addrow[OF - B ], insert $k$ (2), force, force, rule sym, rule det-multrow[of-n + ? $m$ ],

```
            insert A1 A2 B3 B4 k, auto)
            also have addrow (-?b) (n+k) (n+?k)?.B'=?B
            proof (rule eq-matI, unfold index-mat-four-block index-mat-multrow in-
dex-mat-addrow, goal-cases)
            case (1 i j)
            show ?case
            proof (cases i<n)
            case True
            thus ?thesis using 1(2) A1 A2 B3 B4 by simp
            next
                    case False
                    hence }i=n+(i-n) by sim
                    then obtain d}\mathrm{ where }i=n+d\mathrm{ by blast
                    thus ?thesis using 1A1 A2 B3 B4 k(2) by simp
                    qed
            qed auto
            finally have det1:?v* det ?A= det ?B .
            from det1 det2 B3' B4'v 0 have ?P ?B3 ?B4 ?v (Suc k) by auto
            thus ?thesis by blast
        qed
    qed
qed
from this[OF le-refl] obtain B3 B4 v where P: ?P B3 B4 v m by blast
let ?B = four-block-mat A1 A2 B3 B4
from P have v:}v\not=0\mathrm{ and det: v* det ?A= det ? B v* det A4 = det B4
    and B3: B3 \in carrier-mat ?m n and B4:B4 \in carrier-mat ?m ?m and 0: ^
i. i<m\LongrightarrowB4$$ (i,m)=0
    by auto
let ?A2 = Om n m
let ?A3 = mat mn( }\lambda\textrm{ij}.\textrm{B3}$$ ij
let ?A4 = mat mm( }\lambda\textrm{i}i.\textrm{B}, $$ ij
let ?B1 = four-block-mat A1 ?A2 ?A3 ?A4
let ?B2 = 0m}(n+m)
let ?B3 = mat 1 ( }n+m)(\lambda(i,j). if j<n then B3 $$ (m,j) else B4 $$ (m,j -
n))
    let ?B4 = mat 11 ( }\lambda\mathrm{ -. B4 $$ (m,m))
    have B44: B4 = four-block-mat ?A4 (0m m 1) (mat 1 m (\lambda (i,j). B4 $$ (m,j)))
?B4
proof (rule eq-matI, unfold index-mat-four-block dim-col-mat dim-row-mat, goal-cases)
    case (1 i j)
    hence [simp]: \negi<m\Longrightarrowi=m\negj<m\Longrightarrowj=m by auto
    from 1 show ?case using B4 0 by auto
    qed (insert B4, auto)
    have ?B = four-block-mat ?B1 ?B2 ?B3 ?B4
proof (rule eq-matI, unfold index-mat-four-block dim-col-mat dim-row-mat, goal-cases)
    case (1 i j)
    then consider (UL) i<n+mj<n+m|(UR)i<n+mj=n+m
                | (LL) i=n +mj<n+m|(LR)i=n+mj=n+musing A1 by
auto linarith
```

```
    thus ?case
    proof cases
    case UL
    hence [simp]: ᄀi<n\Longrightarrowi-n<m
        \neg j < n \Longrightarrow j - n < m \neg j < n \Longrightarrow j - n < S u c ~ m ~ b y ~ a u t o
    from UL show ?thesis using A1 A20 B3 B4 by simp
    next
    case LL
    hence [simp]: }\negj<n\Longrightarrowj-n<m\negj<n\Longrightarrowj-n<Suc m by aut
    from LL show ?thesis using A1 A2 B3 B4 by simp
    next
    case LR
    thus ?thesis using A1 A2 B3 B4 by simp
    next
    case UR
    hence [simp]: \negi<n\Longrightarrow \ i-n<m by auto
    from UR show ?thesis using A1 A20 0 B3 B4 by simp
    qed
    qed (insert B4, auto)
    hence det ?B = det (four-block-mat ?B1 ?B2 ?B3 ?B4) by simp
    also have ... = det ?B1 * det ?B4
    by (rule det-four-block-mat-upper-right-zero-col[of-n + m], insert A1 A2 B3
B4, auto)
    also have det ?B1 = det A1 * det (mat mm(($$)B4))
        by (rule Suc(1), insert B3 B4, auto)
    also have \ldots. * det ?B4 = det A1 * (det (mat m m (($$) B4)) * det ?B4) by
simp
    also have det (mat m m (($$) B4))* det ?B4 = det B4
        unfolding arg-cong[OF B44, of det]
    by (subst det-four-block-mat-upper-right-zero-col[OF - refl],auto)
    finally have id: det ?B = det A1 * det B4.
    from this[folded det] have v* det?A=v*(\operatorname{det A1 * det A4) by simp}
    with}v\mathrm{ show det ?A = det A1* det A4 by simp
qed
lemma det-four-block-mat-lower-left-zero: fixes A1 :: 'a :: idom mat
    assumes A1: A1 \in carrier-mat n n
    and A2:A2 \in carrier-mat nm and A30:A3 = 0mmn
    and A4:A4 \in carrier-mat m m
shows det (four-block-mat A1 A2 A3 A4) = det A1 * det A4
proof -
    have A3: A3 \in carrier-mat m n using A30 by auto
    show ?thesis
        apply (subst det-transpose[OF four-block-carrier-mat[OF A1 A4], symmetric])
        apply (subst transpose-four-block-mat[OF A1 A2 A3 A4])
        apply (subst det-four-block-mat-upper-right-zero[of-n -m],
            insert A1 A2 A30 A4, auto simp: det-transpose)
        done
qed
```

```
context
begin
private lemma det-four-block-mat-preliminary: assumes }A\mathrm{ : ( }A\mathrm{ :: ' ' a :: idom mat)
\in carrier-mat n n
    and B:B\incarrier-mat n n
    and C:C\incarrier-mat n n
    and D:D\incarrier-mat n n
    and commute: C*D=D*C
    and detD: det D\not=0
shows det (four-block-mat A B C D) = det (A*D - B*C)
proof -
    let ?m = n+n
    let ?M = four-block-mat A B CD
    let ?N = four-block-mat D (0m n n) (-C) (1m n)
    have M: ?M \in carrier-mat ?m ?m using A B C D by auto
    have N:?N \in carrier-mat ?m ?m using A B C D by auto
    have det ?M * det ?N = det (?M * ?N) using det-mult[OF M N] ..
    also have ?M* ?N = four-block-mat }(A*D-B*C)B(0m n n) D
        by (subst mult-four-block-mat[OF A B C D D], insert A B C D, auto simp:
commute)
    also have det \ldots= det (A*D - B*C)*\operatorname{det}D
    by (rule det-four-block-mat-lower-left-zero, insert A B C D, auto)
    also have det ?N N = det D
    by (subst det-four-block-mat-upper-right-zero[OF D], insert C, auto)
    finally have det ?M*\operatorname{det}D=\operatorname{det}(A*D-B*C)*\operatorname{det}D.
    with detD show ?thesis by simp
qed
lemma det-four-block-mat: assumes A:(A :: 'a :: idom mat) \in carrier-mat n n
    and B:B\incarrier-mat n n
    and C:C\in carrier-mat n n
    and D:D\in carrier-mat n n
    and commute: C*D=D*C
shows det (four-block-mat A B C D) = det (A*D-B*C)
proof (cases n=0)
    case True
    hence four-block-mat A B C D=A*D - B*C using four-block-carrier-mat[OF
A D, of B C]
    A B C D by auto
    thus?thesis by simp
next
    case n: False
    define l where l= map-mat ( }\lambdax::' 'a. [: x :])
    have coeff:coeff [: x :] n= (if n=0 then x else 0) for }x:: ' a and 
        by (simp add: coeff-eq-0)
    have l-mult:l }(A*B)=lA*lB\mathrm{ if }A\in\mathrm{ carrier-mat n n B carrier-mat n n
for A B :: 'a mat
    unfolding l-def using that
```

apply (intro eq-matI, auto simp: scalar-prod-def coeff-sum intro!: poly-eqI) subgoal for $i j n$
by (cases $n$, auto simp: coeff ac-simps)
done
let ?p0 $=$ map-mat $(\lambda p$. poly $p(0:: ' a))$
have poly-det: poly $(\operatorname{det} A) 0=\operatorname{det}(? p A)$ if $A \in$ carrier-mat $n n$ for $A n$
apply (subst (1 2) det-def', use that in force, rule that)
unfolding poly-sum poly-mult poly-prod using that
by (intro sum.cong[OF refl] arg-cong2[of - - (*)] prod.cong[OF refl], auto
simp: sign-def)
let $? A=l A$
let $? B=l B$
let ? $C=l C$
let ? $D=l D$
let ? $D x=$ ? $D+$ monom $11 \cdot m 1_{m} n$
from $A B C D$
have
lA: ? $A \in$ carrier-mat $n n$ and
lB: ? $B \in$ carrier-mat $n n$ and
$l C: ? C \in$ carrier-mat $n n$ and
$l D: ? D \in$ carrier-mat $n n$ and
lDx: ? $D x \in$ carrier-mat $n n$
unfolding $l$-def by auto
have ? $C * ? D x=? C * ? D+? C *\left(\right.$ monom $\left.11 \cdot m 1_{m} n\right)$
by (subst mult-add-distrib-mat[OF lC lD], auto)
also have ? $C * ? D=l(C * D)$ using $l$-mult $[O F C D]$ by simp
also have $\ldots=l(D * C)$ using commute by auto
also have $\ldots=? D * ? C$ using $l$-mult $[O F D C]$ by simp
also have ? $C *\left(\right.$ monom $\left.11 \cdot m 1_{m} n\right)=\left(\right.$ monom $\left.11 \cdot_{m} 1_{m} n\right) * ? C$ using $l C$ by auto
also have $? D * ? C+\ldots=? D x * ? C$
by (subst add-mult-distrib-mat $[O F l D-l C]$, auto)
finally have comm: ? $C * ? D x=? D x * ? C$.
have det (four-block-mat ?A ?B ? $C$ ? $D x$ ) $=$
$\operatorname{det}(? A * ? D x-? B * ? C)$
proof (rule det-four-block-mat-preliminary[OF lA lB lC lDx comm])
define $f$ where $f=\left(\lambda p\right.$. of-int $(\operatorname{sign} p) *\left(\prod i=0 . .<n .(l D+\right.$ monom 11
-m $\left.\left.\left.1_{m} n\right) \$ \$(i, p i)\right)\right)$
have deg: degree ([:D \$\$ ij:] + monom 1 (Suc 0)) = 1 for $i j$
by (subst degree-add-eq-right, auto simp: degree-monom-eq)
have deg-id: degree $(f i d)=n$ unfolding $f$-def
apply (simp, subst degree-prod-eq-sum-degree, insert D, auto simp: l-def deg)
subgoal for $i$ using $\operatorname{deg}[$ of $(i, i)]$ by auto done
have degree $(\operatorname{det} ? D x)=\operatorname{degree}($ sum $f\{p . p$ permutes $\{0 . .<n\}\})$
by (subst det-def', use lD in force, auto simp: $f$-def)
also have sum $f\{p . p$ permutes $\{0 . .<n\}\}=f$ id $+\operatorname{sum} f(\{p . p$ permutes $\{0 . .<n\}\}-\{i d\})$
by (rule sum.remove, auto intro: finite-permutations)
also have degree $\ldots=$ degree ( $f$ id)
proof (rule degree-add-eq-left, unfold deg-id,
rule le-less-trans $[$ of $-n-1]$, rule degree-sum-le, force intro: finite-permutations)
fix $p$
assume $p: p \in\{p$. p permutes $\{0 . .<n\}\}-\{i d\}$
then obtain $i$ where $i: i \in\{0 . .<n\}$ and $p i: p i \neq i$
by (metis Diff-iff mem-Collect-eq permutes-empty permutes-superset singletonI)
from $p i$ have pin: $p i<n$ by auto
define $g$ where $g=\left(\lambda i .\left(l D+\right.\right.$ monom $\left.\left.11 \cdot m 1_{m} n\right) \$ \$(i, p i)\right)$
have degree $(f p) \leq$ degree (of-int (sign $p$ ) :: 'a poly) + degree $\left(\prod i=0 . .<n\right.$. $g i)$ unfolding $f$-def $g$-def by (rule degree-mult-le)
also have degree (of-int (sign p) :: 'a poly) $=0$ unfolding sign-def by auto
also have $0+$ degree $\left(\prod i=0 . .<n . g i\right)=$ degree $\left(\prod i=0 . .<n . g i\right)$ by simp
also have $\left(\prod i=0 . .<n . g i\right)=g i *(\operatorname{prod} g(\{0 . .<n\}-\{i\}))$ by (subst prod.remove $[O F-i]$, auto)
also have degree $\ldots \leq$ degree $(g i)+$ degree $(\operatorname{prod} g(\{0 . .<n\}-\{i\}))$ by (rule degree-mult-le)
also have $\ldots=$ degree $(\operatorname{prod} g(\{0 . .<n\}-\{i\}))$ unfolding $g$-def $l$-def using i $D$ pi pin by auto
also have $\ldots \leq \operatorname{sum}($ degree $\circ g)(\{0 . .<n\}-\{i\})$
by (rule degree-prod-sum-le, auto)
also have $\ldots \leq \operatorname{sum}(\lambda i .1)(\{0 . .<n\}-\{i\})$
by (rule sum-mono, insert p $D$, auto simp: $g$-def o-def l-def deg)
also have $\ldots=n-1$ using $i$ by simp
finally show degree ( $f p) \leq n-1$.
qed (insert $n$, auto)
also have $\ldots=n$ by fact
finally show det ? $D x \neq 0$ using $n$ by auto
qed
hence poly (det (four-block-mat ?A ?B ?C ?Dx)) $0=\operatorname{poly}(\operatorname{det}(? A *$ ?Dx - ?B

* ?C)) 0 by $\operatorname{simp}$
also have $\ldots=\operatorname{det}\left(? p 0\left(? A *\left(? D+\right.\right.\right.$ monom $\left.\left.\left.11 \cdot m 1_{m} n\right)-? B * ? C\right)\right)$
by (rule poly-det, use $l A l B l C l D$ in force)
also have ? $A *\left(? D+\right.$ monom $\left.11 \cdot m 1_{m} n\right)=? A * ? D+l A *($ monom 11
-m $1_{m} n$ )
by (rule mult-add-distrib-mat[OF lA lD], auto)
also have ? $A *\left(\right.$ monom $\left.11 \cdot m 1_{m} n\right)=$ monom $11 \cdot m$ ? A using $l A$ by auto
also have ? $p 0(? A * ? D+\ldots-? B * ? C)=? p 0(? A * ? D)-? p 0(? B * ? C)$
by (intro eq-matI, insert $l A l B l C l D$, auto simp: poly-monom)
also have $? A * ? D=l(A * D)$ using $l$-mult $[O F A D]$ by auto
also have ? $p 0 \ldots=A * D$ unfolding $l$-def by fastforce
also have ? $B * ? C=l(B * C)$ using $l$-mult $[O F B C]$ by auto
also have ? $p 0 \ldots=B * C$ unfolding $l$-def by fastforce
also have poly (det (four-block-mat ?A ?B ?C ?Dx)) $0=\operatorname{det}$ (?p0 (four-block-mat ?A ?B ? $C$ ? D $x)$ )
by (rule poly-det[OF four-block-carrier-mat[OF lA]], insert lD, auto)
also have ?p0 (four-block-mat ?A ?B ?C ?Dx) = four-block-mat (?p0 ?A) (?p0 ?B) (?p0 ?C) (?p0 ?D $x$ )
by (rule map-four-block-mat $[$ OF $l A l B l C]$, insert $l D$, auto)
also have ?p0 ? $A=A$ unfolding $l$-def by fastforce
also have ? 0 0 ? $B=B$ unfolding $l$-def by fastforce
also have ?p0 ? $C=C$ unfolding $l$-def by fastforce
also have ? $p 0$ ? $D x=D$ unfolding $l$-def using $D$ by (intro eq-matI, auto simp: poly-monom)
finally show ?thesis.
qed
end
lemma det-swapcols:
assumes $*: k<n l<n k \neq l$ and $A: A \in$ carrier-mat $n n$
shows $\operatorname{det}($ swapcols $k l A)=-\operatorname{det} A$
proof -
let $? B=$ transpose-mat $A$
let ? $C=$ swaprows $k l$ ? $B$
let $? D=$ transpose-mat ? $C$
have $C: ? C \in$ carrier-mat $n n$ and $B: ? B \in$ carrier-mat $n n$
unfolding transpose-carrier-mat swaprows-carrier using $A$ by auto
show ?thesis
unfolding
swapcols-is-transp-swap-rows[OF $A *(1-2)]$
det-transpose $[O F C]$ det-swaprows $[O F * B]$ det-transpose $[O F A]$..
qed
lemma swap-row-to-front-det: $A \in$ carrier-mat $n n \Longrightarrow I<n \Longrightarrow \operatorname{det}$ (swap-row-to-front A I)

$$
=(-1)^{\wedge} I * \operatorname{det} A
$$

proof (induct I arbitrary: A)
case (Suc I A)
from $\operatorname{Suc}(3)$ have $I: I<n$ by auto
let ? $I=$ Suc $I$
let ? $A=$ swaprows I ?I $A$
have $A A: ? A \in$ carrier-mat $n n$ using $S u c(2)$ by simp
have $\operatorname{det}($ swap-row-to-front $A(S u c I))=\operatorname{det}$ (swap-row-to-front ? A I) by $\operatorname{simp}$
also have $\ldots=(-1)^{\wedge} I * \operatorname{det}$ ? A by (rule Suc(1)[OF AA I])
also have det ? $A=-1 * \operatorname{det} A$ using det-swaprows[OF I Suc(3) - Suc(2)] by $\operatorname{simp}$
finally show? case by simp
qed $\operatorname{simp}$
lemma swap-col-to-front-det: $A \in$ carrier-mat $n n \Longrightarrow I<n \Longrightarrow \operatorname{det}$ (swap-col-to-front A I)

$$
=(-1)^{\wedge} I * \operatorname{det} A
$$

proof (induct I arbitrary: A)

```
    case (Suc I A)
    from Suc(3) have I:I<n by auto
    let ?I = Suc I
    let ?A = swapcols I ?I A
    have AA: ?A \in carrier-mat n n using Suc(2) by simp
    have det (swap-col-to-front A (Suc I)) = det (swap-col-to-front ?A I) by simp
    also have \ldots= (-1)^I* det ?A by (rule Suc(1)[OF AA I])
    also have det ?A = -1* det A using det-swapcols[OF I Suc(3) - Suc(2)] by
simp
    finally show ?case by simp
qed simp
lemma swap-row-to-front-four-block: assumes A1: A1 \in carrier-mat n m1
    and A2: A2 \in carrier-mat n m2
    and A3:A3 \in carrier-mat 1 m1
    and A4:A4 \in carrier-mat 1 m2
    shows swap-row-to-front (four-block-mat A1 A2 A3 A4) n = four-block-mat A3
A4 A1 A2
    by (subst swap-row-to-front-result[OF four-block-carrier-mat[OF A1 A4]], force,
    rule eq-matI, insert A1 A2 A3 A4,auto)
lemma swap-col-to-front-four-block: assumes A1: A1 \in carrier-mat n1 m
    and A2: A2 \in carrier-mat n1 1
    and A3:A3 \in carrier-mat n2 m
    and A4:A4 \in carrier-mat n2 1
    shows swap-col-to-front (four-block-mat A1 A2 A3 A4) m = four-block-mat A2
A1 A4 A3
    by (subst swap-col-to-front-result[OF four-block-carrier-mat[OF A1 A4]], force,
    rule eq-matI, insert A1 A2 A3 A4, auto)
lemma det-four-block-mat-lower-right-zero-col: assumes A1: A1 \in carrier-mat 1
n
    and A2: A2 \in carrier-mat 1 1
    and A3:A3 \in carrier-mat n n
    and A40:A4 = ( (0m n 1)
    shows det (four-block-mat A1 A2 A3 A4) = (-1)`n * det A2 * det A3 (is det
?A = -)
proof -
    let ?B = four-block-mat A3 A4 A1 A2
    from four-block-carrier-mat[OF A3 A2]
    have B: ?B \in carrier-mat (Suc n) (Suc n) by simp
    from A40 have A4:A4 \in carrier-mat n 1 by auto
    from arg-cong[OF swap-row-to-front-four-block[OF A3 A4 A1 A2], of det]
    swap-row-to-front-det[OF B, of n]
    have det?A = (-1)` n * det ? B by auto
    also have det ? B = det A3*\operatorname{det A2}
    by (rule det-four-block-mat-upper-right-zero-col[OF A3 A40 A1 A2])
    finally show ?thesis by simp
```


## qed

lemma det-four-block-mat-lower-left-zero-col: assumes A1: A1 $\in$ carrier-mat 11
and A2: A2 $\in$ carrier-mat 1 n
and $A 30: A 3=\left(0_{m} n 1\right)$
and A4: A4 $\in$ carrier-mat $n n$
shows $\operatorname{det}($ four-block-mat A1 A2 A3 A4 $)=\operatorname{det} A 1 * \operatorname{det} A 4($ is $\operatorname{det} ? A=-)$
proof -
from A30 have A3: A3 $\in$ carrier-mat $n 1$ by auto
let $? B=$ four-block-mat A2 A1 A4 A3
from four-block-carrier-mat[OF A2 A3]
have B: ?B $\in$ carrier-mat (Suc n) (Suc n) by simp
from arg-cong[OF swap-col-to-front-four-block[OF A2 A1 A4 A3], of det] swap-col-to-front-det[OF B, of $n]$
have det ? $A=(-1)^{\wedge} n *$ det $? B$ by auto
also have $\operatorname{det} ? B=(-1)^{\wedge} n * \operatorname{det} A 1 * \operatorname{det} A 4$
by (rule det-four-block-mat-lower-right-zero-col[OF A2 A1 A4 A30])
also have $(-1) \widehat{n} * \ldots=(-1 *-1) \widehat{n} * \operatorname{det} A 1 * \operatorname{det} A 4$
unfolding power-mult-distrib by (simp add: ac-simps)
finally show? thesis by simp
qed
lemma det-addcol[simp]:
assumes $l: l<n$ and $k: k \neq l$ and $A: A \in$ carrier-mat $n n$
shows $\operatorname{det}(a d d c o l a k l A)=\operatorname{det} A$
proof -
have addcol a $k l A=A *$ addrow-mat $n$ a $l k$
using addcol-mat[OF A l].
also have $\operatorname{det}(A * a d d r o w-m a t n a l k)=\operatorname{det} A * \operatorname{det}($ addrow-mat nalk) by (rule det-mult $[O F A]$, auto)
also have det (addrow-mat nalk)=1
using det-addrow-mat[OF $k[$ symmetric $]]$.
finally show? ?thesis using $A$ by simp
qed
definition insert-index $i \equiv \lambda i^{\prime}$. if $i^{\prime}<i$ then $i^{\prime}$ else Suc $i^{\prime}$
definition delete-index $i \equiv \lambda i^{\prime}$. if $i^{\prime}<i$ then $i^{\prime}$ else $i^{\prime}-$ Suc 0
lemma insert-index[simp]:
$i^{\prime}<i \Longrightarrow$ insert-index $i i^{\prime}=i^{\prime}$
$i^{\prime} \geq i \Longrightarrow$ insert-index $i i^{\prime}=S u c i^{\prime}$
unfolding insert-index-def by auto
lemma delete-insert-index[simp]:
delete-index $i$ (insert-index $\left.i i^{\prime}\right)=i^{\prime}$
unfolding insert-index-def delete-index-def by auto

```
lemma insert-delete-index
    assumes }\mp@subsup{i}{}{\prime}i:\mp@subsup{i}{}{\prime}\not=
    shows insert-index i(delete-index i i})=\mp@subsup{i}{}{\prime
    unfolding insert-index-def delete-index-def using i'i by auto
definition delete-dom pi\equiv\lambdai'. p(insert-index i i')
definition delete-ran pj\equiv\lambdai.delete-index j (pi)
definition permutation-delete p i=delete-ran (delete-dom p i) (p i)
definition insert-ran p j\equiv \i. insert-index j (pi)
definition insert-dom pij\equiv
    \lambdai'.. if i'< i then p i' else if }\mp@subsup{i}{}{\prime}=i\mathrm{ then j else p ( }\mp@subsup{i}{}{\prime}-1
definition permutation-insert i j p \equiv insert-dom(insert-ran p j) i j
lemmas permutation-delete-expand =
    permutation-delete-def[unfolded delete-dom-def delete-ran-def insert-index-def delete-index-def]
lemmas permutation-insert-expand =
    permutation-insert-def[unfolded insert-dom-def insert-ran-def insert-index-def delete-index-def]
lemma permutation-insert-inserted[simp]:
    permutation-insert (i::nat) j pi=j
    unfolding permutation-insert-expand by auto
lemma permutation-insert-base:
    assumes p:p permutes {0..<n}
    shows permutation-insert n n p = p
proof (rule ext)
    fix x show permutation-insert n n p x = p x
        apply (cases rule: linorder-cases[of x n])
        unfolding permutation-insert-expand
        using permutes-others[OF p] p by auto
qed
lemma permutation-insert-row-step:
    <permutation-insert (Suc i) jp\circ transpose i (Suc i)= permutation-insert i j p>
(is <?l = ?r>)
proof (rule ext)
    fix }
    consider }\langlex<i\rangle|\langlex=i\rangle|{x=Suc i\rangle|\Suc i<x
            using Suc-lessI by (cases rule: linorder-cases [of x i]) blast+
    then show <?l x = ?r x>
        by cases (simp-all add: permutation-insert-expand)
qed
```

```
lemma permutation-insert-column-step:
    assumes \(p\) : p permutes \(\{0 . .<n\}\) and \(j<n\)
    shows transpose \(j\) (Suc \(j\) ) ○ permutation-insert \(i(S u c j) p=\) permutation-insert
i j p
    (is \(? l=? r\) )
proof (rule ext)
    fix \(x\) show ?l \(x=\) ? \(r x\)
    proof (cases rule: linorder-cases[of \(x i]\) )
        case less note \(x=\) this
            show ?thesis
                apply (cases rule: linorder-cases [of \(p x j]\) )
                unfolding permutation-insert-expand using \(x\) by simp+
        next case equal thus? ?hesis by simp
        next case greater note \(x=\) this
            show ?thesis
                apply (cases rule: linorder-cases \([\) of \(p(x-1) j]\) )
                unfolding permutation-insert-expand using \(x\) by simp +
    qed
qed
lemma delete-dom-image:
    assumes \(i: i \in\{0 . .<\) Suc \(n\}(\) is \(-\in ? N)\)
    assumes iff: \(\forall i^{\prime} \in\) ? \(N . f i^{\prime}=f i \longrightarrow i^{\prime}=i\)
    shows delete-dom \(f i\) ' \(\{0 . .<n\}=f\) '? \(N-\{f i\}(\) is \(? L=? R)\)
proof (unfold set-eq-iff, intro allI iffI)
    fix \(j^{\prime}\)
    \{ assume \(L: j^{\prime} \in ? L\)
        then obtain \(i^{\prime}\) where \(i^{\prime}: i^{\prime} \in\{0 . .<n\}\) and \(d j^{\prime}: \operatorname{delete-dom~} f i i^{\prime}=j^{\prime}\) by auto
        show \(j^{\prime} \in\) ? \(R\)
        proof \(\left(\right.\) cases \(\left.i^{\prime}<i\right)\)
            case True
                show ?thesis
                unfolding image-def
                unfolding Diff-iff
                unfolding mem-Collect-eq singleton-iff
                    proof (intro conjI bexI)
                        show \(j^{\prime} \neq f i\)
                        proof
                            assume \(j^{\prime}: j^{\prime}=f i\)
                            hence \(f i^{\prime}=f i\)
                            using \(d j\) '[unfolded delete-dom-def insert-index-def] using True by simp
                                    thus False using iff \(i\) True by auto
                    qed
                    show \(j^{\prime}=f i^{\prime}\)
                    using dj' True unfolding delete-dom-def insert-index-def by simp
                qed (insert \(i^{\prime}\), simp)
            next case False
                show ?thesis
                    unfolding image-def
```

```
            unfolding Diff-iff
            unfolding mem-Collect-eq singleton-iff
            proof(intro conjI bexI)
            show Si': Suc i'}\mp@subsup{i}{}{\prime}\in?N\mathrm{ using }\mp@subsup{i}{}{\prime}\mathrm{ by auto
            show }\mp@subsup{j}{}{\prime}\not=f
            proof
                    assume j': j' = fi
                    hence f(Suc i')=fi
                    using dj'[unfolded delete-dom-def insert-index-def] j' False by simp
                    thus False using iff Si' False by auto
            qed
            show j' = f(Suc i')
                using dj' False unfolding delete-dom-def insert-index-def by simp
        qed
    qed
}
{ assume R: j' }\in
    then obtain }\mp@subsup{i}{}{\prime
        where }\mp@subsup{i}{}{\prime}:\mp@subsup{i}{}{\prime}\in?N\mathrm{ and }\mp@subsup{j}{}{\prime}f:\mp@subsup{j}{}{\prime}\not=fi\mathrm{ and }\mp@subsup{j}{}{\prime}\mp@subsup{f}{}{\prime}:\mp@subsup{j}{}{\prime}=f\mp@subsup{i}{}{\prime}\mathrm{ by auto
    hence }\mp@subsup{i}{}{\prime}i:\mp@subsup{i}{}{\prime}\not=i\mathrm{ using iff by auto
    hence n: n>0 using i i' by auto
    show j'\in?L
    proof (cases i'<i)
        case True show ?thesis
            proof
            show j' = delete-dom fi i'
                unfolding delete-dom-def insert-index-def using True j'fi' by simp
            qed (insert True i, simp)
        next case False show ?thesis
            proof
                show }\mp@subsup{i}{}{\prime}-1\in{0..<n}\mathrm{ using }\mp@subsup{i}{}{\prime}n\mathrm{ by auto
                show j}\mp@subsup{j}{}{\prime}=\mathrm{ delete-dom fi (i'-1)
                    unfolding delete-dom-def insert-index-def using False j'fi' i'i by auto
            qed
    qed
    }
qed
lemma delete-ran-image:
    assumes j:j\in{0..<Suc n} (is - \in ?N)
    assumes fimg: f'{0..<n}=?N - {j}
    shows delete-ran fj'{0..<n}={0..<n} (is ?L =?R)
proof(unfold set-eq-iff, intro allI iffI)
    fix j}\mp@subsup{j}{}{\prime
    { assume L: j' }\in
        then obtain i where i:i\in{0..<n} and ij':delete-ran fji= j' by auto
        have fi\in?N-{j} using fimg i by blast
        thus j' }\in\mathrm{ ?R using ij' j unfolding delete-ran-def delete-index-def by auto
    }
```

```
    { assume R: j' }\in?R\mathrm{ show j' }\mp@subsup{j}{}{\prime}\in
        proof (cases j'<j)
        case True
            hence }\mp@subsup{j}{}{\prime}\in?N-{j} using R by aut
            then obtain i}\mathrm{ where fij':fi= j' and i:i}{{0..<n
                unfolding fimg[symmetric] by auto
                have delete-ran fji= j'
                    unfolding delete-ran-def delete-index-def unfolding fij' using True by
simp
            thus ?thesis using i by auto
        next case False
            hence Suc j' }\in?N-{j} using R by aut
            then obtain }i\mathrm{ where fj':}fi=Suc j' and i:i\in{0..<n
                unfolding fimg[symmetric] by auto
            have delete-ran fji= j'
                unfolding delete-ran-def delete-index-def unfolding fi' using False by
simp
            thus ?thesis using i by auto
        qed
    }
qed
lemma delete-index-inj-on:
    assumes iS:i\not\inS
    shows inj-on (delete-index i) S
proof(intro inj-onI)
    fix }x
    assume eq: delete-index i }x=\mathrm{ delete-index i y and x: x }\inS\mathrm{ and }y:y\in
    have }x\not=iy\not=i\mathrm{ using x y iS by auto
    thus }x=
        using eq unfolding delete-index-def
        by(cases x<i; cases y<i;simp)
qed
lemma insert-index-inj-on:
    shows inj-on (insert-index i) S
proof(intro inj-onI)
    fix }x
    assume eq: insert-index i }x=\mathrm{ insert-index i y and x: x &S and y: y GS
    show }x=
        using eq unfolding insert-index-def
        by(cases }x<i\mathrm{ ; cases y<i;simp)
qed
lemma delete-dom-inj-on:
    assumes i:i\in{0..<Suc n} (is - }\in\mathrm{ ?N)
    assumes inj: inj-onf ?N
    shows inj-on(delete-dom f i) {0..<n}
```

```
proof (rule eq-card-imp-inj-on)
    have card ?N = card ( f'?N) using card-image[OF inj]..
    hence card {0..<n} = card (f'?N - {fi}) using i by auto
    also have ... = card (delete-dom fi'{ {0..<n})
        apply(subst delete-dom-image[symmetric])
        using i inj unfolding inj-on-def by auto
    finally show card (delete-dom fi''{0..<n})= card {0..<n}..
qed simp
lemma delete-ran-inj-on:
    assumes j: j\in{0..<Suc n} (is - \in ?N)
    assumes img: f'{0..<n} = ?N - {j}
    shows inj-on (delete-ran fj) {0..<n}
    apply (rule eq-card-imp-inj-on)
    unfolding delete-ran-image[OF j img] by simp+
lemma permutation-delete-bij-betw:
    assumes i: i\in{0 ..<Suc n} (is - \in?N)
    assumes bij:bij-betw p ?N ?N
    shows bij-betw(permutation-delete p i) {0..<n} {0..<n} (is bij-betw ?p - -)
proof-
    have inj: inj-on p ?N using bij-betw-imp-inj-on[OF bij].
    have ran: p'?N = ?N using bij-betw-imp-surj-on[OF bij].
    hence j:pi\in?N using i by auto
    have }\forall\mp@subsup{i}{}{\prime}\in?N.p\mp@subsup{i}{}{\prime}=pi\longrightarrow\mp@subsup{i}{}{\prime}=i\mathrm{ using inj i unfolding inj-on-def by auto
    from delete-dom-image[OF i this]
    have delete-dom pi'{ {0..<n} =?N - {pi} unfolding ran.
    from delete-ran-inj-on[OF j this] delete-ran-image[OF j this]
    show ?thesis unfolding permutation-delete-def
        using bij-betw-imageI by blast
qed
lemma permutation-delete-permutes:
assumes \(p\) : \(p\) permutes \(\{0\).. \(<\) Suc \(n\}\) (is - permutes ? \(N\) ) and \(i: i<\) Suc \(n\)
    shows permutation-delete p i permutes {0..<n} (is?p permutes ? N')
proof (rule bij-imp-permutes, rule permutation-delete-bij-betw)
    have pi:pi< Suc n using pi by auto
    show bij-betw p ?N ?N using permutes-imp-bij[OF p].
    fix }x\mathrm{ assume }x\not\in{0..<n} hence x: x\geqn by sim
        show ?p x = x
        proof(cases x<i)
            case True thus ?thesis
                unfolding permutation-delete-def using x i by simp
            next case False
                    hence p(Suc x) = Suc x using x permutes-others [OF p] by auto
                    thus ?thesis
                    unfolding permutation-delete-expand using False pi x by simp
        qed
```

```
qed (insert i,auto)
lemma permutation-insert-delete:
    assumes p:p permutes {0..<Suc n}
        and i:i<Suc n
    shows permutation-insert i (pi)(permutation-delete p i)=p
    (is ?l = -)
proof
    fix }\mp@subsup{i}{}{\prime
    show ?l i' = p i'
    proof (cases rule: linorder-cases[of i' i])
        case less note i'i = this
            show ?thesis
            proof (cases pi=pi')
                case True
                    hence }i=\mp@subsup{i}{}{\prime}\mathrm{ using permutes-inj[OF p] injD by metis
                    hence False using i'i by auto
                    thus ?thesis by auto
                next case False thus ?thesis
                unfolding permutation-insert-expand permutation-delete-expand
                using i'i by auto
        qed
    next case equal thus ?thesis unfolding permutation-insert-expand by simp
    next case greater hence }\mp@subsup{i}{}{\prime}i:\mp@subsup{i}{}{\prime}>i\mathrm{ by auto
            hence cond: \neg i' - 1<i using i'i by simp
            show ?thesis
            proof (cases rule: linorder-cases[of p i' p i])
            case less
                hence pd: permutation-delete pi(i'-1)=p i'
                    unfolding permutation-delete-expand
                    using i'i cond by auto
                    show ?thesis
                        unfolding permutation-insert-expand pd
                using i'i less by simp
            next case equal
                hence i= i' using permutes-inj[OF p]injD by metis
                hence False using i'i by auto
                    thus ?thesis by auto
            next case greater
                hence pd: permutation-delete p i (i'-1) =p pi'-1
                    unfolding permutation-delete-expand
                using i'i cond by simp
                    show ?thesis
                unfolding permutation-insert-expand pd
                using i'i greater by auto
        qed
    qed
qed
```

```
lemma insert-index-exclude[simp]:
    insert-index i }\mp@subsup{i}{}{\prime}\not=i\mathrm{ unfolding insert-index-def by auto
lemma insert-index-image:
    assumes i:i<Suc n
    shows insert-index i'{0..<n}={0..<Suc n}-{i} (is ?L =?R)
proof(unfold set-eq-iff, intro allI iffI)
    let ?N = {0..<Suc n}
    fix }\mp@subsup{i}{}{\prime
    { assume L: i' }\in
        then obtain }\mp@subsup{i}{}{\prime\prime
            where ins: }\mp@subsup{i}{}{\prime}=\mathrm{ insert-index }i\mp@subsup{i}{}{\prime\prime}\mathrm{ and }\mp@subsup{i}{}{\prime\prime}:\mp@subsup{i}{}{\prime\prime}\in{0..<n} by aut
        show i'\in?N - {i}
        proof(rule DiffI)
            show i'\in?N using ins unfolding insert-index-def using i" by auto
            show }\mp@subsup{i}{}{\prime}\not\in{i
                unfolding singleton-iff
                    unfolding ins unfolding insert-index-def by auto
        qed
    }
    { assume R: i' }\in
        show i'}\mp@subsup{i}{}{\prime}\in
        proof(cases rule: linorder-cases[of i' i])
            case less
                hence }\mp@subsup{i}{}{\prime}:\mp@subsup{i}{}{\prime}\in{0..<n}\mathrm{ using }i\mathrm{ by auto
                hence insert-index i i' = i' unfolding insert-index-def using less by auto
                    thus ?thesis using i' by force
            next case equal
                    hence False using R by auto
                    thus ?thesis by auto
            next case greater
                hence }\mp@subsup{i}{}{\prime\prime}:\mp@subsup{i}{}{\prime}-1\in{0..<n}\mathrm{ using }iR\mathrm{ by auto
                    hence insert-index i (i'-1)= = i'
                    unfolding insert-index-def using greater by auto
                    thus ?thesis using i"}\mathrm{ by force
        qed
    }
qed
lemma insert-ran-image:
    assumes j: j< Suc n
    assumes img: f'{0..<n}={0..<n}
    shows insert-ran fj'{0..<n} ={0..<Suc n} - {j} (is ?L = ?R)
proof -
    have ?L}=(\lambdai. insert-index j(fi))'{0..<n} unfolding insert-ran-def..
    also have ... =(insert-index j\circf)'{0..<n} by auto
    also have ... = insert-index j' f'{0..<n} by auto
    also have ... = insert-index j' {0..<n} unfolding img by auto
    finally show ?thesis using insert-index-image[OF j] by auto
```


## qed

```
lemma insert-dom-image:
    assumes \(i: i<\) Suc \(n\) and \(j: j<\) Suc \(n\)
        and img: \(f\) ' \(\{0 . .<n\}=\{0 . .<\) Suc \(n\}-\{j\}\) (is \(-=? N--)\)
    shows insert-dom \(f i j\) '? \(N=? N\) (is ? f' \(-=-\) )
proof(unfold set-eq-iff,intro allI iffI)
    fix \(j^{\prime}\)
    \{ assume \(j^{\prime} \in ? f\) '?N
        then obtain \(i^{\prime}\) where \(i^{\prime}: i^{\prime} \in ? N\) and \(j^{\prime}: j^{\prime}=? f i^{\prime}\) by auto
        show \(j^{\prime} \in ? N\)
        proof (cases rule:linorder-cases \(\left[\right.\) of \(\left.i^{\prime} i\right]\) )
            case less
                hence \(i^{\prime} \in\{0 . .<n\}\) using \(i\) by auto
                hence \(f i^{\prime}<S u c n\) using imageI \(\left[\right.\) of \(\left.i^{\prime}\{0 . .<n\} f\right]\) img by auto
                    thus ?thesis
                        unfolding \(j^{\prime}\) unfolding insert-dom-def using less by auto
            next case equal
                thus ?thesis unfolding \(j^{\prime}\) insert-dom-def using \(j\) by auto
            next case greater
                    hence \(i^{\prime}-1 \in\{0 . .<n\}\) using \(i^{\prime}\) by auto
                    hence \(f\left(i^{\prime}-1\right)<\) Suc \(n\) using imageI \(\left[\right.\) of \(\left.i^{\prime}-1\{0 . .<n\} f\right] \mathrm{img}\) by auto
                    thus ?thesis
                    unfolding \(j^{\prime}\) insert-dom-def using greater by auto
        qed
    \}
    \{ assume \(j^{\prime}: j^{\prime} \in ? N\) show \(j^{\prime} \in ? f\) ? ? \(N\)
        proof (cases \(j^{\prime}=j\) )
            case True
                        hence ?f \(i=j^{\prime}\) unfolding insert-dom-def by auto
                thus ?thesis using \(i\) by auto
            next case False
                hence \(j^{\prime}: j^{\prime} \in ? N-\{j\}\) using \(j j^{\prime}\) by auto
                    then obtain \(i^{\prime}\) where \(j^{\prime} f: j^{\prime}=f i^{\prime}\) and \(i^{\prime}: i^{\prime} \in\{0 . .<n\}\)
                        unfolding img[symmetric] by auto
                show ?thesis
                proof (cases \(i^{\prime}<i\) )
                        case True thus ?thesis unfolding \(j^{\prime} f i\) insert-dom-def using \(i^{\prime}\) by auto
                    next case False
                        hence ?f (Suc \(i^{\prime}\) ) \(=j^{\prime}\) unfolding \(j^{\prime} f i\) insert-dom-def using \(i^{\prime}\) by auto
                        thus ?thesis using \(i^{\prime}\) by auto
            qed
        qed
    \}
qed
lemma insert-ran-inj-on:
    assumes \(\operatorname{inj}\) : inj-on \(f\{0 . .<n\}\) and \(j: j<\) Suc \(n\)
    shows inj-on (insert-ran \(f\) ) \(\{0 . .<n\}\) (is inj-on ?f -)
```

```
proof (rule inj-onI)
    fix }i\mp@subsup{i}{}{\prime
    assume i:i}i\in{0..<n} and \mp@subsup{i}{}{\prime}:\mp@subsup{i}{}{\prime}\in{0..<n} and eq:?f i=?f i'
    note eq2 = eq[unfolded insert-ran-def insert-index-def]
    have fi=fi'
    proof (cases f i<j)
        case True
            moreover have fi'<j apply (rule ccontr) using eq2 True by auto
            ultimately show ?thesis using eq2 by auto
        next case False
            moreover have }\negf\mp@subsup{i}{}{\prime}<j\mathrm{ apply (rule ccontr) using eq2 False by auto
            ultimately show ?thesis using eq2 by auto
    qed
    from inj-onD[OF inj this i i] show i= i'.
qed
lemma insert-dom-inj-on:
    assumes inj: inj-on f {0..<n}
        and i:i< Suc n and j:j<Suc n
        and img: f'{0..<n} ={0..<Suc n} - {j} (is - = ?N - -)
    shows inj-on (insert-dom fij) ?N
    apply(rule eq-card-imp-inj-on)
    unfolding insert-dom-image[OF i j img] by simp+
lemma permutation-insert-bij-betw:
    assumes q:q permutes {0..<n} and i:i<Suc n and j: j<Suc n
    shows bij-betw (permutation-insert i j q) {0..<Suc n} {0..<Suc n}
        (is bij-betw?q ?N -)
proof (rule bij-betw-imageI)
    have img:q'{0..<n}={0..<n} using permutes-image [OF q].
    show ?q' ?N = ?N
        unfolding permutation-insert-def
        using insert-dom-image[OF i j insert-ran-image[OF j permutes-image[OF q]]].
    have inj: inj-on q {0..<n}
        apply(rule subset-inj-on) using permutes-inj[OF q] by auto
    show inj-on ?q ?N
        unfolding permutation-insert-def
        using insert-dom-inj-on[OF - i j]
        using insert-ran-inj-on[OF inj j] insert-ran-image[OF j img] by auto
qed
lemma permutation-insert-permutes:
    assumes q:q permutes {0..<n}
        and i:i<Suc n and j:j<Suc n
    shows permutation-insert i j q permutes {0..<Suc n} (is ?p permutes ?N)
    using permutation-insert-bij-betw[OF q i j]
proof (rule bij-imp-permutes)
    fix }\mp@subsup{i}{}{\prime}\mathrm{ assume }\mp@subsup{i}{}{\prime}\not\in?
    moreover hence q(i'-1) = i'-1 using permutes-others[OF q] by auto
```

ultimately show ? $p i^{\prime}=i^{\prime}$
unfolding permutation-insert-expand using $i j$ by auto qed
lemma permutation-fix:
assumes $i: i<$ Suc $n$ and $j: j<S u c n$
shows $\{p . p$ permutes $\{0 . .<$ Suc $n\} \wedge p i=j\}=$ permutation-insert $i j$ ' $\{q . q$ permutes $\{0 . .<n\}\}$
(is ? $L=? R$ )
unfolding set-eq-iff
proof (intro allI iffI)
let $? N=\{0 . .<$ Suc $n\}$
fix $p$
$\{$ assume $p \in ? L$
hence $p: p$ permutes ? $N$ and $p i j: p i=j$ by auto
show $p \in$ ? $R$
unfolding mem-Collect-eq
using permutation-delete-permutes $[$ OF $p i]$
using permutation-insert-delete[OF p i,symmetric]
unfolding pij by auto
\}
$\{$ assume $p \in ? R$
then obtain $q$ where $p q: p=$ permutation-insert $i j q$ and $q: q$ permutes $\{0 . .<n\}$ by auto
hence $p i=j$ unfolding permutation-insert-expand by simp
thus $p \in$ ? $L$
using pq permutation-insert-permutes[OF qij] by auto
\}
qed
lemma permutation-split-ran:
assumes $j: j \in S$
shows $\{p . p$ permutes $S\}=(\bigcup i \in S .\{p . p$ permutes $S \wedge p i=j\})$
(is ? $L=? R$ )
unfolding set-eq-iff
$\operatorname{proof}($ intro allI iffI)
fix $p$
$\{$ assume $p \in ? L$
hence $p$ : $p$ permutes $S$ by auto
obtain $i$ where $i: i \in S$ and pij:pi=j using $j$ permutes-image $[O F p]$ by force
thus $p \in ? R$ using $p$ by auto
\}
$\{$ assume $p \in ? R$
then obtain $i$
where $p$ : $p$ permutes $S$ and $i: i \in S$ and pij: $p i=j$
by auto
show $p \in$ ? $L$
unfolding mem-Collect-eq using $p$.

```
    }
qed
lemma permutation-disjoint-dom:
    assumes }i:i\inS\mathrm{ and }\mp@subsup{i}{}{\prime}:\mp@subsup{i}{}{\prime}\inS\mathrm{ and j:j S S and }i\mp@subsup{i}{}{\prime}:i\not=\mp@subsup{i}{}{\prime
    shows {p.p permutes S\wedge pi=j}\cap{p.p permutes S\wedgep\mp@subsup{i}{}{\prime}=j}={}
        (is ?L\cap?R={})
proof -
    {
        fix p assume p}\in?,L\cap?
        hence p: p permutes S and pi=j and pi'=j by auto
        hence pi=p i' by auto
        note injD[OF permutes-inj[OF p] this]
        hence False using ii' by auto
    }
    thus ?thesis by auto
qed
lemma permutation-disjoint-ran:
    assumes }i:i\inS\mathrm{ and j:j 
    shows {p. p permutes S\wedgepi=j}\cap{p.p permutes S\wedgepi=\mp@subsup{j}{}{\prime}}={}
        (is ?L\cap?R={})
proof -
    {
        fix p assume p\in?L\cap?R
        hence p permutes S and pi=j and pi= j' by auto
        hence False using jj' by auto
    }
    thus ?thesis by auto
qed
lemma permutation-insert-inj-on:
    assumes i< Suc n
    assumes j< Suc n
    shows inj-on (permutation-insert ij) {q. q permutes {0..<n} }
    (is inj-on ?f ?S)
proof (rule inj-onI)
    fix q q '
    assume q\in?S q' \in?S
    hence q:q permutes {0..<n} and q': q' permutes {0..<n} by auto
    assume ?f q=?f q
    hence eq: permutation-insert i j q = permutation-insert i j q' by auto
    note eq = cong[OF eq refl, unfolded permutation-insert-expand]
    show }q\mp@subsup{q}{}{\prime}:q=\mp@subsup{q}{}{\prime
    proof(rule ext)
        fix }
        have foo: Suc x-1=x by auto
        show q x = q' x
        proof(cases x<i)
```

case True thus ?thesis apply(cases $q x<j$;cases $q^{\prime} x<j$ ) using eq[of $\left.x\right]$ by auto
next case False thus ?thesis
apply (cases $q x<j$;cases $q^{\prime} x<j$ ) using eq[of Suc $\left.x\right]$ by auto

## qed

qed
qed
lemma signof-permutation-insert:
assumes $p$ : p permutes $\{0 . .<n\}$ and $i: i<$ Suc $n$ and $j: j<$ Suc $n$
shows signof (permutation-insert i $j p)=\left(-1::^{\prime} a::^{\prime} \text { ring-1) }\right)^{( }(i+j) *$ signof $p$
proof -
\{ fix $j$ assume $j \leq n$
hence signof (permutation-insert $n(n-j) p)=\left(-1:^{\prime} a\right)^{\wedge}(n+(n-j)) *$ signof $p$ proof (induct $j$ )
case 0 show ?case using permutation-insert-base $[O F$ p] by (simp add:
mult-2[symmetric])
next case (Suc j)
hence Sjn: Suc $j \leq n$ and $j: j<n$ and Sj: $n-S u c j<n$ by auto
hence $n 0: n>0$ by auto
have ease: Suc $(n-S u c j)=n-j$ using $j$ by auto
let ?swap $=$ transpose $(n-$ Suc $j)(n-j)$
let ?prev $=$ permutation-insert $n(n-j) p$
have signof (permutation-insert $n(n-S u c j) p)=$ signof (?swap $\circ$ ?prev)
unfolding permutation-insert-column-step [OF p Sj, unfolded ease] ..
also have $\ldots=$ signof ? swap $*$ signof ? prev
proof(rule signof-compose)
show ?swap permutes $\{0 . .<$ Suc $n\}$ by (rule permutes-swap-id,auto)
show ?prev permutes $\{0 . .<$ Suc $n\}$ by (rule permutation-insert-permutes $[O F$
$p]$,auto)
qed
also have signof ? swap $=-1$
proof-
have $n-S u c j<n-j$ using Sjn by simp
thus ?thesis unfolding sign-swap-id by simp
qed
also have signof ?prev $=\left(-1::^{\prime} a\right) \wedge(n+(n-j)) *$ signof $p$ using Suc(1)
$j$ by auto
also have $(-1) * \ldots=(-1)^{\wedge}(1+n+(n-j)) *$ signof $p$ by simp
also have $n-j=1+(n-$ Suc $j)$ using $j$ by simp
also have $1+n+\ldots=2+(n+(n-S u c j))$ by simp
also have $\left(-1::^{\prime} a\right) \wedge \ldots=(-1) \wedge 2 *(-1) \wedge(n+(n-S u c j))$ by simp also have $\ldots=(-1)^{\wedge}(n+(n-S u c j))$ by $\operatorname{simp}$
finally show? case.

## qed

\}
note $\mathrm{col}=$ this
have $n j$ : $n-j \leq n$ using $j$ by auto
have row-base: signof (permutation-insert $\left.n j p)=\left(-1::^{\prime} a\right) \mathcal{}\right)(n+j) *$ signof $p$

$$
\text { using } \operatorname{col}[O F n j] \text { using } j \text { by } \operatorname{simp}
$$

\{ fix $i$ assume $i \leq n$
hence signof (permutation-insert $(n-i) j p)=\left(-1::^{\prime} a\right)^{\wedge}((n-i)+j) *$ signof $p$ proof (induct i)
case 0 show ?case using row-base by auto
next case (Suc i)
hence Sin: Suc $i \leq n$ and $i: i \leq n$ and Si: $n-S u c i<n$ by auto
have ease: Suc $(n-$ Suc $i)=n-i$ using Sin by auto
let ?prev $=$ permutation-insert $(n-i) j p$
let ?swap $=$ transpose $(n-$ Suc $i)(n-i)$
have signof (permutation-insert $(n-S u c i) j p)=$ signof (?prev $\circ$ ?swap) using permutation-insert-row-step $[$ of $n-S u c i]$ unfolding ease by auto
also have $\ldots=$ signof ?prev $*$ signof ?swap
proof(rule signof-compose)
show ?swap permutes $\{0 . .<$ Suc $n\}$ by (rule permutes-swap-id,auto)
show ? prev permutes $\{0 . .<$ Suc $n\}$
apply (rule permutation-insert-permutes $[O F p]$ ) using $j$ by auto
qed
also have signof ? swap $=(-1)$
proof-
have $n-$ Suc $i<n-i$ using Sin by simp
thus ?thesis unfolding sign-swap-id by simp
qed
also have signof ? prev $=\left(-1::^{\prime} a\right)^{\wedge}(n-i+j) *$ signof $p$
using $\operatorname{Suc}(1)[O F i]$.
also have $\ldots *(-1)=(-1)^{\wedge}$ Suc $(n-i+j) *$ signof $p$
by auto
also have $\operatorname{Suc}(n-i+j)=\operatorname{Suc}(S u c(n-S u c i+j))$
using Sin by auto
also have $(-1:: \text { int })^{\wedge} \ldots=(-1)^{\wedge}(n-S u c i+j)$ by auto
ultimately show ?case by auto
qed
\}
note row $=$ this
have $n i: n-i \leq n$ using $i$ by auto
show ?thesis using row [OF ni] using $i$ by simp
qed
lemma foo:
assumes $i: i<S u c n$ and $j: j<S u c n$
assumes $q: q$ permutes $\{0 . .<n\}$
shows $\left\{\left(i^{\prime}\right.\right.$, permutation-insert ijq $\left.i^{\prime}\right) \mid i^{\prime} . i^{\prime} \in\{0 . .<$ Suc $\left.n\}-\{i\}\right\}=$
$\left\{\left(\right.\right.$ insert-index $i i^{\prime \prime}$, insert-index $\left.\left.j\left(q i^{\prime \prime}\right)\right) \mid i^{\prime \prime} . i^{\prime \prime}<n\right\}($ is $? L=? R)$
unfolding set-eq-iff
proof (intro allI iffI)
fix $i j$
\{ assume $i j \in ? L$
then obtain $i^{\prime}$
where $i j: i j=\left(i^{\prime}\right.$, permutation-insert $\left.i j q i^{\prime}\right)$ and $i^{\prime}: i^{\prime}<$ Suc $n$ and $i^{\prime} i: i^{\prime}$

```
\not=i
    by auto
    show ij }\in\mathrm{ ?R unfolding mem-Collect-eq
    proof(intro exI conjI)
        show ij = (insert-index i (delete-index i i'), insert-index j (q (delete-index i
i
            using ij unfolding insert-delete-index[OF i'i] using i'i
            unfolding permutation-insert-expand insert-index-def delete-index-def by
auto
            show delete-index i i'<n using i' i i'i unfolding delete-index-def by auto
        qed
    }
    { assume ij \in?R
        then obtain i'
            where ij: ij = (insert-index i i'\prime, insert-index j (q i'|)) and i': i' i' < n
            by auto
    show ij \in?L unfolding mem-Collect-eq
    proof(intro exI conjI)
            show insert-index i i'\prime}\in{0..<Suc n}-{i
                unfolding insert-index-image[OF i,symmetric] using i' by auto
            have insert-index j (q i') = permutation-insert i j q (insert-index i i')
                unfolding permutation-insert-expand insert-index-def by auto
            thus ij = (insert-index i i', permutation-insert i j q (insert-index i i'\prime))
            unfolding ij by auto
    qed
    }
qed
definition mat-delete A ij\equiv
    mat (dim-row A - 1) (dim-col A - 1) ( }\lambda(\mp@subsup{i}{}{\prime},\mp@subsup{j}{}{\prime})
    A $$(if i'< i then i' else Suc }\mp@subsup{i}{}{\prime}\mathrm{ , if }\mp@subsup{j}{}{\prime}<j\mathrm{ then j' else Suc j'))
lemma mat-delete-dim[simp]:
    dim-row (mat-delete A i j) = dim-row A - 1
    dim-col (mat-delete A ij) = dim-col A - 1
    unfolding mat-delete-def by auto
lemma mat-delete-carrier:
    assumes A:A\in carrier-mat m n
    shows mat-delete A i j f carrier-mat (m-1) (n-1) unfolding mat-delete-def
using A by auto
lemma mat-delete-index:
    assumes A:A carrier-mat (Suc n) (Suc n)
    and i: i<Suc n and j:j<Suc n
    and }\mp@subsup{i}{}{\prime}:\mp@subsup{i}{}{\prime}<n\mathrm{ and }\mp@subsup{j}{}{\prime}:\mp@subsup{j}{}{\prime}<
shows A $$ (insert-index i i', insert-index j j')= mat-delete A i j $$ (i', j')
unfolding mat-delete-def
unfolding permutation-insert-expand
```

unfolding insert-index-def using $A i j i^{\prime} j^{\prime}$ by auto
definition cofactor $A i j=(-1) \uparrow(i+j) * \operatorname{det}($ mat-delete $A i j)$
lemma laplace-expansion-column:
assumes $A:(A:: ' a::$ comm-ring-1 mat $) \in$ carrier-mat $n n$
and $j: j<n$
shows $\operatorname{det} A=\left(\sum i<n . A \$ \$(i, j) *\right.$ cofactor $\left.A i j\right)$
proof -
define $l$ where $l=n-1$
have $A: A \in$ carrier-mat (Suc l) (Suc l)
and $j l: j<S u c l$ using $A j$ unfolding $l$-def by auto
let $? N=\{0$.. $<$ Suc $l\}$
define $f$ where $f=(\lambda p i$. $A \$ \$(i, p i))$
define $g$ where $g=(\lambda p$. prod $(f p)$ ? $N)$
define $h$ where $h=(\lambda p$. signof $p * g p)$
define $Q$ where $Q=\{q . q$ permutes $\{0 . .<l\}\}$
have $j N: j \in$ ? $N$ using $j l$ by auto
have disj: $\forall i \in ? N . \forall i^{\prime} \in ? N . i \neq i^{\prime} \longrightarrow$
$\{p . p$ permutes $? N \wedge p i=j\} \cap\left\{p . p\right.$ permutes $\left.? N \wedge p i^{\prime}=j\right\}=\{ \}$
using permutation-disjoint-dom[OF-jN] by auto
have fin: $\forall i \in ? N$. finite $\{p$. p permutes ? $N \wedge p i=j\}$
using finite-permutations[of ?N] by auto
have $\operatorname{det} A=\operatorname{sum} h\{p . p$ permutes ? $N\}$
using det-def'[OF A] unfolding $h$-def $g$-def $f$-def using atLeastOLessThan by auto
also have $\ldots=\operatorname{sum} h(\bigcup i \in ? N .\{p . p$ permutes $? N \wedge p i=j\})$
unfolding permutation-split-ran $[$ OF jN]..
also have $\ldots=\left(\sum i \in ? N\right.$. sum $h\{p \mid p . p$ permutes ? $\left.N \wedge p i=j\}\right)$
using sum.UNION-disjoint[OF - fin disj] by auto
also \{
fix $i$ assume $i \in ? N$
hence $i: i<S u c l$ by auto
have sum $h\{p \mid p . p$ permutes $? N \wedge p i=j\}=\operatorname{sum} h$ (permutation-insert $i$ $j$ ' $Q$ )
using permutation-fix[OF ijl] unfolding $Q$-def by auto
also have $\ldots=\operatorname{sum}(h \circ$ permutation-insert $i j$ ) $Q$
unfolding $Q$-def using sum.reindex[OF permutation-insert-inj-on[OF i jl]].
also have $\ldots=\left(\sum q \in Q\right.$.
signof (permutation-insert ijq)*prod $(f($ permutation-insert $i j q)) ? N)$
unfolding $h$-def $g$-def $Q$-def by simp
also \{
fix $q$ assume $q \in Q$
hence $q: q$ permutes $\{0 . .<l\}$ unfolding $Q$-def by auto
let ? $p=$ permutation-insert i j q
have fin: finite (?N $-\{i\})$ by auto

```
    have notin: i\not\in?N - {i} by auto
    have close: insert i(?N - {i})=?N using notin i by auto
    have prod (f?p) ?N = f?pi*\operatorname{prod}(f?p)(?N-{i})
        unfolding prod.insert[OF fin notin, unfolded close] by auto
    also have ... = A $$ (i,j)* prod ( f ? p) (?N-{i})
        unfolding f-def Q-def using permutation-insert-inserted by simp
    also have prod (f ? P) (?N-{i}) = prod ( }\lambda\mp@subsup{i}{}{\prime}.A$$(\mp@subsup{i}{}{\prime},\mathrm{ permutation-insert i j
qi'))(?N-{i})
        unfolding f-def..
    also have ... = prod (\lambdaij. A $$ ij) ((\lambdai'. (i', permutation-insert i j q i'))`
(?N-{i}))
        (is - = prod - ?part)
        unfolding prod.reindex[OF inj-on-convol-ident] o-def..
    also have ?part ={(\mp@subsup{i}{}{\prime},\mathrm{ permutation-insert ij q i')| i'. i' }\mp@subsup{i}{}{\prime}\in?N-{i}}
        unfolding image-def by metis
    also have ... = {(insert-index i i'\prime, insert-index j (q i'\prime}))|\mp@subsup{i}{}{\prime\prime}.\mp@subsup{i}{}{\prime\prime}<l
        unfolding foo[OF i jl q]..
    also have ... = ((\lambdai'\prime.(insert-index i i'\prime, insert-index j (q i'\prime})))'{0..<l}
        unfolding image-def by auto
    also have prod ( }\lambdaij.A$$ij)\ldots= prod ((\lambdaij. A $$ ij) ○ ( \lambdai'". (insert-index i
i'\prime, insert-index j (q i'\)))) {0..<l}
        proof(subst prod.reindex[symmetric])
            have 1: inj ( }\lambda\mp@subsup{i}{}{\prime\prime}.(\mp@subsup{i}{}{\prime\prime},\mathrm{ insert-index j (q i'\))) using inj-on-convol-ident.
            have 2: inj ( }\lambda(\mp@subsup{i}{}{\prime\prime},j).(\mathrm{ insert-index }i\mp@subsup{i}{}{\prime\prime},j)
                apply (intro injI) using injD[OF insert-index-inj-on[of - UNIV]] by
auto
            have inj ( }\lambda\mp@subsup{i}{}{\prime\prime}.(\mathrm{ (insert-index i }\mp@subsup{i}{}{\prime\prime}\mathrm{ , insert-index j (q i')})
                using inj-compose[OF 2 1] unfolding o-def by auto
            thus inj-on ( }\lambda\mp@subsup{i}{}{\prime\prime}\mathrm{ . (insert-index i i'', insert-index j (q i'|}))\mathrm{ ) {0..<l}
                using subset-inj-on by auto
        qed auto
        also have ... = prod ( }\lambda\mp@subsup{i}{}{\prime\prime}.A$$(\mathrm{ insert-index i i',}\mathrm{ insert-index j (q i'f})
{0..<l}
            by auto
    also have ... = prod (\lambda\mp@subsup{i}{}{\prime\prime}.mat-delete A ij $$ (i'\prime,qi\mp@subsup{i}{}{\prime\prime})){0..<l}
    proof (rule prod.cong[OF refl], unfold atLeastLessThan-iff, elim conjE)
        fix }x\mathrm{ assume }x:x<
        show A $$ (insert-index i x, insert-index j (q x)) = mat-delete A i j $$ (x,
q x)
            apply(rule mat-delete-index[OF A i jl]) using qx by auto
    qed
    finally have prod (f ?p) ?N =
        A $$ (i,j)*(\prod\mp@subsup{i}{}{\prime\prime}=0..<l. mat-delete A ij $$ (i'\prime,qi'\prime}
        by auto
    hence signof ?p * prod (f ?p) ?N = (-1::'a)^( i+j)* signof q * ...
        unfolding signof-permutation-insert[OF q i jl] by auto
    }
    hence ... = (\sumq & Q. (-1)`(i+j)* signof q *
    A$$ (i,j)* (\prod\mp@subsup{i}{}{\prime\prime}=0..< l. mat-delete A ij $$ (i'\prime, q i'\prime}))
```

```
    by(intro sum.cong[OF refl],auto)
    also have ... = ( \sumq q\inQ.A $$ (i,j)* (-1)^(i+j)*
        ( signof q*(\prod\mp@subsup{i}{}{\prime\prime}=0..<l. mat-delete A ij $$ (i',},q\mp@subsup{i}{}{\prime\prime})))
        by (intro sum.cong[OF refl],auto)
    also have ... = A $$ (i,j)* (-1)^(i+j)*
        ( \sumq\inQ.signof q*(\prod\mp@subsup{i}{}{\prime\prime}=0..<l. mat-delete A ij$$ (i'\prime,q\mp@subsup{i}{}{\prime\prime})))
        unfolding sum-distrib-left by auto
    also have \ldots. = (A$$ (i,j)*(-1)^(i+j)*\operatorname{det}(mat-delete A i j))
        unfolding det-def'[OF mat-delete-carrier[OF A]]
        unfolding Q-def by auto
    finally have sum h{p| p. p permutes ? N \wedge pi=j}=A$$(i,j)* cofactor
A ij
    unfolding cofactor-def by auto
}
hence ... = (\sumi\in?N. A $$ (i,j)* cofactor A i j) by auto
    finally show ?thesis unfolding atLeastOLessThan using A j unfolding l-def
by auto
qed
lemma laplace-expansion-row:
    assumes A:(A :: 'a :: comm-ring-1 mat) \incarrier-mat n n
        and i: i<n
    shows det A=(\sumj<n.A $$ (i,j)* cofactor A i j)
proof -
    have }\operatorname{det}A=\operatorname{det}(\mp@subsup{A}{}{T})\mathrm{ using det-transpose[OF A] by simp
    also have \ldots. = (\sumj<n. AT $$ (j,i)* cofactor A}\mp@subsup{A}{}{T}ji
    by (rule laplace-expansion-column[OF-i], insert A, auto)
    also have \ldots. = (\sumj<n.A$$ (i,j) * cofactor A ij) unfolding cofactor-def
    proof (rule sum.cong[OF refl], rule arg-cong2[of-- - \xy.x*y],goal-cases)
        case (1 j)
        thus ?case using A i by auto
    next
        case (2 j)
    have det (mat-delete AT ji)=\operatorname{det}((mat-delete AT}ji\mp@subsup{)}{}{T}
        by (subst det-transpose, insert A, auto simp: mat-delete-def)
    also have (mat-delete AT j i)
        unfolding mat-delete-def using A by auto
    finally show ?case by (simp add: ac-simps)
    qed
    finally show ?thesis.
qed
```

lemma degree-det-le: assumes $\wedge i j . i<n \Longrightarrow j<n \Longrightarrow \operatorname{degree}(A \$ \$(i, j)) \leq k$
and $A: A \in$ carrier-mat $n n$
shows degree $(\operatorname{det} A) \leq k * n$
proof -
\{
fix $p$

```
    assume p:p permutes {0..<n}
    have }(\sumx=0..<n.degree (A$$ (x,px)))\leq(\sumx=0..<n.k
        by (rule sum-mono[OF assms(1)], insert p, auto)
    also have \ldots=k*n unfolding sum-constant by simp
    also note calculation
    } note * = this
    show ?thesis unfolding det-def'[OF A]
    apply (rule degree-sum-le)
        apply (simp-all add: finite-permutations)
    apply (drule *)
    apply (rule order.trans [OF degree-mult-le])
    apply simp
    apply (rule order.trans [OF degree-prod-sum-le])
    apply simp-all
    done
qed
lemma upper-triangular-imp-det-eq-0-iff:
    fixes }A :: 'a :: idom mat
    assumes }A\in\mathrm{ carrier-mat }nn\mathrm{ and upper-triangular }
    shows }\operatorname{det}A=0\longleftrightarrow0\in\operatorname{set}(\mathrm{ diag-mat A)
    using assms by (auto simp: det-upper-triangular)
lemma det-identical-columns:
    assumes A: A\in carrier-mat n n
        and ij:i\not=j
        and}i:i<n and j:j<
        and r: col A i = col A j
    shows }\operatorname{det}A=
proof-
    have }\operatorname{det}A=\operatorname{det}\mp@subsup{A}{}{T}\mathbf{using}\operatorname{det-transpose[OF A]..
    also have ... = 0
    proof (rule det-identical-rows[of - n i j])
        show row (transpose-mat A) i = row (transpose-mat A) j
            using A ijr by auto
    qed (auto simp add: assms)
    finally show ?thesis .
qed
definition adj-mat :: ' a :: comm-ring-1 mat => 'a mat where
    adj-mat A = mat (dim-row A) (dim-col A) (\lambda (i,j).cofactor A j i)
lemma adj-mat: assumes A: A carrier-mat n n
    shows adj-mat A \in carrier-mat n n
    A*adj-mat A = det A *m 1m}
    adj-mat A*A = det A 后 1m}
proof -
    from A have dims: dim-row }A=n\mathrm{ dim-col }A=n\mathrm{ by auto
    show aA: adj-mat A \in carrier-mat n n unfolding adj-mat-def dims by simp
```

fix $i j$
assume $i j: i<n j<n$
define $B$ where $B=$ mat $n n\left(\lambda\left(i^{\prime}, j^{\prime}\right)\right.$. if $i^{\prime}=j$ then $A \$ \$\left(i, j^{\prime}\right)$ else $A \$ \$$ $\left(i^{\prime}, j^{\prime}\right)$ )
have $(A *$ adj-mat $A) \$ \$(i, j)=\left(\sum k<n . A \$ \$(i, k) *\right.$ cofactor $\left.A j k\right)$
unfolding times-mat-def scalar-prod-def adj-mat-def using ij $A$ by (auto intro: sum.cong)
also have $\ldots=\left(\sum k<n . A \$ \$(i, k) *(-1) \uparrow(j+k) * \operatorname{det}\right.$ (mat-delete $A j$ k))
unfolding cofactor-def by (auto intro: sum.cong)
also have $\ldots=\left(\sum k<n . B \$ \$(j, k) *(-1) \smile(j+k) * \operatorname{det}\right.$ (mat-delete $B j$ k))
by (rule sum.cong[OF reff], intro arg-cong2[of -- $-\lambda x y . y *-* \operatorname{det} x]$, insert A ij,
auto simp: B-def mat-delete-def)
also have $\ldots=\left(\sum k<n . B \$(j, k) *\right.$ cofactor $\left.B j k\right)$
unfolding cofactor-def by (simp add: ac-simps)
also have $\ldots=\operatorname{det} B$
by (rule laplace-expansion-row[symmetric], insert ij, auto simp: B-def)
also have $\ldots=($ if $i=j$ then $\operatorname{det} A$ else 0$)$
proof (cases $i=j$ )
case True
hence $B=A$ using $A$ by (auto simp add: B-def)
with True show? ?thesis by simp
next
case False
have $\operatorname{det} B=0$
by (rule Determinant.det-identical-rows[OF - False ij], insert A ij, auto simp: B-def)
with False show ?thesis by simp
qed
also have $\ldots=\left(\operatorname{det} A \cdot m 1_{m} n\right) \$(i, j)$ using $i j$ by auto
finally have $(A * \operatorname{adj-mat} A) \$(i, j)=\left(\operatorname{det} A \cdot_{m} 1_{m} n\right) \$ \$(i, j)$.
$\}$ note main $=$ this
show $A * \operatorname{adj}-m a t A=\operatorname{det} A \cdot m 1_{m} n$
by (rule eq-matI[OF main], insert A aA, auto)

## \{

fix $i j$
assume $i j: i<n j<n$
define $B$ where $B=$ mat $n n\left(\lambda\left(i^{\prime}, j^{\prime}\right)\right.$. if $j^{\prime}=i$ then $A \$\left(i^{\prime}, j\right)$ else $A \$ \$$ $\left(i^{\prime}, j^{\prime}\right)$ )
have $($ adj-mat $A * A) \$ \$(i, j)=\left(\sum k<n . A \$ \$(k, j) *\right.$ cofactor $\left.A k i\right)$
unfolding times-mat-def scalar-prod-def adj-mat-def using ij $A$ by (auto intro: sum.cong)
also have $\ldots=\left(\sum k<n . A \$ \$(k, j) *(-1) \curlyvee(k+i) * \operatorname{det}\right.$ (mat-delete $A k$ i))
unfolding cofactor-def by (auto intro: sum.cong)
also have $\ldots=\left(\sum k<n . B \$ \$(k, i) *(-1) \uparrow(k+i) * \operatorname{det}\right.$ (mat-delete $B k$ i))
by (rule sum.cong[OF refl], intro arg-cong2[of $--\lambda x y . y *-* \operatorname{det} x]$, insert A ij, auto simp: B-def mat-delete-def)
also have $\ldots=\left(\sum k<n . B \$ \$(k, i) *\right.$ cofactor $\left.B k i\right)$
unfolding cofactor-def by (simp add: ac-simps)
also have $\ldots=\operatorname{det} B$
by (rule laplace-expansion-column[symmetric], insert ij, auto simp: B-def)
also have $\ldots=($ if $i=j$ then $\operatorname{det} A$ else 0$)$
proof (cases $i=j$ )
case True
hence $B=A$ using $A$ by (auto simp add: $B$-def)
with True show ?thesis by simp
next
case False
have $\operatorname{det} B=0$
by (rule Determinant.det-identical-columns[OF - False ij], insert A ij, auto simp: $B$-def)
with False show ?thesis by simp
qed
also have $\ldots=\left(\operatorname{det} A \cdot{ }_{m} 1_{m} n\right) \$ \$(i, j)$ using $i j$ by auto
finally have $(\operatorname{adj}-\mathrm{mat} A * A) \$ \$(i, j)=\left(\operatorname{det} A \cdot{ }_{m} 1_{m} n\right) \$ \$(i, j)$.
\} note main $=$ this
show adj-mat $A * A=\operatorname{det} A \cdot m 1_{m} n$
by (rule eq-matI[OF main], insert $A$ a $A$, auto)
qed
definition replace-col $A b k=\operatorname{mat}($ dim-row $A)(\operatorname{dim}-\operatorname{col} A)(\lambda(i, j)$ if $j=k$ then $b \$ i$ else $A \$ \$(i, j))$
lemma cramer-lemma-mat:
assumes $A: A \in$ carrier-mat $n n$
and $x: x \in$ carrier-vec $n$
and $k: k<n$
shows $\operatorname{det}\left(\right.$ replace-col $\left.A\left(A *_{v} x\right) k\right)=x \$ k * \operatorname{det} A$
proof -
define $b$ where $b=A *_{v} x$
have $b: b \in$ carrier-vec $n$ using $A x$ unfolding $b$-def by auto
let ? $A b=$ replace-col $A b k$
have $A b: ? A b \in$ carrier-mat $n n$ using $A$ by (auto simp: replace-col-def)
have $x \$ k * \operatorname{det} A=\left(\operatorname{det} A \cdot{ }_{v} x\right) \$ k$ using $A k x$ by auto
also have $\operatorname{det} A \cdot{ }_{v} x=\operatorname{det} A \cdot v\left(1_{m} n *_{v} x\right)$ using $x$ by auto
also have $\ldots=\left(\operatorname{det} A{ }_{m} 1_{m} n\right) *_{v} x$ using $A x$ by auto
also have $\ldots=(\operatorname{adj}-m a t A * A) *_{v} x$ using adj-mat $[O F A]$ by simp
also have $\ldots=$ adj-mat $A *_{v} b$ using $\operatorname{adj}$-mat $[O F A] A x$ unfolding $b$-def by (metis assoc-mult-mat-vec)
also have $\ldots \$ k=\operatorname{row}(a d j-m a t A) k \cdot b$ using $\operatorname{adj-mat}[O F A] b k$ by auto
also have $\ldots=\operatorname{det}$ (replace-col $A b k$ ) unfolding scalar-prod-def using $b k A$
by (subst laplace-expansion-column[OF Ab k], auto intro!: sum.cong arg-cong[of - - det]
$\arg -\mathrm{cong}[o f--\lambda x .-* x]$ eq-matI
simp: replace-col-def adj-mat-def Matrix.row-def cofactor-def mat-delete-def ac-simps)
finally show ?thesis unfolding $b$-def by simp
qed
end

## 10 Code Equations for Determinants

We compute determinants on arbitrary rings by applying elementary rowoperations to bring a matrix on upper-triangular form. Then the determinant can be determined by multiplying all entries on the diagonal. Moreover the final result has to be divided by a factor which is determined by the rowoperations that we performed. To this end, we require a division operation on the element type.

The algorithm is parametric in a selection function for the pivot-element, e.g., for matrices over polynomials it turned out that selecting a polynomial of minimal degree is beneficial.

```
theory Determinant-Impl
imports
    Polynomial-Interpolation.Missing-Polynomial
    HOL-Computational-Algebra.Polynomial-Factorial
    Determinant
begin
type-synonym 'a det-selection-fun = (nat }\times\mp@subsup{}{}{\prime}a)\mathrm{ list }=>\mathrm{ nat
definition det-selection-fun :: 'a det-selection-fun }=>\mathrm{ bool where
    det-selection-fun f}=(\forallxs.xs\not=[]\longrightarrowfxs\infst'set xs
lemma det-selection-funD: det-selection-fun }f\Longrightarrowxs\not=[]\Longrightarrowfxs\infst'set x
    unfolding det-selection-fun-def by auto
definition mute-fun :: (' }a::\mathrm{ comm-ring-1 }\mp@subsup{=>}{}{\prime}a=>\mp@subsup{|}{}{\prime}a\times\mp@subsup{}{}{\prime}a\times' a) => bool wher
    mute-fun }f=(\forallxyy\mp@subsup{x}{}{\prime}\mp@subsup{y}{}{\prime}g.fxy=(\mp@subsup{x}{}{\prime},\mp@subsup{y}{}{\prime},g)\longrightarrowy\not=
        \longrightarrow x = x ^ { \prime } * g \wedge y * x ^ { \prime } = x * y ^ { \prime } )
context
    fixes sel-fun :: 'a ::idom-divide det-selection-fun
begin
```


### 10.1 Properties of triangular matrices

Each column of a triangular matrix should satisfy the following property.

```
definition triangular-column ::nat \(\Rightarrow{ }^{\prime}\) a mat \(\Rightarrow\) bool
    where triangular-column \(j A \equiv \forall i . j<i \longrightarrow i<\) dim-row \(A \longrightarrow A \$ \$(i, j)=0\)
```

lemma triangular-columnD [dest]:
triangular-column $j A \Longrightarrow j<i \Longrightarrow i<$ dim-row $A \Longrightarrow A \$ \$(i, j)=0$
unfolding triangular-column-def by auto
lemma triangular-columnI [intro]:
$(\bigwedge i . j<i \Longrightarrow i<$ dim-row $A \Longrightarrow A \$ \$(i, j)=0) \Longrightarrow$ triangular-column $j A$
unfolding triangular-column-def by auto
The following predicate states that the first $k$ columns satisfy triangu-
larity.
definition triangular-to:: nat $\Rightarrow{ }^{\prime}$ a mat $\Rightarrow$ bool
where triangular-to $k A==\forall j . j<k \longrightarrow$ triangular-column $j A$
lemma triangular-to-triangular: upper-triangular $A=$ triangular-to (dim-row $A$ )
A
unfolding triangular-to-def triangular-column-def upper-triangular-def
by auto
lemma triangular-toD [dest]:
triangular-to $k A \Longrightarrow j<k \Longrightarrow j<i \Longrightarrow i<\operatorname{dim}-$ row $A \Longrightarrow A \$ \$(i, j)=0$
unfolding triangular-to-def triangular-column-def by auto
lemma triangular-toI [intro]:
$(\bigwedge i j . j<k \Longrightarrow j<i \Longrightarrow i<$ dim-row $A \Longrightarrow A \$ \$(i, j)=0) \Longrightarrow$ triangular-to
$k A$
unfolding triangular-to-def triangular-column-def by auto
lemma triangle-growth:
assumes tri:triangular-to $k A$
and col:triangular-column $k A$
shows triangular-to (Suc k) A
unfolding triangular-to-def
proof (intro alli impI)
fix $i$ assume $i S k: i<S u c k$
show triangular-column i A
proof (cases $i=k$ )
case True
then show ?thesis using col by auto next
case False
then have $i<k$ using $i S k$ by auto
thus ?thesis using tri unfolding triangular-to-def by auto
qed
qed
lemma triangle-trans: triangular-to $k A \Longrightarrow k>k^{\prime} \Longrightarrow$ triangular-to $k^{\prime} A$ by (intro triangular-toI, elim triangular-to $D$, auto)

### 10.2 Algorithms for Triangulization

## context

fixes $m f::{ }^{\prime} a \Rightarrow^{\prime} a \Rightarrow{ }^{\prime} a \times{ }^{\prime} a \times{ }^{\prime} a$
begin
private fun mute $::$ ' $a \Rightarrow n a t \Rightarrow n a t \Rightarrow{ }^{\prime} a \times{ }^{\prime} a m a t \Rightarrow{ }^{\prime} a \times{ }^{\prime} a$ mat where mute $A$-ll $k l(r, A)=($ let $p=A \$ \$(k, l)$ in if $p=0$ then $(r, A)$ else case mf $A$-ll $p$ of $\left(q^{\prime}, p^{\prime}, g\right) \Rightarrow$ $\left(r * q^{\prime}\right.$, addrow $\left(-p^{\prime}\right) k l\left(\right.$ multrow $\left.\left.\left.k q^{\prime} A\right)\right)\right)$
lemma mute-preserves-dimensions:
assumes mute qkl(r,A)=($\left.r^{\prime}, A^{\prime}\right)$
shows $[$ simp $]$ : dim-row $A^{\prime}=\operatorname{dim}$-row $A$ and $[$ simp $]$ : dim-col $A^{\prime}=\operatorname{dim}-\operatorname{col} A$
using assms by (auto simp: Let-def split: if-splits prod.splits)
Algorithm mute $k l$ makes $k$-th row $l$-th column element to 0 .
lemma mute-makes-0 :
assumes mute-fun: mute-fun mf
assumes mute $(A \$ \$(l, l)) k l(r, A)=\left(r^{\prime}, A^{\prime}\right)$
$l<$ dim-row $A$
$l<\operatorname{dim}-\operatorname{col} A$
$k<$ dim-row $A$
$k \neq l$
shows $A^{\prime} \$ \$(k, l)=0$
proof -
define $a$ where $a=A \$ \$(l, l)$
define $b$ where $b=A \$ \$(k, l)$
let ? $m f=m f(A \$ \$(l, l))(A \$ \$(k, l))$
obtain $q^{\prime} p^{\prime} g$ where $i d$ : ? $m f=\left(q^{\prime}, p^{\prime}, g\right)$ by (cases ? $m f$, auto)
note $m f=$ mute-fun[unfolded mute-fun-def, rule-format, OF id]
from assms show ?thesis
unfolding mat-addrow-def using mf id by (auto simp: ac-simps Let-def split:
if-splits)
qed
It will not touch unexpected rows.
lemma mute-preserves:
mute $q k l(r, A)=\left(r^{\prime}, A^{\prime}\right) \Longrightarrow$
$i<$ dim-row $A \Longrightarrow$
$j<\operatorname{dim}-\operatorname{col} A \Longrightarrow$
$l<$ dim-row $A \Longrightarrow$
$k<$ dim-row $A \Longrightarrow$
$i \neq k \Longrightarrow$
$A^{\prime} \$ \$(i, j)=A \$ \$(i, j)$
by (auto simp: Let-def split: if-splits prod.splits)

It preserves 0s in the touched row.
lemma mute-preserves- 0 :

```
mute \(q k l(r, A)=\left(r^{\prime}, A^{\prime}\right) \Longrightarrow\)
    \(i<\) dim-row \(A \Longrightarrow\)
    \(j<\operatorname{dim}-\operatorname{col} A \Longrightarrow\)
    \(l<\) dim-row \(A \Longrightarrow\)
    \(k<\) dim-row \(A \Longrightarrow\)
    \(A \$ \$(i, j)=0 \Longrightarrow\)
    \(A \$ \$(l, j)=0 \Longrightarrow\)
    \(A^{\prime} \$ \$(i, j)=0\)
    by (auto simp: Let-def split: if-splits prod.splits)
```

Hence, it will respect partially triangular matrix.
lemma mute-preserves-triangle:
assumes $r A^{\prime}$ : mute qkl(r,A)=($\left.r^{\prime}, A^{\prime}\right)$
and triA: triangular-to $l A$
and $l k: l<k$
and $k r$ : $k<$ dim-row $A$
and $l r: l<$ dim-row $A$
and $l c: l<d i m-c o l A$
shows triangular-to $l A^{\prime}$
proof (rule triangular-toI)
fix $i j$
assume $j l: j<l$ and $j i: j<i$ and $i r^{\prime}: i<d i m$-row $A^{\prime}$
then have $A 0: A \$ \$(i, j)=0$ using triA $r A^{\prime}$ by auto
moreover have $A \$ \$(l, j)=0$ using triA jl jl lr by auto
moreover have $j c: j<\operatorname{dim}$-col $A$ using $j l l c$ by auto
moreover have $i r$ : $i<$ dim-row $A$ using $\mathrm{ir}^{\prime} r A^{\prime}$ by auto
ultimately show $A^{\prime} \$ \$(i, j)=0$
using mute-preserves- $0[O F r A 〕 l r k r$ by auto
qed
Recursive application of mute
private fun sub1 :: ' $a \Rightarrow n a t \Rightarrow n a t \Rightarrow{ }^{\prime} a \times$ ' $a$ mat $\Rightarrow{ }^{\prime} a \times{ }^{\prime} a$ mat
where sub1 q 0 l $r A=r A$
| sub1 $q($ Suc k) $l r A=$ mute $q(l+$ Suc $k) l(s u b 1 q k l r A)$
lemma sub1-preserves-dimensions[simp]:
sub1 qkl $(r, A)=\left(r^{\prime}, A^{\prime}\right) \Longrightarrow$ dim-row $A^{\prime}=$ dim-row $A$
sub1 qkl $(r, A)=\left(r^{\prime}, A^{\prime}\right) \Longrightarrow d i m-c o l A^{\prime}=d i m-\operatorname{col} A$
proof (induction $k$ arbitrary: $r^{\prime} A^{\prime}$ )
case (Suc k)
moreover obtain $r^{\prime} A^{\prime}$ where $r A^{\prime}$ : sub1 q $k l(r, A)=\left(r^{\prime}, A^{\prime}\right)$ by force
moreover fix $r^{\prime \prime} A^{\prime \prime}$ assume sub1 $q$ (Suc k) $l(r, A)=\left(r^{\prime \prime}, A^{\prime \prime}\right)$
ultimately show dim-row $A^{\prime \prime}=\operatorname{dim}$-row $A \operatorname{dim}-\operatorname{col} A^{\prime \prime}=\operatorname{dim}-c o l A$ by auto
qed auto
lemma sub1-closed [simp]:
sub1 qkl(r,A) $=\left(r^{\prime}, A^{\prime}\right) \Longrightarrow A \in$ carrier-mat $m n \Longrightarrow A^{\prime} \in$ carrier-mat $m n$
unfolding carrier-mat-def by auto

```
lemma sub1-preserves-diagnal:
    assumes sub1 qkl(r,A)=( \(\left.r^{\prime}, A^{\prime}\right)\)
    and \(l<\operatorname{dim}-c o l A\)
    and \(k+l<\) dim-row \(A\)
    shows \(A^{\prime} \$ \$(l, l)=A \$ \$(l, l)\)
using assms
proof -
    show \(k+l<\) dim-row \(A \Longrightarrow\) sub1 q \(k l(r, A)=\left(r^{\prime}, A^{\prime}\right) \Longrightarrow\)
        \(A^{\prime} \$ \$(l, l)=A \$ \$(l, l)\)
    proof (induction \(k\) arbitrary: \(r^{\prime} A^{\prime}\) )
        case (Suc k)
            obtain \(r^{\prime \prime} A^{\prime \prime}\) where \(r A^{\prime \prime}[\operatorname{simp}]\) : sub1 \(q k l(r, A)=\left(r^{\prime \prime}, A^{\prime \prime}\right)\) by force
            have \([\) simp \(]\) :dim-row \(A^{\prime \prime}=\operatorname{dim}\)-row \(A\) and \(\left[\right.\) simp] \(: \operatorname{dim}\)-col \(A^{\prime \prime}=\operatorname{dim}\)-col \(A\)
                using snd-conv sub1-preserves-dimensions \(\left[O F r A^{\prime}\right]\) by auto
            have \(A^{\prime \prime} \$ \$(l, l)=A \$ \$(l, l)\) using assms Suc by auto
            have \(r A^{\prime}\) : mute \(q(l+\) Suc \(k) l\left(r^{\prime \prime}, A^{\prime \prime}\right)=\left(r^{\prime}, A^{\prime}\right)\)
                using Suc by auto
            show ?case using subst mute-preserves[OF rA ] Suc assms by auto
    qed auto
qed
```

Triangularity is respected by sub1.

```
lemma sub1-preserves-triangle:
    assumes sub1 qkl(r,A)=( \(\left.r^{\prime}, A^{\prime}\right)\)
    and tri: triangular-to \(l A\)
    and \(l r: l<\) dim-row \(A\)
    and \(l c: l<d i m-\operatorname{col} A\)
    and \(l k r: l+k<\) dim-row \(A\)
    shows triangular-to \(l A^{\prime}\)
using assms
proof -
    show sub1 qkl(r,A)=(r', \(\left.A^{\prime}\right) \Longrightarrow l+k<\) dim-row \(A \Longrightarrow\)
        triangular-to \(l A^{\prime}\)
    proof (induction \(k\) arbitrary: \(r^{\prime} A^{\prime}\) )
    case (Suc k)
        then have sub1 \(q(\) Suc \(k) l(r, A)=\left(r^{\prime}, A^{\prime}\right)\) by auto
        moreover obtain \(r^{\prime \prime} A^{\prime \prime}\)
            where \(r A^{\prime \prime}\) : sub1 q \(k l(r, A)=\left(r^{\prime \prime}, A^{\prime \prime}\right)\) by force
        ultimately
            have \(r A^{\prime}\) : mute \(q(\) Suc \((l+k)) l\left(r^{\prime \prime}, A^{\prime \prime}\right)=\left(r^{\prime}, A^{\prime}\right)\) by auto
        have triangular-to \(l A^{\prime \prime}\) using \(r A^{\prime \prime}\) Suc by auto
        thus ?case
            using Suc assms mute-preserves-triangle [OF rA'] \(r A^{\prime \prime}\) by auto
        qed (insert assms,auto)
qed
context
```

```
    assumes mf: mute-fun mf
begin
lemma sub1-makes-0s:
    assumes sub1 (A $$ (l,l))kl (r,A)=(r',A')
    and lr:l<dim-row A
    and lc:l<dim-col A
    and li:l<i
    and}i\leqk+
    and }k+l<dim-row A
    shows }\mp@subsup{A}{}{\prime}$$(i,l)=
using assms
proof -
    show sub1 (A$$(l,l))kl(r,A)=(r',}\mp@subsup{A}{}{\prime})\Longrightarrowi\leqk+l\Longrightarrowk+l<dim-row
A\Longrightarrow
        A'$$(i,l)=0
    using lr lc li
    proof (induction k arbitrary: }\mp@subsup{r}{}{\prime}\mp@subsup{A}{}{\prime}
    case (Suc k)
        obtain }\mp@subsup{r}{}{\prime}\mp@subsup{A}{}{\prime}\mathrm{ where }r\mp@subsup{A}{}{\prime}:\operatorname{sub1}(A$$(l,l))kl(r,A)=(\mp@subsup{r}{}{\prime},\mp@subsup{A}{}{\prime})\mathrm{ by force
    fix }\mp@subsup{r}{}{\prime\prime}\mp@subsup{A}{}{\prime\prime
    from sub1-preserves-diagnal[OF rA] have AA':A $$ (l,l) = A' $$ (l,l) using
Suc(2-) by auto
    assume sub1 (A $$ (l,l)) (Suc k) l (r,A) = (r'\prime},\mp@subsup{A}{}{\prime\prime}
    then have rA":mute (A $$ (l,l)) (Suc (l+k))l(r', A')=(r'\prime,}\mp@subsup{A}{}{\prime\prime}
            using rA' by simp
    have ir: i< dim-row A using Suc by auto
    have il:i\not=l using li by auto
    have lr': l< dim-row A' using lr rA' by auto
    have lc':l<dim-col A' using lc rA' by auto
    have Slkr': Suc (l+k)<dim-row A' using Suc rA' by auto
    show }\mp@subsup{A}{}{\prime\prime}$$(i,l)=
    proof (cases Suc(l+k)=i)
            case True {
            have l:Suc (l+k)\not=l by auto
            show ?thesis
                using mute-makes-0[OF mf rA''[unfolded AA' lr' lc'Slkr' l] ir il rA'
                by (simp add:True)
            } next
            case False {
                    then have ikl: i\leqk+l using Suc by auto
            have ir':}i<\mathrm{ dim-row }\mp@subsup{A}{}{\prime}\mathrm{ using ir rA' by auto
            have lc': l< dim-col A' using lc rA' by auto
            have IH:A' $$(i,l)=0 using rA'Suc False by auto
            thus ?thesis using mute-preserves[OFrA" ir' lc ] rA' False Suc
                by simp
            }
            qed
    qed auto
qed
```

```
lemma sub1-triangulizes-column:
    assumes rA': sub1 (A $$ (l,l)) (dim-row A - Suc l) l (r,A) = (r',}\mp@subsup{A}{}{\prime}
    and tri:triangular-to l A
    and r:dim-row }A>
    and lr:l<dim-row A
    and lc:l<dim-col A
    shows triangular-column l A'
proof (intro triangular-columnI)
    fix }
    assume li:l<i
    assume ir: i< dim-row A'
        also have ... = dim-row A using sub1-preserves-dimensions[OF rA'] by auto
        also have ... = dim-row }A-l+l\mathrm{ using lr li by auto
        finally have ir2: i\leq dim-row A -l+l by auto
    show }\mp@subsup{A}{}{\prime}$$(i,l)=
        apply (subst sub1-makes-0s[OF rA' lr lc])
        using li ir assms
        by auto
qed
```

The algorithm sub1 increases the number of columns that form triangle.
lemma sub1-grows-triangle:
assumes $r A^{\prime}: \operatorname{sub1}(A \$ \$(l, l))($ dim-row $A-S u c l) l(r, A)=\left(r^{\prime}, A^{\prime}\right)$
and $r$ : dim-row $A>0$
and tri:triangular-to $l A$
and $l r: l<d i m$-row $A$
and $l c: l<\operatorname{dim}-\operatorname{col} A$
shows triangular-to (Suc l) $A^{\prime}$
proof -
have triangular-to $l A^{\prime}$
using sub1-preserves-triangle[OF rA] assms by auto
moreover have triangular-column $l A^{\prime}$
using sub1-triangulizes-column $[O F r A]$ assms by auto
ultimately show ?thesis by (rule triangle-growth)
qed
end

### 10.3 Finding Non-Zero Elements

private definition find-non0 :: nat $\Rightarrow$ 'a mat $\Rightarrow$ nat option where
find-non0 $l A=($ let is $=[$ Suc $l . .<\operatorname{dim}$-row $A]$;
Ais $=$ filter $(\lambda(i, A i l)$. Ail $\neq 0)(\operatorname{map}(\lambda i .(i, A \$ \$(i, l))) i s)$
in case Ais of []$\Rightarrow$ None $\mid-\Rightarrow$ Some (sel-fun Ais))
lemma find-non0: assumes sel-fun: det-selection-fun sel-fun
and res: find-non0 $l A=$ Some $m$
shows $A \$ \$(m, l) \neq 0 l<m m<$ dim-row $A$
proof -
let ?xs $=$ filter $(\lambda(i$, Ail $) . A i l \neq 0)(\operatorname{map}(\lambda i .(i, A \$ \$(i, l)))[$ Suc l.. $<$ dim-row A])
from res[unfolded find-non0-def Let-def]
have $x s:$ ? $x s \neq[]$ and $m: m=$ sel-fun ?xs
by (cases ? xs, auto) +
from det-selection-funD[OF sel-fun $x s$, folded $m$ ] show $A \$ \$(m, l) \neq 0 l<m$ $m<\operatorname{dim}$-row $A$ by auto
qed
If find-non0 $l A$ fails, then $A$ is already triangular to $l$-th column.
lemma find-non0-all0:
find-non0 l $A=$ None $\Longrightarrow$ triangular-column $l$ A
proof (intro triangular-columnI)
fix $i$
let ? $x s=$ filter $(\lambda(i$, Ail $)$. Ail $\neq 0)(\operatorname{map}(\lambda i .(i, A \$ \$(i, l)))[$ Suc l.. $<$ dim-row A])
assume none: find-non0 $l A=$ None and $l i: l<i i<d i m-r o w ~ A$
from none have $x s: ? x s=[]$
unfolding find-non0-def Let-def by (cases ?xs, auto)
from li have $(i, A \$ \$(i, l)) \in \operatorname{set}(\operatorname{map}(\lambda i .(i, A \$ \$(i, l)))[$ Suc l.. $<$ dim-row $A]$ ) by auto
with $x s$ show $A \$(i, l)=0$
by (metis (mono-tags) xs case-prodI filter-empty-conv)
qed

### 10.4 Determinant Preserving Growth of Triangle

The algorithm sub1 does not preserve determinants when it hits a 0 -valued diagonal element. To avoid this case, we introduce the following operation:

```
private fun sub2 :: nat \(\Rightarrow\) nat \(\Rightarrow{ }^{\prime} a \times\) 'a mat \(\Rightarrow{ }^{\prime} a \times\) 'a mat
    where sub2 \(d l(r, A)=(\)
        case find-non0 \(l\) A of None \(\Rightarrow(r, A)\)
        | Some \(m \Rightarrow\) let \(A^{\prime}=\) swaprows \(m l A\) in sub1 \(\left(A^{\prime} \$ \$(l, l)\right)(d-S u c l) l(-r\),
\(\left.A^{\prime}\right)\) )
```

lemma sub2-preserves-dimensions[simp]:
assumes $r A^{\prime}$ : sub2 d $l(r, A)=\left(r^{\prime}, A^{\prime}\right)$
shows dim-row $A^{\prime}=\operatorname{dim}$-row $A \wedge \operatorname{dim}-\operatorname{col} A^{\prime}=\operatorname{dim}-\operatorname{col} A$
proof (cases find-non0 l A)
case None then show ?thesis using $r A^{\prime}$ by auto next
case (Some $m$ ) then show ?thesis using $r A^{\prime}$ by (cases $m=l$, auto simp:
Let-def)
qed
lemma sub2-closed [simp]:
sub2 d $l(r, A)=\left(r^{\prime}, A^{\prime}\right) \Longrightarrow A \in$ carrier-mat $m n \Longrightarrow A^{\prime} \in$ carrier-mat $m n$
unfolding carrier-mat-def by auto
context
assumes sel-fun: det-selection-fun sel-fun
begin
lemma sub2-preserves-triangle:
assumes $r A^{\prime}$ : sub2 $d l(r, A)=\left(r^{\prime}, A^{\prime}\right)$
and tri: triangular-to $l A$
and $l c: l<d i m-c o l A$
and $l d: l<d$
and $d r: d \leq$ dim-row $A$
shows triangular-to $l A^{\prime}$
proof -
have $l r: l<d i m$-row $A$ using $l d d r$ by auto
show ?thesis
proof (cases find-non0 $l$ A)
case None then show ?thesis using $r A^{\prime}$ tri by simp next
case (Some m) \{
have $l m: l<m$ and $m r: m<$ dim-row $A$
using find-non0 $[O F$ sel-fun Some $]$ by auto let ? $A 1=$ swaprows $m l A$
have tri ${ }^{\prime \prime}$ : triangular-to $l$ ?A1
proof (intro triangular-toI)
fix $i j$
assume $j l: j<l$ and $j i: j<i$ and ir1: $i<$ dim-row? A1
have $j m: j<m$ using $j l m$ by auto
have ir: $i<$ dim-row $A$ using ir1 by auto
have $j c: j<d i m-c o l ~ A$ using $j l l c$ by auto
show ?A1 $\$ \$(i, j)=0$
proof (cases $m=i$ )
case True \{
then have $l i: l \neq i$ using $l m$ by auto
hence ? A1 $\$ \$(i, j)=A \$ \$(l, j)$ using ir $j c<m=i\rangle$ by auto
also have $\ldots=0$ using tri jl lr by auto
finally show ?thesis. \} next
case False show ?thesis
proof (cases $l=i$ )
case True \{
then have ?A1 $\$ \$(i, j)=A \$ \$(m, j)$
using ir $j c\langle m \neq i\rangle$ by auto
thus? A1 $\$ \$(i, j)=0$ using tri $j l j m \mathrm{mr}$ by auto
\} next
case False \{
then have ? $A 1 \$ \$(i, j)=A \$ \$(i, j)$
using ir $j c\langle m \neq i\rangle$ by auto
thus ? A1 $\$ \$(i, j)=0$ using tri jl ji ir by auto
\}
qed
qed

```
        qed
        let ?rA3 = sub1 (?A1 $$ (l,l)) (d-Suc l) l (-r,?A1)
        have [simp]: dim-row ?A1 = dim-row A ^ dim-col ?A1 = dim-col A by auto
        have rA'2: ?rA3 = ( }\mp@subsup{r}{}{\prime},\mp@subsup{A}{}{\prime})\mathrm{ using rA' Some by (simp add: Let-def)
        have l}+(d-Sucl)<dim-row A using ld dr by aut
        thus ?thesis
            using sub1-preserves-triangle[OF rA'2 tri'\ lr lc rA' by auto
        }
    qed
qed
lemma sub2-grows-triangle:
    assumes mf: mute-fun mf
    and rA': sub2 (dim-row A) l (r,A) = (r', A')
    and tri: triangular-to l A
    and lc:l<dim-col A
    and lr:l<dim-row A
    shows triangular-to (Suc l) A'
proof (rule triangle-growth)
    show triangular-to l A'
        using sub2-preserves-triangle[OF rA' tri lc lr] by auto
        next
    have r0: 0<dim-row A using lr by auto
    show triangular-column l A'
        proof (cases find-non0 l A)
        case None {
            then have }\mp@subsup{A}{}{\prime}=A\mathrm{ using }r\mp@subsup{A}{}{\prime}\mathrm{ by simp
            moreover have triangular-column l A using find-non0-allO[OF None].
            ultimately show ?thesis by auto
        } next
        case (Some m) {
            have lm:l<m and mr:m<dim-row A
                using find-non0[OF sel-fun Some] by auto
            let ?A = swaprows m l A
            have tri2: triangular-to l ?A
                proof
                fix i j assume jl: j<l and ji:j<i and ir: i< dim-row?A
                    show ?A $$ (i,j)=0
                proof (cases i=m)
                    case True {
                        then have ?A $$ (i,j)=A$$(l,j)
                        using jl lc ir by simp
                            also have ... = 0
                            using triangular-toD[OF tri jl jl] lr by auto
                            finally show ?thesis by auto
                    } next
                        case False show ?thesis
                        proof (cases i=l)
```

```
                    case True {
                        then have ?A $$ (i,j)=A$$(m,j)
                            using jl lc ir by auto
                        also have ... = 0
                            using triangular-toD[OF tri jl] jl lm mr by auto
                            finally show ?thesis by auto
                        } next
                        case False {
                            then have ?A $$ (i,j)=A$$(i,j)
                            using <i\not=m> jl lc ir by auto
                            thus ?thesis using tri jl ji ir by auto
                    }
                    qed
                qed
            qed
            have rA'2: sub1 (?A $$ (l,l)) (dim-row ?A - Suc l) l (-r,?A) = (r',A')
                using lm Some rA' by (simp add: Let-def)
            show ?thesis
            using sub1-triangulizes-column[OF mf rA'2 tri2] r0 lr lc by auto
        }
    qed
qed
end
```


### 10.5 Recursive Triangulization of Columns

Now we recursively apply sub2 to make the entire matrix to be triangular.

```
private fun sub3 :: nat }=>\mathrm{ nat }=>\mp@subsup{}{}{\prime}a\times'a mat = ' a > 'a mat
    where sub3 d 0 rA =rA
    | sub3 d (Suc l)rA= sub2 d l (sub3 d l rA)
lemma sub3-preserves-dimensions[simp]:
    sub3 d l (r,A) =( }\mp@subsup{r}{}{\prime},\mp@subsup{A}{}{\prime})\Longrightarrow\mathrm{ dim-row }\mp@subsup{A}{}{\prime}=\mathrm{ dim-row }
    sub3 d l (r,A) = (r',}\mp@subsup{A}{}{\prime})\Longrightarrowdim-col A' = dim-col A
proof (induction l arbitrary: r' }\mp@subsup{A}{}{\prime}\mathrm{ )
    case (Suc l)
        obtain r'A' where rA': sub3 d l (r,A) = (r', A') by force
        fix }\mp@subsup{r}{}{\prime\prime}\mp@subsup{A}{}{\prime\prime}\mathrm{ assume rA'": sub3 d (Suc l) (r,A) = (r'', A')
        then show dim-row }\mp@subsup{A}{}{\prime\prime}=dim-row A dim-col A" = dim-col A
        using Suc rA' by auto
qed auto
lemma sub3-closed[simp]:
    sub3 kl(r,A)=( r', A')\LongrightarrowA\in carrier-mat m n \Longrightarrow A' \in carrier-mat m n
    unfolding carrier-mat-def by auto
lemma sub3-makes-triangle:
    assumes mf: mute-fun mf
    and sel-fun: det-selection-fun sel-fun
```

```
    and sub3 (dim-row A) l (r,A) = (r',}\mp@subsup{A}{}{\prime}
    and l}\leq\mathrm{ dim-row }
    and l\leqdim-col A
    shows triangular-to l A'
    using assms
proof -
    show sub3 (dim-row A) l(r,A)=(r',}\mp@subsup{A}{}{\prime})\Longrightarrowl\leq\operatorname{dim-row A\Longrightarrowl\leqdim-col A
C
    triangular-to l A'
    proof (induction l arbitrary:r' }\mp@subsup{A}{}{\prime}\mathrm{ )
    case (Suc l)
        then have Slr:Suc l \leq dim-row A and Slc: Suc l\leqdim-col A by auto
        hence lr:l<dim-row A and lc:l<dim-col A by auto
        moreover obtain r" }\mp@subsup{|}{}{\prime\prime
            where r\mp@subsup{A}{}{\prime\prime}: sub3 (dim-row A) l (r,A) = (r'\prime},\mp@subsup{A}{}{\prime\prime})\mathrm{ by force
            ultimately have IH: triangular-to l }\mp@subsup{A}{}{\prime\prime}\mathrm{ using Suc by auto
            have [simp]:dim-row A'\prime = dim-row A and [simp]:dim-col A" = dim-col A
                using Slr Slc rA" by auto
            fix r r'A
            assume sub3 (dim-row A) (Suc l) (r,A) = (r', A')
            then have rA': sub2 (dim-row A') l ( }\mp@subsup{r}{}{\prime\prime},\mp@subsup{A}{}{\prime\prime})=(\mp@subsup{r}{}{\prime},\mp@subsup{A}{}{\prime}
                using rA'" by auto
            show triangular-to (Suc l) A'
            using sub2-grows-triangle[OF sel-fun mf rA` lr lc rA"IH by auto
    qed auto
qed
```


### 10.6 Triangulization

definition triangulize :: 'a mat $\Rightarrow{ }^{\prime} a \times{ }^{\prime} a$ mat
where triangulize $A=$ sub3 (dim-row $A$ ) (dim-row $A)(1, A)$
lemma triangulize-preserves-dimensions [simp]:
triangulize $A=\left(r^{\prime}, A^{\prime}\right) \Longrightarrow$ dim-row $A^{\prime}=$ dim-row $A$
triangulize $A=\left(r^{\prime}, A^{\prime}\right) \Longrightarrow d i m-c o l A^{\prime}=\operatorname{dim}-\operatorname{col} A$
unfolding triangulize-def by auto
lemma triangulize-closed $[$ simp $]$ :
triangulize $A=\left(r^{\prime}, A^{\prime}\right) \Longrightarrow A \in$ carrier-mat $m n \Longrightarrow A^{\prime} \in$ carrier-mat $m n$ unfolding carrier-mat-def by auto
context
assumes $m f$ : mute-fun $m f$
and sel-fun: det-selection-fun sel-fun
begin
theorem triangulized:
assumes $A \in$ carrier-mat $n n$
and triangulize $A=\left(r^{\prime}, A^{\prime}\right)$

```
    shows upper-triangular A'
proof (cases 0<n)
    case True
        have rA': sub3 (dim-row A) (dim-row A) (1,A)=( (r',A')
            using assms unfolding triangulize-def by auto
        have nr:n=dim-row A and nc:n=\operatorname{dim}\mathrm{ -col A and nr':n=dim-row A'}
    using assms by auto
    thus ?thesis
            unfolding triangular-to-triangular
            using sub3-makes-triangle[OF mf sel-fun rA] True by auto
    next
    case False
    then have nr':dim-row }\mp@subsup{A}{}{\prime}=0\mathrm{ using assms by auto
    thus ?thesis unfolding upper-triangular-def by auto
qed
```


### 10.7 Divisor will not be 0

Here we show that each sub-algorithm will not make $r$ of the input/output pair $(r, A)$ to 0 . The algorithm sub1 $A$-ll $k l(r, A)$ requires $A_{l, l} \neq 0$.

```
lemma sub1-divisor [simp]:
    assumes \(r A^{\prime}\) : sub1 q kl( \(\left.r, A\right)=\left(r^{\prime}, A^{\prime}\right)\)
    and \(r 0: r \neq 0\)
    and All: \(q \neq 0\)
    and \(k+l<\) dim-row \(A\)
    and \(l c: l<d i m-\operatorname{col} A\)
    shows \(r^{\prime} \neq 0\)
using assms
proof -
    show sub1 qkl(r,A)=(\(\left.r^{\prime}, A^{\prime}\right) \Longrightarrow k+l<\operatorname{dim}\)-row \(A \Longrightarrow r^{\prime} \neq 0\)
    proof (induction \(k\) arbitrary: \(r^{\prime} A^{\prime}\) )
        case (Suc \(k\) )
        obtain \(r^{\prime \prime} A^{\prime \prime}\) where \(r A^{\prime \prime}\) : sub1 qkl(r,A)=( \(\left.r^{\prime \prime}, A^{\prime \prime}\right)\) by force
        then have \(I H: r^{\prime \prime} \neq 0\) using Suc by auto
        obtain \(q^{\prime} l^{\prime} g\) where \(m f\)-id: mf \(q\left(A^{\prime \prime} \$ \$(S u c(l+k), l)\right)=\left(q^{\prime}, l^{\prime}, g\right)\) by (rule
prod-cases3)
        define fact where fact \(=\left(\right.\) if \(A^{\prime \prime} \$ \$(S u c(l+k), l)=0\) then 1 else \(\left.q^{\prime}\right)\)
        note \(m f=m f[u n f o l d e d\) mute-fun-def, rule-format, OF \(m f\)-id]
        have All: \(q \neq 0\)
            using sub1-preserves-diagnal \(\left[\right.\) OF \(\left.r A^{\prime \prime} l c\right]\) All Suc by auto
        moreover have fact \(\neq 0\) unfolding fact-def using All mf by auto
        moreover assume sub1 \(q\) (Suc \(k) l(r, A)=\left(r^{\prime}, A^{\prime}\right)\)
            then have mute \(q(S u c(l+k)) l\left(r^{\prime \prime}, A^{\prime \prime}\right)=\left(r^{\prime}, A^{\prime}\right)\)
                using \(r A^{\prime \prime}\) by auto
            hence \(r^{\prime \prime} *\) fact \(=r^{\prime}\)
                unfolding mute.simps fact-def Let-def mf-id using IH by (auto split:
if-splits)
        ultimately show \(r^{\prime} \neq 0\) using \(I H\) by auto
    qed (insert r0, simp)
```


## qed

The algorithm sub2 will not require such a condition.

```
lemma sub2-divisor [simp]:
    assumes \(r A^{\prime}\) : sub2 \(k l(r, A)=\left(r^{\prime}, A^{\prime}\right)\)
    and \(l k: l<k\)
    and \(k r: k \leq\) dim-row \(A\)
    and \(l c: l<d i m-\operatorname{col} A\)
    and \(r 0: r \neq 0\)
    shows \(r^{\prime} \neq 0\)
using assms
proof (cases find-non0 l A) \{
    case (Some m)
    then have Aml0: A \(\$ \$(m, l) \neq 0\) using find-non0[OF sel-fun] by auto
    have \(m d: m<\) dim-row \(A\) using find-non0[OF sel-fun Some] \(l k k r\) by auto
    let \(? A^{\prime \prime}=\) swaprows \(m l A\)
    have \(r A^{\prime}\) 2: sub1 \(\left(? A^{\prime \prime} \$ \$(l, l)\right)(k-S u c l) l\left(-r, ? A^{\prime \prime}\right)=\left(r^{\prime}, A^{\prime}\right)\)
        using \(r A^{\prime}\) Some by (simp add: Let-def)
    have Allo: ? \(A^{\prime \prime} \$ \$(l, l) \neq 0\) using Aml0 md lk kr lc by auto
    show ?thesis using sub1-divisor[OF rA'2-All0] r0 lk kr lc by simp
\} qed auto
lemma sub3-divisor [simp]:
    assumes sub3 \(d l(r, A)=\left(r^{\prime \prime}, A^{\prime \prime}\right)\)
    and \(l \leq d\)
    and \(d \leq\) dim-row \(A\)
    and \(l \leq \operatorname{dim}-c o l A\)
    and \(r 0: r \neq 0\)
    shows \(r^{\prime \prime} \neq 0\)
    using assms
proof -
    show
        sub3 \(d l(r, A)=\left(r^{\prime \prime}, A^{\prime \prime}\right) \Longrightarrow\)
        \(l \leq d \Longrightarrow d \leq\) dim-row \(A \Longrightarrow l \leq\) dim-col \(A \Longrightarrow\) ?thesis
    proof (induction l arbitrary: \(r^{\prime \prime} A^{\prime \prime}\) )
        case 0
            then show ?case using r0 by simp
            next
    case (Suc l)
                obtain \(r^{\prime} A^{\prime}\) where \(r A^{\prime}\) : sub3 \(d l(r, A)=\left(r^{\prime}, A^{\prime}\right)\) by force
                then have \([\) simp \(]\) :dim-row \(A^{\prime}=\operatorname{dim}\)-row \(A\) and \([\operatorname{simp}]: d i m-c o l ~ A^{\prime}=d i m-c o l\)
A
            by auto
            from \(r A^{\prime}\) have \(r^{\prime} \neq 0\) using Suc r0 by auto
            moreover have sub2 \(d l\left(r^{\prime}, A^{\prime}\right)=\left(r^{\prime \prime}, A^{\prime \prime}\right)\)
            using \(r A^{\prime}\) Suc by simp
            ultimately show ?case using sub2-divisor using Suc by simp
    qed
qed
```

```
theorem triangulize-divisor:
    assumes A: A c carrier-mat d d
    shows triangulize }A=(\mp@subsup{r}{}{\prime},\mp@subsup{A}{}{\prime})\Longrightarrow\mp@subsup{r}{}{\prime}\not=
unfolding triangulize-def
proof -
    assume rA': sub3 (dim-row A) (dim-row A) (1, A) = (r', A')
    show ?thesis using sub3-divisor[OF rA] A by auto
qed
```


### 10.8 Determinant Preservation Results

For each sub-algorithm $f$, we show $f(r, A)=\left(r^{\prime}, A^{\prime}\right)$ implies $r * \operatorname{det} A^{\prime}=r^{\prime}$ * $\operatorname{det} A$.
lemma mute-det:
assumes $A \in$ carrier-mat $n n$
and $r A^{\prime}:$ mute $q k l(r, A)=\left(r^{\prime}, A^{\prime}\right)$
and $k<n$
and $l<n$
and $k \neq l$
shows $r * \operatorname{det} A^{\prime}=r^{\prime} * \operatorname{det} A$
proof (cases $A \$ \$(k, l)=0)$
case True
thus ?thesis using assms by auto
next
case False
obtain $p^{\prime} q^{\prime} g$ where $m f$-id: $m f q(A \$ \$(k, l))=\left(q^{\prime}, p^{\prime}, g\right)$ by (rule prod-cases3)
let ?All $=q^{\prime}$
let ? $A k l=-p^{\prime}$
let $? B=$ multrow $k$ ?All $A$
let ? $C=$ addrow ? $A k l k l$ ? $B$
have $r * \operatorname{det} A^{\prime}=r * \operatorname{det} ? C$ using assms by (simp add: Let-def mf-id False)
also have det ? $C=$ det ?B using assms by (auto simp: det-addrow)
also have $\ldots=$ ?All $* \operatorname{det} A$ using assms det-multrow by auto
also have $r * \ldots=(r *$ ? All $) * \operatorname{det} A$ by $\operatorname{simp}$
also have $r: r *$ ? All $=r^{\prime}$ using assms by (simp add: Let-def mf-id False)
finally show ?thesis.
qed
lemma sub1-det:
assumes $A: A \in$ carrier-mat $n n$
and sub1: sub1 q $k l(r, A)=\left(r^{\prime \prime}, A^{\prime \prime}\right)$
and $r 0: r \neq 0$
and All0: $q \neq 0$
and $l: l+k<n$
shows $r * \operatorname{det} A^{\prime \prime}=r^{\prime \prime} * \operatorname{det} A$
using sub1 $l$
proof (induction $k$ arbitrary: $A^{\prime \prime} r^{\prime \prime}$ )
case 0

```
    then show ?case by auto
next
    case (Suc k)
    let ?rA' = sub1 qkl (r,A)
    obtain }\mp@subsup{r}{}{\prime}\mp@subsup{A}{}{\prime}\mathrm{ where }r\mp@subsup{A}{}{\prime}:?r\mp@subsup{A}{}{\prime}=(\mp@subsup{r}{}{\prime},\mp@subsup{A}{}{\prime})\mathrm{ by force
    have }\mp@subsup{A}{}{\prime}:\mp@subsup{A}{}{\prime}\in\mathrm{ carrier-mat n n using sub1-closed [OF rA] A by auto
    have IH:r*\operatorname{det A'}=\mp@subsup{r}{}{\prime}*\operatorname{det}A\mathrm{ using Suc assms rA' by auto}
    assume sub1 q (Suc k) l (r,A) = (r'\prime,}\mp@subsup{A}{}{\prime\prime}
    then have rA'\prime:mute q(Suc (l+k)) l (r',A')=( (r',}\mp@subsup{A}{}{\prime\prime})\mathrm{ using rA' by auto
    hence lem: }\mp@subsup{r}{}{\prime}*\operatorname{det}\mp@subsup{A}{}{\prime\prime}=\mp@subsup{r}{}{\prime\prime}*\operatorname{det}\mp@subsup{A}{}{\prime
        using assms Suc A' mute-det[OF A'rA'` by auto
    hence r* r'* det A'首 =r* 年* det A' by auto
    also from IH have ... = r'\prime* r'* det A by auto
    finally have *: r* r'* det A'\prime}=\mp@subsup{r}{}{\prime\prime}*\mp@subsup{r}{}{\prime}*\operatorname{det}A\mathrm{ .
    then have r* r'* det A'\prime div r'}=\mp@subsup{r}{}{\prime\prime}*\mp@subsup{r}{}{\prime}*\operatorname{det}A\mathrm{ div r' by presburger
    moreover have r'}=
        using r0 sub1-divisor[OF rA] AllO Suc A by auto
    ultimately show ?case using * by auto
qed
lemma sub2-det:
    assumes A: A \in carrier-mat d d
    and rA': sub2 d l (r,A) = (r',A')
    and r0:r\not=0
    and ld:l<d
    shows r*\operatorname{det}\mp@subsup{A}{}{\prime}=\mp@subsup{r}{}{\prime}*\operatorname{det}A
proof (cases find-non0 l A)
    case None then show ?thesis using assms by auto next
    case (Some m) {
        then have lm:l<m and md:m<d
            using A find-non0[OF sel-fun Some] ld by auto
    hence m\not=l by auto
    let ? }\mp@subsup{A}{}{\prime\prime}=\mathrm{ swaprows m l A
    have rA'2: sub1 (?A'\prime $$ (l,l)) (d - Suc l) l (-r,? ?'') = ( r',}\mp@subsup{A}{}{\prime}
            using rA' Some by (simp add: Let-def)
    have }\mp@subsup{A}{}{\prime\prime}:?\mp@subsup{A}{}{\prime\prime}\in\mathrm{ carrier-mat }dd\mathrm{ using }A\mathrm{ by auto
    hence A'll0: ?A"' $$ (l,l)\not=0
            using find-non0[OF sel-fun Some] ld by auto
    hence -r*\operatorname{det}\mp@subsup{A}{}{\prime}=\mp@subsup{r}{}{\prime}*\operatorname{det}?\mp@subsup{A}{}{\prime\prime}
            using sub1-det[OF A'trA'2] ld A r0 by auto
    also have r*\ldots= ..r* r'* det A
            using det-swaprows[OF md ld <m\not=l>A] by auto
    finally have r*-r*\operatorname{det A'}=-r*\mp@subsup{r}{}{\prime}*\operatorname{det}A\mathrm{ by auto}
    thus ?thesis using r0 by auto
    }
qed
lemma sub3-det:
    assumes A:A\in carrier-mat d d
```

```
    and sub3 \(d l(r, A)=\left(r^{\prime \prime}, A^{\prime \prime}\right)\)
    and \(r 0: r \neq 0\)
    and \(l \leq d\)
    shows \(r * \operatorname{det} A^{\prime \prime}=r^{\prime \prime} * \operatorname{det} A\)
    using assms
proof -
    have \(d: d=\) dim-row \(A\) using \(A\) by auto
    show sub3 \(d l(r, A)=\left(r^{\prime \prime}, A^{\prime \prime}\right) \Longrightarrow l \leq d \Longrightarrow r * \operatorname{det} A^{\prime \prime}=r^{\prime \prime} * \operatorname{det} A\)
    proof (induction l arbitrary: \(r^{\prime \prime} A^{\prime \prime}\) )
    case (Suc l)
        let ? \(r A^{\prime}=\operatorname{sub3} d l(r, A)\)
        obtain \(r^{\prime} A^{\prime}\) where \(r A^{\prime}\) ? ? \(r A^{\prime}=\left(r^{\prime}, A^{\prime}\right)\) by force
        then have \(r A^{\prime \prime}: \operatorname{sub2} d l\left(r^{\prime}, A^{\prime}\right)=\left(r^{\prime \prime}, A^{\prime \prime}\right)\)
            using Suc by auto
        have \(A^{\prime}: A^{\prime} \in\) carrier-mat \(d d\) using \(A r A^{\prime} r A^{\prime \prime}\) by auto
        have \(r^{\prime} 0: r^{\prime} \neq 0\) using r0 sub3-divisor[OF rA] A Suc by auto
        have \(r^{\prime} * \operatorname{det} A^{\prime \prime}=r^{\prime \prime} * \operatorname{det} A^{\prime}\)
            using Suc \(r^{\prime} 0 A \operatorname{by}\left(\right.\) subst sub2- \(\operatorname{det}\left[O F A^{\prime} r A^{\prime \prime}\right]\), auto)
        also have \(r * \ldots=r^{\prime \prime} *\left(r * \operatorname{det} A^{\prime}\right)\) by auto
        also have \(r * \operatorname{det} A^{\prime}=r^{\prime} * \operatorname{det} A\) using Suc r \(A^{\prime}\) by auto
        also have \(r^{\prime \prime} * \ldots\) div \(r^{\prime}=r^{\prime \prime} * r^{\prime} * \operatorname{det} A\) div \(r^{\prime}\) by (simp add: ac-simps)
        finally show \(r * \operatorname{det} A^{\prime \prime}=r^{\prime \prime} * \operatorname{det} A\) using \(r^{\prime} 0\)
            by (metis \(\left\langle r * \operatorname{det} A^{\prime}=r^{\prime} * \operatorname{det} A\right\rangle\left\langle r^{\prime} * \operatorname{det} A^{\prime \prime}=r^{\prime \prime} * \operatorname{det} A^{\prime}\right\rangle\)
                mult.left-commute mult-cancel-left)
    qed \(\operatorname{simp}\)
qed
theorem triangulize-det:
    assumes \(A: A \in\) carrier-mat \(d d\)
    and \(r A^{\prime}\) : triangulize \(A=\left(r^{\prime}, A^{\prime}\right)\)
    shows \(\operatorname{det} A * r^{\prime}=\operatorname{det} A^{\prime}\)
proof -
    have \(r A^{\prime}\) 2: sub3 \(d d(1, A)=\left(r^{\prime}, A^{\prime}\right)\)
        using \(A r A^{\prime}\) unfolding triangulize-def by auto
    show ?thesis
    proof (cases \(d=0\) )
        case True
            then have \(A^{\prime}: A^{\prime} \in\) carrier-mat 00 using \(A r A^{\prime} 2\) by auto
            have \(r A^{\prime} 3:\left(r^{\prime}, A^{\prime}\right)=(1, A)\) using True \(r A^{\prime} 2\) by simp
            thus ?thesis by auto
            next
        case False
            then show ?thesis using sub3-det[OF A rA'2] assms by auto
        qed
qed
end
```


### 10.9 Determinant Computation

definition det-code :: 'a mat $\Rightarrow{ }^{\prime} a$ where
det-code $A=$ (if dim-row $A=\operatorname{dim}$-col $A$ then
case triangulize $A$ of $\left(m, A^{\prime}\right) \Rightarrow$ prod-list (diag-mat $A^{\prime}$ ) div m
else 0)
lemma det-code[simp]: assumes sel-fun: det-selection-fun sel-fun
and $m f$ : mute-fun $m f$
shows $d e t$-code $A=\operatorname{det} A$
using det-code-def[simp]
proof (cases dim-row $A=\operatorname{dim}-\operatorname{col} A$ )
case True
then have $A: A \in$ carrier-mat (dim-row $A$ ) (dim-row $A$ ) unfolding carrier-mat-def by auto
obtain $r^{\prime} A^{\prime}$ where $r A^{\prime}$ : triangulize $A=\left(r^{\prime}, A^{\prime}\right)$ by force
from triangulize-divisor $[$ OF mf sel-fun $A] r A^{\prime}$ have $r^{\prime} 0: r^{\prime} \neq 0$ by auto
from triangulize- $\operatorname{det}\left[O F m f\right.$ sel-fun $\left.A r A^{\prime}\right]$ have $\operatorname{det}^{\prime}: \operatorname{det} A * r^{\prime}=\operatorname{det} A^{\prime}$ by auto
from triangulized $\left[O F m f\right.$ sel-fun $A$, unfolded $\left.r A^{\prime}\right]$ have tri': upper-triangular $A^{\prime}$ by $\operatorname{simp}$
have $A^{\prime}: A^{\prime} \in$ carrier-mat (dim-row $\left.A^{\prime}\right)\left(\right.$ dim-row $\left.A^{\prime}\right)$
using triangulize-closed $\left[O F r A^{\prime} A\right]$ by auto
from tri' have tr: triangular-to (dim-row $A^{\prime}$ ) $A^{\prime}$ by auto
have det-code $A=$ prod-list (diag-mat $A^{\prime}$ ) div $r^{\prime}$ using $r A^{\prime}$ True by simp
also have prod-list $\left(\operatorname{diag}-m a t A^{\prime}\right)=\operatorname{det} A^{\prime}$
unfolding det-upper-triangular[OF tri' $A$ ] ..
also have $\ldots=\operatorname{det} A * r^{\prime}$ by ( $\left.\operatorname{simp} a d d: \operatorname{det}^{\prime}\right)$
also have $\ldots$ div $r^{\prime}=\operatorname{det} A$ using $r^{\prime} 0$ by auto
finally show ?thesis .
qed (simp add: det-def)
end
end
Now we can select an arbitrary selection and mute function. This will be important for computing resultants over polynomials, where usually a polynomial with small degree is preferable.

The default however is to use the first element.
definition trivial-mute-fun :: ' $a::$ comm-ring- $1 \Rightarrow{ }^{\prime} a \Rightarrow^{\prime} a \times{ }^{\prime} a \times{ }^{\prime} a$ where
trivial-mute-fun $x y=(x, y, 1)$
lemma trivial-mute-fun[simp,intro]: mute-fun trivial-mute-fun unfolding mute-fun-def trivial-mute-fun-def by auto
definition fst-sel-fun :: 'a det-selection-fun where $f s t-s e l-f u n x=f s t(h d x)$

```
lemma fst-sel-fun[simp]: det-selection-fun fst-sel-fun
    unfolding det-selection-fun-def fst-sel-fun-def by auto
context
    fixes measure :: 'a }=\mathrm{ nat
begin
private fun select-min-main where
    select-min-main mi ((j,p) # xs)=(let n= measure p in if n < m then se-
lect-min-main n j xs
    else select-min-main m i xs)
| select-min-main mi [] = i
definition select-min :: (nat }\times\mp@subsup{}{}{\prime}a) list => nat wher
    select-min xs =( case xs of ((i,p)#ys)=>(select-min-main (measure p)iys))
lemma select-min[simp]:det-selection-fun select-min
    unfolding det-selection-fun-def
proof (intro allI impI)
    fix xs :: (nat > 'a)list
    assume xs \not= []
    then obtain i p ys where xs: xs =((i,p) # ys) by (cases xs,auto)
    then obtain m}\mathrm{ where id: select-min xs=select-min-main m i ys unfolding
select-min-def by auto
    have i\infst' set xs set ys \subseteq set xs unfolding xs by auto
    thus select-min xs \infst'set xs unfolding id
    proof (induct ys arbitrary:m i)
        case (Cons jp ys m i)
        obtain j p where jp: jp=(j,p) by force
        obtain kn where res: select-min-main mi (jp # ys) = select-min-main n k
ys
                and k: k\infst'set xs
                using Cons(2-) unfolding jp by (cases measure p<m; force simp: Let-def)
        from Cons(1)[OF k, of n] Cons(3)
        show ?case unfolding res by auto
    qed simp
qed
end
```

For the code equation we use the trivial mute and selection function as this does not impose any further class restrictions.
lemma det-code-fst-sel-fun[code]: det $A=$ det-code fst-sel-fun trivial-mute-fun $A$ by $\operatorname{simp}$

But we also provide specialiced functions for more specific carriers.
definition field-mute-fun :: ' $a::$ field $\Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a \times{ }^{\prime} a \times{ }^{\prime} a$ where
field-mute-fun $x y=(x / y, 1, y)$
lemma field-mute-fun[simp,intro]: mute-fun field-mute-fun
unfolding mute-fun-def field-mute-fun-def by auto

```
definition det-field :: 'a :: field mat }=>\mp@subsup{}{}{\prime}'a\mathrm{ where
    det-field A = det-code fst-sel-fun field-mute-fun A
lemma det-field[simp]: det-field = det
    by (intro ext, auto simp: det-field-def)
definition gcd-mute-fun :: ' }a::\mathrm{ ring-gcd }=>\mp@subsup{}{}{\prime}a=>'a\times' 'a\times' a wher
    gcd-mute-fun x y = (let g=gcd x y in (x div g, y div g,g))
lemma gcd-mute-fun[simp,intro]: mute-fun gcd-mute-fun
    unfolding mute-fun-def gcd-mute-fun-def by (auto simp: Let-def div-mult-swap
mult.commute)
definition det-int :: int mat }=>\mathrm{ int where
    det-int A = det-code (select-min ( }\lambda\mathrm{ x. nat (abs x))) gcd-mute-fun A
lemma det-int[simp]: det-int = det
    by (intro ext, auto simp: det-int-def)
definition det-field-poly :: 'a :: {field,field-gcd} poly mat }=>\mathrm{ ' 'a poly where
    det-field-poly A = det-code (select-min degree) gcd-mute-fun A
lemma det-field-poly[simp]: det-field-poly = det
    by (intro ext, auto simp: det-field-poly-def)
end
```


## 11 Converting Matrices to Strings

We just instantiate matrices in the show-class by printing them as lists of lists.

```
theory Show-Matrix
imports
    Show.Show
    Matrix
begin
definition shows-vec :: ' }a\mathrm{ :: show vec }=>\mathrm{ shows where
    shows-vec v \equivshows (list-of-vec v)
instantiation vec :: (show) show
begin
definition shows-prec p (v :: 'a vec) \equiv shows-vec v
definition shows-list (vs :: 'a vec list) \equiv shows-sep shows-vec (shows '", ') vs
```

instance

```
by standard (simp-all add: shows-vec-def show-law-simps shows-prec-vec-def shows-list-vec-def)
end
definition shows-mat :: 'a :: show mat }=>\mathrm{ shows where
    shows-mat A \equiv shows (mat-to-list A)
instantiation mat :: (show) show
begin
definition shows-prec p (A :: 'a mat) \equiv shows-mat A
definition shows-list (As :: 'a mat list) \equiv shows-sep shows-mat (shows '", ') As
instance
    by standard (simp-all add: shows-mat-def show-law-simps shows-prec-mat-def
shows-list-mat-def)
end
end
```


## 12 Characteristic Polynomial

We define eigenvalues, eigenvectors, and the characteristic polynomial. We further prove that the eigenvalues are exactly the roots of the characteristic polynomial. Finally, we apply the fundamental theorem of algebra to show that the characteristic polynomial of a complex matrix can always be represented as product of linear factors $x-a$.

```
theory Char-Poly
imports
    Polynomial-Factorization.Fundamental-Theorem-Algebra-Factorized
    Polynomial-Interpolation.Missing-Polynomial
    Polynomial-Interpolation.Ring-Hom-Poly
    Determinant
    Complex-Main
begin
```

definition eigenvector :: 'a :: comm-ring-1 mat $\Rightarrow{ }^{\prime} a$ vec $\Rightarrow{ }^{\prime} a \Rightarrow$ bool where
eigenvector $A v k=\left(v \in\right.$ carrier-vec $(\operatorname{dim}-r o w A) \wedge v \neq 0_{v}(\operatorname{dim}-r o w A) \wedge A *_{v}$
$v=k \cdot v v$ )
lemma eigenvector-pow: assumes $A: A \in$ carrier-mat $n n$
and ev: eigenvector $A v(k:: ' a$ :: comm-ring-1)
shows $A \widehat{m}_{m} i *_{v} v=k \widehat{\imath} \cdot{ }_{v} v$
proof -
let ? $G=$ monoid-vec TYPE ('a) n
from $A$ have $\operatorname{dim}$ : dim-row $A=n$ by auto
from ev[unfolded eigenvector-def dim]
have $v: v \in$ carrier-vec $n$ and $A v: A *_{v} v=k \cdot{ }_{v} v$ by auto
interpret $v$ : comm-group? $G$ by (rule comm-group-vec)

```
show ?thesis
proof (induct i)
    case 0
    show ?case using v dim by simp
next
    case (Suc i)
    define P where P=A \widehat{m}}\mp@subsup{}{}{\prime
    have P:P\incarrier-mat n n using A unfolding P-def by simp
    have A\widehat{m}}\mp@subsup{}{m}{}\mathrm{ Suc i}=P*A\mathrm{ unfolding P-def by simp
    also have \ldots*vv
    also have }A\mp@subsup{*}{v}{}v=k\cdotvvv by (rule Av
    also have P** (k\cdotv}v)=k\cdotv(P**v
        by (rule eq-vecI, insert v P, auto)
    also have (P**v) = (k^i) 汭v unfolding P-def by (rule Suc)
```



```
        by (rule eq-vecI, insert v, auto)
    also have k* k` i=k^ (Suc i) by auto
    finally show ?case .
    qed
qed
```

definition eigenvalue :: ' $a$ :: comm-ring-1 mat $\Rightarrow{ }^{\prime} a \Rightarrow$ bool where eigenvalue $A k=(\exists v$. eigenvector $A v k)$
definition char-matrix $::$ ' $a::$ field mat $\Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a$ mat where char-matrix $A e=A+\left((-e) \cdot m\left(1_{m}(\right.\right.$ dim-row $\left.\left.A)\right)\right)$
lemma char-matrix-closed[simp]: $A \in$ carrier-mat $n n \Longrightarrow$ char-matrix $A e \in$ carrier-mat $n n$
unfolding char-matrix-def by auto
lemma eigenvector-char-matrix: assumes $A:(A:: ' a$ :: field mat $) \in$ carrier-mat $n n$
shows eigenvector $A v e=\left(v \in\right.$ carrier-vec $n \wedge v \neq 0_{v} n \wedge$ char-matrix $A e *_{v}$
$v=0_{v} n$ )
proof -
from $A$ have dim: dim-row $A=n \operatorname{dim}-\operatorname{col} A=n$ by auto \{
assume $v: v \in$ carrier-vec $n$
hence dimv: dim-vec $v=n$ by auto
have $\left(A *_{v} v=e \cdot v v\right)=\left(A *_{v} v+(-e) \cdot v v=0_{v} n\right)(\mathbf{i s}$ ? id1 $=? i d 2)$
proof
assume ?id1
from arg-cong[OF this, of $\lambda w . w+(-e) \cdot v v]$
show ?id2 using $A v$ by auto
next

```
        assume ?id2
        have}A*\mp@subsup{v}{v}{}v+-e\cdotvvv+e e\cdotv v=A*\mp@subsup{v}{v}{}v\mathrm{ using A v by auto
        from arg-cong[OF<?id2\rangle, of \lambda w.w+e\cdotv v, unfolded this]
        show ?id1 using A v by simp
        qed
    also have (A*vv+(-e)\cdotvv)=char-matrix A e *vv unfolding char-matrix-def
            by (rule eq-vecI, insert v A dim, auto simp: add-scalar-prod-distrib[of - n])
        finally have (A *vv =e 汭v)=(char-matrix A e *vv= 价 n).
    }
    thus ?thesis unfolding eigenvector-def dim by blast
qed
lemma eigenvalue-char-matrix: assumes A: (A :: 'a :: field mat) }\in\mathrm{ carrier-mat n
n
    shows eigenvalue A e=(\existsv.v\incarrier-vec n}\wedgev\not=\mp@subsup{O}{v}{}n\wedge\mathrm{ char-matrix A e
*v}v=\mp@subsup{0}{v}{}n
    unfolding eigenvalue-def eigenvector-char-matrix[OF A] ..
definition find-eigenvector :: ' }a::\mathrm{ field mat }=>\mp@subsup{|}{}{\prime}a>\mp@subsup{|}{}{\prime}a\mathrm{ vec where
    find-eigenvector A e=
        find-base-vector (fst (gauss-jordan (char-matrix A e) (0m (dim-row A) 0)))
lemma find-eigenvector: assumes A: A\incarrier-mat n n
    and ev: eigenvalue A e
    shows eigenvector A (find-eigenvector A e) e
proof -
    define B where B=char-matrix A e
    from ev[unfolded eigenvalue-char-matrix[OF A]] obtain v where
        v: v\incarrier-vec n v}=\mp@subsup{O}{v}{}n\mathrm{ and Bv: B *vv= Ov n unfolding B-def by
auto
    have B: B\incarrier-mat n n using A unfolding B-def by simp
    let ?z = 0m (dim-row A)0
    obtain C D where gauss: gauss-jordan B?z=(C,D) by force
    define w}\mathrm{ where w= find-base-vector C
    have res: find-eigenvector }Ae=w\mathrm{ unfolding w-def find-eigenvector-def Let-def
gauss B-def[symmetric]
        by simp
    have ?z \in carrier-mat n 0 using A by auto
    note gauss-0 = gauss-jordan[OF B this gauss]
    hence C:C\incarrier-mat n n by auto
    from gauss-0(1)[OF v(1)] Bv have Cv: C *vv= 0v n by auto
    {
        assume C:C=1 }=1
        have False using id Cv v unfolding C by auto
    }
    hence C1:C\not= 1m n by auto
    from find-base-vector-not-1[OF gauss-jordan-row-echelon[OF B gauss] C C1]
    have w:w\incarrier-vec n w\not= 0v n and id: C *v w = 0 v n unfolding w-def
by auto
```

from gauss- $0(1)[$ OF $w(1)]$ id have $B w: B *_{v} w=0_{v} n$ by simp from $w B w$ have eigenvector $A w e$
unfolding eigenvector-char-matrix $[O F A] B$-def by auto
thus ?thesis unfolding res .
qed
lemma eigenvalue-imp-nonzero-dim: assumes $A \in$ carrier-mat $n n$ and eigenvalue $A$ ev
shows $n>0$
proof (cases $n$ )
case 0
from assms obtain $v$ where eigenvector $A v$ ev unfolding eigenvalue-def by auto
from this[unfolded eigenvector-def] assms 0
have $v \in$ carrier-vec $0 v \neq 0_{v} 0$ by auto
hence False by auto
thus ?thesis by auto
qed $\operatorname{simp}$
lemma eigenvalue-det: assumes $A:\left(A::{ }^{\prime} a::\right.$ field mat $) \in$ carrier-mat $n$ shows eigenvalue $A \quad e=(\operatorname{det}($ char-matrix $A e)=0)$
proof -
from $A$ have $c A$ : char-matrix $A e \in$ carrier-mat $n n$ by auto
show ?thesis
unfolding eigenvalue-char-matrix $[O F A]$
unfolding id det-0-negate[OF cA] det-0-iff-vec-prod-zero $[O F ~ c A]$
eigenvalue-def by auto
qed
definition char-poly-matrix :: ' $a$ :: comm-ring-1 mat $\Rightarrow{ }^{\prime}$ ' poly mat where char-poly-matrix $A=\left(\left([: 0,1:] \cdot m 1_{m}(\right.\right.$ dim-row $\left.A)\right)+\operatorname{map-mat}(\lambda a .[:-a:])$ A)
lemma char-poly-matrix-closed[simp]: $A \in$ carrier-mat $n n \Longrightarrow$ char-poly-matrix A $\in$ carrier-mat $n n$
unfolding char-poly-matrix-def by auto
definition char-poly :: ' $a$ :: comm-ring-1 mat $\Rightarrow$ 'a poly where
char-poly $A=(\operatorname{det}($ char-poly-matrix $A))$
lemmas char-poly-defs $=$ char-poly-def char-poly-matrix-def
lemma (in comm-ring-hom) char-poly-matrix-hom: assumes $A: A \in$ carrier-mat $n n$
shows char-poly-matrix $\left(\right.$ mat $\left._{h} A\right)=$ map-mat (map-poly hom) (char-poly-matrix A)
unfolding char-poly-defs
by (rule eq-matI, insert $A$, auto simp: smult-mat-def hom-distribs)

```
lemma (in comm-ring-hom) char-poly-hom: assumes A: A \in carrier-mat n n
    shows char-poly (map-mat hom A) = map-poly hom (char-poly A)
proof -
    interpret map-poly-hom: map-poly-comm-ring-hom hom..
    show ?thesis
    unfolding char-poly-def map-poly-hom.hom-det[symmetric] char-poly-matrix-hom[OF
A] ..
qed
context inj-comm-ring-hom
begin
lemma eigenvector-hom: assumes A:A\incarrier-mat n n
    and ev: eigenvector A v ev
    shows eigenvector (math A) (vech v) (hom ev)
proof -
    let ?A = math }
    let ?v = vech v
    let ?ev = hom ev
    from ev[unfolded eigenvector-def] A
    have v:v\incarrier-vec n v\not= Ov n A *vv=ev \cdotv v by auto
    from v(1) have v1:?v }\in\mathrm{ carrier-vec n by simp
    from v(1-2) obtain }i\mathrm{ where }i<n\mathrm{ and }v$i\not=0\mathrm{ by force
    with v(1) have ?v $ i\not=0 by auto
    hence v2: ?v }\not=\mp@subsup{0}{v}{}n\mathrm{ using <i< n> v(1) by force
    from arg-cong[OFv(3), of vec}\mp@subsup{h}{h}{},\mathrm{ unfolded mult-mat-vec-hom[OF A v(1)] vec-hom-smult]
    have v3: ?A *v ?v = ?ev \cdotv ?v.
    from v1 v2 v3
    show ?thesis unfolding eigenvector-def using A by auto
qed
lemma eigenvalue-hom: assumes A: A carrier-mat n n
    and ev: eigenvalue A ev
    shows eigenvalue (math A) (hom ev)
    using eigenvector-hom[OF A, of - ev] ev
    unfolding eigenvalue-def by auto
lemma eigenvector-hom-rev: assumes A:A\incarrier-mat n n
    and ev: eigenvector (math A) (vech v) (hom ev)
    shows eigenvector A v ev
proof -
    let ?A = math}
    let ?v = vech}
    let ?ev = hom ev
    from ev[unfolded eigenvector-def] A
    have v:v\incarrier-vec n ?v}\not=\mp@subsup{0}{v}{}n?A*\mp@subsup{*}{v}{}\mathrm{ ?v = ?ev }\cdotv ?v by aut
    from v(1-2) obtain i where }i<n\mathrm{ and v$i}=0\mathrm{ by force
    with v(1) have v$i\not=0 by auto
    hence v2:v\not= 0v n using <i<n`v(1) by force
```

```
    from vec-hom-inj[OF v(3)[folded mult-mat-vec-hom[OF A v(1)] vec-hom-smult]]
    have v3: A *vv=evv vv v.
    from v(1) v2 v3
    show ?thesis unfolding eigenvector-def using A by auto
qed
end
lemma poly-det-cong: assumes A:A carrier-mat n n
    and B:B\incarrier-mat n n
    and poly: \bigwedge i j. i<n\Longrightarrowj<n\Longrightarrow poly (B $$ (i,j)) k=A $$ (i,j)
    shows poly (det B) k=\operatorname{det}A
proof -
    show ?thesis
    unfolding det-def'[OF A] det-def '[OF B] poly-sum poly-mult poly-prod
    proof (rule sum.cong[OF refl])
        fix }
        assume x: x \in{p.p permutes {0..<n}}
        let ?l = \prodka=0..<n.poly (B$$ (ka,x ka)) k
        let ?r = \i=0..<n. A $$ (i,xi)
        have id: ?l = ?r
            by (rule prod.cong[OF refl poly], insert x, auto)
    show poly (signof x) k*?l = signof x * ?r
        by (cases x rule: sign-cases) (simp-all add: id)
    qed
qed
lemma char-poly-matrix: assumes \(A:\left(A::{ }^{\prime} a\right.\) :: field mat \() \in\) carrier-mat \(n n\) shows poly (char-poly \(A) k=\operatorname{det}(-(\) char-matrix \(A k))\) unfolding char-poly-def by (rule poly-det-cong[of-n], insert A, auto simp: char-poly-matrix-def char-matrix-def)
lemma eigenvalue-root-char-poly: assumes \(A:(A\) :: 'a :: field mat \() \in\) carrier-mat n \(n\)
shows eigenvalue \(A k \longleftrightarrow\) poly \((\) char-poly \(A) k=0\)
unfolding eigenvalue-det[OF A] char-poly-matrix[OF A]
by (subst det-0-negate[of-n], insert \(A\), auto)
context
fixes \(A\) :: ' \(a\) :: comm-ring-1 mat and \(n\) :: nat
assumes \(A: A \in\) carrier-mat \(n n\)
and ut: upper-triangular \(A\)
begin
lemma char-poly-matrix-upper-triangular: upper-triangular (char-poly-matrix A)
using \(A\) ut unfolding upper-triangular-def char-poly-matrix-def by auto
lemma char-poly-upper-triangular:
char-poly \(A=\left(\prod a \leftarrow \operatorname{diag}-m a t A .[:-a, 1:]\right)\)
proof -
```

from $A$ have $c A$ : char-poly-matrix $A \in$ carrier-mat $n n$ by simp show ?thesis
unfolding char-poly-def det-upper-triangular [OF char-poly-matrix-upper-triangular $c A]$
by (rule arg-cong[where $f=$ prod-list $]$, unfold list-eq-iff-nth-eq, insert $c A A$, auto simp: diag-mat-def
char-poly-matrix-def)
qed
end
lemma map-poly-mult: assumes $A: A \in$ carrier-mat $n r n$
and $B: B \in$ carrier-mat $n n c$
shows
map-mat $(\lambda a .[: a:])(A * B)=$ map-mat $(\lambda a .[: a:]) A * \operatorname{map-mat}(\lambda a .[:$ $a:]) B$ (is ? id)
map-mat $(\lambda a .[: a:] * p)(A * B)=$ map-mat $(\lambda a .[: a:] * p) A *$ map-mat ( $\lambda a$. [: $a:]) B$ (is ?left)
map-mat $(\lambda a .[: a:] * p)(A * B)=\operatorname{map-mat}(\lambda a .[: a:]) A * \operatorname{map-mat}(\lambda a$. [: $a:]$ *p) $B$ (is ?right)
proof -
from $A B$ have dim: dim-row $A=n r \operatorname{dim}-c o l ~ A=n d i m$-row $B=n \operatorname{dim}$-col $B$ $=n c$ by auto
\{
fix $i j$
have $i<n r \Longrightarrow j<n c \Longrightarrow$
row (map-mat ( $\lambda$. [:a:]) A) $i \cdot \operatorname{col}($ map-mat $(\lambda a .[: a:]) B) j=[:($ row $A i \cdot$ col B j):]
unfolding scalar-prod-def
by (auto simp: dim ac-simps, induct $n$, auto)
$\}$ note $i d=t h i s$
\{
fix $i j$
have $i<n r \Longrightarrow j<n c \Longrightarrow$
$[:(\operatorname{row} A i \cdot \operatorname{col} B j):] * p=\operatorname{row}($ map-mat $(\lambda a .[: a:] * p) A) i \cdot \operatorname{col}($ map-mat ( $\lambda a .[: a:]) B) j$
unfolding scalar-prod-def
by (auto simp: dim ac-simps smult-sum)
$\}$ note left $=$ this
\{
fix $i j$
have $i<n r \Longrightarrow j<n c \Longrightarrow$
$[:($ row $A i \cdot \operatorname{col} B j):] * p=\operatorname{row}($ map-mat $(\lambda a .[: a:]) A) i \cdot \operatorname{col}($ map-mat $(\lambda a .[: a:] * p) B) j$
unfolding scalar-prod-def
by (auto simp: dim ac-simps smult-sum)
$\}$ note right $=$ this
show ?id
by (rule eq-matI, insert id, auto simp: dim)
show ?left

```
    by (rule eq-matI, insert left, auto simp: dim)
    show ?right
    by (rule eq-matI, insert right, auto simp: dim)
qed
lemma char-poly-similar: assumes similar-mat A ( }B\mathrm{ :: 'a :: comm-ring-1 mat)
    shows char-poly }A=\mathrm{ char-poly }
proof -
    from similar-matD[OF assms] obtain n P Q where
    carr: {A,B,P,Q}\subseteqcarrier-mat n n (is - \subseteq?C)
    and PQ:P*Q=1m}
    and AB:A=P*B*Q by auto
    hence A:A\in?C and B:B\in?C and P:P\in?C and Q:Q\in?C by auto
    let ?m=\lambdaa.[:-a:]
    let ?P = map-mat (\lambda a. [:a :]) P
    let ?Q = map-mat (\lambdaa. [:a a:]) Q
    let ?B = map-mat ?m B
    let ?I = map-mat (\lambdaa.[:a:]) (1m n)
    let ?XI= [:0, 1:] m m 1m n
    from A B have dim: dim-row }A=n\mathrm{ dim-row }B=n\mathrm{ by auto
    have cong: \xyz. x=y\Longrightarrowx*z=y*z by auto
    have id: ?m = (\lambda a ::' 'a. [: a :] * [:-1 :]) by (intro ext, auto)
    have char-poly A = det (?XI + map-mat (\lambdaa. [:-a:]) (P*B*Q)) unfolding
    char-poly-defs dim
    by (simp add: AB)
    also have ?XI =?P * ?XI * ?Q (is - = ?left)
    proof -
    have ?P * ?XI = [:0, 1:] m (?P * 1m n)
        by (rule mult-smult-distrib[of-nn-n], insert P, auto)
    also have ?P * 1m n=?P using P by simp
    also have ([:0,1:] 旃?P)*?Q = [:0, 1:] m}(?P* * ?Q
        by (rule mult-smult-assoc-mat, insert P Q, auto)
    also have ?P * ?Q = ?I unfolding PQ [symmetric }
        by (rule map-poly-mult[symmetric, OF P Q ])
    also have [:0,1:] m ?I = ?XI
        by rule auto
    finally show ?thesis ..
qed
also have map-mat ?m (P*B*Q)=?P*?B*?Q (is - = ?right)
    unfolding id
    by (subst map-poly-mult[OF mult-carrier-mat [OF P B] Q],
        subst map-poly-mult(3)[OF P B], simp)
also have ?left + ?right =(?P* ?XI + ?P * ?B) * ?Q
    by (rule add-mult-distrib-mat[symmetric, of - n n], insert B P Q, auto)
also have ?P * ?XI +?P *?B =?P * (?XI +?B)
    by (rule mult-add-distrib-mat[symmetric, of - n n], insert B P Q, auto)
also have det (?P*(?XI + ?B)*?Q) = det ?P * det (?XI + ?B)*\operatorname{det}?Q
    by (rule trans[OF det-mult [of - n] cong[OF det-mult]], insert P Q B, auto)
also have ... = (det ?P * det ?Q)*\operatorname{det (?XI + ?B) by (simp add: ac-simps)}
```

also have $\operatorname{det}(? X I+? B)=$ char-poly $B$ unfolding char-poly-defs dim by simp also have $\operatorname{det} ? P * \operatorname{det} ? Q=\operatorname{det}(? P * ? Q)$
by (rule det-mult $[$ symmetric], insert $P Q$, auto)
also have ? $P * ? Q=$ ? $I$ unfolding $P Q[$ symmetric $]$
by (rule map-poly-mult $[$ symmetric, OF $P \quad Q]$ )
also have det $\ldots=$ prod-list (diag-mat ?I)
by (rule det-upper-triangular $[o f-n]$, auto)
also have $\ldots=1$ unfolding prod-list-diag-prod
by (rule prod.neutral) simp
finally show? ?thesis by simp
qed
lemma degree-signof-mult $[$ simp $]$ : degree $(\operatorname{signof} p * q)=$ degree $q$
by (cases $p$ rule: sign-cases) simp-all
lemma degree-monic-char-poly: assumes $A: A \in$ carrier-mat $n n$
shows degree (char-poly $A$ ) $=n \wedge$ coeff (char-poly $A$ ) $n=1$
proof -
from $A$ have $A^{\prime}:[: 0,1:] \cdot m 1_{m}($ dim-row $A)+$ map-mat $(\lambda a .[:-a:]) A \in$ carrier-mat $n n$ by auto
from $A$ have $d A$ : dim-row $A=n$ by simp
show ?thesis
unfolding char-poly-defs det-def ${ }^{\prime}[O F A]$
proof (rule degree-lcoeff-sum [of - id], auto simp: finite-permutations permutes-id dA)
have both: degree $\left(\prod i=0 . .<n .\left([: 0,1:] \cdot m 1_{m} n+\right.\right.$ map-mat $\left.(\lambda a .[:-a:]) A\right)$ $\$ \$(i, i))=n \wedge$
coeff $\left(\prod i=0 . .<n .\left([: 0,1:] \cdot m 1_{m} n+\operatorname{map-mat}(\lambda a .[:-a:]) A\right) \$(i, i)\right)$ $n=1$
by (rule degree-prod-monic, insert $A$, auto)
from both show degree ( $\Pi i=0 . .<n$. ([:0, 1:] ${ }_{m} 1_{m} n+$ map-mat ( $\lambda a$. [:a:]) A) $\$ \$(i, i))=n .$.
from both show coeff $\left(\prod i=0 . .<n .\left([: 0,1:]{ }_{m} 1_{m} n+\right.\right.$ map-mat ( $\lambda a .[:-$ a:]) A) $\$ \$(i, i)) n=1 .$.
next
fix $p$
assume $p$ : p permutes $\{0 . .<n\}$
and $p \neq i d$
then obtain $i$ where $i: i<n$ and $p i: p i \neq i$
by (metis atLeastLessThan-iff order-refl permutes-natset-le)
show degree $\left(\prod i=0 . .<n\right.$. ([:0, 1:] $\mathrm{m}_{\mathrm{m}} 1_{m} n+\operatorname{map-mat}(\lambda a .[:-a:])$ A) $\$ \$$
$(i, p i))<n$
by (rule degree-prod-sum-lt-n $[O F-i]$, insert p i pi $A$, auto)
qed
qed
lemma char-poly-factorized: fixes $A$ :: complex mat
assumes $A: A \in$ carrier-mat $n n$
shows $\exists$ as. char-poly $A=\left(\prod a \leftarrow\right.$ as. $\left.[:-a, 1:]\right) \wedge$ length as $=n$

```
proof -
    let ?p = char-poly A
    from fundamental-theorem-algebra-factorized[of ?p] obtain as
    where Polynomial.smult (coeff ?p (degree ?p)) (\proda\leftarrowas. [:- a, 1:]) = ?p by
blast
    also have coeff ?p (degree ?p) = 1 using degree-monic-char-poly[OF A] by simp
    finally have cA:?p=(\proda\leftarrowas. [:-a,1:]) by simp
    from degree-monic-char-poly[OF A] have degree ?p = n ..
    with degree-linear-factors[of uminus as, folded cA] have length as = n by auto
    with }cA\mathrm{ show ?thesis by blast
qed
lemma char-poly-four-block-zeros-col: assumes A1: A1 f carrier-mat 1 1
    and A2:A2 \in carrier-mat 1 n and A3:A3 \in carrier-mat n n
    shows char-poly (four-block-mat A1 A2 ( ( Om n 1) A3) = char-poly A1 * char-poly
A3
    (is char-poly ?A = ?cp1 * ?cp3)
proof -
    let ?cm = \lambda A. [:0, 1:] m m 1m (dim-row A) +
        map-mat (\lambdaa. [:-a:]) A
    let ?B2 = map-mat (\lambdaa. [:- a:]) A2
    have char-poly ?A = det (?cm ?A)
        unfolding char-poly-defs using A1 A3 by simp
    also have ?cm ?A = four-block-mat (?cm A1) ?B2 ( (0m n 1) (?cm A3)
    by (rule eq-matI, insert A1 A2 A3, auto simp: one-poly-def)
    also have det ... = det (?cm A1)*\operatorname{det (?cm A3)}
        by (rule det-four-block-mat-lower-left-zero-col[OF - refl], insert A1 A2 A3,
auto)
    also have ... = ?cp1 * ?cp3 unfolding char-poly-defs ..
    finally show ?thesis.
qed
lemma char-poly-transpose-mat[simp]: assumes \(A: A \in\) carrier-mat \(n n\)
shows char-poly (transpose-mat \(A\) ) \(=\) char-poly \(A\)
proof -
    let ?A = [:0, 1:] 白 1m}(\mathrm{ dim-row A) + map-mat ( }\lambdaa.[:-a:])
    have A': ?A \in carrier-mat n n using A by auto
    show ?thesis unfolding char-poly-defs
        by (subst det-transpose[symmetric, OF A ], rule arg-cong[of--det],
        insert A, auto)
qed
lemma pderiv-char-poly: fixes }A\mathrm{ :: ' }a\mathrm{ :: idom mat
    assumes A:A\in carrier-mat n n
    shows pderiv (char-poly A) = (\sumi<n.char-poly (mat-delete A i i))
proof -
    let ?det = Determinant.det
    let ?m = map-mat (\lambdaa. [:- a:])
    let ?lam = [:0, 1:] m 1m n :: 'a poly mat
```

from $A$ have $i d$ : dim-row $A=n$ by auto
define $m A$ where $m A=? m A$
define lam where lam =?lam
let ? sum $=l a m+m A$
define Sum where Sum $=$ ?sum
have $m A: m A \in$ carrier-mat $n n$ and
lam: lam $\in$ carrier-mat $n n$ and
Sum: Sum $\in$ carrier-mat $n n$
using $A$ unfolding $m A$-def Sum-def lam-def by auto
let $? P=\{p$. p permutes $\{0 . .<n\}\}$
let $? e=\lambda p .\left(\prod i=0 . .<n\right.$. Sum $\left.\$ \$(i, p i)\right)$
let ?f $=\lambda p$ a. signof $p *\left(\prod i \in\{0 . .<n\}-\{a\}\right.$. Sum $\left.\$ \$(i, p i)\right) *$ pderiv (Sum \$\$( $a, p a)$ )
let $? g=\lambda p a$. signof $p *\left(\prod i \in\{0 . .<n\}-\{a\} . \operatorname{Sum} \$ \$(i, p i)\right)$
define $f$ where $f=$ ? $f$
define $g$ where $g=? g$
\{
fix $p$
assume $p$ : p permutes $\{0 \quad . .<n\}$
have pderiv (signof $p::$ 'a poly) $=0$
by (cases $p$ rule: sign-cases) (simp-all add: pderiv-minus)
hence pderiv (signof $p *$ ? e $p)=\operatorname{signof} p * \operatorname{pderiv}\left(\prod i=0 . .<n\right.$. Sum $\$ \$(i$, pi))
unfolding pderiv-mult by auto
also have signof $p *$ pderiv $\left(\prod i=0 . .<n\right.$. Sum $\left.\$ \$(i, p i)\right)=$ ( $\left.\sum a=0 . .<n . f p a\right)$
unfolding pderiv-prod sum-distrib-left f-def by (simp add: ac-simps)
also note calculation
$\}$ note to- $f=$ this
\{
fix $a$
assume $a$ : $a<n$
have Psplit: ? $P=\{p . p$ permutes $\{0 . .<n\} \wedge p a=a\} \cup(? P-\{p . p a=a\})$
(is - = ? Pa $\cup$ ? Pz) by auto
\{
fix $p$
assume $p$ : p permutes $\{0 . .<n\} p a \neq a$
hence pderiv (Sum $\$ \$(a, p a))=0$ unfolding Sum-def lam-def mA-def
using a $p A$ by auto
hence $f p a=0$ unfolding $f$-def by auto
$\}$ note $0=$ this
\{
fix $p$
assume $p$ : p permutes $\{0 . .<n\} p a=a$
hence pderiv (Sum $\$ \$(a, p a))=1$ unfolding Sum-def lam-def mA-def using a $p A$
by (auto simp: pderiv-pCons)
hence $f p a=g p a$ unfolding $f$-def $g$-def by auto

```
    \} note \(f g=\) this
    let ? \(n=n-1\)
    from \(a\) have \(n\) : Suc \(? n=n\) by simp
    let ? \(B=[: 0,1:] \cdot m 1_{m}\) ? \(n+\) ? \(m\) (mat-delete \(\left.A a a\right)\)
    have \(B: ? B \in\) carrier-mat ?n ?n using \(A\) by auto
    have \(\operatorname{sum}(\lambda p . f p a) ? P=\operatorname{sum}(\lambda p . f p a) ? P a+\operatorname{sum}(\lambda p . f p a) ? P z\)
    by (subst sum.union-disjoint[symmetric], auto simp: finite-permutations Psplit[symmetric])
    also have \(\ldots=\operatorname{sum}(\lambda p . f p a) ? P a\)
    by (subst (2) sum.neutral, insert 0, auto)
    also have \(\ldots=\operatorname{sum}(\lambda p . g p a) ? P a\)
    by (rule sum.cong, auto simp: fg)
    also have \(\ldots=\) ? det ? \(B\)
    unfolding det-def \({ }^{\prime}[O F B]\)
    unfolding permutation-fix \(\left[\begin{array}{lll}\text { of } a & ? n & a \text {, unfolded } n \text {, OF } a\end{array}\right.\) a]
    unfolding sum.reindex[OF permutation-insert-inj-on[of a ?n a, unfolded n,
OF a a al] o-def
    proof (rule sum.cong[OF refl \(]\) )
    fix \(p\)
    let \(? Q=\{p . p\) permutes \(\{0 . .<? n\}\}\)
    assume \(p \in\) ? \(Q\)
    hence \(p\) : \(p\) permutes \(\{0 . .<\) ? \(n\}\) by auto
    let \(? p=\) permutation-insert a a \(p\)
    let ? \(i=\) insert-index a
    have sign: signof ? \(p=\) signof \(p\)
        unfolding signof-permutation-insert[OF p, unfolded \(n\), OF a a] by simp
    show \(g\) (permutation-insert a a \(p\) ) \(a=\) signof \(p *\left(\prod i=0 . .<? n\right.\) ? \(B \$ \$(i\),
pi))
        unfolding \(g\)-def sign
    proof (rule arg-cong[of - (*) (signof p)])
        have \(\left(\prod i \in\{0 . .<n\}-\{a\}\right.\). Sum \(\$ \$(i\), ?p \(\left.i)\right)=\)
            \(\operatorname{prod}((\$ \$)\) Sum \()((\lambda x .(x\), ?p \(x))\) ' \((\{0 . .<n\}-\{a\}))\)
        unfolding prod.reindex[OF inj-on-convol-ident, of - ?p] o-def ..
        also have \(\ldots=\left(\prod\right.\) ii \(\in\left\{\left(i^{\prime}, ? p i^{\prime}\right) \mid i^{\prime} . i^{\prime} \in\{0 . .<n\}-\{a\}\right\}\). Sum \(\$ \$\) ii)
        by (rule prod.cong, auto)
        also have \(\ldots=\operatorname{prod}((\$ \$) \operatorname{Sum})((\lambda i .(? i \quad i, ? i(p i))) \cdot\{0 . .<? n\})\)
            unfolding Determinant.foo[of a ? \(n\) a, unfolded \(n\), OF a a \(p\) ]
            by (rule arg-cong[of - prod -], auto)
        also have \(\ldots=\operatorname{prod}(\lambda i\). Sum \(\$ \$(? i i, ? i(p i)))\{0 . .<? n\}\)
        proof (subst prod.reindex, unfold o-def)
        show inj-on ( \(\lambda i\). (?i \(i\), ? \(\left.i\binom{p}{i}\right)\) ) \(\{0 . .<? n\}\) using insert-index-inj-on[of a]
            by (auto simp: inj-on-def)
        qed simp
        also have \(\ldots=\left(\prod i=0 . .<? n\right.\). ? \(\left.B \$ \$(i, p i)\right)\)
        proof (rule prod.cong[OF refl], rename-tac \(i\) )
            fix \(j\)
            assume \(j \in\{0 . .<? n\}\)
            hence \(j: j<\) ? \(n\) by auto
            with \(p\) have \(p j: p j<? n\) by auto
            from \(j p j\) have \(j j\) : ?i \(j<n\) ? \(i(p j)<n\) by (auto simp: insert-index-def)
```

```
            let ?jj = (?i j, ?i ( }pj\mathrm{ ) )
            note index-adj = mat-delete-index[of - ?n, unfolded n, OF - a a j pj]
            have Sum $$ ? jj = lam $$ ?jj + mA $$ ?jj unfolding Sum-def using jj
A lam mA by auto
            also have \ldots=? . $$$(j, p j)
            unfolding index-adj[OF mA] index-adj[OF lam] using j pj A
            by (simp add: mA-def lam-def mat-delete-def)
            finally show Sum $$ ? jj = ?B $$ (j, pj).
                    qed
                    finally
            show }(\prodi\in{0..<n}-{a}.Sum $$(i,?p i))=(\prodi=0..<?n.?B $$ (i
pi)).
            qed
    qed
    also have ... = char-poly (mat-delete A a a) unfolding char-poly-def char-poly-matrix-def
        using }A\mathrm{ by simp
    also note calculation
    } note to-char-poly = this
    have pderiv (char-poly A) = pderiv (?det Sum)
        unfolding char-poly-def char-poly-matrix-def id lam-def mA-def Sum-def by
auto
    also have ... = sum ( }\lambda\mathrm{ p. pderiv (signof p * ?e p)) ?P unfolding det-def'[OF
Sum]
    pderiv-sum by (rule sum.cong, auto)
    also have ... = sum ( }\lambda\mathrm{ p. ( \a=0..<n.f pa)) ?P
    by (rule sum.cong[OF refl], subst to-f, auto)
    also have ... = (\suma=0..<n.sum (\lambda p.fpa) ?P)
        by (rule sum.swap)
    also have \ldots=( \suma<n.char-poly (mat-delete A a a))
    by (rule sum.cong, auto simp: to-char-poly)
    finally show ?thesis .
qed
lemma char-poly-0-column: fixes A :: 'a :: idom mat
    assumes 0: \bigwedgej.j<n\LongrightarrowA$$(j,i)=0
    and A:A\incarrier-mat n n
    and i:i<n
shows char-poly A = monom 1 1 * char-poly(mat-delete A i i)
proof -
    let ? n = n-1
    let ?A = mat-delete A i i
    let ?sum = [:0, 1:] 白 1 m n + map-mat (\lambdaa. [:-a:]) A
    define Sum where Sum = ?sum
    let ?f = \lambda j. Sum $$ (j,i)* cofactor Sum j i
    have Sum:Sum \in carrier-mat n n using A unfolding Sum-def by auto
    from }A\mathrm{ have id: dim-row }A=n\mathrm{ by auto
    have char-poly A=(\sumj<n. ?f j)
    unfolding char-poly-def[of A] char-poly-matrix-def
    using laplace-expansion-column[OF Sum i] unfolding Sum-def using A by
```

```
simp
    also have ... = ?f i + sum ?f ({..<n}-{i})
        by (rule sum.remove[of-i], insert i, auto)
    also have ... = ?f i
    proof (subst sum.neutral, intro ballI)
        fix }
        assume j\in{..<n}-{i}
    hence j:j<n and ji:j\not=i by auto
    show ?f j = 0 unfolding Sum-def using ji j i A O[OF j] by simp
    qed simp
    also have ?f i}=[:0,1:]*(cofactor Sum i i)
        unfolding Sum-def using i A O[OF i] by simp
    also have cofactor Sum i i= det (mat-delete Sum i i)
        unfolding cofactor-def by simp
    also have ... = char-poly ?A
        unfolding char-poly-def char-poly-matrix-def Sum-def
    proof (rule arg-cong[of - det])
        show mat-delete ?sum i i = [:0, 1:] m 1m (dim-row?A) + map-mat (\lambdaa. [:-
a:]) ?A
            using i A by (auto simp: mat-delete-def)
    qed
    also have [:0, 1:] = (monom 1 1 :: 'a poly) by (rule x-as-monom)
    finally show ?thesis .
qed
definition mat-erase :: 'a :: zero mat }=>\mathrm{ nat }=>\mathrm{ nat }=>\mathrm{ ' 'a mat where
    mat-erase A i j = Matrix.mat (dim-row A) (dim-col A)
    (\lambda(i',}\mp@subsup{j}{}{\prime})\mathrm{ . if }\mp@subsup{i}{}{\prime}=i\vee\mp@subsup{j}{}{\prime}=j\mathrm{ then 0 else A $$ ( }\mp@subsup{i}{}{\prime},\mp@subsup{j}{}{\prime})
lemma mat-erase-carrier[simp]: (mat-erase A ij) \in carrier-mat nr nc \longleftrightarrow}\longleftrightarrowA
carrier-mat nr nc
    unfolding mat-erase-def carrier-mat-def by simp
lemma pderiv-char-poly-mat-erase: fixes }A :: 'a :: idom mat
    assumes A: A \in carrier-mat n n
    shows monom 11* pderiv (char-poly A)=(\sumi<n.char-poly (mat-erase A i
i))
proof -
    show ?thesis unfolding pderiv-char-poly[OF A] sum-distrib-left
    proof (rule sum.cong[OF refl])
        fix }
        assume i\in{..<n}
        hence i: i<n by simp
    have mA: mat-erase A i i G carrier-mat n n using A by simp
    show monom 11* char-poly (mat-delete A i i) = char-poly(mat-erase A i i)
    by (subst char-poly-0-column[OF-mA i], (insert i A, force simp: mat-erase-def),
            rule arg-cong[of - - \lambda x.f* char-poly x for f],
            auto simp: mat-delete-def mat-erase-def)
    qed
```


## qed

end

## 13 Jordan Normal Form

This theory defines Jordan normal forms (JNFs) in a sparse representation, i.e., as block-diagonal matrices. We also provide a closed formula for powers of JNFs, which allows to estimate the growth rates of JNFs.

```
theory Jordan-Normal-Form
imports
    Matrix
    Char-Poly
    Polynomial-Interpolation.Missing-Unsorted
```

begin
definition jordan-block :: nat $\Rightarrow$ ' $a::\{$ zero,one $\} \Rightarrow$ 'a mat where
jordan-block $n a=$ mat $n n(\lambda(i, j)$. if $i=j$ then a else if Suc $i=j$ then 1 else
0)
lemma jordan-block-index[simp]: $i<n \Longrightarrow j<n \Longrightarrow$
jordan-block na $\$ \$(i, j)=($ if $i=j$ then a else if Suc $i=j$ then 1 else 0$)$
dim-row $($ jordan-block $n k)=n$
dim-col $($ jordan-block $n k)=n$
unfolding jordan-block-def by auto
lemma jordan-block-carrier[simp]: jordan-block $n k \in$ carrier-mat $n n$
unfolding carrier-mat-def by auto
lemma jordan-block-char-poly: char-poly (jordan-block $n$ a) $=[:-a, 1:]$ n
unfolding char-poly-defs by (subst det-upper-triangular[of-n], auto simp: prod-list-diag-prod)
lemma jordan-block-pow-carrier [simp]:
jordan-block $n a{ }_{m} r \in$ carrier-mat $n n$ by auto
lemma jordan-block-pow-dim[simp]:
dim-row (jordan-block $n$ a $\left.\widehat{m}_{m} r\right)=n$ dim-col (jordan-block $\left.n a \widehat{m}_{m} r\right)=n$ by
auto
lemma jordan-block-pow: (jordan-block $n$ ( $a$ :: 'a :: comm-ring-1)) ${ }_{m} r=$
mat $n n(\lambda(i, j)$. if $i \leq j$ then of-nat ( $r$ choose $(j-i)) * a \wedge(r+i-j)$ else 0$)$
proof (induct $r$ )
case 0
\{
fix $i j$ :: nat
assume $i \neq j i \leq j$
hence $j-i>0$ by auto
hence 0 choose $(j-i)=0$ by simp
$\}$ note $[$ simp $]=$ this

```
show ?case
    by (simp, rule eq-matI, auto)
next
    case (Suc r)
    let ?jb = jordan-block n a
    let ?rij = \lambdarij. of-nat (r choose ( j - i)) *a^ (r + i-j)
    let ?v = \lambda i j. if i\leqj then of-nat (r choose ( j - i))*a^ (r+i-j) else 0
    have ?jb \widehat{m}}\mathrm{ Suc r= mat n n ( }\lambda(i,j)\mathrm{ . if i m j then ?rij r i j else 0) * ?jb by
(simp add: Suc)
    also have ... = mat n n ( }\lambda(i,j). if i\leq j then ?rij (Suc r) i j else 0)
    proof -
    {
        fix }
        assume j:j<n
        hence col: col (jordan-block n a) j=vec n (\lambdai. if i = j then a else if Suc i
= j then 1 else 0)
            unfolding jordan-block-def col-mat[OF j] by simp
        fix f
        have vec nf col (jordan-block na) j=(fj*a+(if j=0 then 0 else f (j
- 1)))
    proof -
        define p}\mathrm{ where p=( }\lambda\mathrm{ i.vec n f$i* col (jordan-block na) j$i)
        have vec nf col (jordan-block n a) j=(\sumi=0 ..< n. pi)
            unfolding scalar-prod-def p-def by simp
        also have ... = pj + sum p({0..<n}-{j}) using j
            by (subst sum.remove[of-j], auto)
        also have p j=fj*a unfolding p-def col using j by auto
        also have sum p ({0..< n}-{j})=(if j=0 then 0 else f (j - 1))
        proof (cases j)
            case 0
            have sum p ({0..< n} - {j})=0
                by (rule sum.neutral, auto simp: p-def col 0)
            thus ?thesis using 0 by simp
        next
            case (Suc jj)
            with j have jj: jj \in{0..<n}-{j} by auto
            have sum p ({0..<n}-{j})=pjj + sum p({0..<n}-{j}-{jj})
                by (subst sum.remove[OF-jj], auto)
            also have p jj =f(j-1) unfolding p-def col using jj
                    by (auto simp: Suc)
            also have sum p ({0 ..<n}-{j}-{jj})=0
                    by (rule sum.neutral, auto simp: p-def col, auto simp: Suc)
            finally show ?thesis unfolding Suc by simp
        qed
        finally show ?thesis .
    qed
    } note scalar-to-sum = this
{
    fix ij
```

```
    assume i: i<n and ij:i>j
    hence }j:j<n\mathrm{ by auto
    have vec n (?v i) • col (jordan-block n a) j=0
        unfolding scalar-to-sum[OF j] using ij ij by auto
    } note easy-case = this
    {
        fix ij
        assume j:j<n and ij:i\leqj
        hence i:i<n and id: \ pq. (if i\leqj then p else q) = p by auto
        have vec n (?v i) • col (jordan-block n a) j=
        (of-nat (r choose (j - i)) * (a^ (Suc (r + i - j)))) +
        (if j = 0 then 0
        else if i\leqj-1 then of-nat (r choose (j-1-i))*a^(r+i-(j-
1)) else 0)
    unfolding scalar-to-sum[OF j]
    using ij by simp
    also have ... = of-nat (Suc r choose (j - i))*a^ (Suc (r +i) - j)
    proof (cases j)
    case (Suc jj)
    {
        assume i\leqSuc jj and }\negi\leqj
        hence i=Suc jj by auto
        hence a*a^(r+i-Suc jj)=a^(r+i-jj) by simp
    }
    moreover
    {
        assume ijj:i\leqjj
        have of-nat (r choose (Suc jj - i)) * (a*a^(r + i - Suc jj))
        +of-nat (r choose (jj - i))*a^ (r +i-jj)=
        of-nat (Suc r choose (Suc jj - i)) *a^ (r+i-jj)
        proof (cases r + i<jj)
            case True
            hence gt: jj - i>r Suc jj - i> r Suc jj - i> Suc r by auto
            show ?thesis
                unfolding binomial-eq-0[OF gt(1)] binomial-eq- 0[OF gt(2)] bino-
mial-eq-0[OF gt(3)]
            by simp
    next
        case False
        hence ge: r +i\geqjj by simp
        show ?thesis
        proof (cases jj =r+i)
            case True
            have gt: r < Suc r by simp
            show ?thesis unfolding True by (simp add: binomial-eq-0[OF gt])
        next
            case False
            with ge have lt: jj<r+i by auto
            hence r + i-jj = Suc (r+i-Suc jj) by simp
```

```
                    hence prod: a*a^ (r+i-Suc jj) = a^ (r + i-jj) by simp
                    from ijj have id: Suc jj - i=Suc ( }jj-i)\mathrm{ by simp
                    have binom: Suc r choose (Suc jj - i)=
                        rchoose (Suc jj - i) + (r choose (jj - i))
                        unfolding id
                        by (subst binomial-Suc-Suc, simp)
                    show ?thesis unfolding prod binom
                        by (simp add: field-simps)
                qed
                qed
            }
            ultimately show ?thesis using ij unfolding Suc by auto
        qed auto
        finally have vec n (?v i) - col (jordan-block n a) j
    =of-nat (Suc r choose (j-i))*a^^(Suc (r+i)-j).
    } note main-case = this
    show ?thesis
    by (rule eq-matI, insert easy-case main-case, auto)
    qed
    finally show ?case by simp
qed
definition jordan-matrix :: (nat > 'a :: {zero,one})list => 'a mat where
    jordan-matrix n-as = diag-block-mat (map (\lambda (n,a).jordan-block n a) n-as)
lemma jordan-matrix-dim[simp]:
    dim-row (jordan-matrix n-as) = sum-list (map fst n-as)
    dim-col (jordan-matrix n-as) = sum-list (map fst n-as)
    unfolding jordan-matrix-def
    by (subst dim-diag-block-mat, auto, (induct n-as, auto simp: Let-def)+)
lemma jordan-matrix-carrier[simp]:
    jordan-matrix n-as \in carrier-mat (sum-list (map fst n-as)) (sum-list (map fst
n-as))
    unfolding carrier-mat-def by auto
lemma jordan-matrix-upper-triangular: i < sum-list (map fst n-as)
    \Longrightarrowj<i\Longrightarrow jordan-matrix n-as $$ (i,j)=0
    unfolding jordan-matrix-def
    by (rule diag-block-upper-triangular, auto simp: jordan-matrix-def[symmetric])
lemma jordan-matrix-pow: (jordan-matrix n-as) \widehat{m}r=
    diag-block-mat (map (\lambda (n,a). (jordan-block n a) ^m}r)n-as
    unfolding jordan-matrix-def
    by (subst diag-block-pow-mat, force, rule arg-cong[of - diag-block-mat], auto)
lemma jordan-matrix-char-poly:
    char-poly (jordan-matrix n-as)=(\prod(n,a)\leftarrown-as.[:- a, 1:] ^ n)
proof -
```

```
let ?n = sum-list (map fst n-as)
have diag-mat
    ([:0, 1:] 质 1m (sum-list (map fst n-as)) + map-mat (\lambdaa. [:-a:]) (jordan-matrix
n-as)) =
    concat (map (\lambda(n,a). replicate n [:- a, 1:]) n-as) unfolding jordan-matrix-def
proof (induct n-as)
    case (Cons na n-as)
    obtain n a where na: na=(n,a) by force
    let ?n2 = sum-list (map fst n-as)
    note fbo = four-block-one-mat
    note mz = zero-carrier-mat
    note mo = one-carrier-mat
        have mA: \bigwedge A. A \in carrier-mat (dim-row A) (dim-col A) unfolding car-
rier-mat-def by auto
    let ?Bs = map ( }\lambda(x,y).jordan-block x y) n-a
    let ?B = diag-block-mat ?Bs
    from jordan-matrix-dim[of n-as, unfolded jordan-matrix-def]
    have dimB: dim-row ?B=?n2 dim-col ?B=?n2 by auto
    hence B:?B \in carrier-mat ?n2 ?n2 unfolding carrier-mat-def by simp
    show ?case unfolding na fbo
    apply (simp add: Let-def fbo[symmetric] del: fbo)
    apply (subst smult-four-block-mat[OF mo mz mz mo])
    apply (subst map-four-block-mat[OF jordan-block-carrier mz mz mA])
    apply (subst add-four-block-mat[of - n n - ?n2 - ?n2], auto simp: }\operatorname{dim}B B
    apply (subst diag-four-block-mat[of - n-?n2], auto simp: dimB B)
    apply (subst Cons, auto simp: jordan-block-def diag-mat-def,
        intro nth-equalityI, auto)
    done
    qed (force simp: diag-mat-def)
    also have prod-list ... = (\prod( n,a)\leftarrown-as. [:- a, 1:]^ n)
    by (induct n-as, auto)
    finally
    show ?thesis unfolding char-poly-defs
    by (subst det-upper-triangular[of - ?n], auto simp: jordan-matrix-upper-triangular)
qed
definition jordan-nf :: 'a :: semiring-1 mat => (nat }\times 'a)list => bool where
    jordan-nf A n-as \equiv(0\not\infst'set n-as ^ similar-mat A (jordan-matrix n-as))
```

lemma jordan-nf-powE: assumes $A: A \in$ carrier-mat $n n$ and jnf: jordan-nf $A$ $n$-as
obtains $P Q$ where $P \in$ carrier-mat $n n Q \in$ carrier-mat $n n$ and
char-poly $A=\left(\prod(n a, a) \leftarrow n\right.$-as. $[:-a, 1:]$ ^ $\left.n a\right)$
$\wedge k . A{ }_{m} k=P *(\text { jordan-matrix } n \text {-as })^{\wedge}{ }_{m} k * Q$
proof -
from $A$ have $\operatorname{dim}$ : dim-row $A=n$ by auto
assume obt: $\bigwedge P Q . P \in$ carrier-mat $n n \Longrightarrow Q \in$ carrier-mat $n n \Longrightarrow$ char-poly $A=\left(\prod(n a, a) \leftarrow n\right.$-as. $\left.[:-a, 1:] \wedge n a\right) \Longrightarrow$
$\left(\bigwedge k . A{ }_{m} k=P *\right.$ jordan-matrix $n$-as $\left.{ }_{m} k * Q\right) \Longrightarrow$ thesis

```
    from jnf[unfolded jordan-nf-def] obtain P Q where
        simw: similar-mat-wit A (jordan-matrix n-as) PQ
        and sim: similar-mat A (jordan-matrix n-as) unfolding similar-mat-def by
blast
    show thesis
    proof (rule obt)
        show \k. A \widehat{m}k=P* jordan-matrix n-as \widehat{m}}
            by (rule similar-mat-wit-pow-id[OF simw])
        show char-poly A=(\prod(na,a)\leftarrown-as. [:- a, 1:] ^ na)
            unfolding char-poly-similar[OF sim] jordan-matrix-char-poly ..
    qed (insert simw[unfolded similar-mat-wit-def Let-def dim], auto)
qed
lemma choose-poly-bound: assumes i\leqd
    shows r choose i\leqmax 1 (r^d)
proof (cases i\leqr)
    case False
    hence r choose i=0 by simp
    thus ?thesis by arith
next
    case True
    show ?thesis
    proof (cases r)
        case (Suc rr)
        from binomial-le-pow[OF True] have r choose i\leq r^ i by simp
        also have ... \leqr^d using power-increasing[OF<i}\leqd\rangle\mathrm{ ,of r] Suc by auto
        finally show ?thesis by simp
    qed (insert True, simp)
qed
context
    fixes b :: ' }a\mathrm{ :: archimedean-field
    assumes b: 0<bb<1
begin
lemma poly-exp-constant-bound: \exists p.\forall x.c*b^ x * of-nat x^ deg \leq p
proof (cases c \leq 0)
    case True
    show ?thesis
        by (rule exI[of - 0], intro allI,
        rule mult-nonpos-nonneg[OF mult-nonpos-nonneg[OF True]], insert b, auto)
next
    case False
    hence c:c\geq0 by simp
    from poly-exp-bound[OF b, of deg] obtain p where \ x.b^^x* of-nat x^ deg
\leqp by auto
    from mult-left-mono[OF this c]
    show ?thesis by (intro exI[of - c*p], auto simp: ac-simps)
qed
```

```
lemma poly-exp-max-constant-bound: \(\exists p . \forall x . c * b \wedge x * \max 1\) (of-nat \(x \wedge\) deg)
\(\leq p\)
proof -
    from poly-exp-constant-bound [of \(c\) deg] obtain \(p\) where
        \(p: \bigwedge x . c * b^{\wedge} x *\) of-nat \(x{ }^{\wedge} \operatorname{deg} \leq p\) by auto
    show ?thesis
    proof (rule exI[of \(-\max p c]\), intro allI)
        fix \(x\)
        let ? \(\exp =\) of-nat \(x\) ^ \(\operatorname{deg}::{ }^{\prime} a\)
        show \(c * b \wedge x * \max 1\) ? exp \(\leq \max p c\)
        proof (cases \(x=0\) )
            case False
            hence ? exp \(\neq\) of-nat 0 by simp
        hence \(? \exp \geq 1\) by (metis less-one not-less of-nat-1 of-nat-less-iff of-nat-power)
            hence \(\max 1 ? \exp =\) ? \(\exp\) by \(\operatorname{simp}\)
            thus ?thesis using \(p[o f x]\) by simp
        qed (cases deg, auto)
    qed
qed
end
context
    fixes \(a\) :: ' \(a\) :: real-normed-field
begin
lemma jordan-block-bound:
    assumes \(i: i<n\) and \(j: j<n\)
    shows norm ((jordan-block \(\left.\left.n a{ }_{m} k\right) \$ \$(i, j)\right)\)
        \(\leq\) norm \(a^{\wedge}(k+i-j) * \max 1\) (of-nat \(\left.k \wedge(n-1)\right)\)
        (is ? lhs \(\leq\) ? \(r h s\) )
proof -
    have id: (jordan-block \(\left.n a \widehat{ }^{{ }_{m}} k\right) \$ \$(i, j)=(i f i \leq j\) then of-nat ( \(k\) choose \((j-\)
\(i)) * a^{\wedge}(k+i-j)\) else 0)
        unfolding jordan-block-pow using \(i j\) by auto
    from \(i j\) have diff: \(j-i \leq n-1\) by auto
    show ?thesis
    proof (cases \(i \leq j\) )
        case False
        thus ?thesis unfolding id by simp
    next
        case True
    hence ?lhs \(=\) norm (of-nat \((k\) choose \((j-i)) * a^{\wedge}(k+i-j)\) ) unfolding id
by \(\operatorname{simp}\)
        also have \(\ldots \leq\) norm (of-nat \((k\) choose \(\left.(j-i))::{ }^{\prime} a\right) * \operatorname{norm}\left(a^{\wedge}(k+i-\right.\)
j))
        by (rule norm-mult-ineq)
        also have \(\ldots \leq(\max 1(\) of-nat \(k \wedge(n-1))) *\) norm \(a^{\wedge}(k+i-j)\)
        proof (rule mult-mono[OF - norm-power-ineq - norm-ge-zero])
        have \(k\) choose \((j-i) \leq \max 1\left(k^{\wedge}(n-1)\right)\)
```

by (rule choose-poly-bound $[$ OF diff $]$ )
hence norm (of-nat ( $k$ choose $(j-i)):: ' a) \leq$ of-nat $(\max 1(k \wedge(n-1)))$ unfolding norm-of-nat of-nat-le-iff .
also have $\ldots=\max 1$ (of-nat $k \wedge(n-1)$ ) by (metis max-def of-nat-1 of-nat-le-iff of-nat-power)
finally show norm (of-nat ( $k$ choose $(j-i)$ ) :: 'a) $\leq \max 1$ (real-of-nat $k{ }^{\wedge}$ $(n-1))$.
qed $\operatorname{simp}$
also have $\ldots=$ ? rhs by simp
finally show ?thesis.
qed
qed
lemma jordan-block-poly-bound:
assumes $i: i<n$ and $j: j<n$ and $a$ : norm $a=1$
shows norm $\left(\left(\right.\right.$ jordan-block $\left.\left.n a{ }_{m} k\right) \$ \$(i, j)\right) \leq \max 1$ (of-nat $k^{\wedge}(n-1)$ )
(is ?lhs $\leq$ ? $r h s$ )
proof -
from jordan-block-bound $[$ OF $i j$, of $k$, unfolded a]
show ?thesis by simp
qed
theorem jordan-block-constant-bound: assumes a: norm a<1
shows $\exists p . \forall i j k . i<n \longrightarrow j<n \longrightarrow \operatorname{norm}\left(\left(\right.\right.$ jordan-block na $\left.\left.{ }^{\wedge}{ }_{m} k\right) \$ \$(i, j)\right)$
$\leq p$
proof (cases $a=0$ )
case True
show ?thesis
proof (rule exI[of - 1], intro allI impI) fix $i j k$ assume $*: i<n j<n$ \{
assume $i j: i \leq j$
have norm $\left(\left(\right.\right.$ of-nat $(k$ choose $\left.\left.(j-i))::{ }^{\prime} a\right) * 0^{\wedge}(k+i-j)\right) \leq 1$ (is norm
$? l h s \leq 1$ )
proof (cases $k+i>j$ )
case True
hence ? $\mathrm{lh} s=0$ by $\operatorname{simp}$
also have norm (...) $\leq 1$ by simp
finally show ?thesis .
next
case False
hence $i d$ : ? lhs $=($ of-nat $(k$ choose $(j-i)):: ' a)$ and $j: j-i \geq k$ by auto
from $j$ have $k$ choose $(j-i)=0 \vee k$ choose $(j-i)=1$ by (simp add:
nat-less-le)
thus norm? lhs $\leq 1$
proof
assume $k$ : $k$ choose $(j-i)=0$

```
                    show ?thesis unfolding id k by simp
            next
                    assume k: k choose ( j - i)=1
            show ?thesis unfolding id unfolding k by simp
            qed
        qed
    }
    thus norm ((jordan-block n a \widehat{m}k)$$(i,j)) \leq 1 unfolding True
        unfolding jordan-block-pow using * by auto
    qed
next
    case False
    hence na: norm a>0 by auto
    define c where c=inverse (norm a^n)
    define deg where deg =n-1
    have c: c>0 unfolding c-def using na by auto
    define b where b= norm a
    from }a\mathrm{ na have }0<bb<1\mathrm{ unfolding b-def by auto
    from poly-exp-max-constant-bound[OF this, of c deg]
    obtain p where ^ k.c* b^ k* max 1 (of-nat k^deg)\leqp by auto
    show ?thesis
    proof (intro exI[of - p], intro allI impI)
    fix ijk
    assume ij: i<nj<n
    from jordan-block-bound[OF this]
    have norm ((jordan-block n a ^}\mp@subsup{m}{m}{k})$$(i,j)
        \leqnorm a^ (k+i-j)* max 1 (real-of-nat k^ (n-1)).
    also have _..\leqc* norm a ^ k* max 1 (real-of-nat k^(n-1))
    proof (rule mult-right-mono)
        from ij have }i-j\leqn\mathrm{ by auto
        show norm a^ (k+i-j)\leqc* norm a^k
        proof (rule mult-left-le-imp-le)
            show 0 < norm a ^ n using na by auto
            let ?lhs = norm a^n n* norm a^(k+i-j)
            let ?rhs = norm a^n n*(c* norm a^ k)
            from ij have ge: n+(k+i-j)\geqk by arith
            have ?lhs = norm a^ ( n + (k+i-j)) by (simp add: power-add)
            also have ...\leqnorm a^ k using ge a na using less-imp-le power-decreasing
by blast
            also have ... = ?rhs unfolding c-def using na by simp
            finally show ?lhs \leq?rhs .
            qed
    qed simp
    also have ... = c* b^ k* max 1 (real-of-nat k^deg) unfolding b-def deg-def
    also have .. Sp by fact
    finally show norm ((jordan-block n a ` m}k)$$(i,j))\leqp
    qed
qed
```

definition norm-bound $::$ ' $a$ mat $\Rightarrow$ real $\Rightarrow$ bool where
norm-bound $A \quad b \equiv \forall i j$. $i<\operatorname{dim}$-row $A \longrightarrow j<\operatorname{dim}-c o l A \longrightarrow \operatorname{norm}(A \$ \$$ $(i, j)) \leq b$
lemma norm-boundI[intro]:
assumes $\bigwedge i j$. $i<$ dim-row $A \Longrightarrow j<\operatorname{dim}$-col $A \Longrightarrow \operatorname{norm}(A \$ \$(i, j)) \leq b$ shows norm-bound $A b$
unfolding norm-bound-def using assms by blast
lemma jordan-block-constant-bound2:
$\exists$. norm $(a::$ ' $a::$ real-normed-field $)<1 \longrightarrow$
$\left(\forall i j k . i<n \longrightarrow j<n \longrightarrow\right.$ norm $\left(\left(\right.\right.$ jordan-block $\left.\left.\left.n a \widehat{m}_{m} k\right) \$ \$(i, j)\right) \leq p\right)$
using jordan-block-constant-bound by auto
lemma jordan-matrix-poly-bound2:
fixes $n$-as :: $\left(\right.$ nat $\left.\times{ }^{\prime} a\right)$ list
assumes $n$-as: $\bigwedge n a .(n, a) \in$ set $n$-as $\Longrightarrow n>0 \Longrightarrow$ norm $a \leq 1$
and $N: \bigwedge n a .(n, a) \in$ set $n$-as $\Longrightarrow$ norm $a=1 \Longrightarrow n \leq N$
shows $\exists c 1 . \forall k$. $\forall e \in$ elements-mat (jordan-matrix $n$-as $\widehat{m}_{m} k$ ).
norm $e \leq c 1+$ of-nat $k^{\wedge}(N-1)$
proof -
from jordan-matrix-carrier [of n-as] obtain $d$ where
jm: jordan-matrix $n$-as $\in$ carrier-mat $d d$ by blast
define $f$ where $f=\left(\lambda n\left(a::^{\prime} a\right) i j k\right.$. norm ( $\left(\right.$ jordan-block $\left.\left.n a{ }_{m} k\right) \$ \$(i, j)\right)$ )
let ? $g=\lambda k c 1 . c 1+$ of-nat $k{ }^{\wedge}(N-1)$
let ? $P=\lambda n\left(a::^{\prime} a\right) ~ i j k c 1$.fnaijks?gkc1
define $Q$ where $Q=\left(\lambda n\left(a::^{\prime} a\right) k c 1 . \forall i j . i<n \longrightarrow j<n \longrightarrow ? P n a i j k c 1\right)$
have $\Lambda c c^{\prime} k n a i j . c \leq c^{\prime} \Longrightarrow ? P n a i j k c \Longrightarrow ? P n a i j k c^{\prime}$ by auto
hence $Q$-mono: $\bigwedge n a c c^{\prime} . c \leq c^{\prime} \Longrightarrow \forall k . Q n a k c \Longrightarrow \forall k . Q n a k c^{\prime}$
unfolding $Q$-def by arith
\{ fix $n$ a assume na: $(n, a) \in$ set $n$-as
obtain $c$ where $c:$ norm $a<1 \longrightarrow(\forall i j k . i<n \longrightarrow j<n \longrightarrow f n a i j k$ $\leq c$ )
apply (rule exE[OF jordan-block-constant-bound2])
unfolding $f$-def using Jordan-Normal-Form.jordan-block-constant-bound2 by metis
define $c 1$ where $c 1=\max 1 c$
then have $c 1 \geq 1 c 1 \geq c$ by auto
have $\exists c 1 . \forall k i j . i<n \longrightarrow j<n \longrightarrow$ ? P n a ijk $k 1$
proof rule+
fix $i j k$ assume $i<n j<n$
then have $0<n$ by auto
let ? jbs $=\operatorname{map}(\lambda(n, a)$. jordan-block $n a) n$-as
have sq-jbs: Ball (set ? jbs) square-mat by auto
have jordan-matrix n-as $\widehat{m}_{m} k=$ diag-block-mat (map $\left(\lambda A . A \widehat{m}_{m}\right.$ ) ?jbs)
unfolding jordan-matrix-def using diag-block-pow-mat $[O F ~ s q-j b s]$ by auto show ?P naijkc1
proof (cases norm $a=1$ )

```
        case True {
            have nN:n-1\leqN-1 using N[OF na] True by auto
            have fna i jk\leqmax 1 (of-nat k^(n-1))
                using Jordan-Normal-Form.jordan-block-poly-bound True \langlei<n>\langlej<n>
                unfolding f}f\mathrm{ -def by auto
            also have .. \leq max 1 (of-nat k^(N-1))
                    proof (cases k=0)
                    case False then show ?thesis
                        by (subst max.mono[OF - power-increasing[OF nN]], auto)
                qed (simp add: power-eq-if)
            also have ... \leq max c1 (of-nat k^ (N-1)) using <c1\geq1> by auto
            also have ... \leqc1 + (of-nat k` (N-1)) using <c1\geq1> by auto
            finally show ?thesis by simp
        } next
        case False {
            then have na1: norm a<1 using n-as[OF na] <0<n> by auto
            hence fn a ijk\leqc using c\langlei<n\rangle\langlej<n\rangle by auto
            also have ... \leqc1 using <c\leqc1\rangle.
            also have ... \leqc1+of-nat k^(N-1) by auto
            finally show ?thesis by auto
        }
        qed
    qed
}
hence }\forallna.\existsc1.na\inset n-as\longrightarrow(\forallk.Q (fst na) (snd na) k c1
    unfolding Q-def by auto
from choice[OF this] obtain c'
    where c': \ na k.na \in set n-as\LongrightarrowQ (fst na) (snd na) k (c'na) by blast
define c where c=max 0 (Max (set (map c' n-as)))
{ fix n a assume na: (n,a)\in set n-as
    then have Q: \forallk.Q n a k (c'(n,a)) using c'[OF na] by auto
    from na have c'}\mp@subsup{c}{}{\prime}(n,a)\in\operatorname{set}(map c'n-as) by aut
    from Max-ge[OF - this] have c' (n,a)\leqc unfolding c-def by auto
    from Q-mono[OF this Q] have }\k.Qnakc by blas
}
hence Q: \kna. (n,a) \in set n-as\LongrightarrowQ nakc by auto
have c0:c\geq0 unfolding c-def by simp
{ fix kn a e
    assume na:(n,a) \in set n-as
    let ?jbk = jordan-block n a \widehat{m}}
    assume e\in elements-mat ?jbk
    from elements-matD[OF this] obtain ij
        where }i<nj<n\mathrm{ and [simp]: e=? jbk $$(i,j)
        by (simp only:pow-mat-dim-square[OF jordan-block-carrier],auto)
    hence norm e\leq?g k c using Q[OF na] unfolding Q-def f-def by simp
}
hence norm-jordan:
    \k.}\forall(n,a)\in set n-as.\foralle\in elements-mat (jordan-block n a 人 m k)
    norm e\leq?g k c by auto
```

```
    {fix k
        let ?jmk = jordan-matrix n-as \widehat{m}
        have dim-row ?jmk = d dim-col ?jmk = d
            using jm by (simp only:pow-mat-dim-square[OF jm])+
    let ?As = (map ( }\lambda(n,a).jordan-block n a \ m k) n-as
    have \e. e\in elements-mat ?.jmk \Longrightarrow norm e\leq??g kc
    proof -
            fix e assume e:e\in elements-mat ?jmk
            obtain ij where ij: i<dj<d and e=? jmk $$ (i,j)
            using elements-matD[OF e] by (simp only:pow-mat-dim-square[OF jm],auto)
            have ?jmk = diag-block-mat ?As
            using jordan-matrix-pow[of n-as k] by auto
    hence elements-mat ?jmk\subseteq{0}\cup\bigcup(set (map elements-mat ?As))
            using elements-diag-block-mat[of ?As] by auto
            hence e-mem: e\in{0}\cup\bigcup (set (map elements-mat ?As))
                using e by blast
            show norm e\leq?g kc
            proof (cases e=0)
            case False
                    then have e\in\bigcup (set (map elements-mat ?As)) using e-mem by auto
                    then obtain na
                        where e\in elements-mat (jordan-block n a 人 m}k\mathrm{ )
                    and na: (n,a) \in set n-as by force
                    thus ?thesis using norm-jordan na by force
        qed (insert c0, auto)
    qed
    }
    thus ?thesis by auto
qed
lemma norm-bound-bridge:
    \foralle\in elements-mat A. norm e\leqb\Longrightarrow norm-bound A b
    unfolding norm-bound-def by force
lemma norm-bound-mult: assumes A1: A1 \in carrier-mat nr n
    and A2: A2 \in carrier-mat n nc
    and b1: norm-bound A1 b1
    and b2: norm-bound A2 b2
    shows norm-bound (A1 * A2) (b1*b2 * of-nat n)
proof
    let ?A = A1 * A2
    let ?n=of-nat n
    fix ij
    assume i: i< dim-row ?A and j:j< dim-col ?A
    define v1 where v1 = ( }\lambdak\mathrm{ . row A1 i$k)
    define v2 where v2 = ( }\lambdak\mathrm{ . col A2 j$k)
    from assms(1-2) have dim: dim-row A1 = nr dim-col A2 = nc dim-col A1 =
n dim-row A2 = n by auto
    {
```

fix $k$
assume $k$ : $k<n$
have $n$ : norm $(v 1 k) \leq b 1$ norm $(v 2 k) \leq b 2$
using $i j k \operatorname{dim} v 1$-def v2-def
b1 [unfolded norm-bound-def, rule-format, of $i k$ ]
b2[unfolded norm-bound-def, rule-format, of $k j$ ] by auto
have norm $(v 1 k * v 2 k) \leq \operatorname{norm}(v 1 k) *$ norm (v2 $k$ ) by (rule norm-mult-ineq)
also have $\ldots \leq b 1 * b 2$ by (rule mult-mono' $[$ OF n], auto)
finally have norm $(v 1 k * v 2 k) \leq b 1 * b 2$.
\} note bound $=$ this
have ?A $\$ \$(i, j)=$ row $A 1 i \cdot \operatorname{col} A 2 j$ using $\operatorname{dim} i j$ by simp
also have $\ldots=\left(\sum k=0 . .<n . v 1 k * v 2 k\right)$ unfolding scalar-prod-def
using $\operatorname{dim} i j v 1-$ def v2-def by $\operatorname{simp}$
also have norm $(\ldots) \leq\left(\sum k=0 . .<n . b 1 * b 2\right)$
by (rule sum-norm-le, insert bound, simp)
also have $\ldots=b 1 * b 2 * ? n$ by simp
finally show norm $(? A \$ \$(i, j)) \leq b 1 * b 2 * ? n$.
qed
lemma norm-bound-max: norm-bound $A(\operatorname{Max}\{\operatorname{norm}(A \$ \$(i, j)) \mid i j . i<$ dim-row $A \wedge j<\operatorname{dim}-\operatorname{col} A\}$ )
(is norm-bound $A$ (Max ?norms))
proof
fix $i j$
have fin: finite ?norms by (simp add: finite-image-set2)
assume $i<d i m$-row $A$ and $j<d i m-c o l ~ A$
hence norm $(A \$ \$(i, j)) \in$ ?norms by auto
from Max-ge[OF fin this] show norm $(A \$ \$(i, j)) \leq$ Max ?norms.
qed
lemma jordan-matrix-poly-bound: fixes $n$-as :: (nat $\times$ 'a)list assumes $n$-as: $\bigwedge n a .(n, a) \in$ set $n$-as $\Longrightarrow n>0 \Longrightarrow$ norm $a \leq 1$
and $N: \bigwedge n a .(n, a) \in$ set $n$-as $\Longrightarrow$ norm $a=1 \Longrightarrow n \leq N$
shows $\exists c 1 . \forall k$. norm-bound (jordan-matrix n-as $\left.\widehat{m}_{m} \widehat{k}\right)(c 1+$ of-nat $k \wedge(N$
-1))
using jordan-matrix-poly-bound2 norm-bound-bridge $N$ n-as
by metis
lemma jordan-nf-matrix-poly-bound: fixes $n$-as :: (nat $\times$ 'a)list
assumes $A: A \in$ carrier-mat $n n$
and $n$-as: $\bigwedge n a .(n, a) \in$ set $n$-as $\Longrightarrow n>0 \Longrightarrow$ norm $a \leq 1$
and $N: \bigwedge n a .(n, a) \in$ set $n$-as $\Longrightarrow$ norm $a=1 \Longrightarrow n \leq N$
and jnf: jordan-nf $A$ n-as
shows $\exists c 1 c 2 . \forall k$. norm-bound $\left(A \widehat{m}_{m} k\right)\left(c 1+c 2 *\right.$ of-nat $\left.k^{\wedge}(N-1)\right)$
proof -
let ? $c>2=\prod(n, a) \leftarrow n$-as. $[:-a, 1:]{ }^{\wedge} n$
let $? J=$ jordan-matrix $n$-as
from jnf[unfolded jordan-nf-def]
have sim: similar-mat $A$ ? $J$ by auto
then obtain $P Q$ where sim-wit: similar-mat-wit $A$ ?J $P Q$ unfolding simi-lar-mat-def by auto
from similar-mat-wit-pow-id[OF this] have pow: $\wedge k . A \widehat{m}_{m} k=P * ? J \widehat{m}_{m} k *$ $Q$.
from sim-wit[unfolded similar-mat-wit-def Let-def] A
have $J: ? J \in$ carrier-mat $n n$ and $P: P \in$ carrier-mat $n n$ and $Q: Q \in$ car-rier-mat $n n$
unfolding carrier-mat-def by force+
have $\exists c 1 . \forall k$. norm-bound $\left(? J \widehat{m}_{m} k\right)\left(c 1+\right.$ of-nat $\left.k{ }^{\wedge}(N-1)\right)$
by (rule jordan-matrix-poly-bound[OF n-as N])
then obtain $c 1$ where
bound-pow: $\bigwedge k$. norm-bound $\left(\left(? J{ }_{m} k\right)\right)\left(c 1+\right.$ of-nat $\left.k{ }^{\wedge}(N-1)\right)$ by blast obtain $b P$ where $b P$ : norm-bound $P b P$ using norm-bound-max $[o f P]$ by auto obtain $b Q$ where $b Q$ : norm-bound $Q b Q$ using norm-bound-max $[o f ~ Q]$ by auto let $? n=$ of-nat $n::$ real let ? $c 2=b P * ? n * b Q * ? n$ let ? $c 1=? c 2 * c 1$
\{
fix $k$
have $J k$ : ? $J{ }_{m} k \in$ carrier-mat $n n$ using $J$ by simp
from norm-bound-mult[OF mult-carrier-mat[OF P Jk] Q
norm-bound-mult[OF P Jk bP bound-pow] bQ, folded pow]
have norm-bound $\left(A \widehat{m}_{m} k\right)(? c 1+? c 2 *$ of-nat $k \wedge(N-1))$ (is norm-bound ? exp)
by (simp add: field-simps)
\} note main $=$ this
show ?thesis
by (intro exI allI, rule main)
qed
end
context
fixes $f$-ty $::$ ' $a$ :: field itself
begin
lemma char-matrix-jordan-block: char-matrix (jordan-block $n$ a) $b=($ jordan-block $n(a-b)$ )
unfolding char-matrix-def jordan-block-def by auto
lemma diag-jordan-block-pow: diag-mat (jordan-block $\left.n(a:: ' a){ }_{m} k\right)=$ replicate $n\left(a^{\wedge} k\right)$
unfolding diag-mat-def jordan-block-pow
by (intro nth-equalityI, auto)
lemma jordan-block-zero-pow: (jordan-block $n(0::$ 'a) $){ }_{m} k=$ (mat $n n(\lambda(i, j)$. if $j \geq i \wedge j-i=k$ then 1 else 0$))$
proof -
\{
fix $i j$
assume $*: j-i \neq k$

```
        have of-nat (k choose (j-i))*0^ (k+i-j)=(0 :: 'a)
        proof (cases k+i-j>0)
            case True thus ?thesis by (cases k+i-j,auto)
        next
            case False
            with * have j - i>k by auto
            thus ?thesis by (simp add: binomial-eq-0)
    qed
    }
    thus ?thesis unfolding jordan-block-pow by (intro eq-matI, auto)
qed
end
lemma jordan-matrix-concat-diag-block-mat: jordan-matrix (concat jbs) = diag-block-mat
(map jordan-matrix jbs)
    unfolding jordan-matrix-def[abs-def]
    by (induct jbs, auto simp: diag-block-mat-append Let-def)
lemma jordan-nf-diag-block-mat: assumes Ms: \bigwedge A jbs. (A,jbs) \in set Ms \Longrightarrow
jordan-nf A jbs
    shows jordan-nf (diag-block-mat (map fst Ms)) (concat (map snd Ms))
proof -
    let ?Ms = map ( }\lambda(A,jbs). (A, jordan-matrix jbs))M
    have id: map fst ?Ms = map fst Ms by auto
    have id2: map snd ?Ms = map jordan-matrix (map snd Ms) by auto
    {
        fix }A
        assume (A,B)\in set ?Ms
        then obtain jbs where mem: (A,jbs)\in set Ms and B:B= jordan-matrix jbs
by auto
    from Ms[OF mem] have similar-mat A B unfolding B jordan-nf-def by auto
    }
    from similar-diag-mat-block-mat[of ?Ms, OF this, unfolded id id2] Ms
    show ?thesis
        unfolding jordan-nf-def jordan-matrix-concat-diag-block-mat by force
qed
```

lemma jordan-nf-char-poly: assumes jordan-nf $A n$-as
shows char-poly $A=(\Pi(n, a) \leftarrow n$-as. $[:-a, 1:] \wedge n)$
unfolding jordan-matrix-char-poly[symmetric]
by (rule char-poly-similar, insert assms[unfolded jordan-nf-def], auto)
lemma jordan-nf-block-size-order-bound: assumes jnf: jordan-nf A n-as
and mem: $(n, a) \in$ set $n$-as
shows $n \leq$ order a (char-poly $A$ )
proof -
from jnf[unfolded jordan-nf-def]
have similar-mat $A$ (jordan-matrix n-as) by auto
from similar-matD[OF this] obtain $m$ where $A \in$ carrier-mat $m$ by auto
from degree-monic-char-poly $[O F$ this $]$ have $A$ : char-poly $A \neq 0$ by auto
from mem obtain as bs where nas: n-as $=a s @(n, a) \# b s$
by (meson split-list)
from jordan-nf-char-poly[OF jnf]
have $c A$ : char-poly $A=\left(\prod(n, a) \leftarrow n\right.$-as. $\left.[:-a, 1:]{ }^{\wedge} n\right)$.
also have $\ldots=[:-a, 1:]$ へ $n *\left(\prod(n, a) \leftarrow a s @ b s\right.$. $[:-a, 1:]$ ^ $\left.n\right)$ unfolding nas by auto
also have [: $-a, 1:]$ ^ $n$ dvd . . . unfolding dvd-def by blast
finally have [: $-a, 1$ :] ^ $n$ dvd char-poly $A$ by auto
from order-max $[O F$ this $A]$ show ?thesis .
qed
lemma similar-mat-jordan-block-smult: fixes $A$ :: 'a :: field mat
assumes similar-mat $A$ (jordan-block $n$ a)
and $k: k \neq 0$
shows similar-mat $(k \cdot m A)($ jordan-block $n(k * a))$
proof -
let ? $J=$ jordan-block $n$ a
let ? Jk $=$ jordan-block $n(k * a)$
let ? $k J=k \cdot{ }_{m}$ jordan-block $n a$
from $k$ have inv: $k^{\wedge} i \neq 0$ for $i$ by auto
let ? $A=$ mat-diag $n(\lambda i . k \uparrow i)$
let ? $B=$ mat-diag $n(\lambda i$. inverse $(k \uparrow i))$
have similar-mat-wit ? Jk ?kJ ? A ?B
proof (rule similar-mat-witI)
show jordan-block $n(k * a)=? A * ? k J * ? B$
by (subst mat-diag-mult-left $[$ of - $n]$, force, subst mat-diag-mult-right $[o f-n]$, insert $k$ inv, auto simp: jordan-block-def field-simps intro!: eq-matI)
qed (auto simp: inv field-simps $k$ )
hence $k J$ : similar-mat? $J k$ ? $k J$
unfolding similar-mat-def by auto
have similar-mat $A$ ? $J$ by fact
hence similar-mat $(k \cdot m A)(k \cdot m$ ? $J)$ by (rule similar-mat-smult)
with $k J$ show ?thesis
using similar-mat-sym similar-mat-trans by blast
qed
lemma jordan-matrix-Cons: jordan-matrix (Cons ( $n, a) n$-as) $=$ four-block-mat (jordan-block $n a) \quad\left(0_{m} n\right.$ (sum-list (map fst $n$-as)) )
( $0_{m}$ (sum-list (map fst $n$-as)) $n$ ) (jordan-matrix $n$-as)
unfolding jordan-matrix-def by (simp, simp add: jordan-matrix-def[symmetric])
lemma similar-mat-jordan-matrix-smult: fixes $n$-as :: (nat $\times{ }^{\prime} a$ :: field) list assumes $k: k \neq 0$
shows similar-mat $(k \cdot m$ jordan-matrix $n$-as) (jordan-matrix (map $(\lambda(n, a) .(n$, $k * a)$ ) $n$-as))
proof (induct $n$-as)

```
    case Nil
    show ?case by (auto simp: jordan-matrix-def intro!: similar-mat-refl)
next
    case (Cons na n-as)
    obtain \(n a\) where \(n a\) : \(n a=(n, a)\) by force
    let ?l \(=\operatorname{map}(\lambda(n, a) .(n, k * a))\)
    let \(? n=\) sum-list (map fst \(n\)-as)
    have \(k \cdot m\) jordan-matrix (Cons na \(n\)-as) \(=k \cdot m\) four-block-mat
        (jordan-block na) ( \(0_{m} n\) ?n)
        \(\left(0_{m}\right.\) ?n \(n\) ) (jordan-matrix \(n\)-as) \(\left(\right.\) is \(? M=-\cdot_{m}\) four-block-mat ?A ?B ?C ?D)
    by (simp add: na jordan-matrix-Cons)
    also have \(\ldots=\) four-block-mat \(\left(k \cdot_{m}\right.\) ? \(\left.A\right)\) ? \(B\) ? \(C(k \cdot m\) ? \(D)\)
    by (subst smult-four-block-mat, auto)
    finally have \(j m: ? M=\) four-block-mat \(\left(k \cdot m\right.\) ?A) ?B ? \(C\left(k \cdot_{m}\right.\) ?D) .
    have \([\) simp \(]\) : fst (case \(x\) of \((n::\) nat, \(a) \Rightarrow(n, k * a))=f\) st \(x\) for \(x\) by (cases \(x\),
auto)
    have jmk: jordan-matrix (?l (Cons na n-as)) = four-block-mat
        (jordan-block \(n(k * a))\) ?B
        ?C (jordan-matrix (?l n-as)) (is ? \(k M=\) four-block-mat ? \(k A-\) ? \(k D\) )
        by (simp add: na jordan-matrix-Cons o-def)
    show ?case unfolding jmk jm
        by (rule similar-mat-four-block-0-0[OF similar-mat-jordan-block-smult \([O F-k]\)
Cons],
            auto intro!: similar-mat-refl)
qed
lemma jordan-nf-smult: fixes \(k::\) ' \(a\) :: field
    assumes jn: jordan-nf \(A\) n-as
    and \(k: k \neq 0\)
    shows jordan-nf \((k \cdot m A)(\operatorname{map}(\lambda(n, a) .(n, k * a)) n\)-as)
proof -
    let ?l \(=\operatorname{map}(\lambda(n, a) .(n, k * a))\)
    from jn[unfolded jordan-nf-def] have sim: similar-mat A (jordan-matrix n-as)
by auto
    from similar-mat-smult [OF this, of \(k\) ] similar-mat-jordan-matrix-smult \([O F k\), of
\(n-a s]\)
    have similar-mat \((k \cdot m A)(\) jordan-matrix \((\operatorname{map}(\lambda(n, a) .(n, k * a)) n\)-as \())\)
        using similar-mat-trans by blast
    with \(j n\) show ?thesis unfolding jordan-nf-def by force
qed
lemma jordan-nf-order: assumes jordan-nf \(A\) n-as
    shows order a (char-poly A) =sum-list (map fst (filter \((\lambda\) na. snd na \(=a)\)
\(n\)-as))
proof -
    let ? \(p=\lambda n\)-as. \(\left(\prod(n, a) \leftarrow n\right.\)-as. \(\left.[:-a, 1:]{ }^{\wedge} n\right)\)
    let \(? s=\lambda n\)-as. sum-list \((\) map fst \((\) filter \((\lambda n a . s n d n a=a) n\)-as \()\) )
    from jordan-nf-char-poly[OF assms]
    have order a (char-poly \(A\) ) \(=\) order a (?p n-as) by simp
```

also have $\ldots=$ ?s $n$-as
proof (induct $n$-as)
case (Cons nb n-as)
obtain $n b$ where $n b: n b=(n, b)$ by force
have order $a(? p(n b \# n$-as $))=$ order $a([:-b, 1:]$ ^ $n *$ ?p $n$-as) unfolding $n b$ by $\operatorname{simp}$
also have $\ldots=$ order $a([:-b, 1:] \wedge n)+\operatorname{order} a(? p n-a s)$
by (rule order-mult, auto simp: prod-list-zero-iff)
also have $\ldots=($ if $a=b$ then $n$ else 0$)+$ ?s n-as unfolding Cons or-der-linear-power by simp
also have $\ldots=$ ? $s(n b \# n$ - $a s$ ) unfolding $n b$ by auto
finally show ? case.
qed simp
finally show ?thesis.
qed

### 13.1 Application for Complexity

lemma factored-char-poly-norm-bound: assumes $A: A \in$ carrier-mat $n n$ and linear-factors: char-poly $A=\left(\prod\left(a::{ }^{\prime} a\right.\right.$ :: real-normed-field $) \leftarrow a s .[:-a$, 1:])
and jnf-exists: $\exists$ n-as. jordan-nf $A n$-as
and $l e-1: \wedge a . a \in$ set $a s \Longrightarrow$ norm $a \leq 1$
and $l e-N: \bigwedge a . a \in$ set $a s \Longrightarrow$ norm $a=1 \Longrightarrow$ length $($ filter $((=) a) a s) \leq N$
shows $\exists c 1 c 2 . \forall k$. norm-bound $\left(A \widehat{m}_{m} k\right)(c 1+c 2 *$ of-nat $k \wedge(N-1))$
proof -
from jnf-exists obtain $n$-as
where jnf: jordan-nf $A n$-as by auto
let ? $c p 1=\left(\prod a \leftarrow a s .[:-a, 1:]\right)$
let ? $c p 2=\prod(n, a) \leftarrow n$-as. $[:-a, 1:]{ }^{\wedge} n$
let ? $J=$ jordan-matrix n-as
from jnf[unfolded jordan-nf-def]
have sim: similar-mat $A$ ? $J$ by auto
from char-poly-similar[OF sim, unfolded linear-factors jordan-matrix-char-poly]
have $c p: ? c p 1=? c p 2$.
show ?thesis
proof (rule jordan-nf-matrix-poly-bound $[$ OF A - jnf])
fix $n a$
assume $n a:(n, a) \in$ set $n$-as
then obtain na1 na2 where n-as: n-as =na1 @ ( $n, a$ ) \# na2
unfolding in-set-conv-decomp by auto
then obtain $p$ where ? $c p 2=[:-a, 1:] \widehat{ } n * p$ unfolding $n$-as by auto
from $c p[$ unfolded this] have dvd: [: $-a, 1:] \wedge n d v d$ ?cp1 by auto
let ?as = filter $((=)$ a) as
let ? $p n=\lambda a s$. $\prod a \leftarrow a s .[:-a, 1:]$
let ? $p=\lambda a s . \prod a \leftarrow a s$. [: a, $\left.1:\right]$
have ?pn as $=$ ? $p$ (map uminus as) by (induct as, auto)
from poly-linear-exp-linear-factors[OF dvd[unfolded this]]
have $n \leq$ length (filter $((=)(-a))$ (map uminus as)).

```
    also have ... = length (filter ((=) a) as)
    by (induct as, auto)
    finally have filt:n\leqlength (filter ((=) a) as).
    {
        assume 0<n
        with filt obtain b bs where ?as = b # bs by (cases ?as, auto)
        from arg-cong[OF this, of set]
        have a\in set as by auto
        from le-1[rule-format, OF this]
        show norm a\leq1.
        note }\langlea\in\mathrm{ set as`
    } note mem = this
    {
        assume norm a = 1
        from le-N[OF mem this] filt show n}\leqN\mathrm{ by (cases n, auto)
    }
    qed
qed
end
```


## 14 Missing Vector Spaces

This theory provides some lemmas which we required when working with vector spaces.

```
theory Missing-VectorSpace
imports
    VectorSpace.VectorSpace
    Missing-Ring
    HOL-Library.Multiset
begin
```

locale comp-fun-commute-on $=$
fixes $f::{ }^{\prime} a \Rightarrow{ }^{\prime} a{ }^{\prime} a$ and $A::^{\prime} a$ set
assumes comp-fun-commute-restrict: $\forall y \in A . \forall x \in A . \forall z \in A . f y(f x z)=f x(f y$
z)
and $f: f: A \rightarrow A \rightarrow A$
begin
lemma comp-fun-commute-on-UNIV:
assumes $A=(U N I V$ :: 'a set)
shows comp-fun-commute $f$
unfolding comp-fun-commute-def
using assms comp-fun-commute-restrict $f$ by auto

```
lemma fun-left-comm:
    assumes }y\inA\mathrm{ and }x\inA\mathrm{ and }z\inA\mathrm{ shows }fy(fxz)=fx(fyz
    using comp-fun-commute-restrict assms by auto
lemma commute-left-comp:
    assumes }y\inA\mathrm{ and }x\inA\mathrm{ and }z\inA\mathrm{ and }g\inA->
    shows fy(fx(gz))=fx(fy(gz))
    using assms by (auto simp add: Pi-def o-assoc comp-fun-commute-restrict)
lemma fold-graph-finite:
    assumes fold-graph f z B y
    shows finite B
    using assms by induct simp-all
lemma fold-graph-closed:
    assumes fold-graph fzB y and B\subseteqA and z\inA
    shows }y\in
    using assms
proof (induct set: fold-graph)
    case emptyI
    then show ?case by auto
next
    case (insertI x B y)
    then show ?case using insertI f by auto
qed
lemma fold-graph-insertE-aux:
    fold-graph fz B y \Longrightarrowa\inB\Longrightarrowz\inA
    \Longrightarrow B \subseteq A
    \Longrightarrow \exists y ^ { \prime } . y = f a y ^ { \prime } \wedge \text { fold-graph f z (B-\{a\}) y'^ y y}
proof (induct set: fold-graph)
    case emptyI
    then show ?case by auto
next
    case (insertI x B y)
    show ?case
    proof (cases x=a)
    case True
    show ?thesis
    proof (rule exI[of - y])
        have B: (insert x B -{a})=B using True insertI by auto
        have f x y = f a y by (simp add: True)
        moreover have fold-graph fz (insert x B - {a}) y by (simp add: B insertI)
        moreover have }y\inA\mathrm{ using insertI fold-graph-closed[of z B] by auto
        ultimately show fxy=fay^ fold-graph fz(insert x B-{a}) y^y\in
A by simp
    qed
    next
```

```
    case False
    then obtain }\mp@subsup{y}{}{\prime}\mathrm{ where y:y=f a y ' and y': fold-graph fz(B-{a}) y' and
y'-in-A: y }\mp@subsup{y}{}{\prime}\in
    using insertI f by auto
    have fxy=fa(fxy')
        unfolding y
    proof (rule fun-left-comm)
        show }x\inA\mathrm{ using insertI by auto
        show }a\inA\mathrm{ using insertI by auto
        show }\mp@subsup{y}{}{\prime}\inA\mathrm{ using }\mp@subsup{y}{}{\prime}-in-A by aut
    qed
    moreover have fold-graph fz (insert x B - {a}) (f x y ')
        using }\mp@subsup{y}{}{\prime}\mathrm{ and }\langlex\not=a\rangle\mathrm{ and }\langlex\not\inB
        by (simp add: insert-Diff-if fold-graph.insertI)
    moreover have (fx y )}\mathrm{ ) & A using insertI f y'-in-A by auto
    ultimately show ?thesis using }\mp@subsup{y}{}{\prime}-in-
        by auto
    qed
qed
lemma fold-graph-insertE:
    assumes fold-graph fz (insert x B) v and x\not\inB and insert x B\subseteqA and z\inA
    obtains y where v=fxy and fold-graph f z B y
    using assms by (auto dest: fold-graph-insertE-aux [OF - insertI1])
lemma fold-graph-determ: fold-graph f z B x \Longrightarrow fold-graph fz B y \Longrightarrow B\subseteqA
\Longrightarrow z \in A \Longrightarrow y = x
proof (induct arbitrary: y set: fold-graph)
    case emptyI
    then show ?case
        by (meson empty-fold-graphE)
next
    case (insertI x B y v)
    from〈fold-graph f z (insert x B) v\rangle and }\langlex\not\inB\rangle\mathrm{ and <insert x B}\subseteqA\rangle\mathrm{ and }\langle
A >
    obtain }\mp@subsup{y}{}{\prime}\mathrm{ where v=fx y}\mp@subsup{y}{}{\prime}\mathrm{ and fold-graph f z B y
        by (rule fold-graph-insertE)
    from <fold-graph f z B y'` and <insert x B\subseteqA〉 have y'}=y\mathrm{ using insertI by
auto
    with }\langlev=fx\mp@subsup{y}{}{\prime}\rangle\mathrm{ show v=fxy
        by simp
qed
lemma fold-equality: fold-graph \(f z B y \Longrightarrow B \subseteq A \Longrightarrow z \in A \Longrightarrow\) Finite-Set.fold
fzB=y
    by (cases finite B)
    (auto simp add: Finite-Set.fold-def intro: fold-graph-determ dest: fold-graph-finite)
```

lemma fold-graph-fold:

```
    assumes f: finite B and BA: B\subseteqA and z:z\inA
    shows fold-graph f z B (Finite-Set.fold f z B)
proof -
    have \existsx. fold-graph f z B x
    by (rule finite-imp-fold-graph[OF f])
    moreover note fold-graph-determ
    ultimately have }\exists\mathrm{ !x. fold-graph fz B x using f BA z by auto
    then have fold-graph fzB(The (fold-graph f z B))
        by (rule theI')
    with assms show ?thesis
        by (simp add: Finite-Set.fold-def)
qed
lemma fold-insert [simp]:
    assumes finite B and x\not\inB and BA: insert x B\subseteqA and z:z\inA
    shows Finite-Set.fold fz (insert x B) =fx(Finite-Set.fold f z B)
    proof (rule fold-equality[OF - BA z])
    from〈finite B> have fold-graph fzB (Finite-Set.fold f z B)
    using BA fold-graph-fold z by auto
    hence fold-graph fz (insert x B) (fx (Finite-Set.fold f z B))
    using BA fold-graph.insertI assms by auto
    then show fold-graph fz (insert x B) (fx (Finite-Set.fold f z B))
    by simp
qed
end
lemma fold-cong:
    assumes f:comp-fun-commute-on f A and g:comp-fun-commute-on g A
        and finite S
        and cong: \bigwedgex. x \inS\Longrightarrowfx=gx
        and s=t and S=T
        and SA:S\subseteqA and s:s\inA
    shows Finite-Set.fold f s S = Finite-Set.fold g t T
proof -
    have Finite-Set.fold f s S = Finite-Set.fold g s S
        using〈finite S〉cong SA s
    proof (induct S)
        case empty
        then show?case by simp
    next
        case (insert x F)
        interpret f:comp-fun-commute-on f A by (fact f)
        interpret g:comp-fun-commute-on g A by (fact g)
        show ?case using insert by auto
    qed
    with assms show ?thesis by simp
qed
```

```
context comp-fun-commute-on
begin
lemma comp-fun-Pi:(\lambdax.fx^~gx) \inA->A->A
proof -
    have (fx~}~~x)y\inA if y:y\inA and x:x\inA for x y
        using x y
    proof (induct g x arbitrary: g)
        case 0
        then show ?case by auto
    next
        case (Suc n g)
        define h where hz=gz-1 for z
        have hyp: (fx ^~hx) y f A
            using h-def Suc.prems Suc.hyps diff-Suc-1 by metis
        have g x = Suc (hx) unfolding h-def
            using Suc.hyps(2) by auto
        then show ?case using f x hyp unfolding Pi-def by auto
    qed
    thus ?thesis by (auto simp add: Pi-def)
qed
```

lemma comp-fun-commute-funpow: comp-fun-commute-on ( $\lambda x . f x \leadsto g x) A$
proof -
have $f:(f y \leadsto g y)((f x \leadsto g x) z)=(f x \leadsto g x)((f y \leadsto g y) z)$
if $x: x \in A$ and $y: y \in A$ and $z: z \in A$ for $x y z$
proof (cases $x=y$ )
case False
show ?thesis
proof (induct $g x$ arbitrary: $g$ )
case (Suc $n g$ )
have hyp 1: $(f y \leadsto g y)(f x k)=f x((f y \leadsto g y) k)$ if $k: k \in A$ for $k$
proof (induct $g$ y arbitrary: $g$ )
case 0
then show? case by simp
next
case (Suc ng)
define $h$ where $h z=g z-1$ for $z$
with Suc have $n=h y$
by $\operatorname{simp}$
with Suc have hyp: $\left(f y{ }^{\wedge} h y\right)(f x k)=f x((f y \leadsto h y) k)$
by auto
from Suc h-def have $g: g y=S u c(h y)$
by $\operatorname{simp}$
have $((f y \sim h y) k) \in A$ using $y k$ comp-fun-Pi[of $h]$ unfolding Pi-def
by auto
then show ?case
by (simp add: comp-assoc g hyp) (auto simp add: o-assoc comp-fun-commute-restrict $x y k)$
qed
define $h$ where $h a=($ if $a=x$ then $g x-1$ else $g a)$ for $a$
with Suc have $n=h x$
by $\operatorname{simp}$
with Suc have $(f y \leadsto h y)((f x \leadsto h x) z)=(f x \leadsto h x)((f y \leadsto h y) z)$
by auto
with False have Suc2: $(f x \leadsto h x)((f y \leadsto g y) z)=(f y \leadsto g y)((f x \leadsto h$ x) z)
using $h$-def by auto
from Suc h-def have $g: g x=S u c(h x)$
by $\operatorname{simp}$
have $(f x \leadsto h x) z \in A$ using comp-fun-Pi[of $h] x z$ unfolding Pi-def by auto
hence $*:(f y \wedge \wedge y)\left(f x\left(\left(f x{ }^{\wedge} h x\right) z\right)\right)=f x\left(\left(f y{ }^{\wedge} g y\right)((f x \wedge h x)\right.$

## z))

using hyp1 by auto
thus ?case using $g$ Suc2 by auto
qed $\operatorname{simp}$
qed $\operatorname{simp}$
thus ?thesis by (auto simp add: comp-fun-commute-on-def comp-fun-Pi o-def) qed
lemma fold-mset-add-mset:
assumes $M A$ : set-mset $M \subseteq A$ and $s: s \in A$ and $x: x \in A$
shows fold-mset $f s($ add-mset $x M)=f x($ fold-mset $f s M)$
proof -
interpret mset: comp-fun-commute-on $\lambda y . f y \leadsto$ count $M$ y $A$
by (fact comp-fun-commute-funpow)
interpret mset-union: comp-fun-commute-on $\lambda y$. f y ${ }^{\sim}$ count (add-mset x M) $y$ A
by (fact comp-fun-commute-funpow)
show ?thesis
proof (cases $x \in$ set-mset $M$ )
case False
then have $*$ : count (add-mset $x$ ) $x=1$
by (simp add: not-in-iff)
have Finite-Set.fold $(\lambda y . f y \sim$ count $($ add-mset $x M) y) s($ set-mset $M)=$
Finite-Set.fold ( $\lambda y . f y \leadsto$ count $M y) s($ set-mset $M)$
by (rule fold-cong[of - A], auto simp add: assms False comp-fun-commute-funpow)
with False * s MA $x$ show ?thesis
by (simp add: fold-mset-def del: count-add-mset)

## next

case True
let ?f $=(\lambda x a . f x a \sim$ count $(a d d-m s e t x M) x a)$
let ?f2 $=(\lambda x . f x$ ^^ count $M x)$
define $N$ where $N=$ set-mset $M-\{x\}$

```
    have F:Finite-Set.fold ?f s (insert x N) = ?f x (Finite-Set.fold ?f s N)
        by (rule mset-union.fold-insert, auto simp add: assms N-def)
    have F2: Finite-Set.fold ?f2 s (insert x N) =?f2 x (Finite-Set.fold ?f2 s N)
        by (rule mset.fold-insert, auto simp add: assms N-def)
    from N-def True have *: set-mset M = insert x N x\not\inN finite N by auto
    then have Finite-Set.fold ( }\lambday.fy~~count (add-mset x M) y) s N
        Finite-Set.fold ( }\lambday.fy~~ count M y) s 
        using MA N-def s
        by (auto intro!: fold-cong comp-fun-commute-funpow)
    with * show ?thesis by (simp add: fold-mset-def del: count-add-mset, unfold
F F2, auto)
    qed
qed
end
```

lemma Diff-not-in: $a \notin A-\{a\}$ by auto
context abelian-group begin
lemma finsum-restrict:
assumes $f A: f: A \rightarrow$ carrier $G$
and restr: restrict $f A=$ restrict $g A$
shows finsum $G f A=$ finsum $G g A$
proof (rule finsum-cong';rule?)
fix $a$ assume $a: a: A$
have $f a=$ restrict $f A$ asing $a$ by simp
also have $\ldots=$ restrict $g A a$ using restr by simp
also have $\ldots=g$ a using $a$ by simp
finally show $f a=g a$.
thus $g a$ : carrier $G$ using $f A a$ by force
qed
lemma minus-nonzero: $x$ : carrier $G \Longrightarrow x \neq \mathbf{0} \Longrightarrow \ominus x \neq \mathbf{0}$
using $r$-neg by force
end
lemma (in ordered-comm-monoid-add) positive-sum:
assumes $X$ : finite $X$
and $f: X \rightarrow\left\{y::{ }^{\prime} a . y \geq 0\right\}$
shows sum $f X \geq 0 \wedge\left(\operatorname{sum} f X=0 \longrightarrow f^{\prime} X \subseteq\{0\}\right)$
using assms
proof (induct set:finite)
case (insert $x X$ )
hence $x 0: f x \geq 0$ and sum0: sum $f X \geq 0$ by auto
hence sum $f$ (insert $x X) \geq 0$ using insert by auto

```
    moreover
    { assume sum f(insert x X)=0
    hence f}x=0\mathrm{ sum fX=0
        using sum0 x0 insert add-nonneg-eq-0-iff by auto
    }
    ultimately show ?case using insert by blast
qed auto
```

lemma insert-union: insert $x X=X \cup\{x\}$ by auto
context vectorspace begin
lemmas lincomb-insert2 $=$ lincomb-insert[unfolded insert-union[symmetric]]
lemma lincomb-restrict:
assumes $U: U \subseteq$ carrier $V$
and $a: a: U \rightarrow$ carrier $K$
and restr: restrict $a U=$ restrict $b U$
shows lincomb a $U=$ lincomb $b U$
proof -
let ?f $=\lambda a u$. $a u \odot_{V} u$
have fa: ?f $a: U \rightarrow$ carrier $V$ using $a U$ by auto
have restrict (?f a) $U=$ restrict (?f b) $U$
proof
fix $u$
have $u: U \Longrightarrow a u=b u$ using restr unfolding restrict-def by metis
thus restrict (?f a) $U u=$ restrict (?f b) $U u$ by auto
qed
thus ?thesis
unfolding lincomb-def using finsum-restrict $[O F$ fa] by auto
qed
lemma lindep-span:
assumes $U: U \subseteq$ carrier $V$ and $\operatorname{fin} U$ : finite $U$
shows lin-dep $U=(\exists u \in U . u \in \operatorname{span}(U-\{u\}))($ is ?l $=? r)$
proof
assume $l$ : ?l show ?r
proof -
from $l[$ unfolded lin-dep-def]
obtain $A$ a u
where finA: finite $A$
and $A U: A \subseteq U$
and $a A: a: A \rightarrow$ carrier $K$
and aA0: lincomb a $A=$ zero $V$
and $u A: u: A$
and au0: a $u \neq$ zero $K$ by auto
define $a^{\prime}$ where $a^{\prime}=(\lambda v$. (if $v: A$ then a $v$ else zero $K)$ )

```
have }\mp@subsup{a}{}{\prime}U:\mp@subsup{a}{}{\prime}:U->\mathrm{ carrier K unfolding }\mp@subsup{a}{}{\prime}\mathrm{ -def using aA by auto
have }uU:u:U\mathrm{ using }uAAU\mathrm{ by auto
have }\mp@subsup{a}{}{\prime}u0:\mp@subsup{a}{}{\prime}u\not=zero K unfolding a'-def using au0 uA by aut
define B}\mathrm{ where B=U-A
have B: B\subseteq carrier V unfolding B-def using U by auto
have UAB: U=A\cupB unfolding B-def using }AU\mathrm{ by auto
have finB: finite B using finU B-def by auto
have }AB:A\capB={}\mathrm{ unfolding B-def by auto
let ?f = \lambdav.a v \odot V v
have fA: ?f : A }->\mathrm{ carrier V unfolding a'-def using aA AU U by auto
let ?f' = \lambdav. a' v \odot V v
have restrict ?f A = restrict ?f' A unfolding a'-def by auto
hence finsum: finsum }V\mathrm{ ?f' }A=\mathrm{ finsum }V\mathrm{ ?f }
    using finsum-restrict[OF fA] by simp
have f}\mp@subsup{f}{}{\prime}A:?\mp@subsup{f}{}{\prime}:A->\mathrm{ carrier }
proof
    fix x assume xA: }x\in
    show ?f' }x\mathrm{ : carrier V unfolding }\mp@subsup{a}{}{\prime}\mathrm{ -def using aA xA AU U by auto
qed
have f'B:?f': }B->\mathrm{ carrier }
proof
    fix }x\mathrm{ assume xB: x:B
    have }x\not\inA\mathrm{ using }\mp@subsup{a}{}{\prime}UxB\mathrm{ unfolding B-def by auto
    thus ?f' }x\mathrm{ : carrier Vusing }xBB\mathrm{ unfolding }\mp@subsup{a}{}{\prime}\mathrm{ -def by auto
qed
have sumB0: finsum V ?f' B = zero V
proof -
    {fix B'
        have finite }\mp@subsup{B}{}{\prime}\Longrightarrow\mp@subsup{B}{}{\prime}\subseteqB\Longrightarrow\mathrm{ finsum V ?f' B' = zero V
        proof(induct set:finite)
            case (insert b B')
                have finB': finite 列}\mathrm{ and }b\mp@subsup{B}{}{\prime}:b\not\in\mp@subsup{B}{}{\prime}\mathrm{ using insert by auto
                    have f'B':? ? ' : 洼 }->\mathrm{ carrier }V\mathrm{ using }\mp@subsup{f}{}{\prime}B\mathrm{ insert by auto
                    have bA: b\not\inA using insert unfolding B-def by auto
                    have b: b: carrier V using insert B by auto
                    have foo: a' }b\odot\mp@subsup{\odot}{V}{}b\in\mathrm{ carrier V unfolding a'-def using bA b by auto
                    have IH: finsum V ?f' B' = zero V using insert by auto
                    show ?case
                        unfolding finsum-insert[OF finB' b\mp@subsup{B}{}{\prime}\mp@subsup{f}{}{\prime}\mp@subsup{B}{}{\prime}\mathrm{ foo]}]
                    using IH a'-def bA b by auto
        qed auto
    }
    thus ?thesis using finB by auto
qed
have a'A0: lincomb a' U = zero V
    unfolding }UA
    unfolding lincomb-def
    unfolding finsum-Un-disjoint[OF finA finB AB f}
    unfolding finsum
```

```
        unfolding aAO[unfolded lincomb-def]
        unfolding sumB0 by simp
    have uU:u:U using uA AU by auto
    moreover have u: span (U-{u})
        using lincomb-isolate(2)[OF finU U a'U uU a'u0 a'A0].
    ultimately show ?r by auto
qed
next assume r: ?r show ?l
proof -
    from r obtain u where uU:u:U and uspan: u: span (U-{u}) by auto
    hence u:u:carrier V using U by auto
    have finUu: finite ( U-{u}) using finU by auto
    have Uu:U-{u}\subseteqcarrier }V\mathrm{ using }U\mathrm{ by auto
    obtain a
    where ulin: u = lincomb a (U-{u})
            and a:a:U-{u} }->\mathrm{ carrier K
        using uspan unfolding finite-span[OF finUu Uu] by auto
    show ?l unfolding lin-dep-def
    proof(intro exI conjI)
        let ?a = \lambdav. if v=u then }\mp@subsup{\ominus}{K}{}\mathrm{ one K else a v
        show ?a :U }->\mathrm{ carrier K using a by auto
        hence a':?a:U-{u}\cup{u} -> carrier K by auto
        have U=U-{u}\cup{u} using uU by auto
        also have lincomb ?a ... = ?a u \odot VV u \oplus \ lincomb ?a ( }U-{u}
            unfolding lincomb-insert[OF finUu Uu a' Diff-not-in u] by auto
        also have restrict a (U-{u}) = restrict ?a (U-{u}) by auto
            hence lincomb ?a (U-{u})= lincomb a (U-{u})
                using lincomb-restrict[OF Uu a] by auto
        also have ?a u \odot }\mp@subsup{V}{V}{}u=\mp@subsup{\ominus}{V}{}u\mathrm{ using smult-minus-1[OF u] by simp
        also have lincomb a (U-{u})=u using ulin..
        also have }\mp@subsup{\ominus}{V}{}u\mp@subsup{\oplus}{V}{}u=\mathrm{ zero V using l-neg[OF u].
        finally show lincomb ?a U = zero V by auto
    qed (insert uU finU, auto)
    qed
qed
lemma not-lindepD:
    assumes ~ lin-dep S
    and finite A A\subseteqSf:A-> carrier K lincomb f A = zero }
    shows f:A->{zero K}
using assms unfolding lin-dep-def by blast
lemma span-mem:
assumes \(E: E \subseteq\) carrier \(V\) and \(u E: u: E\) shows \(u: \operatorname{span} E\) unfolding span-def
proof (rule,intro exI conjI)
show \(u=\) lincomb ( \(\lambda\)-. one \(K\) ) \(\{u\}\) unfolding lincomb-def using \(u E E\) by auto show \(\{u\} \subseteq E\) using \(u E\) by auto
```


## lemma lincomb-distrib:

## assumes $U: U \subseteq$ carrier $V$

and $a: a: U \rightarrow$ carrier $K$
and $c: c:$ carrier $K$
shows $c \odot_{V}$ lincomb a $U=\operatorname{lincomb}\left(\lambda u . c \otimes_{K} a u\right) U$
(is $-=$ lincomb ?b $U$ )
using $U a$
proof (induct $U$ rule: infinite-finite-induct)
case empty show ?case unfolding lincomb-def using $c$ by simp next
case (insert u $U$ )
then have $U: U \subseteq$ carrier $V$
and $u: u$ : carrier $V$
and $a: a$ : insert $u U \rightarrow$ carrier $K$
and fin $U$ : finite $U$ by auto
hence $a U: a: U \rightarrow$ carrier $K$ by auto
have $b$ : ?b : insert $u ~ U \rightarrow$ carrier $K$ using $a c$ by auto
show ?case
unfolding lincomb-insert2[OF fin $U$ U $a \prec u \notin U \succ u$ ]
unfolding lincomb-insert2[OF finU $U b\langle u \notin U\rangle u$ ]
using insert $U$ aU c u smult-r-distr smult-assoc1 by auto next
case (infinite $U$ )
thus ?case unfolding lincomb-def using assms by simp
qed
lemma span-swap:
assumes finE[simp]: finite $E$
and $E[$ simp $]: E \subseteq$ carrier $V$
and $u[$ simp $]: u:$ carrier $V$
and usE: $u \notin \operatorname{span} E$
and $v[$ simp $]$ : $v$ : carrier $V$
and usEv: $u: \operatorname{span}($ insert $v E)$
shows span $($ insert $u E) \subseteq \operatorname{span}($ insert $v E)($ is $? L \subseteq ? R)$
proof -
have $E u[$ simp $]$ : insert u $E \subseteq$ carrier $V$ by auto
have $E v[$ simp $]$ : insert $v E \subseteq$ carrier $V$ by auto
have finEu: finite (insert $u$ E) and finEv: finite (insert $v E$ )
using finE by auto
have $u E: u \notin E$ using usE span-mem by force
have $v E: v \notin E$
proof
assume $v: E$
hence EvE: insert v $E=E$ using insert-absorb by auto
hence $u$ : span $E$ using usEv by auto
thus False using usE by auto
qed
obtain ua
where $u a[$ simp $]: u a:($ insert $v E) \rightarrow$ carrier $K$
and $u u a: u=$ lincomb $u a$ (insert $v E$ )
using usEv finite-span $[O F$ finEv Ev] by auto
hence $u a E[\operatorname{simp}]: u a: E \rightarrow$ carrier $K$ and $[$ simp $]: u a v:$ carrier $K$
by auto

```
show ? \(L \subseteq ? R\)
proof
    fix \(x\) assume \(x\) : ? \(L\)
    then obtain \(x a\)
    where \(x a: x a\) : insert \(u E \rightarrow\) carrier \(K\)
    and \(x x a: x=\) lincomb \(x a\) (insert \(u E\) )
    unfolding finite-span[OF finEu Eu] by auto
    hence \(x a E[\operatorname{simp}]: x a: E \rightarrow\) carrier \(K\) and \(x a u[\operatorname{simp}]: x a u:\) carrier \(K\) by auto
    show \(x\) : span (insert \(v E\) )
    unfolding finite-span[OF finEv Ev]
    proof (rule,intro exI conjI)
    define \(a\) where \(a=\left(\lambda e . x a u \otimes_{K} u a e\right)\)
    define \(a^{\prime}\) where \(a^{\prime}=\left(\lambda e . a e \oplus_{K} x a e\right)\)
    define \(a^{\prime \prime}\) where \(a^{\prime \prime}=\left(\lambda e\right.\). if \(e=v\) then xa \(u \otimes_{K} u a v\) else \(\left.a^{\prime} e\right)\)
    have \(a E: a: E \rightarrow\) carrier \(K\) unfolding \(a\)-def using xau uaE \(u\) by blast
    hence \(a^{\prime} E: a^{\prime}: E \rightarrow\) carrier \(K\) unfolding \(a^{\prime}\)-def using \(x a E\) by blast
    thus \(a^{\prime \prime}: a^{\prime \prime}:\) insert \(v E \rightarrow\) carrier \(K\) unfolding \(a^{\prime \prime}\)-def by auto
    have restr: restrict \(a^{\prime} E=\) restrict \(a^{\prime \prime} E\)
            unfolding \(a^{\prime \prime}\)-def using \(v E\) by auto
    have \(x=x a u \odot_{V} u \oplus_{V}\) lincomb xa \(E\)
        using xxa lincomb-insert2 finE E xa \(u E u\) by auto
    also have
        \(x a u \odot_{V} u=x a u \odot_{V}\) lincomb \(u a(\) insert \(v E)\)
        using uua by auto
    also have lincomb ua (insert \(v E)=u a v \odot_{V} v \oplus_{V}\) lincomb ua \(E\)
        using lincomb-insert2 finE E ua vEv by auto
    also have \(x a u \odot_{V} \ldots=x a u \odot_{V}\left(u a v \odot_{V} v\right) \oplus_{V} x a u \odot_{V}\) lincomb ua \(E\)
        using smult-r-distr by auto
    also have \(x a u \odot_{V}\) lincomb ua \(E=\) lincomb a \(E\)
        unfolding \(a\)-def using lincomb-distrib \([O F E]\) by auto
    finally have \(x=x a u \odot_{V}\left(u a v \odot_{V} v\right) \oplus_{V}\left(\right.\) lincomb a \(E \oplus_{V}\) lincomb xa \(\left.E\right)\)
        using a-assoc xau vaE xaE by auto
    also have lincomb a \(E \oplus_{V}\) lincomb xa \(E=\) lincomb \(a^{\prime} E\)
        unfolding \(a^{\prime}\)-def using lincomb-sum \([\) OF finE E aE xaE]..
    also have \(\ldots=\) lincomb \(a^{\prime \prime} E\)
        using lincomb-restrict \(\left[O F E a^{\prime} E\right]\) restr by auto
    finally have \(x=\left(\left(x a u \otimes_{K}\right.\right.\) ua \(\left.\left.v\right) \odot_{V} v\right) \oplus_{V}\) lincomb \(a^{\prime \prime} E\)
            using smult-assoc1 by auto
    also have \(x a u \otimes_{K} u a v=a^{\prime \prime} v\) unfolding \(a^{\prime \prime}\)-def by simp
    also have \(\left(a^{\prime \prime} v \odot_{V} v\right) \oplus_{V}\) lincomb \(a^{\prime \prime} E=\) lincomb \(a^{\prime \prime}\) (insert \(v E\) )
            using lincomb-insert2 \(\left[O F\right.\) finE \(\left.E a^{\prime \prime} v E\right]\) by auto
    finally show \(x=\) lincomb \(a^{\prime \prime}(\) insert \(v E)\).
    qed
qed
```


## qed

## lemma basis-swap:

assumes finE[simp]: finite $E$
and $u[$ simp $]: u$ : carrier $V$
and $u E[\operatorname{simp}]: u \notin E$
and $b$ : basis (insert $u E$ )
and $v[$ simp $]$ : $v:$ carrier $V$
and $u E v: u:$ span (insert $v E)$
shows basis (insert v E)
unfolding basis-def
proof (intro conjI)
have $E u[$ simp $]$ : insert $u E \subseteq$ carrier $V$
and spanEu: carrier $V=\operatorname{span}$ (insert $u E$ )
and indEu: ~ lin-dep (insert u E)
using $b[$ unfolded basis-def] by auto
hence $E[s i m p]: E \subseteq$ carrier $V$ by auto
thus $E v[$ simp $]$ : insert $v E \subseteq$ carrier $V$ using $v$ by auto
have finEu: finite (insert u E)
and finEv: finite (insert $v E$ ) using finE by auto
have usE: $u \notin \operatorname{span} E$
proof
assume $u$ : span $E$
hence $u$ : span (insert $u E-\{u\}$ ) using $u E$ by auto
moreover have $u$ : insert $u E$ by auto
ultimately have lin-dep (insert u E)
unfolding lindep-span[OF Eu finEu] by auto
thus False using $b$ unfolding basis-def by auto
qed
obtain $u a$
where $u a[$ simp $]: u a:$ insert $v E \rightarrow$ carrier $K$ and uua: $u=$ lincomb ua (insert v $E$ )
using $u E v$ finite-span $[O F$ finEv Ev] by auto
hence $u a E[\operatorname{simp}]: u a: E \rightarrow$ carrier $K$
and uav[simp]: ua $v$ : carrier $K$
by auto
have $v E$ : $v \notin E$
proof
assume $v: E$
hence EvE: insert v $E=E$ using insert-absorb by auto
hence $u$ : span $E$ using $u E v$ by auto
thus False using usE by auto
qed
have $v s E: v \notin \operatorname{span} E$
proof
assume $v: \operatorname{span} E$
then obtain $v a$
where va[simp]: va: $E \rightarrow$ carrier $K$
and vva: $v=$ lincomb va $E$
using finite-span [OF finE E] by auto
define $v a^{\prime}$ where $v a^{\prime}=\left(\lambda e\right.$. ua $\left.v \otimes_{K} v a e\right)$
define $v a^{\prime \prime}$ where $v a^{\prime \prime}=\left(\lambda e . v a^{\prime} e \oplus_{K} u a e\right)$
have $v a^{\prime \prime}[$ simp $]: v a^{\prime}: E \rightarrow$ carrier $K$
unfolding $v a^{\prime}$-def using uav va by blast
hence $v a^{\prime \prime}[$ simp $]: v a^{\prime \prime}: E \rightarrow$ carrier $K$
unfolding $v a^{\prime \prime}$-def using $u a$ by blast
from uua
have $u=u a v \odot_{V} v \oplus_{V}$ lincomb ua $E$
using lincomb-insert2 $v E$ by auto
also have $u a v \odot_{V} v=u a v \odot_{V}($ lincomb va $E)$
using vva by auto
also have $\ldots=$ lincomb $v a^{\prime} E$
unfolding $v a^{\prime}$-def using lincomb-distrib by auto
finally have $u=$ lincomb $v a^{\prime \prime} E$
unfolding $v a^{\prime \prime}$-def
using lincomb-sum $[O F$ finE $E]$ by auto
hence $u$ : span $E$ using finite-span $[O F$ finE $E] v a^{\prime \prime}$ by blast
hence lin-dep (insert $u E$ ) using lindep-span by simp
then show False using indEu by auto
qed
have indE: ~ lin-dep E using indEu subset-li-is-li by auto
show $\sim$ lin-dep (insert $v E)$ using lin-dep-iff-in-span[OF E indE $v v E] v s E$ by auto

```
show span \((\) insert \(v E)=\) carrier \(V(\) is \(? L=? R)\)
proof (rule)
    show ? \(L \subseteq ? R\)
    proof
        have finEv: finite (insert \(v E\) ) using finE by auto
        fix \(x\) assume \(x\) : ? \(L\)
        then obtain \(a\)
            where \(a: a:\) insert \(v E \rightarrow\) carrier \(K\)
            and \(x: x=\) lincomb \(a(\) insert \(v E)\)
            unfolding finite-span [OF finEv Ev] by auto
        from \(a\) have av: a \(v\) : carrier \(K\) by auto
        from \(a\) have \(a 2: a: E \rightarrow\) carrier \(K\) by auto
        show \(x\) : ? \(R\)
            unfolding \(x\)
            unfolding lincomb-insert2[OF finE E a vE v]
            using lincomb-closed[OF E a2] av v
            by auto
    qed
    show ? \(R \subseteq ? L\)
        using span-swap[OF finE Eu usEvuEv] spanEu by auto
```

```
    qed
qed
lemma span-empty: span {} = {zero V}
    unfolding finite-span[OF finite.emptyI empty-subsetI]
    unfolding lincomb-def by simp
lemma span-self: assumes [simp]:v:carrier V shows v: span {v}
proof -
    have v}=\mathrm{ lincomb ( }\lambdax\mathrm{ . one K) {v} unfolding lincomb-def by auto
    thus ?thesis using finite-span by auto
qed
lemma span-zero:zero V : span U unfolding span-def lincomb-def by force
definition emb where emb fD x = (if x : D then f x else zero K)
lemma emb-carrier[simp]: f:D->R\Longrightarrowemb fD:D}->
    apply rule unfolding emb-def by auto
lemma emb-restrict: restrict (emb f D) D = restrict f D
    apply rule unfolding restrict-def emb-def by auto
lemma emb-zero: emb fD:X - D }->{\mathrm{ zero K }
    apply rule unfolding emb-def by auto
lemma lincomb-clean:
    assumes A: A\subseteqcarrier V
    and Z:Z\subseteqcarrier V
    and finA: finite A
    and finZ: finite Z
    and aA:a:A-> carrier K
    and aZ:a:Z 
    shows lincomb a (A\cupZ)= lincomb a A
    using finZ Z aZ
proof(induct set:finite)
    case empty thus ?case by simp next
    case (insert z Z) show ?case
        proof (cases z:A)
            case True hence }A\cup\mathrm{ insert z Z=A UZ by auto
                thus ?thesis using insert by simp next
                case False
                    have finAZ: finite ( }A\cupZ)\mathrm{ using finA insert by simp
                            have AZ:A\cupZ\subseteqcarrier V using A insert by simp
                have a: a : insert z (A\cupZ) -> carrier K using insert aA by force
                    have az= zero K using insert by auto
                    also have ... \odot }\mp@subsup{V}{}{z}=\mathrm{ zero V using insert by auto
                also have \ldots}\mp@subsup{\oplus}{V}{}\mathrm{ lincomb a (A UZ)= lincomb a (A GZ)
                    using insert AZ aA by auto
```

```
            also have ... = lincomb a A using insert by simp
            finally have az \odot V z }\mp@subsup{\oplus}{V}{}\mathrm{ lincomb a (A U Z)=lincomb a A.
            thus ?thesis
                using lincomb-insert2[OF finAZ AZ a] False insert by auto
    qed
qed
lemma span-add1:
    assumes U:U\subseteqcarrier V and v:v:span U and w:w:span U
    shows v}\mp@subsup{\oplus}{V}{}w:\operatorname{span}
proof -
    from v obtain a A
        where finA: finite }
            and va: lincomb a }A=
            and }AU:A\subseteq
            and a:a:A carrier K
            unfolding span-def by auto
    hence A:A\subseteq carrier V using U by auto
    from w obtain b B
            where finB: finite B
            and wb: lincomb b B =w
            and BU:B\subseteqU
            and b:b:B-> carrier K
            unfolding span-def by auto
    hence B: B\subseteq carrier V using U by auto
    have B-A:B-A\subseteqcarrier V and A-B:A - B\subseteqcarrier V using A B by
auto
    have a': emb a A:A\cupB-> carrier K
        apply (rule Pi-I) unfolding emb-def using a by auto
    hence a'A: emb a A : A }->\mathrm{ carrier K by auto
    have a'Z: emb a A : B - A }->{\mathrm{ zero K }
        apply (rule Pi-I) unfolding emb-def using a by auto
    have b': emb b B:A\cupB->carrier K
        apply (rule Pi-I) unfolding emb-def using b by auto
    hence \mp@subsup{b}{}{\prime}B: emb b B:B->carrier K by auto
    have b'Z: emb b B:A - B }->\mathrm{ {zero K }
        apply (rule Pi-I) unfolding emb-def using b by auto
    show ?thesis
    unfolding span-def
    proof (rule,intro exI conjI)
        let ?v = lincomb (emb a A) (A\cupB)
        let ?w = lincomb (emb b B) (A\cupB)
        let ?ab = \lambdau. (emb a A) u \oplus 
        show finAB: finite ( }A\cupB)\mathrm{ using finA finB by auto
        show }A\cupB\subseteqU\mathrm{ using }AUBU\mathrm{ by auto
```

```
        show ?ab:A\cupB->carrier K using a' b' by auto
        have v=?v
            using va lincomb-restrict[OF A a emb-restrict[symmetric]]
            using lincomb-clean[OF A B-A] a'A a'Z finA finB by simp
            moreover have w=?w
            apply (subst Un-commute)
            using wb lincomb-restrict[OF B b emb-restrict[symmetric]]
            using lincomb-clean[OF B A-B] finA finB b}\mp@subsup{b}{}{\prime}B\mp@subsup{b}{}{\prime}Z\mathrm{ by simp
            ultimately show v}\mp@subsup{\oplus}{V}{}w=lincomb ?ab (A\cupB
            using lincomb-sum[OF finAB] A B a' b' by simp
        qed
qed
lemma span-neg:
    assumes U:U\subseteqcarrier V and vU:v: span U
    shows }\mp@subsup{\ominus}{V}{}v:\mathrm{ span }
proof -
    have v:v:carrier V using vU U unfolding span-def by auto
    from }vU[unfolded span-def
    obtain a A
        where finA: finite A
            and }AU:A\subseteq
            and a:a\inA-> carrier K
            and va:v= lincomb a A by auto
    hence A: A\subseteq carrier }V\mathrm{ using }U\mathrm{ by simp
    let ?a = \lambdau.}\mp@subsup{\ominus}{K}{}\mathrm{ one }K\mp@subsup{\otimes}{K}{}a
    have }\mp@subsup{\ominus}{V}{}v=\mp@subsup{\ominus}{K}{}\mathrm{ one K }\mp@subsup{\odot}{V}{}v\mathrm{ using smult-minus-1-back[OF v].
    also have ... = Ө}\mp@subsup{\ominus}{K}{}\mathrm{ one K © V lincomb a A using va by simp
    finally have main: }\mp@subsup{\ominus}{V}{}v=lincomb ?a 
        unfolding lincomb-distrib[OF A a R.a-inv-closed[OF R.one-closed]] by auto
    show ?thesis
        unfolding span-def
        apply rule
        using main a finA AU by force
qed
lemma span-closed[simp]:U\subseteq carrier V\Longrightarrowv: span U\Longrightarrowv:carrier }
    unfolding span-def by auto
lemma span-add:
    assumes U:U\subseteqcarrier V and vU:v: span U and w[simp]:w: carrier V
    shows }w:\operatorname{span}U\longleftrightarrowv\mp@subsup{\oplus}{V}{}w:\operatorname{span}U(\mathrm{ is ? L }\longleftrightarrow?R
proof
    show ?L \Longrightarrow?R using span-add1[OF U vU] by auto
    assume R: ?R show ?L
    proof -
    have v[simp]: v : carrier V using vU U by simp
    have w=zero V }\mp@subsup{\oplus}{V}{}w\mathrm{ using M.l-zero by auto
```

```
    also have \ldots. = \ominus}\mp@subsup{\ominus}{V}{}v\mp@subsup{\oplus}{V}{}v\mp@subsup{\oplus}{V}{}w\mathrm{ using M.l-neg by auto
    also have ... = \ominus}\mp@subsup{V}{V}{}v\mp@subsup{\oplus}{V}{}(v\mp@subsup{\oplus}{V}{}w
            using M.l-zero M.a-assoc M.a-closed by auto
    also have ... : span U using span-neg[OF U vU] span-add1[OF U] R by auto
    finally show ?thesis.
    qed
qed
lemma lincomb-union:
    assumes U:U\subseteq carrier V
    and U'[simp]: U'` carrier V
    and disj: U\cap U'={}
    and finU: finite }
    and finU': finite }\mp@subsup{U}{}{\prime
    and a:a:U\cupU'}->\mathrm{ carrier K
    shows lincomb a (U\cupU')= lincomb a U \oplus \ lincomb a U'
    using finU U disj a
proof (induct set:finite)
    case empty thus ?case by (subst(2) lincomb-def, simp) next
    case (insert u U) thus ?case
        unfolding Un-insert-left
        using lincomb-insert2 finU' insert a-assoc by auto
qed
lemma span-union1:
    assumes U:U\subseteqcarrier V and U': U'\subseteqcarrier V and UU': span U = span
U'
    and W:W\subseteqcarrier V and W': W' W}\subseteq\mathrm{ carrier V and W W': span W =
span W'
    shows span }(U\cupW)\subseteq\operatorname{span}(\mp@subsup{U}{}{\prime}\cup\mp@subsup{W}{}{\prime})(\mathrm{ is ? }L\subseteq?R
proof
    fix x assume x:?L
    then obtain a A
        where finA: finite A
            and AUW:A\subseteqU\cupW
            and x:x= lincomb a A
            and a:a:A carrier K
    unfolding span-def by auto
    let ?}AU=A\capU\mathrm{ and ?}AW=A\capW-
    have AU: ?AU\subseteq carrier }V\mathrm{ using }U\mathrm{ by auto
    have AW: ?AW\subseteqcarrier V using W by auto
    have disj:?AU \cap?AW = {} by auto
    have }\mp@subsup{U}{}{\prime}\mp@subsup{W}{}{\prime}:\mp@subsup{U}{}{\prime}\cup\mp@subsup{W}{}{\prime}\subseteq\mathrm{ carrier }V\mathrm{ using }\mp@subsup{U}{}{\prime}\mp@subsup{W}{}{\prime}\mathrm{ by auto
    have ?AU\cup?AW = A using AUW by auto
    hence }x=\mathrm{ lincomb a (?AU U?AW) using }x\mathrm{ by auto
    hence }x=\mathrm{ lincomb a ?AU }\mp@subsup{\oplus}{V}{}\mathrm{ lincomb a ?AW
        using lincomb-union[OF AU AW disj] finA a by auto
```

```
    moreover
    have lincomb a ?AU : span U and lincomb a ?AW : span W
        unfolding span-def using AU a finA by auto
    hence lincomb a ?AU : span U' and lincomb a ?AW : span W'
        using UU' WW'' by auto
    hence lincomb a ?AU :?R and lincomb a ?AW :?R
        using span-is-monotone[OF Un-upper1, of U ]
        using span-is-monotone[OF Un-upper2, of W W by auto
    ultimately
    show x : ?R using span-add1[OF U'W ] by auto
qed
lemma span-Un:
    assumes U:U\subseteqcarrier V and U':}\mp@subsup{U}{}{\prime}\subseteq\mathrm{ carrier }V\mathrm{ and }U\mp@subsup{U}{}{\prime}: span U = span
U'
        and W:W\subseteqcarrier V and W': W' \subseteqcarrier V and WW': span W =
span W'
    shows span (U\cupW)=\operatorname{span}(\mp@subsup{U}{}{\prime}\cup\mp@subsup{W}{}{\prime})(\mathrm{ is ? L = ? R)}
    using span-union1[OF assms]
    using span-union1[OF U' U UU'[symmetric] W' W WW'[symmetric]]
    by auto
lemma lincomb-zero:
    assumes }U:U\subseteq\mathrm{ carrier }V\mathrm{ and }a:a:U->{\mathrm{ zero K}
    shows lincomb a }U=\mathrm{ zero }
    using U a
proof (induct U rule: infinite-finite-induct)
    case empty show ?case unfolding lincomb-def by auto next
    case (insert u U)
        hence }a\in\mathrm{ insert }u|->\mathrm{ carrier K using zero-closed by force
    thus ?case using insert by (subst lincomb-insert2; auto)
qed (auto simp: lincomb-def)
end
context module
begin
lemma lincomb-empty[simp]: lincomb a {}=\mp@subsup{\mathbf{0}}{M}{}
    unfolding lincomb-def by auto
end
context linear-map
begin
interpretation Ker: vectorspace K (V.vs kerT)
    using kerT-is-subspace
    using V.subspace-is-vs by blast
```

```
interpretation im : vectorspace \(K\) ( \(W\).vs \(i m T\) )
    using imT-is-subspace
    using W.subspace-is-vs by blast
lemma inj-imp-Ker0:
assumes inj-on \(T\) (carrier \(V\) )
shows carrier \((V\).vs ker \(T)=\left\{\mathbf{0}_{V}\right\}\)
    unfolding mod-hom.ker-def
    using assms inj-on-contraD by fastforce
lemma Ke0-imp-inj:
assumes \(c\) : carrier ( \(V\).vs ker \(T\) ) \(=\left\{\mathbf{0}_{V}\right\}\)
shows inj-on \(T\) (carrier \(V\) )
proof (auto simp add: inj-on-def)
    fix \(x y\)
    assume \(x: x \in\) carrier \(V\) and \(y: y \in\) carrier \(V\)
    and \(T x-T y: T x=T y\)
    hence \(T x \ominus_{W} T y=\mathbf{0}_{W}\) using \(W\).module.M.minus-other-side by auto
    hence \(T\left(x \ominus_{V} y\right)=\mathbf{0}_{W}\) by (simp add: \(x y\) )
    hence \(x \ominus_{V} y \in\) carrier ( \(V . v s\) kerT) by (simp add: mod-hom.ker-def \(x\) y)
    hence \(x \ominus_{V} y=\mathbf{0}_{V}\) using \(c\) by fast
    thus \(x=y\) by (simp add: \(x y\) )
qed
corollary Ke0-iff-inj: inj-on \(T\) (carrier \(V)=\left(\right.\) carrier \(\left.(V . v s \operatorname{ker} T)=\left\{\mathbf{0}_{V}\right\}\right)\)
using inj-imp-Ker0 Ke0-imp-inj by auto
lemma inj-imp-dim-ker0:
assumes inj-on \(T\) (carrier \(V\) )
shows vectorspace.dim \(K(V . v s \operatorname{ker} T)=0\)
proof (unfold Ker.dim-def, rule Least-eq-0, rule exI[of - \{\}])
    have Ker-rw: carrier (V.vs ker \(T\) ) \(=\left\{\mathbf{0}_{V}\right\}\)
        unfolding mod-hom.ker-def
        using assms inj-on-contraD by fastforce
    have finite \(\}\) by simp
    moreover have card \(\}=0\) by simp
    moreover have \(\} \subseteq\) carrier ( V .vs kerT) by simp
    moreover have Ker.gen-set \{\} unfolding Ker-rw by (simp add: Ker.span-empty)
        ultimately show finite \(\} \wedge\) card \(\}=0 \wedge\{ \} \subseteq \operatorname{carrier}(V . v s \operatorname{ker} T) \wedge\)
Ker.gen-set \(\}\) by simp
qed
```

lemma surj-imp-imT-carrier:
assumes surj: $T^{\prime}($ carrier $V)=$ carrier $W$
shows $($ imT $)=$ carrier $W$
by (simp add: surj im-def)

```
lemma dim-eq:
assumes fin-dim-V:V.fin-dim
and i: inj-on T (carrier V) and surj: T' (carrier V) = carrier W
shows V.dim = W.dim
proof -
    have dim0: vectorspace.dim K (V.vs kerT) = 0
        by (rule inj-imp-dim-kerO[OF i])
    have imT-W: (imT) = carrier W
        by (rule surj-imp-imT-carrier[OF surj])
    have rnt: vectorspace.dim K (W.vs imT) + vectorspace.dim K (V.vs kerT) =
V.dim
    by (rule rank-nullity[OF fin-dim-V])
    hence V.dim = vectorspace.dim K (W.vs imT) using dim0 by auto
    also have ... = W.dim using imT-W by auto
    finally show ?thesis using fin-dim-V by auto
qed
lemma lincomb-linear-image:
assumes inj-T: inj-on \(T\) (carrier \(V\) )
assumes A-in-V:A\subseteqcarrier V and a: a\in(T'A) }->\mathrm{ carrier K
assumes f: finite }
shows W.module.lincomb a (T'A) =T (V.module.lincomb (a\circT) A)
using f using A-in-V a
proof (induct A)
    case empty thus ?case by auto
next
    case (insert x A)
    have T-insert-rw: T`'insert x A = insert (T x) (T'A) by simp
    have W.module.lincomb a (T`insert x A ) = W.module.lincomb a (insert (T x)
(T'A))
        unfolding T-insert-rw ..
    also have ... = a (Tx) \odot W
    proof (rule W.lincomb-insert2)
        show finite (T'A) by (simp add: insert.hyps(1))
        show T'A\subseteqcarrier W using insert.prems(1) by auto
        show a \in insert (Tx) (T'`A) -> carrier K
            using insert.prems(2) by blast
        show Tx\not\inT``A
            by (meson inj-T inj-on-image-mem-iff insert.hyps(2) insert.prems(1) in-
sert-subset)
        show T x carrier W using insert.prems(1) by blast
    qed
    also have ... =a(Tx) \odot W (Tx) \oplus W (T (V.module.lincomb ( a\circT)A))
        using insert.hyps(3) insert.prems(1) insert.prems(2) by fastforce
    also have ... =T(a(Tx) \odot V x) \oplus}\mp@subsup{W}{W}{}(T(V.module.lincomb (a\circT)A))
        using insert.prems(1) insert.prems(2) by auto
    also have ... =T ((a(T x) \odot V x) \oplus}\mp@subsup{V}{V}{}(V.module.lincomb (a\circT)A)
    proof (rule T-add[symmetric])
```

```
    show a (T x) \odot V x carrier V using insert.prems(1) insert.prems(2) by
auto
    show V.module.lincomb ( }a\circT)A\in\mathrm{ carrier V
    proof (rule V.module.lincomb-closed)
        show A\subseteq carrier V using insert.prems(1) by blast
        show }a\circT\inA->\mathrm{ carrier K using coeff-in-ring insert.prems(2) by auto
    qed
    qed
    also have ... =T(V.module.lincomb ( }a\circT)(\mathrm{ insert x A )}
    proof (rule arg-cong[of - - T])
    have a\circT\in insert x A C carrier K
        using comp-def insert.prems(2) by auto
    then show }a(Tx)\mp@subsup{\odot}{V}{}x\mp@subsup{\oplus}{V}{}V.module.lincomb ( a\circT)
            =V.module.lincomb (a\circT) (insert x A)
            using V.lincomb-insert2 insert.hyps(1) insert.hyps(2) insert.prems(1) by
force
    qed
    finally show ?case .
qed
```

lemma surj-fin-dim:
assumes fd: V.fin-dim and surj: $T^{6}($ carrier $V)=$ carrier $W$
shows image-fin-dim: W.fin-dim
using rank-nullity-main(2)[OF fd surj].
lemma linear-inj-image-is-basis:
assumes inj-T: inj-on $T($ carrier $V)$ and surj: $T^{\prime}($ carrier $V)=$ carrier $W$
and basis-B: V.basis $B$
and fin-dim- $V$ : V.fin-dim
shows W.basis ( $T^{\prime} B$ )
proof (rule W.dim-li-is-basis)
have $l m$ : linear-map $K V W T$ by intro-locales
have inj-TB: inj-on $T B$
by (meson basis-B inj-T subset-inj-on V.basis-def)
show W.fin-dim by (rule surj-fin-dim[OF fin-dim-V surj])
show finite ( $T$ ' $B$ )
proof (rule finite-imageI, rule V.fin $[O F$ fin-dim- $V$ ])
show V.module.lin-indpt $B$ using basis- $B$ unfolding $V$.basis-def by auto
show $B \subseteq$ carrier $V$ using basis- $B$ unfolding $V$.basis-def by auto
qed
show $T$ ' $B \subseteq$ carrier $W$ using basis- $B$ unfolding $V$.basis-def by auto
show $W \cdot \operatorname{dim} \leq \operatorname{card}\left(T^{\prime} B\right)$
proof -
have $d: V . \operatorname{dim}=W . \operatorname{dim}$ by (rule dim-eq[OF fin-dim-V inj-T surj])
have card $\left(T^{\prime} B\right)=$ card $B$ by ( simp add: card-image inj-TB)
also have $\ldots=V$.dim using basis-B fin-dim-V V.basis-def V.dim-basis $V$.fin by auto

```
    finally show ?thesis using \(d\) by simp
    qed
    show \(W\).module.lin-indpt ( \(T\) ‘ \(B\) )
    proof (rule W.module.finite-lin-indpt2)
    show fin-TB: finite \(\left(T^{\prime} B\right)\) by fact
    show \(T B-W: T\) ' \(B \subseteq\) carrier \(W\) by fact
    fix \(a\) assume \(a: a \in T\) ' \(B \rightarrow\) carrier \(K\) and \(l c-a\) : \(W\).module.lincomb \(a\) ( \(T\)
\(B)=\mathbf{0}_{W}\)
    show \(\forall v \in T\) ' \(B . a v=\mathbf{0}_{K}\)
    proof (rule ballI)
    fix \(v\) assume \(v: v \in T\) ' \(B\)
    have \(W\).module.lincomb \(a(T\) ' \(B)=T\) (V.module.lincomb \((a \circ T) B)\)
    proof (rule lincomb-linear-image \([O F\) inj-T])
    show \(B \subseteq\) carrier \(V\) using \(V\).vectorspace-axioms basis- \(B\) vectorspace.basis-def
by blast
            show \(a \in T^{\prime} B \rightarrow\) carrier \(K\) by (simp add: a)
            show finite \(B\) using fin-TB finite-image-iff inj-TB by blast
        qed
            hence \(T\)-lincomb: \(T\) (V.module.lincomb \((a \circ T) B)=\mathbf{0}_{W}\) using \(l c-a\) by
simp
    have lincomb-0: V.module.lincomb \((a \circ T) B=\mathbf{0}_{V}\)
        proof -
            have \(a \circ T \in B \rightarrow\) carrier \(K\)
            using \(a\) by auto
            then show ?thesis
                by (metis V.module.M.zero-closed V.module.lincomb-closed
                    T-lincomb basis-B f0-is-0 inj-T inj-onD V.basis-def)
    qed
    have \((a \circ T) \in B \rightarrow\left\{\mathbf{0}_{K}\right\}\)
    proof (rule V.not-lindep \(D[O F-\) - - lincomb-0])
            show V.module.lin-indpt \(B\) using \(V\).basis-def basis- \(B\) by blast
            show finite \(B\) using fin-TB finite-image-iff inj-TB by auto
            show \(B \subseteq B\) by auto
            show \(a \circ T \in B \rightarrow\) carrier \(K\) using \(a\) by auto
            qed
            thus \(a v=\mathbf{0}_{K}\) using \(v\) by auto
    qed
    qed
qed
end
lemma (in vectorspace) dim1I:
assumes gen-set \(\{v\}\)
assumes \(v \neq \mathbf{0}_{V} v \in\) carrier \(V\)
shows \(\operatorname{dim}=1\)
proof -
    have basis \(\{v\}\) by (metis assms(1) assms(2) assms(3) basis-def empty-iff empty-subsetI
    finite.emptyI finite-lin-indpt2 insert-iff insert-subset insert-union lin-dep-iff-in-span
```

```
    span-empty)
    then show ?thesis using dim-basis by force
qed
lemma (in vectorspace) dim0I:
assumes gen-set {0}\mp@subsup{0}{V}{}
shows dim = 0
proof -
    have basis {} unfolding basis-def using already-in-span assms finite-lin-indpt2
span-zero by auto
    then show ?thesis using dim-basis by force
qed
lemma (in vectorspace) dim-le1I:
assumes gen-set {v}
assumes v\in carrier V
shows dim \leq 1
by (metis One-nat-def assms(1) assms(2) bot.extremum card.empty card.insert
empty-iff finite.intros(1)
finite.intros(2) insert-subset vectorspace.gen-ge-dim vectorspace-axioms)
definition find-indices where find-indices x xs \equiv[i\leftarrow[0..<length xs]. xs!i=x]
lemma find-indices-Nil [simp]:
    find-indices x [] = []
    by (simp add: find-indices-def)
lemma find-indices-Cons:
    find-indices x (y#ys)=(if x=y then Cons 0 else id) (map Suc (find-indices x
ys))
apply (unfold find-indices-def length-Cons, subst upt-conv-Cons, simp)
apply (fold map-Suc-upt, auto simp: filter-map o-def) done
lemma find-indices-snoc [simp]:
    find-indices x (ys@[y])= find-indices x ys @ (if x=y then [length ys] else [])
    by (unfold find-indices-def, auto intro!: filter-cong simp: nth-append)
lemma mem-set-find-indices [simp]: i set (find-indices x xs)\longleftrightarrow \longleftrightarrow < length xs
\wedges!!i=x
    by (auto simp: find-indices-def)
lemma distinct-find-indices: distinct (find-indices x xs)
    unfolding find-indices-def by simp
context abelian-monoid begin
definition sumlist
    where sumlist xs \equiv foldr ( }\oplus\mathrm{ ) xs 0
```

```
lemma [simp]:
    shows sumlist-Cons: sumlist (x#xs)=x\oplus sumlist xs
        and sumlist-Nil: sumlist [] = 0
    by (simp-all add: sumlist-def)
lemma sumlist-carrier [simp]:
    assumes set xs \subseteqcarrier G shows sumlist xs \in carrier G
    using assms by (induct xs, auto)
lemma sumlist-neutral:
    assumes set xs\subseteq{0} shows sumlist xs = 0
proof (insert assms, induct xs)
    case (Cons x xs)
    then have }x=\mathbf{0}\mathrm{ and set xs }\subseteq{\mathbf{0}}\mathrm{ by auto
    with Cons.hyps show ?case by auto
qed simp
lemma sumlist-append:
    assumes set xs\subseteqcarrier G and set ys\subseteqcarrier G
    shows sumlist (xs @ ys) = sumlist xs \oplus sumlist ys
proof (insert assms, induct xs arbitrary: ys)
    case (Cons x xs)
    have sumlist (xs @ ys)=sumlist xs \oplus sumlist ys
        using Cons.prems by (auto intro: Cons.hyps)
    with Cons.prems show ?case by (auto intro!: a-assoc[symmetric])
qed auto
lemma sumlist-snoc:
    assumes set xs \subseteqcarrier G and x\in carrier G
    shows sumlist (xs @ [x]) = sumlist xs }\oplus
    by (subst sumlist-append, insert assms, auto)
lemma sumlist-as-finsum:
    assumes set xs\subseteqcarrier G and distinct xs shows sumlist xs =(\bigoplusx\inset xs. x)
    using assms by (induct xs, auto intro:finsum-insert[symmetric])
lemma sumlist-map-as-finsum:
    assumes f: set xs -> carrier G and distinct xs
    shows sumlist (map fxs)=(\bigoplusx\in set xs. f x )
    using assms by (induct xs, auto)
definition summset where summset M\equiv fold-mset ( }\oplus\mathrm{ ) 0 M
lemma summset-empty [simp]: summset {#} = 0 by (simp add: summset-def)
lemma fold-mset-add-carrier: a carrier }G\Longrightarrow\mathrm{ set-mset M}\subseteq\mathrm{ carrier }G
fold-mset ( }\oplus\mathrm{ ) a M E carrier G
proof (induct M arbitrary:a)
```

```
case (add x M)
thus ?case by
    (subst comp-fun-commute-on.fold-mset-add-mset[of - carrier G], unfold-locales,
auto simp: a-lcomm)
qed simp
lemma summset-carrier[intro]: set-mset M\subseteq carrier G\Longrightarrow summset M c carrier
G
    unfolding summset-def by (rule fold-mset-add-carrier, auto)
lemma summset-add-mset[simp]:
    assumes a:a\in carrier G and MG: set-mset M\subseteq carrier G
    shows summset (add-mset a M)=a\oplus summset M
    using assms
    by (auto simp add: summset-def)
    (rule comp-fun-commute-on.fold-mset-add-mset, unfold-locales, auto simp add:
a-lcomm)
lemma sumlist-as-summset:
    assumes set xs\subseteq carrier G shows sumlist xs = summset (mset xs)
    by (insert assms, induct xs, auto)
lemma sumlist-rev:
    assumes set xs \subseteqcarrier G
    shows sumlist (rev xs) = sumlist xs
    using assms by (simp add: sumlist-as-summset)
lemma sumlist-as-fold:
    assumes set xs \subseteqcarrier G
    shows sumlist xs = fold ( }\oplus\mathrm{ ) xs 0
    by (fold sumlist-rev[OF assms], simp add: sumlist-def foldr-conv-fold)
end
context Module.module begin
definition lincomb-list
where lincomb-list c vs = sumlist (map (\lambdai.c i }\mp@subsup{\odot}{M}{}\mathrm{ vs ! i) [0..<length vs])
lemma lincomb-list-carrier:
    assumes set vs\subseteqcarrier M and c:{0..<length vs} }->\mathrm{ carrier R
    shows lincomb-list c vs }\in\mathrm{ carrier M
    by (insert assms, unfold lincomb-list-def, intro sumlist-carrier, auto intro!: smult-closed)
lemma lincomb-list-Nil [simp]: lincomb-list c [] = 0}\mp@subsup{\mathbf{0}}{M}{
    by (simp add: lincomb-list-def)
lemma lincomb-list-Cons [simp]:
```



```
    by (unfold lincomb-list-def length-Cons, subst upt-conv-Cons, simp, fold map-Suc-upt,
simp add:o-def)
lemma lincomb-list-eq-0:
    assumes \bigwedgei.i< length vs \Longrightarrowci}\mp@subsup{\odot}{M}{}vs!i=\mp@subsup{\mathbf{0}}{M}{
    shows lincomb-list c vs = 0}\mp@subsup{\mathbf{0}}{M}{
proof (insert assms, induct vs arbitrary:c)
    case (Cons v vs)
    from Cons.prems[of 0] have [simp]:c 0 \odot < v = 0
    from Cons.prems[of Suc -] Cons.hyps have lincomb-list (c\circSuc) vs = 0}\mp@subsup{\mathbf{0}}{M}{}\mathrm{ by
auto
    then show ?case by (simp add: o-def)
qed simp
definition mk-coeff where mk-coeff vs c v \equivR.sumlist (map c (find-indices v vs))
lemma mk-coeff-carrier:
    assumes c:{0..<length vs} }->\mathrm{ carrier R shows mk-coeff vs c w carrier R
    by (insert assms, auto simp: mk-coeff-def find-indices-def intro!:R.sumlist-carrier
elim!:funcset-mem)
lemma mk-coeff-Cons:
    assumes c:{0..<length (v#vs)} }->\mathrm{ carrier R
    shows mk-coeff (v#vs) c=(\lambdaw. (if w=v then c 0 else 0) }\oplusmk\mathrm{ -coeff vs (c o
Suc) w)
proof-
    from assms have c o Suc:{0..<length vs }}->\mathrm{ carrier R by auto
    from mk-coeff-carrier[OF this] assms
    show ?thesis by (auto simp add: mk-coeff-def find-indices-Cons)
qed
lemma mk-coeff-O[simp]:
    assumes v& set vs
    shows mk-coeff vs c v=\mathbf{0}
proof -
    have (find-indices v vs)=[] using assms unfolding find-indices-def
        by (simp add: in-set-conv-nth)
    thus ?thesis unfolding mk-coeff-def by auto
qed
lemma lincomb-list-as-lincomb:
    assumes vs-M: set vs\subseteqcarrier M and c:c:{0..<length vs }}->\mathrm{ carrier }
    shows lincomb-list c vs = lincomb (mk-coeff vs c) (set vs)
proof (insert assms, induct vs arbitrary: c)
    case (Cons v vs)
    have mk-coeff-Suc-closed: mk-coeff vs (c\circSuc) a carrier R for a
        apply (rule mk-coeff-carrier)
        using Cons.prems unfolding Pi-def by auto
    have x-in: x carrier M if x: x\in set vs for x using Cons.prems x by auto
```

```
    show ?case apply (unfold mk-coeff-Cons[OF Cons.prems(2)] lincomb-list-Cons)
    apply (subst Cons) using Cons apply (force, force)
    proof (cases v\in set vs,auto simp:insert-absorb)
    case False
    let ?f = (\lambdava. ((if va=v then c 0 else 0) }\oplusmk-coeff vs (c\circSuc) va) \odot < va)
    have mk-0: mk-coeff vs (c\circSuc) v=0 using False by auto
    have [simp]:(c0}0\oplus\mathbf{0})=c
        using Cons.prems(2) by force
    have finsum-rw: (\bigoplus Mva\ininsert v (set vs). ?f va) = (?f v) }\mp@subsup{\oplus}{M}{}(\mp@subsup{\bigoplus}{M}{
vs). ?f va)
    proof (rule finsum-insert, auto simp add: False, rule smult-closed, rule R.a-closed)
        fix }
        show mk-coeff vs (c\circSuc) x carrier R
        using mk-coeff-Suc-closed by auto
        show c 0 \odot }\mp@subsup{M}{}{v}v\in\mathrm{ carrier M
        proof (rule smult-closed)
        show c 0 \in carrier R
            using Cons.prems(2) by fastforce
        show v}\in\mathrm{ carrier M
            using Cons.prems(1) by auto
        qed
    show 0 }\in\mathrm{ carrier R
        by simp
        assume x: x\in set vs show }x\in\mathrm{ carrier M
        using Cons.prems(1) x by auto
    qed
    have finsum-rw2:
```



```
va)
    proof (rule finsum-cong2, auto simp add: False)
        fix i assume i:i\in set vs
        have coSuc }\in{0..<length vs } -> carrier R using Cons.prems by aut
        then have [simp]:mk-coeff vs (c\circSuc) i\in carrier R
            using mk-coeff-Suc-closed by auto
        have 0}\oplusmk\mathrm{ -coeff vs (c○Suc) i=mk-coeff vs (c○Suc) i by (rule R.l-zero,
simp)
        then show (0 \oplus mk-coeff vs (c\circSuc) i) \odot }\mp@subsup{M}{M}{}i=mk\mathrm{ -coeff vs (c○Suc)i }\mp@subsup{\odot}{M}{
i
        by auto
        show (0 }\oplusmk\mathrm{ -coeff vs (c ○Suc) i) }\mp@subsup{\odot}{M}{}i\in\mathrm{ carrier M
            using Cons.prems(1) i by auto
    qed
    show c 0 \odot }\mp@subsup{M}{}{v}v\mp@subsup{\oplus}{M}{}\mathrm{ lincomb (mk-coeff vs (c ○Suc)) (set vs)=
    lincomb ( }\lambdaa.(\mathrm{ if }a=v then c 0 else 0) \oplus mk-coeff vs (c\circSuc) a) (insert v (se
vs))
        unfolding lincomb-def
        unfolding finsum-rw mk-0
        unfolding finsum-rw2 by auto
next
```

case True
let ?f $=\lambda v a .(($ if $v a=v$ then c 0 else $\mathbf{0}) \oplus m k$-coeff $v s(c \circ S u c) v a) \odot_{M} v a$
have $r w:(c 0 \oplus m k$-coeff $v s(c \circ S u c) v) \odot_{M} v$
$=\left(\begin{array}{c}c\end{array} \odot_{M} v\right) \oplus_{M}(m k$-coeff $v s(c \circ S u c) v) \odot_{M} v$
using Cons.prems(1) Cons.prems(2) atLeast0-lessThan-Suc-eq-insert-0
using $m k$-coeff-Suc-closed smult-l-distr by auto
have rw2: $\left((m k\right.$-coeff $\left.v s(c \circ S u c) v) \odot_{M} v\right)$
$\oplus_{M}\left(\bigoplus_{M^{v a}}(\right.$ set vs $-\{v\})$. ?f $\left.v a\right)=\left(\bigoplus_{M^{v} \in \text { set vs. mk-coeff } v s(c \circ S u c) v}\right.$ $\left.\odot_{M} v\right)$
proof -
have $\left(\bigoplus_{M} v a \in(\right.$ set $v s-\{v\})$. ?f $\left.v a\right)=\left(\bigoplus_{M} v \in\right.$ set $v s-\{v\}$. mk-coeff vs $(c$

- Suc) $v \odot_{M} v$ )
by (rule finsum-cong2, unfold Pi-def, auto simp add: mk-coeff-Suc-closed
$x$-in)
moreover have $\left(\bigoplus_{M} v \in s e t ~ v s . m k\right.$-coeff $\left.v s(c \circ S u c) v \odot_{M} v\right)=((m k$-coeff vs $\left.(c \circ S u c) v) \odot_{M} v\right)$
$\oplus_{M}\left(\bigoplus_{M^{v}} \in\right.$ set vs $-\{v\} . m k$-coeff $\left.v s(c \circ S u c) v \odot_{M} v\right)$
by (rule M.add.finprod-split, auto simp add: mk-coeff-Suc-closed True $x$-in)
ultimately show ?thesis by auto
qed
have lincomb $(\lambda a$. (if $a=v$ then c 0 else $\mathbf{0}) \oplus m k$-coeff vs $(c \circ$ Suc) $)$ ) (set vs)
$=\left(\bigoplus_{M} v a \in s e t\right.$ vs. ?f va) unfolding lincomb-def ..
also have $\ldots=$ ?f $v \oplus_{M}\left(\bigoplus_{M} v a \in(\right.$ set $v s-\{v\})$. ?f $\left.v a\right)$
proof (rule M.add.finprod-split)
have c0-mkcoeff-in: $c 0 \oplus m k$-coeff vs $(c \circ$ Suc) $v \in \operatorname{carrier} R$
proof (rule R.a-closed)
show c $0 \in$ carrier $R$ using Cons.prems by auto
show $m k$-coeff vs $(c \circ S u c) v \in \operatorname{carrier} R$
using $m k$-coeff-Suc-closed by auto
qed
moreover have $(\mathbf{0} \oplus m k$-coeff vs $(c \circ S u c) v a) \odot_{M}$ va $\in \operatorname{carrier} M$
if $v a: v a \in$ carrier $M$ for $v a$
by (rule smult-closed[OF - va], rule R.a-closed, auto simp add: mk-coeff-Suc-closed)
ultimately show ?f 'set vs $\subseteq$ carrier $M$ using Cons.prems(1) by auto
show finite (set vs) by simp
show $v \in$ set $v s$ using True by simp
qed
also have $\ldots=(c 0 \oplus m k$-coeff vs $(c \circ S u c) v) \odot_{M} v$
$\oplus_{M}\left(\bigoplus_{M} v a \in(\right.$ set vs $-\{v\})$. ?f va) by auto
also have $\ldots=\left(\left(c r 0 \odot_{M} v\right) \oplus_{M}(m k\right.$-coeff vs $\left.(c \circ S u c) v) \odot_{M} v\right)$
$\oplus_{M}\left(\bigoplus_{M} v a \in(\right.$ set $v s-\{v\})$. ?f $\left.v a\right)$ unfolding $r w$ by simp
also have $\ldots=\left(\begin{array}{lll}c & 0 & \odot_{M} v\end{array}\right) \oplus_{M}\left(\left((m k\right.\right.$-coeff vs $(c \circ$ Suc $\left.) v) \odot_{M} v\right)$
$\oplus_{M}\left(\bigoplus_{M} v a \in(\right.$ set $v s-\{v\})$. ?f $\left.\left.v a\right)\right)$
proof (rule M.a-assoc)
show $c 0 \odot_{M} v \in \operatorname{carrier} M$
using Cons.prems(1) Cons.prems(2) by auto
show $m k$-coeff vs $(c \circ S u c) v \odot_{M} v \in$ carrier $M$
using Cons.prems(1) mk-coeff-Suc-closed by auto

```
        show (\bigoplus Mva\inset vs - {v}.((if va=v then c 0 else 0)
        \oplusmk-coeff vs (c\circSuc)va)\odot < va)\in carrier M
        by (rule M.add.finprod-closed) (auto simp add: mk-coeff-Suc-closed x-in)
    qed
```



```
        unfolding rw2 ..
    also have ... = c0 © \odot Mv }\mp@subsup{\oplus}{M}{}\mathrm{ lincomb (mk-coeff vs (c ○ Suc)) (set vs)
    unfolding lincomb-def ..
    finally show c 0 \odot M v 䖝 lincomb (mk-coeff vs (c\circSuc)) (set vs)
    = lincomb ( }\lambdaa.(\mathrm{ if }a=v\mathrm{ then c 0 else 0) }\oplusmk\mathrm{ -coeff vs (c○Suc) a) (set vs)
    qed
qed simp
definition span-list vs \equiv{lincomb-list c vs | c. c:{0..<length vs } }->\mathrm{ carrier R }
lemma in-span-listI:
    assumes c:{0..<length vs} }->\mathrm{ carrier R and v= lincomb-list c vs
    shows v}\in\mathrm{ span-list vs
    using assms by (auto simp: span-list-def)
lemma in-span-listE:
    assumes v\in span-list vs
        and \bigwedgec.c:{0..<length vs }}->\mathrm{ carrier R }\Longrightarrowv=lincomb-list c vs \Longrightarrow thesi
    shows thesis
    using assms by (auto simp: span-list-def)
lemmas lincomb-insert2 = lincomb-insert[unfolded insert-union[symmetric]]
lemma lincomb-zero:
    assumes U:U\subseteqcarrier M and a:a:U->{zero R}
    shows lincomb a U = zero M
    using U a
proof (induct U rule: infinite-finite-induct)
    case empty show ?case unfolding lincomb-def by auto next
    case (insert u U)
        hence a \in insert u U Carrier R using zero-closed by force
        thus ?case using insert by (subst lincomb-insert2; auto)
qed (auto simp: lincomb-def)
end
hide-const (open) Multiset.mult
end
```


## 15 Matrices as Vector Spaces

This theory connects the Matrix theory with the VectorSpace theory of Holden Lee. As a consequence notions like span, basis, linear dependence, etc. are available for vectors and matrices of the Matrix-theory.

```
theory VS-Connect
imports
    Matrix
    Missing-VectorSpace
    Determinant
begin
hide-const (open) Multiset.mult
hide-const (open) Polynomial.smult
hide-const (open) Modules.module
hide-const (open) subspace
hide-fact (open) subspace-def
named-theorems class-ring-simps
abbreviation class-ring :: 'a :: {times,plus,one,zero} ring where
    class-ring \equiv\ carrier = UNIV,mult = (*), one = 1, zero = 0, add = (+) D
interpretation class-semiring: semiring class-ring :: ' }a::\mathrm{ semiring-1 ring
    rewrites [class-ring-simps]: carrier class-ring = UNIV
        and [class-ring-simps]: mult class-ring = (*)
        and [class-ring-simps]: add class-ring = (+)
        and [class-ring-simps]: one class-ring = 1
    and [class-ring-simps]: zero class-ring = 0
    and [class-ring-simps]: pow (class-ring :: 'a ring) = (`)
    and [class-ring-simps]: finsum (class-ring :: 'a ring) = sum
proof -
    let ?r = class-ring :: 'a ring
    show semiring ?r
        by (unfold-locales, auto simp: field-simps)
    then interpret semiring?r.
    {
        fix xy
        have x[}?? y = x^ y
            by (induct y, auto simp: power-commutes)
    }
    thus ([`]?r) = (`) by (intro ext)
    {
        fix f and }A:: 'b se
        have finsum?r f A = sumf }
            by (induct A rule: infinite-finite-induct, auto)
        }
    thus finsum ?r = sum by (intro ext)
qed auto
```

```
interpretation class-ring: ring class-ring :: ' \(a\) :: ring-1 ring
    rewrites carrier class-ring \(=\) UNIV
    and mult class-ring \(=(*)\)
    and add class-ring \(=(+)\)
    and one class-ring \(=1\)
    and zero class-ring \(=0\)
    and [class-ring-simps]: a-inv (class-ring \(:: ~ ' a ~ r i n g) ~=~ u m i n u s ~\)
    and [class-ring-simps]: a-minus (class-ring :: 'a ring) \(=\) minus
    and pow (class-ring :: 'a ring) \(=(\mathcal{)}\)
    and finsum (class-ring \(::\) 'a ring) \(=\) sum
proof -
    let \(? r=\) class-ring \(::\) 'a ring
    interpret semiring? \(r\)..
    show finsum ?r \(=\) sum pow \(? r=(`\) by (simp-all add: class-ring-simps \()\)
    \{
    fix \(x::{ }^{\prime} a\)
    have \(\exists y . x+y=0\) by (rule exI \([\) of \(-x]\), auto)
    \(\}\) note \([\) simp \(]=\) this
    show ring? r
    by (unfold-locales, auto simp: field-simps Units-def)
    then interpret ring? .
    \{
        fix \(x::{ }^{\prime} a\)
        have \(\ominus_{{ }^{2} r} x=-x\) unfolding \(a\)-inv-def m-inv-def
            by (rule the1-equality, rule ex1I[of - x], auto simp: minus-unique)
    \(\}\) note ainv \(=\) this
    thus inv: a-inv ?r = uminus by (intro ext)
    \{
        fix \(x y::^{\prime} a\)
        have \(x \ominus{ }_{? r} y=x-y\)
            apply (subst a-minus-def)
            using inv by auto
    \}
    thus \(\left(\lambda x y . x \ominus ?_{?} y\right)=\) minus by (intro ext)
qed (auto simp: class-ring-simps)
interpretation class-cring: cring class-ring :: 'a :: comm-ring-1 ring
    rewrites carrier class-ring \(=U N I V\)
    and mult class-ring \(=(*)\)
    and add class-ring \(=(+)\)
    and one class-ring \(=1\)
    and zero class-ring \(=0\)
    and \(a\)-inv (class-ring \(::\) 'a ring) \(=\) uminus
    and \(a\)-minus (class-ring \(:: ~ ' a ~ r i n g) ~=~ m i n u s ~\)
    and pow (class-ring \(::{ }^{\prime} a\) ring \()=(\mathcal{)}\)
    and finsum (class-ring :: 'a ring) = sum
    and [class-ring-simps]: finprod class-ring \(=\) prod
proof -
```

```
    let ?r = class-ring :: 'a ring
    interpret ring ?r ..
    show cring?r
    by (unfold-locales, auto)
    then interpret cring ?r .
    show a-inv (class-ring :: 'a ring) = uminus
    and a-minus (class-ring :: 'a ring) = minus
    and pow (class-ring :: 'a ring)=(`)
    and finsum (class-ring :: 'a ring) = sum by (simp-all add: class-ring-simps)
{
    fix f}\mathrm{ and }A::'b se
    have finprod ?r f A = prod f A
        by (induct A rule: infinite-finite-induct, auto)
    }
    thus finprod ?r = prod by (intro ext)
qed (auto simp: class-ring-simps)
definition div0 :: ' a :: {one,plus,times,zero } where
    div0 \equivm-inv (class-ring :: 'a ring) 0
lemma class-field: field (class-ring :: ' a :: field ring) (is field ?r)
proof -
    interpret cring ?r ..
    {
        fix }x:: ' a
        have }x\not=0\Longrightarrow\existsxa.xa*x=1\wedgex*xa=
            by (intro exI[of - inverse x], auto)
    } note [simp] = this
    show field?r
        by (unfold-locales, auto simp: Units-def)
qed
interpretation class-field: field class-ring :: 'a :: field ring
    rewrites carrier class-ring =UNIV
        and mult class-ring =(*)
        and add class-ring = (+)
        and one class-ring = 1
        and zero class-ring =0
        and a-inv class-ring = uminus
        and a-minus class-ring = minus
        and pow class-ring=(`)
        and finsum class-ring = sum
        and finprod class-ring = prod
    and [class-ring-simps]:m-inv (class-ring :: 'a ring) x =
        (if x = 0 then div0 else inverse x)
proof -
    let ?r = class-ring :: 'a ring
    show field ?r using class-field.
```

```
    then interpret field ?r.
    show a-inv ?r = uminus
    and a-minus ?r = minus
    and pow?r = (`)
    and finsum?r = sum
    and finprod ?r = prod by (fact class-ring-simps)+
    show inv?r }x=(\mathrm{ if }x=0\mathrm{ then div0 else inverse }x
    proof (cases x=0)
        case True
        thus ?thesis unfolding div0-def by simp
    next
    case False
    thus ?thesis unfolding m-inv-def
        by (intro the1-equality ex1I[of-inverse x], auto simp: inverse-unique)
    qed
qed (auto simp: class-ring-simps)
lemmas matrix-vs-simps = module-mat-simps class-ring-simps
definition class-field :: 'a :: field ring
    where [class-ring-simps]:class-field }\equiv\mathrm{ class-ring
locale matrix-ring =
    fixes n :: nat
        and field-type :: 'a :: field itself
begin
abbreviation R where R \equivring-mat TYPE('a) n n
sublocale ring R
    rewrites carrier R = carrier-mat n n
        and add R = (+)
        and mult R = (*)
        and one R=1 m}
        and zero }R=\mp@subsup{O}{m}{}n
    using ring-mat by (auto simp: ring-mat-simps)
end
lemma matrix-vs: vectorspace (class-ring :: 'a :: field ring) (module-mat TYPE('a)
nr nc)
proof -
    interpret abelian-group module-mat TYPE('a) nr nc
    by (rule abelian-group-mat)
    show ?thesis unfolding class-field-def
        by (unfold-locales, unfold matrix-vs-simps,
            auto simp: add-smult-distrib-left-mat add-smult-distrib-right-mat)
qed
```

```
locale vec-module =
    fixes f-ty::'a::comm-ring-1 itself
    and n::nat
begin
abbreviation V where V module-vec TYPE('a) n
sublocale Module.module class-ring :: 'a ring V
    rewrites carrier V = carrier-vec n
    and add V = (+)
    and zero }V=0,0v
    and module.smult V = ( }\cdotv
    and carrier class-ring = UNIV
    and monoid.mult class-ring = (*)
    and add class-ring = (+)
    and one class-ring = 1
    and zero class-ring = 0
    and a-inv (class-ring :: 'a ring) = uminus
    and a-minus (class-ring :: 'a ring) = (-)
    and pow (class-ring :: 'a ring) = (`)
    and finsum (class-ring :: 'a ring) = sum
    and finprod (class-ring :: 'a ring) = prod
    and }\bigwedgeX.X\subseteqUNIV = Tru
    and }\bigwedgex.x\inUNIV=Tru
    and \a A. a \inA -> UNIV \equivTrue
    and }\bigwedgeP.P\wedge True \equiv
    and }\bigwedgeP.(\mathrm{ True }\LongrightarrowP)\equiv Trueprop P
    apply unfold-locales
    apply (auto simp: module-vec-simps class-ring-simps Units-def add-smult-distrib-vec
        smult-add-distrib-vec intro!:bexI[of - - -])
    done
end
locale matrix-vs=
    fixes nr :: nat
        and nc :: nat
        and field-type :: ' a :: field itself
begin
abbreviation \(V\) where \(V \equiv\) module-mat \(\operatorname{TYPE}\left({ }^{\prime} a\right) n r n c\)
sublocale
    vectorspace class-ring V
    rewrites carrier V = carrier-mat nr nc
    and add V = (+)
    and mult V = (*)
```

```
    and one V = 1m nr
    and zero V = 0 m nr nc
    and smult V = ( }\cdotm\mathrm{ )
    and carrier class-ring = UNIV
    and mult class-ring = (*)
    and add class-ring = (+)
    and one class-ring = 1
    and zero class-ring =0
    and a-inv (class-ring :: 'a ring) = uminus
    and a-minus (class-ring :: 'a ring) = minus
    and pow (class-ring :: 'a ring) = (`)
    and finsum (class-ring :: 'a ring) = sum
    and finprod (class-ring :: 'a ring) = prod
    and m-inv (class-ring :: 'a ring) }x
    (if }x=0\mathrm{ then div0 else inverse x)
    by (rule matrix-vs, auto simp: matrix-vs-simps class-field-def)
end
lemma vec-module: module (class-ring :: ' a :: field ring)(module-vec TYPE('a) n)
proof -
    interpret abelian-group module-vec TYPE('a) n
        apply (unfold-locales)
        unfolding module-vec-def Units-def
        using add-inv-exists-vec by auto
    show ?thesis
        unfolding class-field-def
        apply (unfold-locales)
        unfolding class-ring-simps
        unfolding module-vec-simps
        using add-smult-distrib-vec
        by (auto simp: smult-add-distrib-vec)
qed
lemma vec-vs: vectorspace (class-ring :: 'a :: field ring) (module-vec TYPE('a) n)
    unfolding vectorspace-def
    using vec-module class-field
    by (auto simp: class-field-def)
locale vec-space =
    fixes f-ty::'a::field itself
    and n::nat
begin
sublocale vec-module f-ty \(n\).
sublocale vectorspace class-ring \(V\)
rewrites \(c V[\) simp \(]\) : carrier \(V=\) carrier-vec \(n\)
and [simp]: add \(V=(+)\)
and \([\operatorname{simp}]\) : zero \(V=0_{v} n\)
```

```
    and [simp]: smult V = ( }\mp@subsup{v}{v}{}
    and carrier class-ring = UNIV
    and mult class-ring = (*)
    and add class-ring = (+)
    and one class-ring = 1
    and zero class-ring = 0
    and a-inv (class-ring :: 'a ring) = uminus
    and a-minus (class-ring :: 'a ring) = minus
    and pow (class-ring :: 'a ring) = (`)
    and finsum (class-ring :: 'a ring) = sum
    and finprod (class-ring :: 'a ring) = prod
    and m-inv (class-ring :: 'a ring) x = (if x =0 then div0 else inverse x)
using vec-vs
unfolding class-field-def
by (auto simp: module-vec-simps class-ring-simps)
lemma finsum-vec[simp]: finsum-vec TYPE('a) n= finsum V
    by (force simp: finsum-vec-def monoid-vec-def finsum-def finprod-def)
lemma finsum-scalar-prod-sum:
    assumes f:f:U-> carrier-vec n
        and w:w:carrier-vec n
    shows finsum Vf U •w = sum (\lambdau.fu\cdotw)U
    using wf
proof (induct U rule: infinite-finite-induct)
    case (insert u U)
    hence f:f:U->carrier-vec n fu:carrier-vec n by auto
    show ?case
            unfolding finsum-insert[OF insert(1) insert(2) f]
            apply (subst add-scalar-prod-distrib) using insert by auto
qed auto
lemma vec-neg[simp]: assumes x : carrier-vec n shows }\mp@subsup{\ominus}{V}{}x=-
    unfolding a-inv-def m-inv-def apply simp
    apply (rule the-equality, intro conjI)
    using assms apply auto
    using M.minus-unique uminus-carrier-vec uminus-r-inv-vec by blast
lemma finsum-dim:
    finite }A\Longrightarrowf\inA->\mathrm{ carrier-vec n }\Longrightarrow\mathrm{ dim-vec (finsum Vf A) =n
proof(induct set:finite)
    case (insert a A)
    hence dfa:dim-vec (f a) = n by auto
    have f:f\inA->carrier-vec n using insert by auto
    hence fa: f a \in carrier-vec n using insert by auto
    show ?case
        unfolding finsum-insert[OF insert(1) insert(2) f fa]
        using insert by auto
qed simp
```

```
lemma lincomb-dim:
    assumes fin: finite }
        and X:X\subseteqcarrier-vec n
    shows dim-vec (lincomb a X)=n
proof -
    let ?f = \lambdav.av v}
    have f:?f }\inX->\mathrm{ carrier-vec n apply rule using }X\mathrm{ by auto
    show ?thesis
        unfolding lincomb-def
        using finsum-dim[OF fin f].
qed
lemma finsum-index:
    assumes i: i<n
        and f:f\inX carrier-vec n
        and X:X\subseteqcarrier-vec n
    shows finsum VfX $i=sum ( }\lambdax.fx$i)
    using Xf
proof (induct X rule: infinite-finite-induct)
    case empty
    then show ?case using i by simp next
    case (insert x X)
        then have Xf: finite X
            and xX:x\not\inX
            and x:x\in carrier-vec n
            and X:X\subseteqcarrier-vec n
            and fx:fx\in carrier-vec n
            and f:f\inX->carrier-vec n by auto
            have i2: i< dim-vec (finsum VfX)
            using i finsum-closed[OF f] by auto
    have ix: i< dim-vec x using x i by auto
    show ?case
            unfolding finsum-insert[OF Xf xX f fx]
            unfolding sum.insert[OF Xf xX]
            unfolding index-add-vec(1)[OF i2]
            using insert lincomb-def
            by auto
qed (insert i, auto)
lemma lincomb-index:
    assumes i: i<n
    and X:X\subseteqcarrier-vec n
    shows lincomb a X $i=sum (\lambdax. a x * x$ i)X
proof -
    let ?f = \lambdax. a x 趹 
    have f: ?f : }X->\mathrm{ carrier-vec n using X by auto
    have point: \v.v\inX\Longrightarrow(av\cdotvv)$i=av*v$i using i X by auto
```

```
    show ?thesis
    unfolding lincomb-def
    unfolding finsum-index[OF if X]
    using point X by simp
qed
lemma append-insert: set (xs @ [x]) = insert x (set xs) by simp
lemma lincomb-units:
    assumes i: i<n
    shows lincomb a (set (unit-vecs n)) $i=a(unit-vec n i)
    unfolding lincomb-index[OF i unit-vecs-carrier]
    unfolding unit-vecs-first
proof -
    let ?f = \lambdami. \sumx\inset (unit-vecs-first n m). a x*x$i
    have zero:\m j.m\leqj\Longrightarrowj<n\Longrightarrow ?f m j=0
    proof -
        fix m
        show }\j.m\leqj\Longrightarrowj<n\Longrightarrow?f mj=
        proof (induction m)
            case (Suc m)
                hence mj:m\leqj and mj':m\not=j and jn:j<n and mn:m<n by auto
                    hence mem: unit-vec n m & set (unit-vecs-first n m)
                            apply(subst unit-vecs-first-distinct) by auto
                show ?case
                        unfolding unit-vecs-first.simps
                    unfolding append-insert
                    unfolding sum.insert[OF finite-set mem]
                    unfolding index-unit-vec(1)[OF mn jn]
                    unfolding Suc(1)[OF mj jn] using mj' by simp
        qed simp
    qed
    { fix m
        have i<m\Longrightarrowm\leqn\Longrightarrow?f mi=a(unit-vec n i)
        proof (induction m arbitrary: i)
            case (Suc m)
                hence iSm: i<Suc m and i:i<n and mn: m<n by auto
                    hence mem: unit-vec n m & set (unit-vecs-first n m)
                            apply(subst unit-vecs-first-distinct) by auto
                    show ?case
                        unfolding unit-vecs-first.simps
                        unfolding append-insert
                        unfolding sum.insert[OF finite-set mem]
                    unfolding index-unit-vec(1)[OF mn i]
                    using zero Suc by (cases i=m,auto)
        qed auto
    }
    thus ?f ni=a (unit-vec n i) using assms by auto
qed
```

```
lemma lincomb-coordinates:
    assumes v: v:carrier-vec n
    defines }a\equiv(\lambdau.v$(THE i.u=unit-vec n i))
    shows lincomb a (set (unit-vecs n))}=
proof -
    have a: a\in set (unit-vecs n) }->\mathrm{ UNIV by auto
    have fvu: \i. i<n\Longrightarrowv$i=a(unit-vec n i)
        unfolding a-def using unit-vec-eq by auto
    show ?thesis
    apply rule
    unfolding lincomb-dim[OF finite-set unit-vecs-carrier]
    using v lincomb-units fvu
    by auto
qed
lemma span-unit-vecs-is-carrier: span (set (unit-vecs n)) = carrier-vec n (is ?L
=?R)
proof (rule;rule)
    fix v assume vsU:v\in?L show }v\in?
    proof -
        obtain a
            where v:v= lincomb a (set (unit-vecs n))
            using vsU
            unfolding finite-span[OF finite-set unit-vecs-carrier] by auto
        thus ?thesis using lincomb-closed[OF unit-vecs-carrier] by auto
    qed
    next fix v::'a vec assume v: v}\in?R\mathrm{ show }v\in?
        unfolding span-def
        using lincomb-coordinates[OF v,symmetric] by auto
qed
lemma fin-dim[simp]: fin-dim
    unfolding fin-dim-def
    apply (intro eqTrueI exI conjI) using span-unit-vecs-is-carrier unit-vecs-carrier
by auto
lemma unit-vecs-basis: basis (set (unit-vecs n)) unfolding basis-def span-unit-vecs-is-carrier
proof (intro conjI)
    show \neg lin-dep (set (unit-vecs n))
    proof
    assume lin-dep (set (unit-vecs n))
    from this[unfolded lin-dep-def] obtain A a v where
                fin: finite A and A:A\subseteq set (unit-vecs n)
        and lc: lincomb a A = Ov n and v:v\inA and av: av\not=0
        by auto
        from v A obtain i where i: i<n and vu:v=unit-vec n i unfolding
unit-vecs-def by auto
    define b}\mathrm{ where b=( }\lambdax\mathrm{ . if }x\inA\mathrm{ then a x else 0)
```

```
    have id: A\cup(set (unit-vecs n) - A) = set (unit-vecs n) using A by auto
    from lincomb-index[OF i unit-vecs-carrier]
    have lincomb b (set (unit-vecs n)) $i=(\sumx\in(A\cup(set (unit-vecs n) - A)).
b x*x$ i)
        unfolding id .
    also have ... = (\sumx\inA.bx*x$i)+(\sumx\in set (unit-vecs n) - A.bx*
x$i)
            by (rule sum.union-disjoint, insert fin, auto)
    also have (\sumx\inA.bx*x$i)=(\sumx\inA.ax*x$i)
        by (rule sum.cong, auto simp: b-def)
    also have ... = lincomb a A $ i
        by (subst lincomb-index[OF i], insert A unit-vecs-carrier, auto)
    also have ... = 0 unfolding lc using i by simp
    also have (\sumx\in set (unit-vecs n) - A.b x * x $ i)=0
        by (rule sum.neutral, auto simp: b-def)
    finally have lincomb b (set (unit-vecs n)) $i=0 by simp
    from lincomb-units[OF i, of b, unfolded this]
    have bv=0 unfolding vu by simp
    with vav show False unfolding b-def by simp
    qed
qed (insert unit-vecs-carrier, auto)
lemma unit-vecs-length[simp]: length (unit-vecs n) = n
    unfolding unit-vecs-def by auto
lemma unit-vecs-distinct: distinct (unit-vecs n)
    unfolding distinct-conv-nth unit-vecs-length
proof (intro allI impI)
    fix ij
    assume *: i<nj<ni\not=j
    show unit-vecs n!i\not= unit-vecs n!j
    proof
    assume unit-vecs n!i= unit-vecs n!j
    from arg-cong[OF this, of \lambdav.v$i]
    show False using * unfolding unit-vecs-def by auto
    qed
qed
lemma dim-is-n: dim =n
    unfolding dim-basis[OF finite-set unit-vecs-basis]
    unfolding distinct-card[OF unit-vecs-distinct]
    by simp
end
locale mat-space =
    vec-space f-ty nc for f-ty::'a::field itself and nc::nat +
    fixes nr :: nat
begin
```

```
    abbreviation M where M = ring-mat TYPE('a) nc nr
end
context vec-space
begin
lemma fin-dim-span:
assumes finite A A\subseteqcarrier V
shows vectorspace.fin-dim class-ring (vs (span A))
proof -
    have vectorspace class-ring (span-vs A)
    using assms span-is-subspace subspace-def subspace-is-vs by simp
    have A\subseteq span A using assms in-own-span by simp
    have submodule class-ring (span A) V using assms span-is-submodule by simp
    have LinearCombinations.module.span class-ring (vs (span A)) A = carrier (vs
(span A))
    using span-li-not-depend(1)[OF <A\subseteq span A><submodule class-ring (span A)
V>] by auto
    then show ?thesis unfolding vectorspace.fin-dim-def[OF<vectorspace class-ring
(span-vs A)>]
    using List.finite-set <A\subseteq span A〉<vectorspace class-ring (vs (span A))>
            vec-vs vectorspace.carrier-vs-is-self[OF <vectorspace class-ring (span-vs A)>]
using assms(1) by auto
qed
lemma fin-dim-span-cols:
assumes A \in carrier-mat n nc
shows vectorspace.fin-dim class-ring (vs (span (set (cols A))))
    using fin-dim-span cols-dim List.finite-set assms carrier-matD(1) module-vec-simps(3)
by force
end
context vec-module
begin
lemma lincomb-list-as-mat-mult:
    assumes }\forallw\in\mathrm{ set ws. dim-vec w=n
    shows lincomb-list c ws = mat-of-cols n ws *v vec (length ws) c (is ?l ws c = ?r
ws c)
proof (insert assms, induct ws arbitrary: c)
    case Nil
    then show ?case by (auto simp: mult-mat-vec-def scalar-prod-def)
next
    case (Cons w ws)
    { fix i assume i:i<n
        have ?l (w#ws) c = c 0 |v w + mat-of-cols n ws *v vec (length ws) (c\circSuc)
            by (simp add: Cons o-def)
        also have ...$i=?r (w#ws) c $i
            using Cons i index-smult-vec
                by (simp add: mat-of-cols-Cons-index-0 mat-of-cols-Cons-index-Suc o-def
```

```
vec-Suc mult-mat-vec-def row-def length-Cons)
    finally have ?l (w#ws)c$i=\ldots..
    }
    with Cons show ?case by (intro eq-vecI, auto)
qed
lemma lincomb-vec-diff-add:
    assumes A:A\subseteqcarrier-vec n
    and BA:B\subseteqA and fin-A: finite }
    and f:f\inA->UNIV shows lincomb fA}=|=lincomb f(A-B)+lincomb f B
proof -
    have }A-B\cupB=A\mathrm{ using }BA\mathrm{ by auto
    hence lincomb fA= lincomb f(A-B\cupB) by simp
    also have ... = lincomb f(A-B)+ lincomb f B
    by (rule lincomb-union, insert assms, auto intro: finite-subset)
    finally show ?thesis.
qed
lemma dim-sumlist:
    assumes }\forallx\in\mathrm{ set xs. dim-vec }x=
    shows dim-vec (M.sumlist xs) = n using assms by (induct xs, auto)
lemma sumlist-nth:
    assumes }\forallx\in\mathrm{ set xs. dim-vec }x=n\mathrm{ and }i<
    shows (M.sumlist xs) $i= sum ( }\lambdaj.(xs!j)$i){0..<length xs
    using assms
proof (induct xs rule: rev-induct)
    case (snoc a xs)
    have [simp]: x\in carrier-vec n if x: x\in set xs for x
        using snoc.prems x unfolding carrier-vec-def by auto
    have [simp]: a\in carrier-vec n
        using snoc.prems unfolding carrier-vec-def by auto
    have hyp:M.sumlist xs $ i=( \sumj=0..<length xs. xs ! j$ i)
    by (rule snoc.hyps, auto simp add: snoc.prems)
    have M.sumlist (xs@ @a])= M.sumlist xs + M.sumlist [a]
    by (rule M.sumlist-append, auto simp add: snoc.prems)
    also have ... = M.sumlist xs +a by auto
    also have ... $ i=(M.sumlist xs $ i) + (a $ i)
    by (rule index-add-vec(1), auto simp add: snoc.prems)
    also have ... = (\sumj=0..<length xs. xs ! j$i) + (a$ i) unfolding hyp by
simp
    also have ... = (\sumj=0..<length (xs @ [a]). (xs @ [a])! j$ i)
        by (auto, rule sum.cong, auto simp add: nth-append)
    finally show ?case .
qed auto
lemma lincomb-as-lincomb-list-distinct:
    assumes s: set ws \subseteqcarrier-vec n and d: distinct ws
    shows lincomb f(set ws) = lincomb-list (\lambdai.f (ws!i)) ws
```

```
proof (insert assms, induct ws)
    case Nil
    then show? case by auto
next
    case (Cons a ws)
    have \([\) simp \(]: \bigwedge v . v \in\) set \(w s \Longrightarrow v \in\) carrier-vec \(n\) using Cons.prems(1) by auto
    then have ws: set ws \(\subseteq\) carrier-vec \(n\) by auto
    have hyp: lincomb \(f(\) set \((w s))=\) lincomb-list \((\lambda i . f(w s!i))\) ws
    proof (intro Cons.hyps ws)
        show distinct ws using Cons.prems(2) by auto
    qed
    have \(\left(\operatorname{map}\left(\lambda i . f(w s!i) \cdot{ }_{v} w s!i\right)[0 . .<l e n g t h w s]\right)=(\operatorname{map}(\lambda v . f v \cdot v v) w s)\)
        by (intro nth-equalityI, auto)
    with ws have sumlist-rw: sumlist (map ( \(\lambda i . f(w s!i) \cdot v\) ws ! i) \([0 . .<\) length ws])
        \(=\operatorname{sumlist}\left(\operatorname{map}\left(\lambda v . f v \cdot{ }_{v} v\right) w s\right)\)
        by (subst (1 2) sumlist-as-summset, auto)
    have lincomb \(f(\operatorname{set}(a \# w s))=\left(\bigoplus_{V} v \in \operatorname{set}(a \# w s) . f v \cdot v v\right)\) unfolding
lincomb-def ..
    also have \(\ldots=\left(\bigoplus_{V} v \in\right.\) insert \(a(\) set ws \(\left.) . f v \cdot{ }_{v} v\right)\) by simp
    also have \(\ldots=\left(f a \cdot{ }_{v} a\right)+\left(\bigoplus_{V^{v}} \in(\right.\) set ws \(\left.) . f v{ }_{v} v\right)\)
        by (rule finsum-insert, insert Cons.prems, auto)
    also have \(\ldots=f a \cdot v a+\) lincomb-list \((\lambda i . f(w s!i))\) ws using hyp lincomb-def
by auto
    also have \(\ldots=f a \cdot{ }_{v} a+\operatorname{sumlist}\left(\operatorname{map}\left(\lambda v . f v \cdot{ }_{v} v\right) w s\right)\)
        unfolding lincomb-list-def sumlist-rw by auto
    also have \(\ldots=\operatorname{sumlist}(\operatorname{map}(\lambda v . f v \cdot v v)(a \# w s))\)
    proof -
        let \(? a=(\operatorname{map}(\lambda v . f v \cdot v v)[a])\)
        have \(a: a \in\) carrier-vec \(n\) using Cons.prems(1) by auto
        have \(f a \cdot{ }_{v} a=\) sumlist (map \(\left.\left(\lambda v . f v \cdot{ }_{v} v\right)[a]\right)\) using Cons.prems(1) by auto
        hence \(f a \cdot_{v} a+\) sumlist (map \(\left.\left(\lambda v . f v \cdot_{v} v\right) w s\right)\)
            \(=\) sumlist \(? a+\operatorname{sumlist}(\operatorname{map}(\lambda v . f v \cdot v v)\) ws) by simp
    also have \(\ldots=\operatorname{sumlist}(? a\) @ (map \((\lambda v . f v \cdot v v) w s))\)
            by (rule sumlist-append[symmetric], auto simp add: a)
        finally show ?thesis by auto
    qed
    also have \(\ldots=\) sumlist \((\operatorname{map}(\lambda i . f((a \#\) ws \()!i) \cdot v(a \# w s)!i)[0 . .<l e n g t h\)
( \(a \# w s)]\) )
    proof -
        have \(u:(\operatorname{map}(\lambda i . f((a \# w s)!i) \cdot v(a \# w s)!i)[0 . .<l e n g t h(a \# w s)])\)
            \(=(\operatorname{map}(\lambda v . f v \cdot v v)(a \# w s))\)
        proof (intro nth-equalityI, goal-cases)
            case (2 i) thus ?case by (smt length-map map-nth nth-map)
        qed auto
        show ?thesis unfolding \(u\)..
    qed
    also have \(\ldots=\) lincomb-list \((\lambda i . f((a \# w s)!i))(a \# w s)\)
        unfolding lincomb-list-def ..
    finally show ?case .
```


## qed

end

```
locale \(i d o m-v e c=\) vec-module \(f\)-ty for \(f\)-ty :: ' \(a\) :: idom itself
begin
lemma lin-dep-cols-imp-det-0':
    fixes \(w s\)
    defines \(A \equiv\) mat-of-cols n ws
    assumes dimv-ws: \(\forall w \in\) set ws. dim-vec \(w=n\)
    assumes \(A: A \in\) carrier-mat \(n n\) and ld-cols: lin-dep (set (cols \(A\) ))
    shows \(\operatorname{det} A=0\)
proof (cases distinct ws)
    case False
    obtain \(i j\) where \(i j: i \neq j\) and \(c: \operatorname{col} A i=\operatorname{col} A j\) and \(i: i<n\) and \(j: j<n\)
        using False \(A\) unfolding \(A\)-def
        by (metis dimv-ws distinct-conv-nth carrier-matD (2)
            col-mat-of-cols mat-of-cols-carrier(3) nth-mem carrier-vecI)
    show ?thesis by (rule det-identical-columns[OF A ij ij c])
next
    case True
    have \(d 1[\) simp \(]: \bigwedge x . x \in\) set \(w s \Longrightarrow x \in\) carrier-vec \(n\) using dimv-ws by auto
    obtain \(A^{\prime} f^{\prime} v\) where \(f^{\prime}\)-in: \(f^{\prime} \in A^{\prime} \rightarrow U N I V\)
        and \(l c-f^{\prime}:\) lincomb \(f^{\prime} A^{\prime}=O_{v} n\) and \(f^{\prime}-v: f^{\prime} v \neq 0\)
        and \(v-A^{\prime}: v \in A^{\prime}\) and \(A^{\prime}\)-in-rows: \(A^{\prime} \subseteq\) set (cols \(A\) )
        using \(l d\)-cols unfolding lin-dep-def by auto
    define \(f\) where \(f \equiv \lambda x\). if \(x \notin A^{\prime}\) then 0 else \(f^{\prime} x\)
    have \(f\)-in: \(f \in(\) set (cols \(A)) \rightarrow U N I V\) using \(f^{\prime}\)-in by auto
    have \(A^{\prime}\)-in-carrier: \(A^{\prime} \subseteq\) carrier-vec \(n\)
    by (metis (no-types) \(A^{\prime}\)-in-rows \(A\)-def cols-dim carrier-matD(1) mat-of-cols-carrier (1)
subset-trans)
    have \(l c-f\) : \(\operatorname{lincomb} f(\operatorname{set}(\operatorname{cols} A))=0_{v} n\)
    proof -
    have 11 : lincomb \(f\left(\operatorname{set}(\operatorname{cols} A)-A^{\prime}\right)=0_{v} n\)
            by (rule lincomb-zero, auto simp add: \(f\)-def, insert \(A\) cols-dim, blast)
    have 12 : lincomb \(f A^{\prime}=O_{v} n\) using \(l c\) - \(f^{\prime}\) unfolding \(f\)-def using \(A^{\prime}\)-in-carrier
by auto
    have lincomb \(f(\operatorname{set}(\operatorname{cols} A))=\operatorname{lincomb} f\left(\operatorname{set}(\operatorname{cols} A)-A^{\prime}\right)+\operatorname{lincomb} f A^{\prime}\)
    proof (rule lincomb-vec-diff-add)
            show set (cols \(A\) ) \(\subseteq\) carrier-vec \(n\)
                using \(A\) cols-dim by blast
            show \(A^{\prime} \subseteq\) set \((\) cols \(A)\)
                using \(A^{\prime}\)-in-rows by blast
    qed auto
    also have \(\ldots=O_{v} n\) using \(l 1\) l2 by auto
    finally show? thesis .
    qed
    have \(v\)-in: \(v \in(\operatorname{set}(\operatorname{cols} A))\) using \(v-A^{\prime} A^{\prime}\)-in-rows by auto
```

have $f v$ : $f v \neq 0$ using $f^{\prime}-v v$ - $A^{\prime}$ unfolding $f$-def by auto
let $? c=(\lambda i . f(w s!i))$
have lincomb $f($ set ws $)=$ lincomb-list ?c ws
by (rule lincomb-as-lincomb-list-distinct $[O F-T r u e]$, auto)
have $\exists v . \quad v \in$ carrier-vec $n \wedge v \neq 0_{v} n \wedge A *_{v} v=0_{v} n$
proof (rule exI[of - vec (length ws) ?c], rule conjI)
show vec (length ws) ?c $\in$ carrier-vec $n$ using $A A$-def by auto
have vec-not0: vec (length ws) ?c $\neq 0_{v} n$
proof -
obtain $i$ where $w s-i:(w s!i)=v$ and $i: i<l e n g t h$ ws using $v$-in unfolding
$A$-def
by (metis d1 cols-mat-of-cols in-set-conv-nth subset-eq)
have vec (length ws) ?c $\$ i=$ ?c $i$ by (rule index-vec[OF $i]$ )
also have $\ldots=f v$ using ws- $i$ by simp
also have $\ldots \neq 0$ using $f v$ by simp
finally show ?thesis
using $A A$-def $i$ by fastforce
qed
have $A *_{v}$ vec (length ws) ?c = mat-of-cols $n w s *_{v}$ vec (length ws) ?c unfolding $A$-def ..
also have ... $=$ lincomb-list ?c ws by (rule lincomb-list-as-mat-mult[symmetric, OF dimv-ws])
also have $\ldots=\operatorname{lincomb} f($ set ws)
by (rule lincomb-as-lincomb-list-distinct[symmetric, OF - True], auto)
also have $\ldots=O_{v} n$
using lc-f unfolding $A$-def using $A$ by (simp add: subset-code(1))
finally show vec (length ws) $(\lambda i . f(w s!i)) \neq 0_{v} n \wedge A *_{v}$ vec (length ws)
$(\lambda i . f(w s!i))=o_{v} n$
using vec-not0 by fast
qed
thus ?thesis unfolding det-0-iff-vec-prod-zero[OF A].
qed
lemma lin-dep-cols-imp-det-0:
assumes $A: A \in$ carrier-mat $n n$ and $l d$ : lin-dep (set (cols $A)$ )
shows $\operatorname{det} A=0$
proof -
have col-rw: $($ cols $($ mat-of-cols $n(\operatorname{cols} A)))=\operatorname{cols} A$
using $A$ by auto
have $m$ : mat-of-cols $n(\operatorname{cols} A)=A$ using $A$ by auto
show ?thesis
by (rule A lin-dep-cols-imp-det-0'[of cols A, unfolded col-rw, unfolded m, OF - A ld])
(metis A cols-dim carrier-matD(1) subsetCE carrier-vecD)
qed
corollary lin-dep-rows-imp-det-0:
assumes $A: A \in$ carrier-mat $n n$ and ld: lin-dep (set (rows $A$ ))
shows $\operatorname{det} A=0$
by (subst det-transpose[OF A, symmetric], rule lin-dep-cols-imp-det-0, auto simp add: $(d$ A)
lemma det-not-0-imp-lin-indpt-rows:
assumes $A: A \in \operatorname{carrier-mat} n n$ and $\operatorname{det}: \operatorname{det} A \neq 0$
shows lin-indpt (set (rows A))
using lin-dep-rows-imp-det- $0[O F A]$ det by auto
lemma upper-triangular-imp-lin-indpt-rows:
assumes $A: A \in$ carrier-mat $n n$
and tri: upper-triangular $A$
and diag: $0 \notin \operatorname{set}(\operatorname{diag}-m a t A)$
shows lin-indpt (set (rows A))
using det-not-0-imp-lin-indpt-rows upper-triangular-imp-det-eq-0-iff assms by auto
lemma lincomb-as-lincomb-list:
fixes $w s f$
assumes $s$ : set ws $\subseteq$ carrier-vec $n$
shows lincomb $f($ set ws $)=$ lincomb-list $(\lambda i$. if $\exists j<i$. ws! $i=$ ws! $j$ then 0 else $f$
(ws!i)) ws
using assms
proof (induct ws rule: rev-induct)
case (snoc a ws)
let ?f $=\lambda i$. if $\exists j<i$. ws ! $i=w s!j$ then 0 else $f(w s!i)$
let ? $g=\lambda i$. (if $\exists j<i$. (ws @ [a])! $i=(w s @[a])!j$ then 0 else $f((w s @[a])!$
i)) $\cdot_{v}(w s @[a])!i$
let ? $g 2=\left(\lambda i\right.$. $($ if $\exists j<i$. ws $!i=$ ws ! $j$ then 0 else $f(w s!i)) \cdot{ }_{v}$ ws!i)
have $[$ simp $]: \bigwedge v . v \in$ set $w s \Longrightarrow v \in$ carrier-vec $n$ using snoc.prems(1) by auto
then have ws: set ws $\subseteq$ carrier-vec $n$ by auto
have hyp: lincomb $f$ (set ws) $=$ lincomb-list ?f ws
by (intro snoc.hyps ws)
show ?case
proof (cases $a \in$ set ws)
case True
have $g$-length: ? $g($ length $w s)=0_{v} n$ using True
by (auto, metis in-set-conv-nth nth-append)
have $(\operatorname{map} ? g[0 . .<$ length $(w s @[a])])=(\operatorname{map} ? g[0 . .<$ length ws $]) @[? g($ length $w s)$ ]
by auto
also have $\ldots=($ map ? $g[0 . .<$ length ws $])$ @ $\left[O_{v} n\right]$ using g-length by simp
finally have map-rw: (map?g $[0 . .<$ length $($ ws @ $[a])])=($ map ? $g[0 . .<$ length $w s])$ @ $\left[\begin{array}{ll}O_{v} & n\end{array}\right]$.
have M.sumlist (map ?g2 $[0 . .<$ length ws $])=$ M.sumlist (map ? $g[0 . .<$ length $w s])$
by (rule arg-cong[of - - M.sumlist], intro nth-equalityI, auto simp add: nth-append)

```
    also have ... = M.sumlist (map ?g [0..<length ws]) + O n n
    by (metis M.r-zero calculation hyp lincomb-closed lincomb-list-def ws)
    also have ... = M.sumlist (map ?g [0..<length ws]@ [ [0v n])
    by (rule M.sumlist-snoc[symmetric], auto simp add: nth-append)
    finally have summlist-rw: M.sumlist (map ?g2 [0..<length ws])
    =M.sumlist (map ?g [0..<length ws]@ [ [ v n]).
    have lincomb f(set (ws @ [a]))= lincomb f(set ws) using True unfolding
lincomb-def
    by (simp add: insert-absorb)
    thus ?thesis
        unfolding hyp lincomb-list-def map-rw summlist-rw
        by auto
    next
    case False
        have g-length:?g (length ws) = f a 汭 a using False by (auto simp add:
nth-append)
    have (map ?g [0..<length (ws@ [a])])=(map ?g [0..<length ws])@ [?g (length
ws)]
            by auto
    also have ... = (map ?g [0..<length ws]) @ [(fa |va)] using g-length by simp
    finally have map-rw: (map ?g [0..<length (ws @ [a])]) = (map ?g [0..<length
ws])@[(f a \cdotv a )].
    have summlist-rw: M.sumlist (map ?g2 [0..<length ws]) = M.sumlist (map ?g
[0..<length ws])
            by (rule arg-cong[of - - M.sumlist], intro nth-equalityI, auto simp add:
nth-append)
    have lincomb f (set (ws @ [a]))= lincomb f (set (a # ws)) by auto
```



```
    also have ... =(\bigoplus\mp@subsup{V}{V}{}v\in\mathrm{ insert a (set ws). fv v}\mp@subsup{v}{v}{}v) by simp
    also have ... = (f a | v}a)+(\bigoplus\mp@subsup{V}{}{v}\in(set ws).fv fvv v
    proof (rule finsum-insert)
        show finite (set ws) by auto
        show a & set ws using False by auto
        show (\lambdav.fv vvv) set ws }->\mathrm{ carrier-vec n
            using snoc.prems(1) by auto
        show f a 汭 a < carrier-vec n using snoc.prems by auto
    qed
    also have ... = (f a 汭 a) + lincomb f (set ws) unfolding lincomb-def ..
    also have ... = (f a v}va)+ lincomb-list ?f ws using hyp by aut
    also have ... = lincomb-list ?f ws + (f a 汭a)
        using M.add.m-comm lincomb-list-carrier snoc.prems by auto
    also have ... = lincomb-list (\lambdai. if \existsj<i. (ws @ [a])!i
        =(ws @ [a])! j then 0 else f ((ws @ [a])! i)) (ws @ [a])
    proof (unfold lincomb-list-def map-rw summlist-rw, rule M.sumlist-snoc[symmetric])
        show set (map ?g [0..<length ws])\subseteq carrier-vec n using snoc.prems
            by (auto simp add: nth-append)
            show fa 汭 a \in carrier-vec n
                using snoc.prems by auto
    qed
```

```
    finally show ?thesis.
    qed
qed auto
lemma span-list-as-span:
    assumes set vs \subseteqcarrier-vec n
    shows span-list vs= span (set vs)
    using assms
proof (auto simp: span-list-def span-def)
    fix f show \existsa A. lincomb-list fvs=lincomb a A ^ finite A}\wedgeA\subseteq set v
        using assms lincomb-list-as-lincomb by auto
next
    fix f::'a vec }\mp@subsup{=>}{}{\prime}a\mathrm{ and }A\mathrm{ assume fA: finite }A\mathrm{ and A:A}\subseteq\mathrm{ set vs
    have [simp]: x carrier-vec n if x:x\inA for x using A x assms by auto
    have [simp]: v\incarrier-vec n if v: v\in set vs for v using assms v by auto
    have set-vs-Un: ((set vs) - A)\cupA = set vs using A by auto
    let ?f = ( }\lambda\mathrm{ x. if }x\in(\mathrm{ set vs ) - A then 0 else f x 
```



```
    have lincomb f A = lincomb ?f A
        by (auto simp add: lincomb-def intro!: finsum-cong2)
    also have ... =( \bigoplus\mp@subsup{V}{}{v}\in(\mathrm{ set vs) - A. ?f v }v\mp@subsup{v}{v}{}v)+(\mp@subsup{\bigoplus}{\mp@subsup{V}{}{v}\inA. ?f v }{v}\mp@subsup{v}{v}{}v)
        unfolding f0 lincomb-def by auto
    also have ... = lincomb ?f (((set vs) - A)\cupA)
        unfolding lincomb-def
        by (rule M.finsum-Un-disjoint[symmetric], auto simp add: fA)
    also have ... = lincomb ?f (set vs) using set-vs-Un by auto
    finally have lincomb f A = lincomb ?f (set vs).
    with lincomb-as-lincomb-list[OF assms]
    show }\exists\textrm{c}\mathrm{ . lincomb f A = lincomb-list c vs by auto
qed
lemma in-spanI[intro]:
    assumes v=lincomb a A finite A A\subseteqW
    shows v\in span W
unfolding span-def using assms by auto
lemma in-spanE:
    assumes v\in span W
    shows \existsa A.v=lincomb a A\wedge finite A\wedgeA\subseteqW
using assms unfolding span-def by auto
declare in-own-span[intro]
lemma smult-in-span:
    assumes W\subseteqcarrier-vec n and insp: x f span W
    shows c}\cdotv,x\in\operatorname{span}
proof -
    from in-spanE[OF insp] obtain a A where a: x=lincomb a A finite A A\subseteqW
by blast
    have }c\cdotv\mp@code{x = lincomb (\lambdax.c*ax) A using a(1) unfolding lincomb-def a
```

apply (subst finsum-smult) using assms a by (auto simp:smult-smult-assoc) thus $c \cdot{ }_{v} x \in \operatorname{span} W$ using $a(2,3)$ by auto qed
lemma span-subsetI: assumes ws: ws $\subseteq$ carrier-vec $n$
$u s \subseteq$ span ws
shows span us $\subseteq$ span ws
by (simp add: assms(1) span-is-submodule span-is-subset subsetI ws)
end
context vec-space begin
sublocale idom-vec.
lemma sumlist-in-span: assumes $W: W \subseteq$ carrier-vec $n$ shows $(\bigwedge x . x \in$ set $x s \Longrightarrow x \in \operatorname{span} W) \Longrightarrow$ sumlist $x s \in$ span $W$
proof (induct xs)
case Nil
thus ?case using $W$ by force
next
case (Cons x xs)
from span-is-subset2[OF W] Cons(2) have $x s: x \in$ carrier-vec $n$ set $x s \subseteq$ car-rier-vec $n$ by auto
from span-add1[OF W Cons(2)[of $x] \operatorname{Cons(1)[OF} \operatorname{Cons(2)]]}$
have $x+$ sumlist $x s \in$ span $W$ by auto
also have $x+$ sumlist $x s=$ sumlist ( $[x] @ x s$ )
by (subst sumlist-append, insert xs, auto)
finally show? case by simp
qed
lemma span-span[simp]:
assumes $W \subseteq$ carrier-vec $n$
shows span $($ span $W)=$ span $W$
proof (standard,standard,goal-cases)
case ( $1 x$ ) with in-spanE obtain a $A$ where $a: x=$ lincomb a $A$ finite $A A \subseteq$ span $W$ by blast
from $a(3)$ assms have $A C: A \subseteq$ carrier-vec $n$ by auto
show ? case unfolding a(1)[unfolded lincomb-def]
proof $($ insert a(3), atomize (full), rule finite-induct[OF a(2)],goal-cases)
case 1
then show ?case using span-zero by auto
next
case (2 $x$ F)
\{ assume $F$ :insert $x$ F span $W$
hence $a x \cdot_{v} x \in \operatorname{span} W$ by (intro smult-in-span[OF assms],auto)
hence $a x \cdot{ }_{v} x+\left(\bigoplus_{V} v \in F . a v \cdot{ }_{v} v\right) \in \operatorname{span} W$
using span-add1 $F 2$ assms by auto
hence $\left(\bigoplus_{V^{v}} \in\right.$ insert $\left.x F . a v \cdot{ }_{v} v\right) \in \operatorname{span} W$
apply(subst $M$.finsum-insert $[$ OF 2(1,2)]) using $F$ assms by auto

```
        }
        then show ?case by auto
        qed
next
    case 2
    show ?case using assms by(intro in-own-span, auto)
qed
lemma upper-triangular-imp-basis:
    assumes A:A\incarrier-mat n n
        and tri: upper-triangular A
        and diag: 0 & set (diag-mat A)
    shows basis (set (rows A))
    using upper-triangular-imp-distinct[OF assms]
    using upper-triangular-imp-lin-indpt-rows[OF assms] A
    by (auto intro: dim-li-is-basis simp: distinct-card dim-is-n set-rows-carrier)
lemma fin-dim-span-rows:
assumes A: A\incarrier-mat nr n
shows vectorspace.fin-dim class-ring (vs (span (set (rows A))))
proof (rule fin-dim-span)
    show set (rows A)\subseteq carrier V using A rows-carrier[of A] unfolding car-
rier-mat-def by auto
    show finite (set (rows A)) by auto
qed
definition row-space B=span (set (rows B))
definition col-space B}=\mathrm{ span (set (cols B))
lemma row-space-eq-col-space-transpose:
    shows row-space A = col-space AT
    unfolding col-space-def row-space-def cols-transpose ..
lemma col-space-eq-row-space-transpose:
    shows col-space A = row-space AT
    unfolding col-space-def row-space-def Matrix.rows-transpose ..
lemma col-space-eq:
    assumes A: A \in carrier-mat n nc
    shows col-space A = {y\incarrier-vec (dim-row A). \existsx\incarrier-vec (dim-col A).
A*v}x=y
proof -
    let ?ws = cols A
    have set-cols-in: set (cols A)\subseteqcarrier-vec n using A unfolding cols-def by
auto
    have lincomb f S\incarrier-vec (dim-row A) if finite S and S:S\subseteq set (cols A)
for fS
```

using lincomb-closed $A$
by (metis (full-types) $S$ carrier-matD(1) cols-dim lincomb-closed subsetCE subsetI)
moreover have $\exists x \in$ carrier-vec (dim-col $A$ ). $A *_{v} x=\operatorname{lincomb} f S$
if fin-S: finite $S$ and $S: S \subseteq \operatorname{set}(\operatorname{cols} A)$ for $f S$
proof -
let ? $g=(\lambda v$. if $v \in S$ then $f v$ else 0$)$
let ? $g^{\prime}=(\lambda i$. if $\exists j<i$. ? ws ! $i=$ ?ws ! $j$ then 0 else ? $g(? w s!i))$
let ? $Z=$ set ? $w s-S$
have union: set ? ws $=S \cup$ ? $Z$ using $S$ by auto
have inter: $S \cap ? Z=\{ \}$ by auto
have lincomb f $S=$ lincomb ? $S$ by (rule lincomb-cong, insert set-cols-in $A S$, auto)
also have $\ldots=$ lincomb ? $g(S \cup ? Z)$
by (rule lincomb-clean[symmetric], insert set-cols-in A $S$ fin-S, auto)
also have $\ldots=$ lincomb ? $g($ set ?ws) using union by auto
also have $\ldots=$ lincomb-list $? g^{\prime}$ ? ws
by (rule lincomb-as-lincomb-list[OF set-cols-in])
also have $\ldots=$ mat-of-cols $n$ ? ws $*_{v}$ vec (length?ws) ? $g^{\prime}$
by (rule lincomb-list-as-mat-mult, insert set-cols-in A, auto)
also have $\ldots=A *_{v}$ (vec (length ? ws) ? $g^{\prime}$ ) using mat-of-cols-cols $A$ by auto
finally show ?thesis by auto
qed
moreover have $\exists f S . A *_{v} x=$ lincomb $f S \wedge$ finite $S \wedge S \subseteq$ set (cols $A$ )
if $A x: A * v x \in$ carrier-vec (dim-row $A$ ) and $x: x \in \operatorname{carrier-vec~(dim-col~} A$ ) for $x$
proof -
let $? c=\lambda i . x \$ i$
have $x$-vec: vec (length ?ws) ?c $=x$ using $x$ by auto
have $A *_{v} x=$ mat-of-cols $n$ ? ws $*_{v}$ vec (length ?ws) ?c using mat-of-cols-cols $A$ x-vec by auto
also have $\ldots=$ lincomb-list ?c ?ws
by (rule lincomb-list-as-mat-mult[symmetric], insert set-cols-in A, auto)
also have $\ldots=$ lincomb ( $m k$-coeff ?ws ?c) (set ?ws)
by (rule lincomb-list-as-lincomb, insert set-cols-in A, auto)
finally show ?thesis by auto
qed
ultimately show ?thesis unfolding col-space-def span-def by auto qed
lemma vector-space-row-space:
assumes $A$ : $A \in$ carrier-mat nr $n$
shows vectorspace class-ring (vs (row-space A))
proof -
have fin: finite (set (rows A)) by auto
have $s$ : set (rows $A$ ) $\subseteq$ carrier $V$ using $A$ unfolding rows-def by auto
have span-vs $($ set $($ rows $A))=v s($ span $($ set $($ rows $A)))$ by auto
moreover have vectorspace class-ring (span-vs (set (rows A)))
using fin s span-is-subspace subspace-def subspace-is-vs by simp
ultimately show ?thesis unfolding row-space-def by auto qed
lemma row-space-eq:
assumes $A: A \in$ carrier-mat $n r n$
shows row-space $A=\{w \in \operatorname{carrier}-v e c(\operatorname{dim}-c o l ~ A) . \exists y \in \operatorname{carrier-vec}(\operatorname{dim}-r o w A)$. $\left.A^{T} *_{v} y=w\right\}$
using $A$ col-space-eq unfolding row-space-eq-col-space-transpose by auto
lemma row-space-is-preserved:
assumes inv- $P$ : invertible-mat $P$ and $P: P \in$ carrier-mat $m m$ and $A: A \in$ carrier-mat m $n$
shows row-space $(P * A)=$ row-space $A$
proof -
have $A t: A^{T} \in$ carrier-mat $n m$ using $A$ by auto
have Pt: $P^{T} \in$ carrier-mat $m$ using $P$ by auto
have $P A: P * A \in$ carrier-mat $m n$ using $P A$ by auto
have $w \in$ row-space $A$ if $w$ : $w \in$ row-space $(P * A)$ for $w$
proof -
have $w$-carrier: $w \in$ carrier-vec (dim-col $(P * A))$
using $w$ mult-carrier-mat $[O F P A]$ row-space-eq by auto
from that and this obtain $y$ where $y: y \in \operatorname{carrier}-v e c(\operatorname{dim}$-row $(P * A)$ )
and $w$-By: $w=(P * A)^{T} *_{v} y$ unfolding row-space-eq[OF PA] by blast
have $y m: y \in$ carrier-vec $m$ using $y P t$ by auto
have $w=\left((P * A)^{T}\right) *_{v} y$ using $w$ - $B y$.
also have $\ldots=\left(A^{T} * P^{T}\right) *_{v} y$ using transpose-mult $[O F P A]$ by auto
also have $\ldots=A^{T} *_{v}\left(P^{T} *_{v} y\right)$ by (rule assoc-mult-mat-vec[OF At Pt], insert Pt $y$, auto)
finally show $w \in$ row-space $A$ unfolding row-space-eq[OF A] using At Pt ym by auto
qed
moreover have $w \in$ row-space $(P * A)$ if $w: w \in$ row-space $A$ for $w$
proof -
have $w$-carrier: $w \in$ carrier-vec (dim-col $A$ ) using $w A$ unfolding row-space-eq[OF
$A]$ by auto
obtain $P^{\prime}$ where $P P^{\prime}$ : inverts-mat $P P^{\prime}$ and $P^{\prime} P$ : inverts-mat $P^{\prime} P$
using inv- $P$ P unfolding invertible-mat-def by blast
have $P^{\prime}: P^{\prime} \in$ carrier-mat $m m$ using $P P^{\prime} P^{\prime} P P$ unfolding inverts-mat-def
by (metis carrier-matD (1) carrier-matD(2) carrier-mat-triv index-mult-mat(3) index-one-mat(3))
from that obtain $y$ where $y: y \in$ carrier-vec (dim-row $A$ ) and
$w-A y: w=A^{T} *_{v} y$ unfolding row-space-eq[OF A] by blast
have Py: $\left(P^{T} *_{v} y\right) \in$ carrier-vec $m$ using $P^{\prime} y A$ by auto
have $w=A^{T} *_{v} y$ using $w-A y$.
also have $\ldots=\left(\left(P^{\prime} * P\right) * A\right)^{T} *_{v} y$
using $P^{\prime} P$ left-mult-one-mat $A P^{\prime}$ unfolding inverts-mat-def by auto
also have $\ldots=\left(\left(P^{\prime} *(P * A)\right)^{T}\right) *_{v} y$ using assoc-mult-mat-vec $P^{\prime} P A$ by auto
also have $\ldots=\left((P * A)^{T} * P^{\prime T}\right) *_{v} y$ using transpose-mult $P$ A $P^{\prime}$ mult-carrier-mat by metis
also have $\ldots=(P * A)^{T} *_{v}\left(P^{T T} *_{v} y\right)$
using assoc-mult-mat-vec A P $P^{\prime}$ y mult-carrier-mat
by (smt carrier-matD (1) transpose-carrier-mat)
finally show $w \in$ row-space $(P * A)$
unfolding row-space-eq[OF PA]
using Py w-carrier A P by fastforce
qed
ultimately show ?thesis by auto
qed
end
context vec-module begin
lemma $R$-sumlist $[$ simp $]:$ R.sumlist $=$ sum-list
proof (intro ext)
fix $x s$
show R.sumlist xs $=$ sum-list $x s$ by (induct xs, auto)
qed
lemma sumlist-dim: assumes $\bigwedge x . x \in$ set $x s \Longrightarrow x \in$ carrier-vec $n$
shows dim-vec (sumlist xs) $=n$
using sumlist-carrier assms
by fastforce
lemma sumlist-vec-index: assumes $\bigwedge x . x \in$ set $x s \Longrightarrow x \in$ carrier-vec $n$ and $i<n$
shows sumlist $x s \$ i=\operatorname{sum-list}(\operatorname{map}(\lambda x . x \$ i) x s)$
unfolding M.sumlist-def using assms(1) proof (induct xs)
case (Cons a xs)
hence cond: $\bigwedge x . x \in$ set $x s \Longrightarrow x \in$ carrier-vec $n$ by auto
from Cons (1) [OF cond] have $I H: f o l d r(+) x s\left(O_{v} n\right) \$ i=\left(\sum x \leftarrow x s . x \$ i\right)$ by auto
have $\left(a+\right.$ foldr $\left.(+) x s\left(O_{v} n\right)\right) \$ i=a \$ i+\left(\sum x \leftarrow x s . x \$ i\right)$
apply (subst index-add-vec) unfolding $I H$
using sumlist-dim[OF cond,unfolded M.sumlist-def] assms by auto
then show? case by auto next
case Nil thus ?case using assms by auto
qed
lemma scalar-prod-left-sum-distrib:
assumes vs: $\bigwedge v . v \in$ set vvs $\Longrightarrow v \in$ carrier-vec $n$ and $w: w \in$ carrier-vec $n$
shows sumlist vvs $\cdot w=\operatorname{sum-list}(\operatorname{map}(\lambda v . v \cdot w) v v s)$
using vs
proof (induct vvs)
case (Cons vvs)
from Cons have $v: v \in$ carrier-vec $n$ and vs: sumlist $v s \in$ carrier-vec $n$

```
    by (auto intro!: sumlist-carrier)
    have sumlist (v# vs) • w = sumlist ([v] @ vs) \cdot w by auto
    also have ... = (v + sumlist vs ) \cdot w
    by (subst sumlist-append, insert Cons v vs, auto)
    also have }\ldots=v\cdotw+(sumlist vs • w
    by (rule add-scalar-prod-distrib[OF v vs w])
    finally show ?case using Cons by auto
qed (insert w, auto)
lemma scalar-prod-right-sum-distrib:
    assumes vs: \v.v\in set vvs \Longrightarrowv\incarrier-vec n and w: w\incarrier-vec n
    shows w\cdotsumlist vvs = sum-list (map (\lambdav.w\cdotv) vvs)
    by (subst comm-scalar-prod[OF w sumlist-carrier], insert vs w, force,
    subst scalar-prod-left-sum-distrib[OF vs w], force,
    rule arg-cong[of - sum-list], rule nth-equalityI,
    auto simp: set-conv-nth intro!: comm-scalar-prod)
lemma lincomb-list-add-vec-2: assumes us: set us\subseteqcarrier-vec n
    and x:x= lincomb-list lc (us [i:=us!i+c\cdot.v us!j])
    and i:j< length us i< length us i\not=j
shows }x=lincomb-list (lc (j:= lc j+lci*c)) us (is - = ? x)
proof -
    let ?xx = lc j + lc i*c
    let ?i=us!i
    let ?j = us! j
    let ?v = ?i + c \cdotv ?j
    let ?ws = us [i:= us! i + c \cdotv us! j]
    from us have usk: k< length us \Longrightarrowus!k\in carrier-vec n for k by auto
    from usk i have ij: ?i i carrier-vec n ?j f carrier-vec n by auto
    hence v:c c.v ?j \in carrier-vec n ?v \in carrier-vec n by auto
    with us have ws: set ?ws \subseteq carrier-vec n unfolding set-conv-nth using i
        by (auto, rename-tac k, case-tac k=i, auto)
    from us have us':}\forallw\inset us. dim-vec w=n by aut
    from ws have ws':}\forallw\inset ?ws. dim-vec w=n by aut
    have mset: mset-set {0..<length us }}={#i#}+{#j#}+(mset-set ({0..<length
us} - {i,j}))
    by (rule multiset-eqI, insert i, auto, rename-tac x, case-tac x }\in{0..< length
us}, auto)
    define M2 where M2 = M.summset
        {#lc ia \cdotv ?ws ! ia. ia }\in#\mathrm{ mset-set ({0..<length us } - {i,j})#}
    define M1 where M1 = M.summset {#(ifi=j then ?xx else lc i)}\cdot\mp@subsup{}{v}{}\mathrm{ us !i.i
\in# mset-set ({0..<length us} - {i,j})#}
    have M1: M1 \in carrier-vec n unfolding M1-def using usk by fastforce
    have M2: M1 = M2 unfolding M2-def M1-def
        by (rule arg-cong[of - - M.summset], rule multiset.map-cong0, insert i usk,
auto)
```



```
        unfolding x lincomb-list-def M2 M2-def
        apply (subst sumlist-as-summset, (insert us ws iv vj, auto simp: set-conv-nth)[1],
```

insert $i$ ij v us ws usk,
simp add: mset smult-add-distrib-vec[OF ij(1) v(1)])
by (subst M.summset-add-mset, auto) +
have $x 2: ? x=? x x \cdot v ? j+\left(l c i{ }_{v} ? i+M 1\right)$
unfolding x lincomb-list-def M1-def
apply (subst sumlist-as-summset, (insert us ws iv ij, auto simp: set-conv-nth)[1],
insert $i$ ij v us ws usk,
simp add: mset smult-add-distrib-vec $[O F i j(1) v(1)])$
by (subst M.summset-add-mset, auto)+
show ?thesis unfolding $x 1$ x2 using M1 ij
by (intro eq-vecI, auto simp: field-simps)
qed
lemma lincomb-list-add-vec-1: assumes us: set us $\subseteq$ carrier-vec $n$
and $x: x=$ lincomb-list lc us
and $i: j<$ length us $i<$ length us $i \neq j$
shows $x=$ lincomb-list $(l c(j:=l c j-l c i * c))(u s[i:=u s!i+c \cdot v u s!j])$ (is
$-=? x$
proof -
let $? i=u s!i$
let ? $j=u s!j$
let ? $v=? i+c \cdot v$ ? $j$
let ?ws $=u s[i:=u s!i+c \cdot v u s!j]$
from us have usk: $k<$ length $u s \Longrightarrow u s!k \in$ carrier-vec $n$ for $k$ by auto
from usk $i$ have $i j$ : ? $i \in$ carrier-vec $n ? j \in$ carrier-vec $n$ by auto
hence $v: c \cdot v$ ? $j \in$ carrier-vec $n ? v \in$ carrier-vec $n$ by auto
with us have ws: set ? ws $\subseteq$ carrier-vec $n$ unfolding set-conv-nth using $i$
by (auto, rename-tac $k$, case-tac $k=i$, auto)
from $u s$ have $u s^{\prime}: \forall w \in$ set us. dim-vec $w=n$ by auto
from ws have $w s^{\prime}: \forall w \in$ set ? ws. dim-vec $w=n$ by auto
have mset: mset-set $\{0 . .<$ length $u s\}=\{\# i \#\}+\{\# j \#\}+($ mset-set $(\{0 . .<$ length $u s\}-\{i, j\}))$
by (rule multiset-eqI, insert $i$, auto, rename-tac $x$, case-tac $x \in\{0 . .<$ length us \}, auto)
define M2 where $M 2=$.summset
$\left\{\#\left(\right.\right.$ if ia $=j$ then lc $j-l c i * c$ else lc ia) $\cdot{ }_{v}$ ? ws ! ia
. ia $\in \#$ mset-set $(\{0 . .<$ length us $\}-\{i, j\}) \#\}$
define M1 where M1 = M.summset $\left\{\# l c i \cdot{ }_{v}\right.$ us $!i . i \in \#$ mset-set $(\{0 . .<$ length $u s\}-\{i, j\}) \#\}$
have M1: M1 $\in$ carrier-vec $n$ unfolding M1-def using usk by fastforce
have M2: M1 = M2 unfolding M2-def M1-def
by (rule arg-cong[of - M.summset], rule multiset.map-cong0, insert i usk, auto)
have $x 1: x=l c j \cdot v ? j+(l c i \cdot v ? i+M 1)$
unfolding $x$ lincomb-list-def M1-def
apply (subst sumlist-as-summset, (insert us ws iv ij, auto simp: set-conv-nth) [1], insert $i$ ij v us ws usk,
simp add: mset smult-add-distrib-vec[OF ij(1) v(1)])
by (subst M.summset-add-mset, auto)+


```
    unfolding x lincomb-list-def M2 M2-def
    apply (subst sumlist-as-summset, (insert us ws iv ij, auto simp: set-conv-nth)[1],
insert i ij v us ws usk,
            simp add: mset smult-add-distrib-vec[OF ij(1) v(1)])
    by (subst M.summset-add-mset, auto)+
    show ?thesis unfolding x1 x2 using M1 ij
    by (intro eq-vecI, auto simp: field-simps)
qed
end
context vec-space
begin
lemma add-vec-span: assumes us: set us \subseteqcarrier-vec n
    and i:j< length us i< length us i\not=j
shows span (set us) = span (set (us [i:=us!i+c\cdotv us!j])) (is - = span (set
?ws))
proof -
    let ?i=us!i
    let ? j = us! j
    let ?v = ?i + c vv?j
    from us i have ij:?i}\in\mathrm{ carrier-vec n ?j }\in\mathrm{ carrier-vec n by auto
    hence v: ?v\in carrier-vec n by auto
    with us have ws: set ?ws \subseteq carrier-vec n unfolding set-conv-nth using i
    by (auto, rename-tac k, case-tac k=i, auto)
    have span (set us) = span-list us unfolding span-list-as-span[OF us] ..
    also have ... = span-list ?ws
    proof -
        {
        fix }
        assume }x\in\mathrm{ span-list us
        then obtain lc where x = lincomb-list lc us by (metis in-span-listE)
        from lincomb-list-add-vec-1[OF us this i, of c]
        have }x\in\mathrm{ span-list ?ws unfolding span-list-def by auto
    }
    moreover
    {
        fix }
        assume x f span-list?ws
        then obtain lc where x = lincomb-list lc ?ws by (metis in-span-listE)
        from lincomb-list-add-vec-2[OF us this i]
        have }x\in\mathrm{ span-list us unfolding span-list-def by auto
    }
    ultimately show ?thesis by blast
    qed
    also have ... = span (set ?ws) unfolding span-list-as-span[OF ws] ..
    finally show ?thesis.
qed
```

```
lemma prod-in-span[intro!]:
    assumes b\incarrier-vec n S\subseteqcarrier-vec na=0\veeb\in span S
    shows }a\cdot\mp@subsup{}{v}{}b\in\operatorname{span}
proof(cases a=0)
    case True
    then show ?thesis by (auto simp:lmult-0[OF assms(1)] span-zero)
next
    case False with assms have b\in span S by auto
    from this[THEN in-spanE]
    obtain aa A where a[intro!]: b= lincomb aa A finite A A\subseteqS by auto
    hence [intro!]:(\lambdav. aa v 汭v) \inA->carrier-vec n using assms by auto
    show ?thesis proof
        show }a\cdotvb=lincomb (\lambdav.a*aav) A using a(1) unfolding lincomb-def
smult-smult-assoc[symmetric]
            by(subst finsum-smult[symmetric]) force+
    qed auto
qed
lemma det-nonzero-congruence:
    assumes eq:A*M=B*M and det:det (M::'a mat) }=
    and M:M\in carrier-mat n n and carr:A \in carrier-mat n n B \in carrier-mat n
n
    shows }A=
proof -
    have 1m}n\in\mathrm{ carrier-mat n n by auto
    from det-non-zero-imp-unit[OF M det] gauss-jordan-check-invertable[OF M this]
    have gj-fst:(fst (gauss-jordan M (1m n)) = 1m n) by metis
    define Mi where Mi = snd (gauss-jordan M (1m n))
    with gj-fst have gj:gauss-jordan M (1m n)=(1m n,Mi)
        unfolding fst-def snd-def by (auto split:prod.split)
    from gauss-jordan-compute-inverse(1,3)[OF M gj]
    have Mi:Mi\in carrier-mat n n and is1:M*Mi=1m n by metis+
    from arg-cong[OF eq, of \lambdaM.M*Mi]
    show }A=B\mathrm{ unfolding carr[THEN assoc-mult-mat[OF - M Mi]] is1 carr[THEN
right-mult-one-mat].
qed
lemma mat-of-rows-mult-as-finsum:
    assumes v\in carrier-vec (length lst) \bigwedgei. i< length lst \Longrightarrowlst ! i\in carrier-vec
n
    defines fl\equiv\operatorname{sum}(\lambdai. ifl=lst ! i then v $ i else 0) {0..<length lst }
    shows mat-of-cols-mult-as-finsum:mat-of-cols n lst *vv = lincomb f (set lst)
proof -
    from assms have }\foralli<length lst. lst ! i\in carrier-vec n by blas
    note an = all-nth-imp-all-set[OF this] hence slc:set lst }\subseteq\mathrm{ carrier-vec n by auto
    hence dn [simp]:\ x. x set lst \Longrightarrowdim-vec }x=n\mathrm{ by auto
    have dl [simp]:dim-vec (lincomb f (set lst)) = n using an by (intro lincomb-dim,auto)
    show ?thesis proof
```

```
    show dim-vec (mat-of-cols n lst *v v) = dim-vec (lincomb f (set lst)) using
assms(1,2) by auto
    fix }i\mathrm{ assume i:i<dim-vec (lincomb f (set lst)) hence }\mp@subsup{i}{}{\prime}:i<n\mathrm{ by auto
    with an have fcarr:(\lambdav.fv vvv) \in set lst }->\mathrm{ carrier-vec n by auto
    from i' have (mat-of-cols n lst *vv)$ i= row (mat-of-cols n lst) i}\cdotv\mathrm{ by auto
    also have ... = (\sumia=0..<dim-vec v.lst!ia $i*v$ ia)
        unfolding mat-of-cols-def row-def scalar-prod-def
        apply(rule sum.cong[OF refl]) using i an assms(1) by auto
    also have ... = (\sumia=0..<length lst.lst!ia $i*v$ia) using assms(1)
by auto
    also have ... = (\sumx\inset lst. f x*x $ i)
            unfolding f-def sum-distrib-right apply (subst sum.swap)
            apply(rule sum.cong[OF refl])
            unfolding if-distrib if-distribR mult-zero-left sum.delta[OF finite-set] by auto
    also have ... = (\sumx\inset lst. ( fx\cdotv x)$ i)
            apply(rule sum.cong[OF refl],subst index-smult-vec) using i slc by auto
    also have \ldots...=(\bigoplus\mp@subsup{V}{}{v\inset lst.f v v}v
            unfolding finsum-index[OF i' fcarr slc] by auto
    finally show (mat-of-cols n lst *vv) $i=lincomb f(set lst)$ i
            by (auto simp:lincomb-def)
        qed
qed
end
end
```


## 16 Gram-Schmidt Orthogonalization

This theory provides the Gram-Schmidt orthogonalization algorithm, that takes the conjugate operation into account. It works over fields like the rational, real, or complex numbers.

```
theory Gram-Schmidt
imports
    VS-Connect
    Missing-VectorSpace
    Conjugate
begin
```


### 16.1 Orthogonality with Conjugates

definition corthogonal vs $\equiv$
$\forall i<$ length vs. $\forall j<$ length vs. vs $!i \cdot c$ vs $!j=0 \longleftrightarrow i \neq j$
lemma corthogonalD $[$ elim]:
corthogonal vs $\Longrightarrow i<$ length $v s \Longrightarrow j<$ length $v s \Longrightarrow$ vs $!i \cdot c$ vs $!j=0 \longleftrightarrow i \neq j$
unfolding corthogonal-def by auto

```
lemma corthogonalI[intro]:
    \((\bigwedge i j . i<\) length \(v s \Longrightarrow j<\) length \(v s \Longrightarrow v s!i \cdot c v s!j=0 \longleftrightarrow i \neq j) \Longrightarrow\)
    corthogonal vs
    unfolding corthogonal-def by auto
lemma corthogonal-distinct: corthogonal us \(\Longrightarrow\) distinct us
proof (induct us)
    case (Cons u us)
        have \(u \notin\) set us
        proof
            assume \(u\) : set us
            then obtain \(j\) where \(u j: u=u s!j\) and \(j: j<\) length \(u s\)
                using in-set-conv-nth by metis
            hence \(j^{\prime}: j+1<\) length ( \(u \# u s\) ) by auto
            have \(u \cdot c u s!j=0\)
                using corthogonalD[OF Cons(2) - \(j^{\prime}\),of 0] by auto
            hence \(u \cdot c u=0\) using \(u j\) by simp
            thus False using corthogonalD[OF Cons(2), of 00\(]\) by auto
    qed
    moreover have distinct us
    proof (rule Cons(1), intro corthogonalI)
            fix \(i j\) assume \(i<\) length (us) \(j<\) length (us)
            hence len: \(i+1<\) length (u\#us) \(j+1<\) length ( \(u \# u s\) ) by auto
            show \((u s!i \cdot c u s!j=0)=(i \neq j)\)
                using corthogonalD[OF Cons(2) len] by simp
    qed
    ultimately show ?case by simp
qed \(\operatorname{simp}\)
lemma corthogonal-sort:
    assumes dist': distinct us \({ }^{\prime}\)
        and mem: set \(u s=\) set \(u s^{\prime}\)
    shows corthogonal us \(\Longrightarrow\) corthogonal us'
proof
    assume orth: corthogonal us
    hence dist: distinct us using corthogonal-distinct by auto
    fix \(i^{\prime} j^{\prime}\) assume \(i^{\prime}: i^{\prime}<\) length \(u s^{\prime}\) and \(j^{\prime}: j^{\prime}<\) length \(u s^{\prime}\)
    obtain \(i\) where \(i i^{\prime}: u s!i=u s^{\prime}!i^{\prime}\) and \(i: i<\) length us
    using mem \(i^{\prime}\) in-set-conv-nth by metis
    obtain \(j\) where \(j j^{\prime}: u s!j=u s^{\prime}!j^{\prime}\) and \(j: j<\) length \(u s\)
    using mem \(j^{\prime}\) in-set-conv-nth by metis
    from corthogonalD[OF orth \(i j]\)
    have (us!i \(\cdot c\) us! \(j=0)=(i \neq j)\).
    hence \(\left(u s^{\prime}!i^{\prime} \cdot\right.\) c us \(\left.!j^{\prime}=0\right)=(i \neq j)\) using \(i i^{\prime} j j^{\prime}\) by auto
    also have \(\ldots=(u s!i \neq u s!j)\) using nth-eq-iff-index-eq dist \(i j\) by auto
    also have \(\ldots=\left(u s^{\prime}!i^{\prime} \neq u s^{\prime}!j^{\prime}\right)\) using \(i i^{\prime} j j^{\prime}\) by auto
    also have \(\ldots=\left(i^{\prime} \neq j^{\prime}\right)\) using nth-eq-iff-index-eq dist' \(i^{\prime} j^{\prime}\) by auto
    finally show \(\left(u s^{\prime}!i^{\prime} \cdot c u s^{\prime}!j^{\prime}=0\right)=\left(i^{\prime} \neq j^{\prime}\right)\).
```


## qed

### 16.2 The Algorithm

fun adjuster :: nat $\Rightarrow{ }^{\prime} a::$ conjugatable-field vec $\Rightarrow{ }^{\prime} a$ vec list $\Rightarrow{ }^{\prime} a$ vec
where adjuster $n$ w []$=0_{v} n$
adjuster $n w(u \# u s)=-(w \cdot c u) /(u \cdot c u) \cdot{ }_{v} u+$ adjuster $n w u s$
The following formulation is easier to analyze, but outputs of the subroutine should be properly reversed.

```
fun gram-schmidt-sub
    where gram-schmidt-sub n us [] = us
    | gram-schmidt-sub n us (w # ws) =
        gram-schmidt-sub n ((adjuster n w us +w) #us) ws
```

definition gram-schmidt :: nat $\Rightarrow{ }^{\prime} a$ :: conjugatable-field vec list $\Rightarrow$ 'a vec list
where gram-schmidt $n$ ws $=$ rev (gram-schmidt-sub $n[]$ ws)

The following formulation requires no reversal.

## fun gram-schmidt-sub2

where gram-schmidt-sub2 $n$ us []$=[]$
| gram-schmidt-sub2 $n$ us $(w \# w s)=$
(let $u=$ adjuster $n w u s+w$ in
u \# gram-schmidt-sub2 $n$ ( $u$ \# us) ws)
lemma gram-schmidt-sub-eq:
rev (gram-schmidt-sub n us ws) = rev us @ gram-schmidt-sub2 nus ws by (induct ws arbitrary:us, auto simp:Let-def)
lemma gram-schmidt-code[code]:
gram-schmidt $n$ ws $=$ gram-schmidt-sub2 $n$ [] ws
unfolding gram-schmidt-def
apply(subst gram-schmidt-sub-eq) by simp

### 16.3 Properties of the Algorithms

locale cof-vec-space $=$ vec-space $f$-ty for
$f$-ty :: 'a :: conjugatable-ordered-field itself
begin
lemma adjuster-finsum:
assumes $U$ : set us $\subseteq$ carrier-vec $n$
and dist: distinct (us :: 'a vec list)
shows adjuster $n$ w us $=$ finsum $V(\lambda u .-(w \cdot c u) /(u \cdot c u) \cdot v u)(s e t u s)$
using assms
proof (induct us)
case Cons show ?case unfolding set-simps
by (subst finsum-insert[OF finite-set], insert Cons, auto)
qed $\operatorname{simp}$

```
lemma adjuster-lincomb:
    assumes w:(w :: 'a vec) : carrier-vec n
        and us: set (us :: 'a vec list)\subseteq carrier-vec n
        and dist: distinct us
    shows adjuster n w us = lincomb (\lambdau. - (w ccu)/(u\cdotcu)) (set us)
        (is - = lincomb ?a -)
    using us dist unfolding lincomb-def
proof (induct us)
    case (Cons u us)
        let ?f = \lambdau. ?a u |}\mp@subsup{v}{}{\prime}
        have ?f : (set us) }->\mathrm{ carrier-vec n and ?f u:carrier-vec n using w Cons by
auto
        moreover have u\not\in set us using Cons by auto
        ultimately show ?case
            unfolding adjuster.simps
            unfolding set-simps
            using finsum-insert[OF finite-set] Cons by auto
qed simp
lemma adjuster-in-span:
    assumes w: (w :: 'a vec) : carrier-vec n
        and us: set (us :: 'a vec list)\subseteq carrier-vec n
        and dist: distinct us
    shows adjuster n w us: span (set us)
    using adjuster-lincomb[OF assms]
    unfolding finite-span[OF finite-set us] by auto
lemma adjuster-carrier[simp]:
    assumes w: (w:: 'a vec) : carrier-vec n
        and us: set (us :: 'a vec list)\subseteq carrier-vec n
        and dist: distinct us
    shows adjuster n w us : carrier-vec n
    using adjuster-in-span span-closed assms by auto
lemma adjust-not-in-span:
    assumes w[simp]: ( w :: 'a vec) : carrier-vec n
        and us: set (us :: 'a vec list)\subseteq carrier-vec n
        and dist: distinct us
        and ind:w\not\in span(set us)
    shows adjuster n w us+w\not\in span (set us)
    using span-add[OF us adjuster-in-span[OF w us dist] w]
    using comm-add-vec ind by auto
lemma adjust-not-mem:
    assumes w[simp]: (w :: 'a vec) : carrier-vec n
        and us: set (us :: 'a vec list)\subseteq carrier-vec n
        and dist: distinct us
        and ind: w }\not=\operatorname{span}(\mathrm{ set us)
```

```
    shows adjuster n wus +w\not\in set us
    using adjust-not-in-span[OF assms] span-mem[OF us] by auto
lemma adjust-in-span:
    assumes w[simp]:(w :: 'a vec) : carrier-vec n
    and us: set (us :: 'a vec list)\subseteqcarrier-vec n
    and dist: distinct us
    shows adjuster n wus+w: span (insert w (set us)) (is ?v + - : span ?U)
proof -
    let ?a = \lambdau. - (w ccu)/(u\cdotc u)
    have ?v = lincomb ?a (set us) using adjuster-lincomb[OF assms].
    hence vU: ?v : span (set us) unfolding finite-span[OF finite-set us] by auto
    hence v[simp]: ?v : carrier-vec n using span-closed[OF us] by auto
    have vU': ?v : span ?U using vU span-is-monotone[OF subset-insertI] by auto
    have {w}\subseteq?U by simp
    from span-is-monotone[OF this]
    have wU':w: span ?U using span-self [OF w] by auto
    have ?U\subseteq carrier-vec n using us w by simp
    from span-add[OF this w\mp@subsup{U}{}{\prime}v] v\mp@subsup{U}{}{\prime}}\mathrm{ comm-add-vec[OF w]
    show ?thesis by simp
qed
lemma adjust-not-lindep:
    assumes w[simp]:(w :: 'a vec) : carrier-vec n
        and us: set (us :: 'a vec list)\subseteqcarrier-vec n
        and dist: distinct us
        and wus:w\not\in\operatorname{span (set us)}
    and ind: ~ lin-dep (set us)
    shows ~ lin-dep (insert (adjuster n w us + w) (set us))
        (is ~ - (insert ?v -))
proof -
    have v: ?v : carrier-vec n using assms by auto
    have ?v & span (set us)
        using adjust-not-in-span[OF w us dist wus]
        using comm-add-vec[OF adjuster-carrier[OF w us dist] w] by auto
    thus ?thesis
        using lin-dep-iff-in-span[OF us ind v] adjust-not-mem[OF w us dist wus] by
auto
qed
lemma adjust-preserves-span:
    assumes w[simp]:(w :: 'a vec) : carrier-vec n
    and us: set (us :: 'a vec list)\subseteqcarrier-vec n
    and dist: distinct us
    shows w: span (set us)\longleftrightarrow adjuster n wus + w: span (set us)
    (is - \longleftrightarrow?v+-:-)
proof -
```

```
    have ?v : span (set us)
    using adjuster-lincomb[OF assms]
    unfolding finite-span[OF finite-set us] by auto
    hence \([\) simp \(]\) : ?v : carrier-vec \(n\) using span-closed \([\) OF us] by auto
    show ?thesis
    using span-add[OF us adjuster-in-span [OF \(w\) us] w] comm-add-vec \([O F w]\) dist
    by auto
qed
lemma in-span-adjust:
    assumes \(w[\operatorname{simp}]:(w::\) 'a vec) : carrier-vec \(n\)
    and us: set (us :: 'a vec list) \(\subseteq\) carrier-vec \(n\)
    and dist: distinct us
    shows \(w\) : span (insert (adjuster \(n w u s+w)(\) set us))
    (is - : span (insert ? \(v-\)-))
proof -
    have \(v\) : ?v: carrier-vec \(n\) using assms by auto
    have \(a[\) simp \(]\) : adjuster \(n\) w us : carrier-vec \(n\)
    and neg: - adjuster \(n\) w us: carrier-vec \(n\) using assms by auto
    hence \(v U\) : insert ? \(v(\) set us \() \subseteq\) carrier-vec \(n\) using us by auto
    have \(a S\) : adjuster \(n\) w us : span (insert ?v (set us))
        using adjuster-in-span[OF wus] span-is-monotone[OF subset-insertI] dist
        by auto
    have negS: - adjuster \(n\) w us : span (insert ?v (set us))
    using span-neg[OF \(v U a S]\) us by simp
    have \([\) simp \(]\) :- adjuster \(n\) wus \(+(\) adjuster \(n w u s+w)=w\)
    unfolding \(a\)-assoc[OF neg a \(w\),symmetric \(]\) by simp
    have \(\{? v\} \subseteq\) insert ?v (set us) by simp
    from span-is-monotone[OF this]
    have \(v S\) : ?v : span (insert ?v (set us)) using span-self \([O F v]\) by auto
    thus ?thesis using span-add[OF vU negS \(v]\) by auto
qed
lemma adjust-zero:
    assumes \(U\) : set (us :: 'a vec list) \(\subseteq\) carrier-vec \(n\)
        and orth: corthogonal us
        and \(w[\) simp \(]: w:\) carrier-vec \(n\)
        and \(i: i<\) length us
    shows (adjuster \(n\) w us \(+w\) ) \(\cdot c\) us \(!i=0\)
proof -
    define \(u\) where \(u=u s!i\)
    have \(u[\) simp \(]: u\) : carrier-vec \(n\) using \(i U u\)-def by auto
    hence cu[simp]: conjugate \(u\) : carrier-vec \(n\) by auto
    have \(u U\) : \(u\) : set us using \(i u\)-def by auto
    let ? \(g=\lambda u^{\prime}::^{\prime} a\) vec. \(\left(-\left(w \cdot c u^{\prime}\right) /\left(u^{\prime} \cdot c u^{\prime}\right) \cdot v u^{\prime}\right)\)
    have \(g: ? g:\) set us \(\rightarrow\) carrier-vec \(n\) using \(w U\) by auto
    hence carrier: finsum \(V\) ?g (set us) : carrier-vec \(n\) by simp
    let ?f \(=\lambda u^{\prime}\). ? \(g u^{\prime} \cdot c u\)
    let ? \(U=\) set \(u s-\{u\}\)
```

```
    { fix }\mp@subsup{u}{}{\prime}\mathrm{ assume }\mp@subsup{u}{}{\prime}:(\mp@subsup{u}{}{\prime}::'a vec) : carrier-vec n
    have [simp]: dim-vec }u=n\mathrm{ by auto
    have ?f }\mp@subsup{u}{}{\prime}=(-(w\cdotc\mp@subsup{u}{}{\prime})/(\mp@subsup{u}{}{\prime}\cdotc\mp@subsup{u}{}{\prime}))*(\mp@subsup{u}{}{\prime}\cdotcu
        using scalar-prod-smult-left[of u' conjugate u]
        unfolding carrier-vecD[OF u] carrier-vecD[OF u\ by auto
    } note conv = this
    have ?f :?U }->{0
proof (intro Pi-I)
    fix }\mp@subsup{u}{}{\prime}\mathrm{ assume }\mp@subsup{u}{}{\prime}Uu:\mp@subsup{u}{}{\prime}:\mathrm{ set us - {u}
    hence }\mp@subsup{u}{}{\prime}U:\mp@subsup{u}{}{\prime}:\mathrm{ set us by auto
    hence }\mp@subsup{u}{}{\prime}[\mathrm{ simp ]: u' : carrier-vec n using U by auto
    obtain j where j: j< length us and u'j: u' =us!j
        using }\mp@subsup{u}{}{\prime}U\mathrm{ in-set-conv-nth by metis
    have i\not=j using }\mp@subsup{u}{}{\prime}Uu\mp@subsup{u}{}{\prime}ju\mathrm{ -def by auto
    hence }\mp@subsup{u}{}{\prime}\cdotcu=
        unfolding }\mp@subsup{u}{}{\prime}j\mathrm{ using corthogonalD [OF orth j i] u-def by auto
    hence ?f }\mp@subsup{u}{}{\prime}=0\mathrm{ using mult-zero-right conv[OF u'] by auto
    thus ?f }\mp@subsup{u}{}{\prime}:{0}\mathrm{ by auto
qed
hence restrict ?f ?U = restrict ( }\lambdau.0)\mathrm{ ? U by force
hence sum?f?U = sum ( \lambdau.0) ?U
    by (intro R.finsum-restrict, auto)
hence fU'0: sum ?f ? U = 0 by auto
have }u\mp@subsup{U}{}{\prime}:u\not\in??U by aut
have set us = insert u ? U
    using insert-Diff-single uU by auto
    hence sum ?f (set us) = ?f u + sum ?f ?U
    using R.finsum-insert[OF-uU\ by auto
also have ... = ?f u using fU'0 by auto
also have ... = - (w ccu)/(u\cdotcu)*(u\cdotcu)
    using conv[OFu] by auto
finally have main: sum ?f (set us) = - (w\cdotcu)
    unfolding u-def
    by (simp add: i orth corthogonalD)
show ?thesis
    unfolding u-def[symmetric]
    unfolding adjuster-finsum[OF U corthogonal-distinct[OF orth]]
    unfolding add-scalar-prod-distrib[OF carrier w cu]
    unfolding finsum-scalar-prod-sum[OF g cu]
    unfolding main
    unfolding comm-scalar-prod[OF cu w]
    using left-minus by auto
qed
lemma adjust-nonzero:
    assumes U: set (us :: 'a vec list)\subseteqcarrier-vec n
        and dist: distinct us
        and w[simp]:w : carrier-vec n
    and wsU:w\not\in span (set us)
```

```
    shows adjuster n w us +w\not=\mp@subsup{0}{v}{}n(\mathrm{ is ? a + - = -)}
proof
    have [simp]: ?a : carrier-vec n using U dist by auto
    have [simp]: - ?a : carrier-vec n by auto
    have [simp]: ?a + w: carrier-vec n by auto
    assume ?a +w= 0}v
    hence - ?a = - ?a + (?a a+w) by auto
    also have ... = (-?a + ?a) + w apply(subst a-assoc) by auto
    also have - ?a + ?a = Ov n using r-neg[OF w] unfolding vec-neg[OF w] by
auto
    finally have - ?a = w by auto
    moreover have - ?a : span (set us)
        using span-neg[OF U adjuster-in-span[OF w U dist]] by auto
    ultimately show False using wsU by auto
qed
lemma adjust-orthogonal:
    assumes U: set (us :: 'a vec list)\subseteqcarrier-vec n
    and orth: corthogonal us
    and w[simp]:w:carrier-vec n
    and wsU:w\not\in span (set us)
    shows corthogonal ((adjuster n w us + w) # us)
    (is corthogonal (?aw # -))
proof
    have dist: distinct us using corthogonal-distinct orth by auto
    have aw[simp]:?aw : carrier-vec n using U dist by auto
    note adjust-nonzero[OF U dist w] wsU
    hence aw0: ?aw •c ?aw \not=0 using conjugate-square-eq-0-vec[OF aw] by auto
    fix ij assume i: i< length (?aw # us) and j: j< length (?aw # us)
    show ((?aw # us)!i c (?aw # us)! j=0)=(i\not=j)
    proof (cases i=0)
        case True note i0 = this
            show ?thesis
            proof (cases j=0)
                case True show ?thesis unfolding True i0 using aw0 by auto
                next case False
                        define }\mp@subsup{j}{}{\prime}\mathrm{ where }\mp@subsup{j}{}{\prime}=j-
                    hence jfold: j= j'+1 using False by auto
                    hence j': j' < length us using j by auto
                    show ?thesis unfolding i0 jfold
                    using adjust-zero[OF U orth w j`] by auto
            qed
    next case False
        define }\mp@subsup{i}{}{\prime}\mathrm{ where }\mp@subsup{i}{}{\prime}=i-
        hence ifold: i= i'+1 using False by auto
        hence }\mp@subsup{i}{}{\prime}:\mp@subsup{i}{}{\prime}<length us using i by aut
        have [simp]: us! i': carrier-vec n using U i' by auto
        hence cu': conjugate (us! ' ') : carrier-vec n by auto
        show ?thesis
```

```
    proof (cases j=0)
        case True
            { assume ?aw \cdotc us ! i'}=
                hence conjugate (?aw c c us ! i') = 0 using conjugate-zero by auto
                hence conjugate ?aw • us ! i'}=
                    using conjugate-sprod-vec[OF aw cu] by auto
        }
        thus ?thesis unfolding True ifold
        using adjust-zero[OF U orth w i']
        by (subst comm-scalar-prod[of-n], auto)
    next case False
        define j' where }\mp@subsup{j}{}{\prime}=j-
        hence jfold: j= j'+1 using False by auto
        hence j': j' < length us using }j\mathrm{ by auto
        show ?thesis
            unfolding ifold jfold
            using orth i' }\mp@subsup{j}{}{\prime}\mathrm{ by (auto simp: corthogonalD)
        qed
    qed
qed
lemma gram-schmidt-sub-span:
    assumes w[simp]:w : carrier-vec n
        and us: set us\subseteqcarrier-vec n
    and dist: distinct us
    shows span (set ((adjuster n wus + w) #us)) = span (set (w#us))
    (is span (set (?v # -)) = span ?wU)
proof (cases w: span (set us))
    case True
        hence ?v: span (set us)
            using adjust-preserves-span[OF assms] by auto
            thus ?thesis using already-in-span[OF us] True by auto next
    case False show ?thesis
    proof
            have wU:?wU\subseteq carrier-vec n using us by simp
            have vswU: ?v : span ?wU using adjust-in-span[OF assms] by auto
            hence v: ?v : carrier-vec n using span-closed[OF wU] by auto
            have wsvU:w : span (insert ?v (set us)) using in-span-adjust[OF assms].
            show span ?wU\subseteq span (set (?v #us))
                using span-swap[OF finite-set us w False v wsvU] by auto
            have ?v & span (set us)
                using False adjust-preserves-span[OF assms] by auto
            thus span (set (?v # us))\subseteq span ?wU
                using span-swap[OF finite-set us v-w] vswU by auto
    qed
qed
lemma gram-schmidt-sub-result:
    assumes gram-schmidt-sub n us ws =us'
```

and set $w s \subseteq$ carrier-vec $n$
and set $u s \subseteq$ carrier-vec $n$
and distinct (us@ws)
and $\sim \operatorname{lin}-\operatorname{dep}(\operatorname{set}(u s @ w s))$
and corthogonal us
shows set us ${ }^{\prime} \subseteq$ carrier-vec $n \wedge$
distinct $u s^{\prime} \wedge$
corthogonal us' $\wedge$
$\operatorname{span}(\operatorname{set}(u s @ w s))=\operatorname{span}\left(\right.$ set us $\left.{ }^{\prime}\right) \wedge$ length us' $=$ length us + length ws
using assms
proof (induct ws arbitrary: us us')
case (Cons w ws)
let ?v = adjuster $n$ w us
have $w W[\operatorname{simp}]$ : set $(w \# w s) \subseteq$ carrier-vec $n$ using Cons by simp
hence $W[$ simp $]$ : set ws $\subseteq$ carrier-vec $n$
and $w[\operatorname{simp}]: w:$ carrier-vec $n$ by auto
have $U[$ simp $]$ : set us $\subseteq$ carrier-vec $n$ using Cons by simp
have $U W$ : set $(u s @ w s) \subseteq$ carrier-vec $n$ by simp
have $w U:$ set $(w \# u s) \subseteq$ carrier-vec $n$ by simp
have dist: distinct (us @ w \# ws) using Cons by simp
hence dist- $U$ : distinct us
and dist- $W$ : distinct ws
and dist- $U W$ : distinct (us @ ws)
and $w-U: w \notin$ set us
and $w-W: w \notin$ set $w s$
and $w-U W: w \notin \operatorname{set}(u s @ w s)$ by auto
have ind: ~ lin-dep (set (us @ $w \#$ ws)) using Cons by simp
have $i n d-U: \sim \operatorname{lin-dep}$ (set us)
and ind-W: ~ lin-dep (set ws)
and $\operatorname{ind}-w U: \sim \operatorname{lin}-d e p($ insert $w($ set $u s))$
and $\operatorname{ind}-U W: \sim \operatorname{lin}-\operatorname{dep}(\operatorname{set}(u s @ w s))$
by (subst subset-li-is-li[OF ind];auto)+
have corth: corthogonal us using Cons by simp
have $U^{\prime}$ def: gram-schmidt-sub $n((? v+w) \# u s) w s=u s^{\prime}$ using Cons by simp
have $v$ : ? $v:$ carrier-vec $n$ using dist- $U$ by auto
hence $v w: ? v+w$ : carrier-vec $n$ by auto
hence $v w U:$ set $((? v+w) \# u s) \subseteq$ carrier-vec $n$ by auto
have $v s U: ? v: \operatorname{span}($ set us) using adjuster-in-span $[O F w]$ dist by auto
hence $v s U W:$ ?v : span (set (us@ ws))
using span-is-monotone[of set us set (us@ws)] by auto
have ws $U$ : w $\notin \operatorname{span}($ set $u s)$
using lin-dep-iff-in-span[OF $U$ ind- $U$ w w- $U]$ ind-w $U$ by auto
hence $v w U: ? v+w \notin \operatorname{span}($ set us) using adjust-not-in-span $[O F w U$ dist- $U]$ by auto
have $w \notin \operatorname{span}($ set (us@ws)) using lin-dep-iff-in-span $[O F-i n d-U W]$ dist ind by auto
hence span: ?v $+w \notin \operatorname{span}($ set $(u s @ w s))$ using span-add $[O F U W$ vs $U W w]$ by

## auto

hence $v w U S$ : ?v $+w \notin$ set (us @ ws) using span-mem by auto
hence ind2: ~ lin-dep $($ set $(((? v+w) \# u s)$ @ ws $))$
using lin-dep-iff-in-span[OF $U W$ ind- $U W$ vw] span by auto
have $v w U$ : set $((? v+w) \# u s) \subseteq$ carrier-vec $n$ using $U w$ dist by auto
have dist2: distinct $(((? v+w) \# u s) @ w s)$ using dist vwUS by simp
have orth2: corthogonal ((adjuster $n w u s+w) \# u s)$
using adjust-orthogonal[OF U corth w wsU].
show ?case
using Cons(1)[OF U'def $W$ vwU dist2 ind2] orth2
using span-Un[OF vwU wU gram-schmidt-sub-span $[O F w U$ dist- $U] W W]$ by auto
qed $\operatorname{simp}$
lemma gram-schmidt-hd [simp]:
assumes $[$ simp $]: w:$ carrier-vec $n$ shows hd (gram-schmidt $n(w \# w s))=w$
unfolding gram-schmidt-code by simp
theorem gram-schmidt-result:
assumes ws: set ws $\subseteq$ carrier-vec $n$
and dist: distinct ws
and ind: ~ lin-dep (set ws)
and us: us = gram-schmidt $n$ ws
shows span (set ws) $=$ span (set us)
and corthogonal us
and set us $\subseteq$ carrier-vec $n$
and length us $=$ length ws
and distinct us
proof -
have main: gram-schmidt-sub $n$ [] ws $=$ rev us
using us unfolding gram-schmidt-def
using gram-schmidt-sub-eq by auto
have orth: corthogonal [] by auto
have span $($ set ws $)=\operatorname{span}($ set $($ rev us) $)$
and orth2: corthogonal (rev us)
and set us $\subseteq$ carrier-vec $n$
and length us $=$ length ws
and dist: distinct us
using gram-schmidt-sub-result[OF main ws]
by (auto simp: assms orth)
thus span $($ set ws) $=$ span (set us) by simp
show set us $\subseteq$ carrier-vec $n$ by fact
show length us $=$ length ws by fact
show distinct us by fact
show corthogonal us

```
    using corthogonal-distinct[OF orth2] unfolding distinct-rev
    using corthogonal-sort[OF - set-rev orth2] by auto
qed
end
end
```


## 17 Schur Decomposition

We implement Schur decomposition as an algorithm which, given a square matrix $A$ and a list eigenvalues, computes $B, P$, and $Q$ such that $A=P B Q$, $B$ is upper-triangular and $P Q=1$. The algorithm works is generic in the kind of field and can be applied on the rationals, the reals, and the complex numbers. The algorithm relies on the method of Gram-Schmidt to create an orthogonal basis, and on the Gauss-Jordan algorithm to find eigenvectors to a given eigenvalue.

The algorithm is a key ingredient to show that every matrix with a linear factorizable characteristic polynomial has a Jordan normal form.

A further consequence of the algorithm is that the characteristic polynomial of a block diagonal matrix is the product of the characteristic polynomials of the blocks.

```
theory Schur-Decomposition
imports
    Polynomial-Interpolation.Missing-Polynomial
    Gram-Schmidt
    Char-Poly
begin
definition vec-inv :: 'a::conjugatable-field vec => 'a vec
    where vec-inv v}=1/(v\cdotcv)\cdotv conjugate 
lemma vec-inv-closed[simp]:v\in carrier-vec n\Longrightarrow vec-inv v carrier-vec n
    unfolding vec-inv-def by auto
lemma vec-inv-dim[simp]: dim-vec (vec-inv v)=dim-vec v
    unfolding vec-inv-def by auto
lemma vec-inv[simp]:
    assumes v: v:carrier-vec n
        and v0:(v::'a::conjugatable-ordered-field vec)}\not=0\mp@subsup{0}{v}{}
    shows vec-inv v • v=1
proof -
    { assume v cc v=0
        hence v}=\mp@subsup{O}{v}{}n\mathrm{ using conjugate-square-eq-0-vec[OF v] by auto
        hence False using v0 by auto
    }
    moreover have conjugate v v v = v ccv
```

apply (rule comm-scalar-prod) using $v$ by auto
ultimately show ?thesis unfolding vec-inv-def apply (subst smult-scalar-prod-distrib)
using assms by auto
qed
lemma corthogonal-inv:
assumes orth: corthogonal (vs ::'a::conjugatable-field vec list) and $V$ : set vs $\subseteq$ carrier-vec $n$
shows inverts-mat (mat-of-rows $n$ (map vec-inv vs)) (mat-of-cols $n v s$ )
(is inverts-mat ? W ?V)
proof -
define $l$ where $l=$ length vs
have $r W[$ simp $]$ : dim-row? $W=l$ using $l$-def by auto
have $c V[$ simp $]:$ dim-col ? $V=l$ using $l$-def by auto
have dim: $\bigwedge i . i<$ length $v s \Longrightarrow v s!i \in$ carrier-vec $n$ using $V$ by auto
show ?thesis
unfolding inverts-mat-def
apply rule
unfolding mat-of-rows-carrier length-map l-def[symmetric]
unfolding index-one-mat
proof -
show dim-row $(? W * ? V)=l$ dim-col $(? W * ? V)=l$
unfolding times-mat-def $r W c V$ by auto
fix $i j$ assume $i: i<l$ and $j: j<l$
hence $i 2: i<$ length vs
and i3: $i<$ length (map vec-inv vs)
and $j 2: j<$ length vs using $l$-def by auto
hence $i d 2:$ vs $!i \in$ carrier-vec $n$
and id3: map vec-inv vs ! i $\in$ carrier-vec $n$
and id4: conjugate (vs $!i$ ) $\in$ carrier-vec $n$
and jd2: vs $!j \in$ carrier-vec $n$ using dim by auto
show $(? W * ? V) \$ \$(i, j)=($ if $i=j$ then 1 else 0$)$
unfolding times-mat-def $r W c V$
unfolding index-mat[OF i j] split
unfolding mat-of-rows-row[OF i3 id3]
unfolding col-mat-of-cols[OF j2 jd2]
unfolding nth-map[OF i2]
unfolding vec-inv-def
unfolding smult-scalar-prod-distrib[OF id4 jd2]
unfolding comm-scalar-prod[OF id4 jd2]
using corthogonalD[OF orth j2 i2] by auto
qed
qed
definition corthogonal-inv :: 'a::conjugatable-field mat $\Rightarrow$ 'a mat where corthogonal-inv $A=$ mat-of-rows (dim-row $A)($ map vec-inv (cols $A))$

```
definition mat-adjoint \(::\) ' \(a\) :: conjugatable-field mat \(\Rightarrow\) 'a mat
    where mat-adjoint \(A \equiv\) mat-of-rows (dim-row \(A\) ) (map conjugate (cols \(A\) ))
definition corthogonal-mat :: 'a::conjugatable-field mat \(\Rightarrow\) bool
    where corthogonal-mat \(A \equiv\)
    let \(B=\) mat-adjoint \(A * A\) in
    diagonal-mat \(B \wedge(\forall i<\operatorname{dim}\)-col \(A . B \$ \$(i, i) \neq 0)\)
lemma corthogonal-matD[elim]:
    assumes orth: corthogonal-mat \(A\)
        and \(i: i<\operatorname{dim}-\operatorname{col} A\)
        and \(j: j<\operatorname{dim}-\operatorname{col} A\)
    shows \((\operatorname{col} A i \cdot c \operatorname{col} A j=0)=(i \neq j)\)
proof
    have \(c i: \operatorname{col} A i:\) carrier-vec (dim-row \(A\) )
    and \(c j\) : col \(A j\) : carrier-vec (dim-row \(A\) ) by auto
    note \([\) simp \(]=\) conjugate-conjugate-sprod \([\) OF ci cj]
    let \(? B=\) mat-adjoint \(A * A\)
    have diag: diagonal-mat ?B and zero: \(\bigwedge i . i<\operatorname{dim}-\operatorname{col} A \Longrightarrow ? B \$ \$(i, i) \neq 0\)
    using orth unfolding corthogonal-mat-def Let-def by auto
    \(\{\) assume \(i=j\)
        hence conjugate \((\operatorname{col} A i) \cdot \operatorname{col} A j \neq 0\)
            using zero \([O F i]\) unfolding mat-adjoint-def using \(i\) by simp
            hence conjugate (conjugate \((\operatorname{col} A i) \cdot \operatorname{col} A j) \neq 0\)
                unfolding conjugate-zero-iff.
            hence \(\operatorname{col} A i \cdot c \operatorname{col} A j \neq 0\) by \(\operatorname{simp}\)
    \}
    thus \(\operatorname{col} A i \cdot c \operatorname{col} A j=0 \Longrightarrow i \neq j\) by auto
    \{ assume \(i \neq j\)
        hence conjugate \((\operatorname{col} A i) \cdot \operatorname{col} A j=0\)
            using diag
            unfolding diagonal-mat-def
            unfolding mat-adjoint-def using \(i j\) by simp
        hence conjugate (conjugate \((\operatorname{col} A i) \cdot \operatorname{col} A j)=0\) by \(\operatorname{simp}\)
        thus \(\operatorname{col} A i \cdot c \operatorname{col} A j=0\) by \(\operatorname{simp}\)
    \}
qed
lemma corthogonal-matI[intro]:
    assumes \((\bigwedge i j . i<d i m-\operatorname{col} A \Longrightarrow j<d i m-\operatorname{col} A \Longrightarrow(\operatorname{col} A i \cdot c \operatorname{col} A j=0)\)
\(=(i \neq j))\)
    shows corthogonal-mat \(A\)
proof -
    \{fix \(i j\) assume \(i: i<\operatorname{dim}-\operatorname{col} A\) and \(j: j<\operatorname{dim}-\operatorname{col} A\) and \(i j: i \neq j\)
        have conjugate \((\operatorname{col} A i) \cdot \operatorname{col} A j=0\)
            by (metis assms col-dim ij ij conjugate-vec-sprod-comm)
    \}
    moreover
```

```
    { fix i assume i<dim-col A
        hence conjugate (col A i) • col A i\not=0
            by (metis assms comm-scalar-prod carrier-vec-conjugate carrier-vecI)
    }
    ultimately show ?thesis
    unfolding corthogonal-mat-def Let-def
    unfolding diagonal-mat-def
    unfolding mat-adjoint-def by auto
qed
lemma corthogonal-inv-result:
    assumes o:corthogonal-mat (A::'a::conjugatable-field mat)
    shows inverts-mat (corthogonal-inv A) A
proof -
    have oc: corthogonal (cols A)
    apply (intro corthogonalI) using corthogonal-matD[OF o] by auto
    show ?thesis unfolding corthogonal-inv-def
    using corthogonal-inv[OF oc cols-dim] by auto
qed
    extends a vector to a basis
definition basis-completion :: 'a::ring-1 vec }=>\mathrm{ ' 'a vec list where
    basis-completion v \equivlet
        n= dim-vec v;
        drop-index =hd ([ i. i<- [0..<n],v$i\not=0]);
        vs=[unit-vec n i. i<-[0..<n],i\not= drop-index]
    in v # vs
lemma (in vec-space) basis-completion: fixes v :: ' a :: field vec
    assumes v: v\in carrier-vec n
        and v0:v\not= 0v n
    shows
    basis (set (basis-completion v))
    set (basis-completion v)\subseteq carrier-vec n
    span (set (basis-completion v)) = carrier-vec n
    distinct (basis-completion v)
    \checkmark ~ l i n - d e p ~ ( s e t ~ ( b a s i s - c o m p l e t i o n ~ v ) )
    length (basis-completion v) =n
    hd (basis-completion v)}=
proof -
    let ?b = basis-completion v
    note d = basis-completion-def Let-def
    from v have dim: dim-vec v=n by auto
    let ?is = [i. i<- [0..<n],v$i\not=0]
    {
    assume empty: set ?is = {}
    have }v=\mp@subsup{O}{v}{}
        by (rule eq-vecI, insert empty v, auto)
    }
```

with $v 0$ obtain $k i d s$ where $i d: ? i s=k \# i d s$ and mem: $k \in$ set ? is by (cases ?is, auto)
from mem have $v k: v \$ k \neq 0$ and $k: k<n$ by auto
\{
fix $i$
assume $i: \neg i<k$
have id: $k$ \# [Suc $k . .<n]=[k . .<n]$ using $k$ by (simp add: upt-conv-Cons)
from $i$ have $i<n \Longrightarrow(k \#[$ Suc $k . .<n])!(i-k)=i$
unfolding id
by (subst nth-upt, auto)
\}
hence split: $[0 . .<n]=[0 . .<k] @ k \#[$ Suc $k . .<n]$
by (intro nth-equality $I$, insert $k$, auto simp: nth-append)
\{
fix as
assume $k \notin$ set as
hence [unit-vec n i. $i<-a s, i \neq k]=[$ unit-vec n i. $i<-a s]$ by (induct as, auto)
\} note conv $=$ this
have $b$-all: $? b=v \#[$ unit-vec $n i . i<-[0 . .<n], i \neq k]$
unfolding $d$ dim id by simp
also have [unit-vec $n$ i. $i<-[0 . .<n], i \neq k]=[$ unit-vec $n i . i<-[0 . .<k]] @$
$[$ unit-vec n i. $i<-[$ Suc $k . .<n]]$
unfolding split by (auto simp: conv)
finally have $b: ? b=v \#[$ unit-vec $n$ i. $i<-[0 . .<k]]$ @ [unit-vec $n i . i<-[$ Suc $k . .<n]$ ] by $\operatorname{simp}$
show carr: set $? b \subseteq$ carrier-vec $n$ (is $? S \subseteq$-)
unfolding $b$ using assms by auto
show $h d ? b=v$ unfolding $b$ by auto
show len: length (basis-completion $v$ ) $=n$ unfolding $b$ using $k$
by auto
define $I$ where $I=(\lambda$ i. if $i<k$ then $i$ else Suc $i)$
have $I$ : $\bigwedge i$. $I i \neq k \wedge i$. Suc $i<n \Longrightarrow I i<n$ unfolding $I$-def by auto
\{
fix $i$
assume $i: i<n$
have ? $b!i=($ if $i=0$ then $v$ else unit-vec $n(I(i-1)))$
unfolding $b$-def using $i$
by (auto split: if-splits simp: nth-append)
\} note $b i=$ this
show dist: distinct ?b unfolding distinct-conv-nth len
proof (intro allI impI)
fix $i j$
assume $i: i<n$ and $j: j<n$ and $i j: i \neq j$
show ?b ! $i \neq ? b!j$
proof
assume $i d 1: ? b!i=? b!j$
hence $i d 2: \wedge l . ? b!i \$ l=? b!j \$ l$ by auto
have $i=j$
proof (cases $i=0$ )
case True
hence biv: ? $b!i=v$ unfolding $b$ by simp
from True ij have $b j: ? b!j=$ unit-vec $n(I(j-1))$ Suc $(j-1)=j$
unfolding $b i[O F j]$ by auto
with id2[of $k$, unfolded biv bj] vk $I[$ of $j-1] k j$
have False by simp
thus ?thesis ..
next
case False note $i 0=$ this
hence $b i^{\prime}: ? b!i=$ unit-vec $n(I(i-1)) S u c(i-1)=i$ unfolding $b i[O F$
i] by auto
show ?thesis
proof (cases $j=0$ )
case True
hence $b j: ? b!j=v$ unfolding $b$ by simp
from $i d 2\left[o f k\right.$, unfolded $\left.b i^{\prime} b j\right] v k I[o f i-1] k i b i^{\prime}$
have False by simp
thus?thesis by simp
next
case False note $j 0=$ this
hence $b j$ : ?b ! $j=$ unit-vec $n(I(j-1))$ Suc $(j-1)=j$ unfolding
$b i[O F j]$ by auto
have $1=? b!i \$ I(i-1)$ unfolding $b i^{\prime}$ using $I[o f i-1] i$ io by auto
also have $\ldots=$ unit-vec $n(I(j-1)) \$ I(i-1)$ unfolding $i d 1$ bj by
simp
also have $\ldots=($ if $I(i-1)=I(j-1)$ then 1 else 0$)$
using $I[o f i-1] I[o f j-1] i 0 j 0 i j$ by auto
finally have $I(i-1)=I(j-1)$ by (auto split: if-splits)
with $i 0 j 0$ show $i=j$ unfolding $I$-def by (auto split: if-splits)
qed
qed
thus False using ij by simp
qed
qed
have span (set ?b) $\subseteq$ carrier-vec $n$ using carr by auto
moreover
\{
fix $w::$ 'a vec
assume $w: w \in$ carrier-vec $n$
define lookup where lookup $=(v, k) \#[($ unit-vec $n i, i) . i<-[0 . .<n], i \neq k]$
define $a$ where $a=(\lambda$ vi. case map-of lookup vi of Some $i \Rightarrow$ if $i=k$ then $w$
$\$ k / v \$ k$ else
$w \$ i-w \$ k / v \$ k * v \$ i)$
have map fst lookup $=? b$ unfolding $b$-all lookup-def
by (auto simp: map-concat o-def if-distrib, unfold list.simps fst-def prod.simps,
simp)
with dist have dist: distinct (map fst lookup) by simp
let ? $w=$ lincomb $a($ set ?b)

```
    have ?w\in carrier-vec n using carr by auto
    with w have dim: dim-vec w=n dim-vec ? w = n by auto
    have w=?w
    proof (rule eq-vecI; unfold dim)
    fix i
    assume i: i<n
    show w$i=?w$i unfolding lincomb-def
    proof (subst finsum-index[OF i-carr])
        show (\lambdav.av v vv v) \in set ?b }->\mathrm{ carrier-vec n using carr by auto
        {
            fix }x:: 'a vec and 
            assume x = unit-vec n j j\not=kj<n
            hence (x,j) \in set lookup unfolding lookup-def by auto
            from map-of-is-SomeI[OF dist this]
            have ax=w$j-w$k/v$k*v$j unfolding a-def using <j\not=k>
by auto
        } note a= this
        have (\sumx\inset ?b. (ax v v ) $i)=(av\cdotvv)$i+(\sumx\in(set ?b) - {v}.
(ax\cdotv}x)$ i
            by (rule sum.remove[OF finite-set], auto simp: b)
        also have av=w$k/v$k unfolding a-def lookup-def by auto
        also have (\ldots\cdotvv)$i=w$k/v$k*v$i using iv by auto
    finally have (\sumx\inset ?b. (ax\cdotv x)$i)=w$k/v$k*v$i+(\sumx\in(set
?b) - {v}. (ax 汭 x)$ i).
    also have ... =w$ i
    proof (cases i=k)
        case True
        hence }w$k/v$k*v$i=w$k\mathrm{ using vk by auto
        moreover have (\sumx\in(set ?b) - {v}. (ax v v x)$ i)=0 unfolding True
        proof (rule sum.neutral, intro ballI)
            fix }
            assume x fet ?b - {v}
            then obtain j where x: x = unit-vec n j j\not=kj<n using k unfolding
b by auto
            show (ax v
auto
            qed
            ultimately show ?thesis unfolding True by simp
        next
            case False
            let ?ui=unit-vec n i :: 'a vec
            {
            assume ?ui = v
            from arg-cong[OF this, of \lambdav.v $k]vk ik False have False by auto
        }
            hence diff: ?ui\not=v by auto
        from a[OF refl False] have ai: (a?ui \cdotv ?ui) $ i=w$ i-w$k/v$k
* v$i
            using i by auto
```

```
                    have ?ui \in set ?b unfolding b-all using False k i by auto
                    with diff have mem: unit-vec n i\in set ?b - {v} by auto
            have w$k/v$k*v$i+(\sumx\in(set ?b) - {v}. (ax\cdotv x)$ i)
                =w$i+(\sumx\in(set ?b) - {v,?ui}. (ax\cdotv x)$i)
                by (subst sum.remove[OF - mem], auto simp: ai intro!: sum.cong)
            also have (\sumx\in(set ?b) - {v,?ui}. (ax v 片 $i)=0
            by (rule sum.neutral, unfold b-all, insert i k, auto)
            finally show ?thesis by simp
            qed
            finally show w$i=(\sumx\inset?b. (ax\cdotv x)$i) by simp
        qed
    qed auto
    hence w\in span (set ?b) unfolding span-def by auto
}
ultimately show span: span (set ?b) = carrier-vec n by blast
show basis (set ?b)
proof (rule dim-gen-is-basis[OF finite-set carr span])
    have card (set ?b) = dim using dist len distinct-card unfolding dim-is-n by
blast
    thus card (set ?b) \leq dim by simp
    qed
    thus ᄀ lin-dep (set ?b) unfolding basis-def by auto
qed
lemma orthogonal-mat-of-cols:
    assumes W: set ws \subseteqcarrier-vec n
        and orth: corthogonal ws
        and len: length ws = n
    shows corthogonal-mat (mat-of-cols n ws) (is corthogonal-mat ?W)
proof
    fix ij assume i:i<dim-col ? W and j: j<dim-col ?W
    hence [simp]: ws!i:carrier-vec n ws ! j:carrier-vec n
        using W len by auto
    have i< length ws and j< length ws using i j using len W by auto
    thus col?W i}\cdot\textrm{c col ? W j=0 \longleftrightarrow 
        using orth
        unfolding corthogonal-def
        by simp
qed
lemma corthogonal-col-ev-0: fixes A :: 'a :: conjugatable-ordered-field mat
    assumes A:A\incarrier-mat n n
    and v:v\in carrier-vec n
    and v0:v\not= 0v n
    and eigen[simp]: A *vv = e v}
    and n:n\not=0
    and hdws: hd ws =v
    and ws: set ws \subseteqcarrier-vec n corthogonal ws length ws =n
    defines W == mat-of-cols n ws
```

defines $W^{\prime}==$ corthogonal-inv $W$
defines $A^{\prime}==W^{\prime} * A * W$
shows col $A^{\prime} 0=$ vec $n(\lambda$ i. if $i=0$ then e else 0$)$
proof -
let $? f=(\lambda$. if $i=0$ then $e$ else 0$)$
from $w s$ have $W: W \in$ carrier-mat $n n$ unfolding $W$-def by auto
from $W$ have $W^{\prime}: W^{\prime} \in$ carrier-mat $n n$ unfolding $W^{\prime}$-def corthogonal-inv-def mat-of-rows-def by auto
from $A W W^{\prime}$ have $A^{\prime}: A^{\prime} \in$ carrier-mat $n n$ unfolding $A^{\prime}$-def by auto
show col $A^{\prime} 0=v e c n ? f$
proof (rule, unfold dim-vec)
show dim: dim-vec $\left(\operatorname{col} A^{\prime} 0\right)=n$ using $A^{\prime}$ by $\operatorname{simp}$
have row0: vec-inv $v \cdot\left(A *_{v} v\right)=e$
using scalar-prod-smult-distrib[OF vec-inv-closed $\left.\left[\begin{array}{lll}O F & v\end{array}\right] v\right]$
using vec-inv[OF v v0] by auto
fix $i$ assume $i: i<n$
hence $i 2: i<$ length ws using ws by auto
let ? wsi $=w s!i$
have $z: 0<d i m$-col $A^{\prime}$ using $A^{\prime} n$ by auto
hence z2: $0<$ length ws using $A^{\prime}$ ws by auto
have wsi[simp]: ws! $i$ : carrier-vec $n$ using ws $i$ by auto
hence ws $0[$ simp $]:$ ws! $0=v$ using hd-conv-nth[symmetric] $h d w s z 2$ by auto
have col $A^{\prime} 0 \$ i=A^{\prime} \$ \$(i, 0)$ using $A^{\prime} i$ by auto
also have $\ldots=\left(W^{\prime} *(A * W)\right) \$ \$(i, 0)$ unfolding $A^{\prime}$-def using $W^{\prime} A W$
by auto
also have $\ldots=$ row $W^{\prime} i \cdot \operatorname{col}(A * W) 0$
apply (subst index-mult-mat) using $W W^{\prime} A i$ by auto
also have row $W^{\prime} i=$ vec-inv ?wsi
unfolding $W^{\prime}$-def $W$-def unfolding corthogonal-inv-def using $i$ ws by auto
also have $\operatorname{col}(A * W) 0=A * v$ col $W 0$ using $A W z A^{\prime}$ by auto
also have col $W 0=v$ unfolding $W$-def using z2 ws0 $n$ col-mat-of-cols $v$ by
blast
also have $A *_{v} v=e \cdot{ }_{v} v$ using eigen.
also have vec-inv ? wsi $\cdot\left(e \cdot{ }_{v} v\right)=e *(v e c-i n v$ ? wsi $\cdot v)$
using scalar-prod-smult-distrib $[$ OF vec-inv-closed $[$ OF wsi $] v]$.
also have $\ldots=$ ?f $i$
proof (cases $i=0$ )
case True thus ?thesis using vec-inv[OF v vo] by simp
next
case False
hence $z: 0<$ length ws using $i$ ws by auto
note cwsi $=$ carrier-vec-conjugate[OF wsi]
have vec-inv ? wsi $\cdot v=1 /($ ?wsi $\cdot c$ ?wsi $) *($ conjugate ?wsi $\cdot v)$
unfolding vec-inv-def unfolding smult-scalar-prod-distrib[OF cwsi v]..
also have conjugate ? wsi $\cdot v=v \cdot c$ ? wsi
using comm-scalar-prod $[$ OF cwsi $v]$.
also have $\ldots=0$
using corthogonalD[OF ws(2) z i2] False unfolding ws0 by auto
finally show ?thesis using False by auto

```
    qed
    also have ... = vec n ?f $ i using i by simp
    finally show col A' 0 $ i = vec n?f $ i .
    qed
qed
```

Schur decomposition

```
fun schur-decomposition :: 'a::conjugatable-field mat \(\Rightarrow{ }^{\prime} a\) list \(\Rightarrow{ }^{\prime} a\) mat \(\times\) 'a mat
\(\times\) 'a mat where
    schur-decomposition \(A[]=\left(A, 1_{m}(\right.\) dim-row \(A), 1_{m}(\) dim-row \(\left.A)\right)\)
| schur-decomposition \(A(e \#\) es \()=\) (let
        \(n=\operatorname{dim}\)-row \(A\);
        \(n 1=n-1\);
        \(v=\) find-eigenvector \(A e ;\)
        \(w s=\) gram-schmidt \(n\) (basis-completion \(v)\);
        \(W=\) mat-of-cols \(n \mathrm{ws}\);
        \(W^{\prime}=\) corthogonal-inv \(W\);
        \(A^{\prime}=W^{\prime} * A * W\);
        \((A 1, A 2, A 0, A 3)=\) split-block \(A^{\prime} 11\);
        \((B, P, Q)=\) schur-decomposition A3 es;
        \(z\)-row \(=\left(0_{m} 1\right.\) n1 \()\);
        \(z-c o l=\left(\begin{array}{lll}0_{m} & n 1 & 1\end{array}\right) ;\)
        one- \(1=1_{m} 1\)
    in (four-block-mat A1 (A2 * P) A0 B,
    \(W *\) four-block-mat one-1 \(z\)-row \(z\)-col \(P\),
    four-block-mat one-1 \(z\)-row \(z\)-col \(\left.Q * W^{\prime}\right)\) )
```

theorem schur-decomposition:
assumes $A$ : (A::'a::conjugatable-ordered-field mat) $\in$ carrier-mat $n n$
and $c:$ char-poly $A=\left(\Pi\left(e::{ }^{\prime} a\right) \leftarrow e s .[:-e, 1:]\right)$
and $B$ : schur-decomposition $A$ es $=(B, P, Q)$
shows similar-mat-wit $A B P Q \wedge$ upper-triangular $B \wedge$ diag-mat $B=$ es
using assms
proof (induct es arbitrary: $n A B P Q$ )
case Nil
with degree-monic-char-poly[of A $n$ ]
show ?case by (auto intro: similar-mat-wit-refl simp: diag-mat-def)
next
case (Cons e es n A CP $Q$ )
let $? n 1=n-1$
from Cons have $A: A \in$ carrier-mat $n n$ and dim: dim-row $A=n$ by auto
let ? $c p=$ char-poly $A$
from Cons(3)
have $c p: ? c p=[:-e, 1:] *\left(\prod e \leftarrow e s .[:-e, 1:]\right)$ by auto
have mon: monic ( $\left.\prod e \leftarrow e s .[:-e, 1:]\right)$ by (rule monic-prod-list, auto)
have deg: degree ? cp $=\operatorname{Suc}\left(\right.$ degree $\left.\left(\prod e \leftarrow e s .[:-e, 1:]\right)\right)$ unfolding $c p$
by (subst degree-mult-eq, insert mon, auto)
with degree-monic-char-poly $[O F A]$ have $n: n \neq 0$ by auto
define $v$ where $v=$ find-eigenvector $A e$
define $b$ where $b=$ basis-completion $v$
define $w s$ where $w s=$ gram-schmidt $n b$
define $W$ where $W=$ mat-of-cols $n$ ws
define $W^{\prime}$ where $W^{\prime}=$ corthogonal-inv $W$
define $A^{\prime}$ where $A^{\prime}=W^{\prime} * A * W$
obtain A1 A2 A0 A3 where splitA': split-block $A^{\prime} 11=(A 1, A 2, A 0, A 3)$
by (cases split-block $A^{\prime} 11$, auto)
obtain $B P^{\prime} Q^{\prime}$ where schur: schur-decomposition $A 3$ es $=\left(B, P^{\prime}, Q^{\prime}\right)$
by (cases schur-decomposition A3 es, auto)
let ? $P^{\prime}=$ four-block-mat $\left(\begin{array}{ll}1_{m} & 1\end{array}\right)\left(\begin{array}{ll}0_{m} & 1 \text { ? } n 1\end{array}\right)\left(\begin{array}{l}0_{m} \text { ?n1 }\end{array}\right.$ 1) $P^{\prime}$
let ? $Q^{\prime}=$ four-block-mat $\left(\begin{array}{ll}1_{m} & 1\end{array}\right)\left(\begin{array}{lll}0_{m} & 1 & ? n 1\end{array}\right)\left(\begin{array}{ll}0_{m} & \text { ?n1 }\end{array}\right) Q^{\prime}$
have $C: C=$ four-block-mat $A 1\left(A 2 * P^{\prime}\right) A 0 B$ and $P: P=W * ? P^{\prime}$ and $Q$ :
$Q=? Q^{\prime} * W^{\prime}$
using Cons(4) unfolding schur-decomposition.simps
Let-def list.sel dim
$v$-def[symmetric] b-def[symmetric] ws-def[symmetric] $W^{\prime}$-def[symmetric] $W$-def[symmetric]
$A^{\prime}$-def[symmetric] split split $A^{\prime}$ schur by auto
have $e$ : eigenvalue $A e$
unfolding eigenvalue-root-char-poly[OF A] cp by simp
from find-eigenvector $[O F A$ e] have ev: eigenvector $A v e$ unfolding $v$-def.
from this[unfolded eigenvector-def]
have $v[$ simp $]: v \in$ carrier-vec $n$ and $v 0: v \neq 0 v n$ using $A$ by auto
interpret cof-vec-space $n$ TYPE ('a) .
from basis-completion[OF v v0, folded b-def]
have span-b: span (set b) $=$ carrier-vec $n$ and dist-b: distinct $b$
and indep: $\neg \operatorname{lin-dep~(set~b)~and~b:~set~} b \subseteq$ carrier-vec $n$ and $h d b: h d b=v$
and len- $b$ : length $b=n$ by auto
from $h d b$ len- $b n$ obtain $v s$ where $b v: b=v \#$ vs by (cases $b$, auto)
from gram-schmidt-result[OF b dist-b indep refl, folded ws-def]
have ws: set $w s \subseteq$ carrier-vec $n$ corthogonal ws length $w s=n$
by (auto simp: len-b)
from gram-schmidt-hd[OF v, of vs, folded bv] have hdws: $h d$ ws $=v$ unfolding ws-def.
have orth-W: corthogonal-mat $W$ using orthogonal-mat-of-cols ws unfolding $W$-def.
have $W: W \in$ carrier-mat $n n$
using ws unfolding $W$-def using mat-of-cols-carrier (1)[of $n$ ws] by auto
have $W^{\prime}: W^{\prime} \in$ carrier-mat $n n$ unfolding $W^{\prime}$-def corthogonal-inv-def using W
by (auto simp: mat-of-rows-def)
from corthogonal-inv-result $[$ OF orth-W]
have $W^{\prime} W$ : inverts-mat $W^{\prime} W$ unfolding $W^{\prime}$-def.
hence $W W^{\prime}$ : inverts-mat $W W^{\prime}$ using mat-mult-left-right-inverse $\left[O F W^{\prime} W\right]$ $W^{\prime} W$
unfolding inverts-mat-def by auto
have $A^{\prime}: A^{\prime} \in$ carrier-mat $n n$ using $W W^{\prime} A$ unfolding $A^{\prime}$-def by auto
have $A^{\prime} A$-wit: similar-mat-wit $A^{\prime} A W^{\prime} W$
by (rule similar-mat-witI[of--n], insert $W W^{\prime} A A^{\prime} W^{\prime} W W W^{\prime}$, auto simp:

```
A'-def
    inverts-mat-def)
    hence }\mp@subsup{A}{}{\prime}A\mathrm{ : similar-mat }\mp@subsup{A}{}{\prime}A\mathrm{ unfolding similar-mat-def by blast
    from similar-mat-wit-sym[OF A'A-wit] have simAA': similar-mat-wit A A' W
W' by auto
    have eigen[simp]:A **v = e vv}v\mathrm{ and v0:v}=0\mp@subsup{0}{v}{}
        using v-def find-eigenvector[OF A e] A
        unfolding eigenvector-def by auto
    let ?f =( }\lambda\mathrm{ i. if }i=0\mathrm{ then e else 0)
    have col0: col A' 0 = vec n?f
        unfolding A'-def W'-def W-def
    using corthogonal-col-ev-0[OF A v v0 eigen n hdws ws].
    from }\mp@subsup{A}{}{\prime}n\mathrm{ have dim-row }\mp@subsup{A}{}{\prime}=1+?n1 dim-col A'=1 +?n1 by aut
    from split-block[OF splitA' this] have A2:A2 \in carrier-mat 1 ?n1
    and A3:A3 \in carrier-mat ?n1 ?n1
    and A'block: A' = four-block-mat A1 A2 A0 A3 by auto
    have A1id: A1 = mat 1 1 ( }\lambda\mathrm{ -. e)
        using splitA'[unfolded split-block-def Let-def] arg-cong[OF col0, of \lambdav.v $ 0]
A'n
    by (auto simp: col-def)
    have A1: A1 \in carrier-mat 1 1 unfolding A1id by auto
    {
        fix }
        assume i<?n1
        with arg-cong[OF col0, of \lambda v.v $ Suc i] A'
        have }\mp@subsup{A}{}{\prime}$$(\mathrm{ Suc i, 0) = 0 by auto
    } note A'0 = this
    have A0id: A0 = 0m ?n1 1
    using splitA'[unfolded split-block-def Let-def] A'0 A' by auto
    have A0:A0 \in carrier-mat?n1 1 unfolding A0id by auto
    from cp char-poly-similar[OF A'A]
    have cp: char-poly }\mp@subsup{A}{}{\prime}=[:-e,1:]*(\prode\leftarrowes.[:-e,1:]) by sim
    also have char-poly A' = char-poly A1 * char-poly A3
        unfolding A'block A0id
        by (rule char-poly-four-block-zeros-col[OF A1 A2 A3])
    also have char-poly A1 = [:-e,1 :]
    by (simp add: A1id char-poly-defs det-def sign-def)
    finally have cp:char-poly A3 }=(\\mathrm{ \ e es. [:- e, 1:])
    by (metis mult-cancel-left pCons-eq-0-iff zero-neq-one)
    from Cons(1)[OF A3 cp schur]
    have simIH: similar-mat-wit A3 B P' Q' and ut: upper-triangular B and diag:
diag-mat B = es by auto
    from similar-mat-witD2[OF A3 simIH]
    have B:B\incarrier-mat ?n1 ?n1 and P': P' \in carrier-mat ?n1 ?n1 and Q':
Q'\in carrier-mat ?n1 ?n1
    and PQ': P'* Q = 1m ?n1 by auto
    have A0-eq:A0 = P'*A0* 1 m 1 unfolding A0id using P' by auto
    have simA'C: similar-mat-wit A' C ? P' ?Q' unfolding A'block C
        by (rule similar-mat-wit-four-block[OF similar-mat-wit-refl[OF A1] simIH -
```

```
A0-eq A1 A3 A0],
    insert PQ'A2 P' Q',auto)
    have ut1: upper-triangular A1 unfolding A1id by auto
    have ut: upper-triangular C unfolding C A0id
    by (intro upper-triangular-four-block[OF - B ut1 ut], auto simp: A1id)
    from A1id have diagA1: diag-mat A1 = [e] unfolding diag-mat-def by auto
    from diag-four-block-mat[OF A1 B] have diag: diag-mat C =e# es unfolding
diag diagA1 C by simp
    from ut similar-mat-wit-trans[OF \operatorname{sim}A\mp@subsup{A}{}{\prime}\operatorname{sim}\mp@subsup{A}{}{\prime}C,\mathrm{ folded P Q] diag}
    show ?case by blast
qed
definition schur-upper-triangular :: 'a::conjugatable-field mat }=>\mp@subsup{}{}{\prime}\mathrm{ 'a list }=>\mathrm{ ' 'a mat
where
    schur-upper-triangular A es =(case schur-decomposition A es of ( }B,-,-)=>B
lemma schur-upper-triangular:
    assumes A: (A :: 'a :: conjugatable-ordered-field mat) \incarrier-mat n n
    and linear: char-poly A}=(\a\leftarrowes.[:- a, 1:]
    defines B: B\equiv schur-upper-triangular }A\mathrm{ es
    shows B\incarrier-mat n n upper-triangular B similar-mat A B
proof -
    let ?B = schur-upper-triangular A es
    obtain C P Q where schur: schur-decomposition A es = (C,P,Q)
    by (cases schur-decomposition A es, auto)
    hence B: B=C using A unfolding schur-upper-triangular-def B by auto
    from schur-decomposition[OF A linear schur]
    have sim: similar-mat-wit A B P Q and B: upper-triangular B unfolding B by
auto
    from sim show similar-mat A B unfolding similar-mat-def by auto
    from similar-mat-witD2[OF A sim] show B \in carrier-mat n n by auto
    show upper-triangular B by fact
qed
lemma schur-decomposition-exists: assumes \(A: A \in\) carrier-mat \(n n\)
    and linear: char-poly A = (П ( a :: 'a :: conjugatable-ordered-field ) }\leftarrow\mathrm{ es. [:- a,
1:])
    shows \existsB\incarrier-mat n n. upper-triangular B ^ similar-mat A B
    using schur-upper-triangular[OF A linear] by blast
lemma char-poly-0-block: fixes A :: 'a :: conjugatable-ordered-field mat
    assumes A:A = four-block-mat B C (0m m n) D
    and linearB: \exists es. char-poly B=(П a\leftarrowes. [:- a, 1:])
    and linearD: \exists es.char-poly D=(П }a\leftarrowes.[:-a,1:]
    and B:B\in carrier-mat n n
    and C:C\in carrier-mat n m
    and D:D\in carrier-mat m m
    shows char-poly A = char-poly B* char-poly D
```

```
proof -
    from linearB obtain bs where cB: char-poly B=(\proda\leftarrowbs. [:- a, 1:]) by auto
    from linearD obtain ds where cD: char-poly D=(\proda\leftarrowds. [:-a, 1:]) by auto
    from schur-decomposition-exists[OF B cB]
    obtain }\mp@subsup{B}{}{\prime}PBQB\mathrm{ where sB: schur-decomposition B bs = ( }\mp@subsup{B}{}{\prime},PB,QB
        by (cases schur-decomposition B bs, auto)
    obtain D' PD QD where sD: schur-decomposition D ds=( }\mp@subsup{D}{}{\prime},PD,QD
        by (cases schur-decomposition D ds, auto)
    from schur-decomposition[OF B cB sB] similar-mat-witD2[OF B, of B] have
        simB: similar-mat B B' and utB: upper-triangular }\mp@subsup{B}{}{\prime}\mathrm{ and diagB: diag-mat B'
= b
        and \mp@subsup{B}{}{\prime}:\mp@subsup{B}{}{\prime}\in\mathrm{ carrier-mat n n}
        by (auto simp: similar-mat-def)
    from schur-decomposition[OF D cD sD] similar-mat-witD2[OF D, of D' have
        simD: similar-mat D D' and utD: upper-triangular D' and diagD: diag-mat D'
= ds
    and D': D' 
    by (auto simp: similar-mat-def)
    let ?z = 0 m m n
    from similar-mat-four-block-0-ex[OF simB simD C B D, folded A]
    obtain B0 where B0: B0 c carrier-mat n m and sim: similar-mat A (four-block-mat
B' B0 ?z D')
    by auto
    let ?block = four-block-mat B' B0 ?z D'
    let ?cp = char-poly
    let ?prod = QB*C*PD
    let ?diag = \lambda A. (\proda\leftarrowdiag-mat A. [:- a, 1:])
    from char-poly-similar[OF sim] have ?cp A = ?cp ?block by simp
    also have ... = ?diag ?block
        by (rule char-poly-upper-triangular[OF four-block-carrier-mat[OF B' D'] up-
per-triangular-four-block[OF B' D' utB utD]])
    also have ... = ?diag B'* ?diag D' unfolding diag-four-block-mat[OF B' D']
        by auto
    also have ? diag B'=?cp B
        by (subst char-poly-upper-triangular[OF B' utB], simp)
    also have ... = ?cp B
        by (rule char-poly-similar[OF similar-mat-sym[OF simB]])
    also have ?diag D' = ?cp D'
        by (subst char-poly-upper-triangular[OF D' utD], simp)
    also have ... = ?cp D
        by (rule char-poly-similar[OF similar-mat-sym[OF simD]])
    finally show ?thesis.
qed
lemma char-poly-0-block': fixes A :: 'a :: conjugatable-ordered-field mat
    assumes A: A = four-block-mat B (0m n m) C D
    and linearB: \exists es. char-poly B=(П a\leftarrowes. [:- a, 1:])
    and linearD: \exists es.char-poly D=(П }a\leftarrowes.[:-a,1:]
```

```
    and B:B\in carrier-mat n n
    and C:C\incarrier-mat mn
    and D:D\incarrier-mat mm
    shows char-poly A = char-poly B* char-poly D
proof -
    let ?A = four-block-mat B (0m n m) C D
    let ?B = transpose-mat B
    let ?D = transpose-mat D
    have AC: ?A A carrier-mat }(n+m)(n+m)\mathrm{ using B D by auto
    from arg-cong[OF transpose-four-block-mat[OF B zero-carrier-mat C D], of char-poly,
        unfolded char-poly-transpose-mat[OF AC], folded A]
    have char-poly A =
        char-poly (four-block-mat ?B (transpose-mat C) ( }0mmm\mp@code{m}\mathrm{ m ?D) by auto
    also have ... = char-poly ? B * char-poly ?D
        by (rule char-poly-0-block[OF refl], insert B C D linearB linearD, auto)
    also have ... = char-poly B* char-poly D using B D
        by simp
    finally show ?thesis.
qed
end
```


## 18 Computing Jordan Normal Forms

```
theory Jordan-Normal-Form-Existence
imports
    Jordan-Normal-Form
    Column-Operations
    Schur-Decomposition
begin
hide-const (open) Coset.order
```

We prove existence of Jordan normal forms by means of first applying Schur's algorithm to convert a matrix into upper-triangular form, and then applying the following algorithm to convert a upper-triangular matrix into a Jordan normal form. It only consists of basic row- and column-operations.

### 18.1 Pseudo Code Algorithm

The following algorithm is used to compute JNFs from upper-triangular matrices. It was generalized from [5, Sect. 11.1.4] where this algorithm was not explicitly specified but only applied on an example. We further introduced step 2 which does not occur in the textbook description.

1. Eliminate entries within blocks besides EV $a$ and above EV $b$ for $a \neq b$ : for $A_{i j}$ with EV $a$ left of it, and EV $b$ below of it, perform
add-col-sub-row $\left(A_{i j} /(b-a)\right) i j$. The iteration should be by first increasing $j$ and the inner loop by decreasing $i$.
2. Move rows of same EV together, can only be done after 1., otherwise triangular-property is lost. Say both rows $i$ and $j(i<j)$ contain EV $a$, but all rows between $i$ and $j$ have different EV. Then perform swap-cols-rows $(i+1) j$, swap-cols-rows $(i+2) j, \ldots$ swap-cols-rows $(j-1) j$. Afterwards row $j$ will be at row $i+1$, and rows $i+1, \ldots, j-1$ will be moved to $i+2, \ldots, j$. The global iteration works by increasing $j$.
3. Transform each EV-block into JNF, do this for increasing upper $n \times k$ matrices, where each new column $k$ will be treated as follows.
a) Eliminate entries $A_{i k}$ in rows of form $0 \ldots 0$ ev $10 \ldots 0 A_{i k}$ : add-col-sub-row $\left(-A_{i k}\right)(i+1) k$. Perform elimination by increasing $i$.
b) Figure out largest JB (of $n-1 \times n-1$ sub-matrix) with lowest row of form $0 \ldots 0$ ev $0 \ldots 0 A_{l k}$ where $A_{l k} \neq 0$, and set $x:=A_{l k}$.
c) If such a JB does not exist, continue with next column. Otherwise, eliminate all other non-zero-entries $y:=A_{i k}$ via row $l$ : add-col-sub-row $(y / x) i l$, add-col-sub-row $(y / x)(i-1)(l-$ 1), add-col-sub-row $(y / x)(i-2)(l-2), \ldots$ where the number of steps is determined by the size of the JB left-above of $A_{i k}$. Perform an iteration over $i$.
d) Normalize value in row $l$ to 1: mult-col-div-row $\left(\left(1::^{\prime} a\right) / x\right) k$.
e) Move the 1 down from row $l$ to row $k-1$ : swap-cols-rows $(l+1)$ $k$, swap-cols-rows $(l+2) k, \ldots$, swap-cols-rows $(k-1) k$.

### 18.2 Real Algorithm

fun lookup-ev :: ' $a \Rightarrow n a t \Rightarrow{ }^{\prime} a$ mat $\Rightarrow$ nat option where
lookup-ev ev 0 A $=$ None
|lookup-ev ev (Suc i) $A=($ if $A \$ \$(i, i)=$ ev then Some $i$ else lookup-ev ev i $A$ )
function swap-cols-rows-block :: nat $\Rightarrow$ nat $\Rightarrow$ 'a mat $\Rightarrow$ 'a mat where
swap-cols-rows-block ij $A=$ (if $i<j$ then swap-cols-rows-block (Suc i) $j$ (swap-cols-rows ij A) else A)
by pat-completeness auto
termination by (relation measure $(\lambda(i, j, A) . j-i)$ ) auto
fun identify-block :: 'a :: one mat $\Rightarrow$ nat $\Rightarrow$ nat where
identify-block A $0=0$
| identify-block $A($ Suc $i)=($ if $A \$ \$(i, S u c i)=1$ then identify-block A i else (Suc i))
function identify-blocks-main $::$ 'a $::$ ring-1 mat $\Rightarrow$ nat $\Rightarrow$ (nat $\times$ nat) list $\Rightarrow$ (nat $\times$ nat) list where
identify-blocks-main A 0 list $=$ list
| identify-blocks-main A (Suc i-end) list $=($
let $i$-begin $=$ identify-block $A$ i-end
in identify-blocks-main A i-begin (( $i$-begin, $i$-end) \# list)
)
by pat-completeness auto
definition identify-blocks :: 'a :: ring-1 mat $\Rightarrow$ nat $\Rightarrow$ (nat $\times$ nat)list where identify-blocks A $i=$ identify-blocks-main A $i[]$
fun find-largest-block :: nat $\times$ nat $\Rightarrow($ nat $\times$ nat $)$ list $\Rightarrow$ nat $\times$ nat where
find-largest-block block [] = block
| find-largest-block (m-start,m-end) ((i-start,i-end) \# blocks) $=$
(if $i$-end $-i$-start $\geq m$-end $-m$-start then
find-largest-block (i-start,i-end) blocks else
find-largest-block (m-start,m-end) blocks)
fun lookup-other-ev :: ' $a \Rightarrow$ nat $\Rightarrow$ 'a mat $\Rightarrow$ nat option where lookup-other-ev ev 0 A = None
|lookup-other-ev ev (Suc i) $A=($ if $A \$ \$(i, i) \neq$ ev then Some $i$ else lookup-other-ev ev $i$ A)
partial-function (tailrec) partition-ev-blocks :: 'a mat $\Rightarrow{ }^{\prime}$ 'a mat list $\Rightarrow{ }^{\prime}$ a mat list where
[code]: partition-ev-blocks $A$ bs $=($ let $n=\operatorname{dim}$-row $A$ in if $n=0$ then $b s$ else (case lookup-other-ev $(A \$ \$(n-1, n-1))(n-1) A$ of
None $\Rightarrow A \#$ bs
| Some $i \Rightarrow$ case split-block A (Suc i) (Suc i) of (UL,-,-,LR) $\Rightarrow$ partition-ev-blocks $U L(L R \# b s)))$

```
context
    fixes n :: nat
    and ty :: 'a :: field itself
begin
function step-1-main :: nat }=>\mathrm{ nat }=>\mathrm{ ' 'a mat }=>\mathrm{ 'a mat where
    step-1-main ij A = (if j\geqn then A else if i=0 then step-1-main (j+1) (j+1)
A
    else let
        i'}=i-1
        ev-left =A $$ ( }\mp@subsup{i}{}{\prime},\mp@subsup{i}{}{\prime})
        ev-below = A $$ (j,j);
        aij = A $$ ( i',j);
        B= if (ev-left }\not=\mathrm{ ev-below }\wedge aij \not=0) then add-col-sub-row (aij / (ev-below
- ev-left)) i' j A else A
```

in step-1-main $i^{\prime} j B$ )
by pat-completeness auto
termination by (relation measures $[\lambda(i, j, A) . n-j, \lambda(i, j, A) . i])$ auto
function step-2-main :: nat $\Rightarrow$ 'a mat $\Rightarrow$ 'a mat where
step-2-main $j A=($ if $j \geq n$ then $A$
else
let ev $=A \$ \$(j, j)$;
$B=$ (case lookup-ev ev $j A$ of
None $\Rightarrow A$
| Some $i \Rightarrow$ swap-cols-rows-block (Suc i) j A )
in step-2-main (Suc j) B)
by pat-completeness auto
termination by (relation measure $(\lambda(j, A) . n-j)$ ) auto
fun step-3- $a::$ nat $\Rightarrow n a t \Rightarrow{ }^{\prime} a$ mat $\Rightarrow{ }^{\prime} a$ mat where
step-3-a $0 j A=A$
| step-3-a (Suc i) $j A=$ (let
$a i j=A \$ \$(i, j)$;
$B=($ if $A \$ \$(i, i+1)=1 \wedge a i j \neq 0$
then add-col-sub-row (-aij) (Suc i) $j$ A else $A$ )
in step-3-a i $j$ B)
fun step-3-c-inner-loop :: 'a $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow n a t \Rightarrow{ }^{\prime} a$ mat $\Rightarrow$ ' $a$ mat where step-3-c-inner-loop val li 0 A $=A$
| step-3-c-inner-loop val li(Suc k) $A=$ step-3-c-inner-loop val $(l-1)(i-1) k$ (add-col-sub-row val il A)
fun step-3-c :: ' $a \Rightarrow n a t \Rightarrow n a t \Rightarrow(n a t \times n a t) l i s t \Rightarrow{ }^{\prime} a$ mat $\Rightarrow{ }^{\prime} a$ mat where step-3-c x lk[] $A=A$
| step-3-c x lk((i-begin,i-end) $\#$ blocks) $A=($ let
$B=($ if $i$-end $=l$ then $A$ else
step-3-c-inner-loop $(A \$ \$(i$-end, $k) / x) l$ i-end $(S u c i$-end $-i$-begin $) A)$ in step-3-c $x l k$ blocks $B$ )
function step-3-main :: nat $\Rightarrow{ }^{\prime}$ a mat $\Rightarrow$ 'a mat where
step-3-main $k A=($ if $k \geq n$ then $A$
else let
$B=$ step-3-a $(k-1) k A ;-3-a$
all-blocks = identify-blocks Bk;
blocks $=$ filter $(\lambda$ block. B \$\$ (snd block,$k) \neq 0)$ all-blocks;
$F=($ if blocks $=[]-$ column $k$ has only 0 s
then $B$
else let
$(l$-start,$l)=$ find-largest-block $(h d$ blocks $)(t l$ blocks $) ;$ - 3-b
$x=B \$ \$(l, k)$;
$C=$ step-3-c $x l k$ blocks $B ;-3-c$

```
            D = mult-col-div-row (inverse x) k C; - 3-d
            E = swap-cols-rows-block (Suc l) k D-3-e
        in E)
    in step-3-main (Suc k) F)
    by pat-completeness auto
termination by (relation measure ( }\lambda(k,A).n-k)) aut
end
definition step-1 :: 'a :: field mat }=>\mp@subsup{}{}{\prime}'a mat where
    step-1 A = step-1-main (dim-row A) 00 A
definition step-2 :: ' }a\mathrm{ :: field mat }=>\mp@subsup{}{}{\prime}'a mat where
    step-2 A = step-2-main (dim-row A) O A
definition step-3 :: 'a :: field mat => 'a mat where
    step-3 A = step-3-main (dim-row A) 1 A
declare swap-cols-rows-block.simps[simp del]
declare step-1-main.simps[simp del]
declare step-2-main.simps[simp del]
declare step-3-main.simps[simp del]
function jnf-vector-main :: nat }\mp@subsup{=>}{}{\prime}\mp@subsup{}{}{\prime}a\mathrm{ :: one mat }=>(nat \times 'a) list wher
    jnf-vector-main 0 A = []
| jnf-vector-main (Suc i-end) A = (let
    i-start = identify-block A i-end
    in jnf-vector-main i-start A @ [(Suc i-end - i-start, A $$ (i-start,i-start))])
    by pat-completeness auto
definition jnf-vector :: ' }a\mathrm{ :: one mat }=>(nat\times'a) list wher
    jnf-vector A = jnf-vector-main (dim-row A) A
definition triangular-to-jnf-vector :: 'a :: field mat }=>(nat \times 'a) list wher
    triangular-to-jnf-vector }A\equiv\mathrm{ let }B=\mathrm{ step-2 (step-1 A)
        in concat (map (jnf-vector o step-3) (partition-ev-blocks B []))
```


### 18.3 Preservation of Dimensions

lemma swap-cols-rows-block-dims-main:
dim-row (swap-cols-rows-block ij $A$ ) dim-row $A \wedge$ dim-col (swap-cols-rows-block
i j $A$ ) $=\operatorname{dim}-\operatorname{col} A$
proof (induct ij A rule: swap-cols-rows-block.induct)
case ( $1 i j A$ )
thus ?case unfolding swap-cols-rows-block.simps[of i $j A]$
by (auto split: if-splits)
qed
lemma swap-cols-rows-block-dims[simp]:

```
    dim-row (swap-cols-rows-block i j A) = dim-row A
    dim-col (swap-cols-rows-block i j A) = dim-col A
    A carrier-mat n n\Longrightarrow swap-cols-rows-block i j A \in carrier-mat n n
    using swap-cols-rows-block-dims-main unfolding carrier-mat-def by auto
lemma step-1-main-dims-main:
    dim-row (step-1-main n i j A) = dim-row A ^dim-col (step-1-main n ij A) =
dim-col A
proof (induct i j A taking: n rule: step-1-main.induct)
    case (1 i j A)
    thus ?case unfolding step-1-main.simps[of n i j A]
        by (auto split: if-splits simp: Let-def)
qed
lemma step-1-main-dims[simp]:
    dim-row (step-1-main n i j A) = dim-row A
    dim-col (step-1-main n i j A) = dim-col A
    using step-1-main-dims-main by blast+
lemma step-2-main-dims-main:
    dim-row (step-2-main n j A) = dim-row A ^dim-col (step-2-main n j A) =
dim-col A
proof (induct j A taking: n rule: step-2-main.induct)
    case (1 j A)
    thus ?case unfolding step-2-main.simps[of n j A]
    by (auto split: if-splits option.splits simp: Let-def)
qed
lemma step-2-main-dims[simp]:
    dim-row (step-2-main n j A) = dim-row A
    dim-col (step-2-main n j A) = dim-col A
    using step-2-main-dims-main by blast+
lemma step-3-a-dims-main:
    dim-row (step-3-a i j A) = dim-row A ^dim-col (step-3-a i j A) = dim-col A
    by (induct i j A rule: step-3-a.induct, auto split: if-splits simp: Let-def)
lemma step-3-a-dims[simp]:
    dim-row (step-3-a i j A) = dim-row A
    dim-col (step-3-a i j A) = dim-col A
    using step-3-a-dims-main by blast+
lemma step-3-c-inner-loop-dims-main:
    dim-row (step-3-c-inner-loop val l i j A) = dim-row A ^ dim-col (step-3-c-inner-loop
val l i j A) = dim-col A
    by (induct val li j A rule: step-3-c-inner-loop.induct,auto)
lemma step-3-c-inner-loop-dims[simp]:
    dim-row (step-3-c-inner-loop val l i j A) = dim-row A
```

```
    dim-col (step-3-c-inner-loop val l i j A) = dim-col A
    using step-3-c-inner-loop-dims-main by blast+
lemma step-3-c-dims-main:
    dim-row (step-3-c x lkiA) = dim-row A ^dim-col(step-3-c x lkiA) = dim-col
A
    by (induct x l k i A rule: step-3-c.induct, auto simp: Let-def)
lemma step-3-c-dims[simp]:
    dim-row (step-3-c x l k i A) = dim-row A
    dim-col (step-3-c x l k i A) = dim-col A
    using step-3-c-dims-main by blast+
lemma step-3-main-dims-main:
    dim-row (step-3-main n k A) = dim-row A ^dim-col (step-3-main n k A) =
dim-col A
proof (induct k A taking: n rule: step-3-main.induct)
    case (1 k A)
    thus ?case unfolding step-3-main.simps[of n k A]
    by (auto split: if-splits prod.splits option.splits simp: Let-def)
qed
lemma step-3-main-dims[simp]:
    dim-row (step-3-main n j A) = dim-row A
    dim-col (step-3-main n j A) = dim-col A
    using step-3-main-dims-main by blast+
lemma triangular-to-jnf-steps-dims[simp]:
    dim-row (step-1 A) = dim-row }
    dim-col (step-1 A) = dim-col A
    dim-row (step-2 A) = dim-row }
    dim-col (step-2 A) = dim-col A
    dim-row (step-3 A) = dim-row A
    dim-col (step-3 A) = dim-col A
    unfolding step-1-def step-2-def step-3-def o-def by auto
```


### 18.4 Properties of Auxiliary Algorithms

```
lemma lookup-ev-Some:
lookup-ev ev \(j A=\) Some \(i \Longrightarrow\)
\(i<j \wedge A \$ \$(i, i)=e v \wedge(\forall k . i<k \longrightarrow k<j \longrightarrow A \$ \$(k, k) \neq e v)\)
by (induct \(j\), auto split: if-splits, case-tac \(k=j\), auto)
lemma lookup-ev-None: lookup-ev ev \(j A=\) None \(\Longrightarrow i<j \Longrightarrow A \$ \$(i, i) \neq e v\) by (induct \(j\), auto split: if-splits, case-tac \(i=j\), auto)
lemma swap-cols-rows-block-index[simp]:
\(i<\) dim-row \(A \Longrightarrow i<\) dim-col \(A \Longrightarrow j<\) dim-row \(A \Longrightarrow j<\operatorname{dim}-c o l A\)
\(\Longrightarrow\) low \(\leq\) high \(\Longrightarrow\) high \(<\) dim-row \(A \Longrightarrow\) high \(<\) dim-col \(A\)
```

```
    " swap-cols-rows-block low high A $$ (i,j) = A $$
    (if i}=\mathrm{ low then high else if i> low }\wedgei\leqhigh then i-1 else i
    if j= low then high else if j> low }\wedgej\leqhigh then j - 1 else j
proof (induct low high A rule: swap-cols-rows-block.induct)
    case (1 low high A)
    let ?r = dim-row A let ?c = dim-col A
    let ?A = swap-cols-rows-block low high A
    let ?B = swap-cols-rows low high A
    let ?C = swap-cols-rows-block (Suc low) high ?B
    note [simp] = swap-cols-rows-block.simps[of low high A]
    from 1(2-) have lh: low \leq high by simp
    show ?case
    proof (cases low < high)
    case False
    with lh have lh:low = high by auto
    from False have ?A = A by simp
    with lh show ?thesis by auto
    next
    case True
    hence A: ?A = ?C by simp
    from True lh have Suc low \leq high by simp
    note IH = 1(1)[unfolded swap-cols-rows-carrier,OF True 1(2-5) this 1(7-)]
    note * = 1(2-)
    let ?i = if i=Suc low then high else if Suc low < i^i\leqhigh then i-1 else
i
    let ?j = if j = Suc low then high else if Suc low < j^j\leq high then j - 1 else
j
    have cong: \bigwedge ij i' j'. i= i' \Longrightarrowj= j'\LongrightarrowA$$(i,j)=A$$( i', j) by auto
    have ij: ?i < ?r ?i < ?c ?j < ?r ?j < ?c low < ?r high < ?r using * True
by auto
    show ?thesis unfolding A IH
        by (subst swap-cols-rows-index[OF ij], rule cong, insert * True, auto)
    qed
qed
lemma find-largest-block-main: assumes find-largest-block block blocks = (m-b,
m-e)
    shows (m-b,m-e) \in insert block (set blocks)
    \wedge (\forallb\in insert block (set blocks). m-e - m-b \geq snd b - fst b)
    using assms
proof (induct block blocks rule: find-largest-block.induct)
    case (2 m-start m-end i-start i-end blocks)
    let ?res = find-largest-block (m-start,m-end) ((i-start,i-end) # blocks)
    show ?case
    proof (cases i-end - i-start \geqm-end - m-start)
        case True
        with 2(3-) have find-largest-block (i-start,i-end) blocks = (m-b,m-e) by auto
        note IH = 2(1)[OF True this]
    thus ?thesis using True by auto
```

```
    next
        case False
        with 2(3-) have find-largest-block (m-start,m-end) blocks =(m-b,m-e) by
auto
    note IH=2(2)[OF False this]
    thus ?thesis using False by auto
    qed
qed simp
lemma find-largest-block: assumes bl: blocks \not= []
    and find: find-largest-block (hd blocks) (tl blocks) = (m-begin, m-end)
    shows (m-begin,m-end) \in set blocks
    \i-begin i-end. (i-begin,i-end ) 
i-begin
proof -
    from bl have id: insert (hd blocks) (set (tl blocks)) = set blocks by (cases blocks,
auto)
    from find-largest-block-main[OF find, unfolded id]
    show (m-begin,m-end) \in set blocks
    \i-begin i-end. (i-begin,i-end ) { set blocks \Longrightarrowm-end - m-begin }\geqi\mathrm{ -end -
i-begin by auto
qed
context
    fixes ev :: 'a :: one
    and A :: 'a mat
begin
lemma identify-block-main: assumes identify-block A j = i
    shows }i\leqj\wedge(i=0\veeA$$(i-1,i)\not=1)\wedge(\forallk.i\leqk\longrightarrowk<j\longrightarrow
$$(k, Suc k) = 1)
        (is ?P j)
    using assms
proof (induct j)
    case (Suc j)
    show ?case
    proof (cases A $$ (j, Suc j)=1)
        case False
        with Suc(2) show ?thesis by auto
    next
        case True
        with Suc(2) have identify-block A j=i by simp
        note IH = Suc(1)[OF this]
        {
        fix }
        assume k\geqi and k<Suc j
        hence A$$(k,Suc k)=1
        proof (cases k<j)
            case True
```

```
            with IH <k\geqi\rangle show ?thesis by auto
        next
            case False
            with «k<Suc j> have k = j by auto
            with True show ?thesis by auto
        qed
    }
    with IH show ?thesis by auto
    qed
qed simp
lemma identify-block-le: identify-block A i\leqi
    using identify-block-main[OF refl] by blast
end
lemma identify-block: assumes identify-block A j=i
    shows i\leqj
    i=0\veeA$$ (i-1,i)\not=1
    i\leqk\Longrightarrowk<j\LongrightarrowA$$(k,Suc k)=1
proof -
    let ?ev = A $$ (j,j)
    note main = identify-block-main[OF assms]
    from main show i\leqj by blast
    from main show }i=0\veeA$$(i-1,i)\not=1\mathrm{ by blast
    assume i\leqk
    with main have main: k<j\LongrightarrowA$$(k,Suck)=1 by blast
    thus }k<j\LongrightarrowA$$(k,Suc k)=1 by blas
qed
lemmas identify-block-le' = identify-block(1)
lemma identify-block-le-rev: j = identify-block A i\Longrightarrowj\leqi
    using identify-block-le'[of A i j] by auto
termination identify-blocks-main by (relation measure ( }\lambda\mathrm{ (-,i,list). i),
    auto simp: identify-block-le le-imp-less-Suc)
termination jnf-vector-main by (relation measure ( }\lambda(i,A).i)
    auto simp: identify-block-le le-imp-less-Suc)
lemma identify-blocks-main: assumes (i-start,i-end) \in set (identify-blocks-main
A i list)
    and \i-s i-e. (i-s,i-e) \in set list \Longrightarrow i-s\leqi-e ^i-e<k
    and i\leqk
    shows i-start }\leqi\mathrm{ -end }\wedgei\mathrm{ -end <k using assms
proof (induct A i list rule: identify-blocks-main.induct)
    case (2 A i-e list)
```

obtain $i$ - $b$ where $i d$ : identify-block $A i-e=i-b$ by force
note $I H=2(1)[$ OF id[symmetric $]]$
let ?res $=$ identify-blocks-main $A($ Suc $i$-e) list
let ?rec $=$ identify-blocks-main A i-b $((i-b, i-e) \#$ list $)$
have res: ? res $=$ ? rec using id by simp
from 2(2)[unfolded res] have (i-start, $i$-end) $\in$ set ?rec.
note $I H=I H[$ OF this $]$
from 2(3-) have iek: $i-e<k$ by simp
from identify-block-le' $[O F$ id] have ibe: $i-b \leq i-e$.
from $i b e ~ i e k$ have $i-b \leq k$ by simp
note $I H=I H[O F-t h i s]$
show ?thesis
by (rule IH, insert ibe iek 2(3-), auto)
qed $\operatorname{simp}$
lemma identify-blocks: assumes (i-start,i-end) $\in \operatorname{set}($ identify-blocks $B k)$
shows $i$-start $\leq i$-end $\wedge i$-end $<k$
using identify-blocks-main[OF assms[unfolded identify-blocks-def]] by auto

### 18.5 Proving Similarity

context
begin
private lemma swap-cols-rows-block-similar: assumes $A \in$ carrier-mat $n n$ and $j<n$ and $i \leq j$
shows similar-mat (swap-cols-rows-block ij A) A
using assms
proof (induct ij A rule: swap-cols-rows-block.induct)
case ( $1 i j A$ )
hence $A: A \in$ carrier-mat $n n$
and $j n: j<n$ and $i j: i \leq j$ by auto
note $[$ simp $]=$ swap-cols-rows-block.simps $[$ of $i j]$
show ? case
proof (cases $i<j$ )
case False
thus ?thesis using similar-mat-refl $[O F A]$ by simp
next
case True note $i j=$ this
let ? $B=$ swap-cols-rows $i j A$
let ? $C=$ swap-cols-rows-block (Suc i) $j$ ? $B$
from swap-cols-rows-similar [OF A-jn, of i] ij jn
have $B A$ : similar-mat ? $B A$ by auto
have $C B$ : similar-mat ? $C$ ? $B$
by (rule $1(1)[O F i j-j n]$, insert ij $A$, auto)
from similar-mat-trans $[O F C B B A]$ show ?thesis using ij by simp qed
qed
private lemma step-1-main-similar: $i \leq j \Longrightarrow A \in$ carrier-mat $n n \Longrightarrow$ simi-

```
lar-mat (step-1-main n i j A) A
proof (induct ij A taking: n rule: step-1-main.induct)
    case (1 i j A)
    note [simp] = step-1-main.simps[of n i j A]
    from 1(3-) have ij:i\leqj and A:A\incarrier-mat n n by auto
    show ?case
    proof (cases j\geqn)
        case True
        thus ?thesis using similar-mat-refl[OF A] by simp
    next
    case False
    hence jn: j<n by simp
    note IH=1(1-2)[OF False]
    show ?thesis
    proof (cases i=0)
        case True
        from IH(1)[OF this - A]
        show ?thesis using jn by (simp, simp add: True)
    next
        case False
        let ?evi=A $$ (i-1,i-1)
        let ?evj = A $$ (j,j)
        let ? B= if ?evi\not=?evj \wedgeA$$ (i-1,j)\not=0 then
                add-col-sub-row (A $$ (i-1,j) / (?evj - ?evi)) (i-1) j A else A
        obtain }B\mathrm{ where }B:B=?B\mathrm{ by auto
        have Bn: B carrier-mat n n unfolding B using }A\mathrm{ by simp
        from False ij jn have *: i-1<nj<ni-1\not=j by auto
        have BA: similar-mat B A unfolding B using similar-mat-refl[OF A]
                add-col-sub-row-similar[OF A *] by auto
        from ij have i-1\leqj by simp
        note IH =IH(2)[OF False refl refl refl refl B this Bn]
        from False jn have id: step-1-main n i j A = step-1-main n (i-1) j B
            unfolding B by (simp add: Let-def)
        show ?thesis unfolding id
            by (rule similar-mat-trans[OF IH BA])
        qed
    qed
qed
private lemma step-2-main-similar: \(A \in\) carrier-mat \(n n \Longrightarrow\) similar-mat (step-2-main
nj A) A
proof (induct j A taking: n rule: step-2-main.induct)
    case (1 j A)
    note [simp]=step-2-main.simps[of n j A]
    from 1(2) have A: A \in carrier-mat n n.
    show ?case
    proof (cases j\geqn)
    case True
    thus ?thesis using similar-mat-refl[OF A] by simp
```

```
    next
    case False
    hence jn: j<n by simp
    note IH=1(1)[OF False]
    let ?look= lookup-ev (A $$ (j,j)) j A
    let ?B= case ?look of
                None }=>
            | Some i m swap-cols-rows-block (Suc i) j A
    obtain B where B: B=?B by auto
    have id: step-2-main n j A = step-2-main n (Suc j) B unfolding B using
False by simp
    have Bn: B carrier-mat n n unfolding B using A by (auto split:op-
tion.splits)
    have BA: similar-mat B A
    proof (cases ?look)
            case None
            thus ?thesis unfolding B using similar-mat-refl[OF A] by simp
    next
            case (Some i)
            from lookup-ev-Some[OF this]
            show ?thesis unfolding B Some
                by (auto intro: swap-cols-rows-block-similar[OF A jn])
    qed
    show ?thesis unfolding id
        by (rule similar-mat-trans[OF IH[OF refl B Bn] BA])
    qed
qed
private lemma step-3-a-similar: A \in carrier-mat n n \Longrightarrowi<j\Longrightarrowj<n\Longrightarrow
similar-mat (step-3-a i j A) A
proof (induct i j A rule: step-3-a.induct)
    case (1 j A)
    thus ?case by (simp add: similar-mat-refl)
next
    case (2 i j A)
    from 2(2-) have A:A\incarrier-mat n n and ij: Suc i<j and j:j<n by
auto
    let ?B= if A $$(i,i+1)=1^A$$(i,j)\not=0
        then add-col-sub-row (- A $$ (i,j)) (Suc i) j A else A
    obtain }B\mathrm{ where }B:B=?B\mathrm{ by auto
    from A have Bn: B\incarrier-mat n n unfolding B by simp
    note IH=2(1)[OF refl B Bn-j]
    have id: step-3-a (Suc i) j A = step-3-a i j B unfolding B by (simp add:
Let-def)
    from ij j have *: Suc i<nj<n Suc i\not=j by auto
    have BA: similar-mat B A unfolding B
    using similar-mat-refl[OF A] add-col-sub-row-similar[OF A *] by auto
    show ?case unfolding id
    by (rule similar-mat-trans[OF IH BA], insert ij, auto)
```


## qed

private lemma step-3-c-inner-loop-similar:

$$
A \in \text { carrier-mat } n n \Longrightarrow l \neq i \Longrightarrow k-1 \leq l \Longrightarrow k-1 \leq i \Longrightarrow l<n \Longrightarrow i
$$

$$
<n \Longrightarrow
$$

similar-mat (step-3-c-inner-loop val likA) A
proof (induct val lik A rule: step-3-c-inner-loop.induct)
case (1 val li A)
thus ?case using similar-mat-refl[of $A]$ by simp
next
case (2 val likA)
let ?res $=$ step-3-c-inner-loop val l $i($ Suc $k)$ A
from 2(2-) have $A: A \in$ carrier-mat $n n$ and $*: l \neq i k \leq l k \leq i l<n i<n$ by auto
let ? $B=$ add-col-sub-row val il $A$
have $B A$ : similar-mat ? $B A$
by (rule add-col-sub-row-similar $[O F A]$, insert *, auto)
have $B: ? B \in$ carrier-mat $n n$ using $A$ unfolding carrier-mat-def by simp
show? case
proof (cases $k$ )
case 0
hence id: ? res $=$ ? $B$ by simp
thus ?thesis using $B A$ by simp
next
case (Suc $k k$ )
with $*$ have $l-1 \neq i-1 k-1 \leq l-1 k-1 \leq i-1 l-1<n i-1<$
$n$ by auto
note $I H=2(1)[O F B$ this]
show ?thesis unfolding step-3-c-inner-loop.simps
by (rule similar-mat-trans[OF IH BA])
qed
qed
private lemma step-3-c-similar:
$A \in$ carrier-mat $n n \Longrightarrow l<k \Longrightarrow k<n$
$\Longrightarrow(\bigwedge i$-begin $i$-end. $(i$-begin, $i$-end $) \in$ set blocks $\Longrightarrow \quad i$-end $\leq k \wedge i$-end -
$i$-begin $\leq l$ )
$\Longrightarrow$ similar-mat (step-3-c x lk blocks A) A
proof (induct $x l k$ blocks A rule: step-3-c.induct)
case ( $1 x l k A$ )
thus ?case using similar-mat-refl[OF 1(1)] by simp
next
case (2 x lki-begin i-end blocks A)
let ?res $=$ step-3-c x lk((i-begin,i-end $) \#$ blocks $) ~ A$
from 2(2-4) have $A: A \in$ carrier-mat $n n$ and $l k i: l<k k<n$ by auto
from 2(5) have $i$ : $i$-end $\leq k i$-end $-i$-begin $\leq l$ by auto
let $? y=A \$ \$(i$-end,$k)$
let ?inner $=$ step-3-c-inner-loop $(? y / x) l$ i-end $(S u c i$-end $-i$-begin) $A$
obtain $B$ where $B$ :

```
    B=(if i-end =l then A else ?inner ) by auto
    hence id:?res = step-3-c x l k blocks B
    by simp
    have BA: similar-mat B A
    proof (cases i-end = l)
        case True
        thus ?thesis unfolding B using similar-mat-refl[OF A] by simp
    next
        case False
        hence B:B=? ?inner and li:l\not=i-end by (auto simp: B)
        show ?thesis unfolding B
        by (rule step-3-c-inner-loop-similar[OF A li], insert lki i, auto)
qed
have Bn: B\incarrier-mat n n using A unfolding B carrier-mat-def by simp
note IH = 2(1)[OF B Bn lki(1-2) 2(5)]
show ?case unfolding id
    by (rule similar-mat-trans[OF IH BA], auto)
qed
private lemma step-3-main-similar: A carrier-mat n n \Longrightarrow k>0\Longrightarrowsimi-
lar-mat (step-3-main n k A) A
proof (induct k A taking: n rule: step-3-main.induct)
    case (1kA)
    from 1(2-) have A:A\incarrier-mat n n and k:k>0 by auto
    note [simp] = step-3-main.simps[of n k A]
    show ?case
    proof (cases k\geqn)
        case True
        thus ?thesis using similar-mat-refl[OF A] by simp
    next
        case False
        hence kn: k< n by simp
        obtain B where B: B= step-3-a (k-1) k A by auto
        note IH=1(1)[OF False B]
        from A have Bn: B c carrier-mat n n unfolding B carrier-mat-def by simp
        from k}\mathrm{ have }k-1<k\mathrm{ by simp
        from step-3-a-similar[OF A this kn] have BA: similar-mat B A unfolding B
    obtain all-blocks where ab: all-blocks = identify-blocks B k by simp
    obtain blocks where blocks: blocks = filter ( }\lambda\mathrm{ block. B $$ (snd block,k)}=00
all-blocks by simp
    obtain F where F:F=(if blocks = [] then B
    else let (l-begin,l) = find-largest-block (hd blocks) (tl blocks); x=B $$ (l, k);
C= step-3-c x l k blocks B;
                    D=mult-col-div-row (inverse x) kC;E=swap-cols-rows-block (Suc l)
k D
            in E) by simp
    note IH=IH[OF ab blocks F]
    have Fn: F \in carrier-mat n n unfolding F Let-def carrier-mat-def using Bn
```

```
    by (simp split: prod.splits)
    have FB: similar-mat F B
    proof (cases blocks=[])
    case True
    thus ?thesis unfolding F using similar-mat-refl[OF Bn] by simp
    next
    case False
    obtain l-start l where l: find-largest-block (hd blocks) (tl blocks) = (l-start,
l) by force
    obtain x where x: x = B $$ (l,k) by simp
    obtain C where C:C=step-3-c x l k blocks B by simp
    obtain D where D: D = mult-col-div-row (inverse x) kC by auto
    obtain E where E: E = swap-cols-rows-block (Suc l) kD by auto
    from find-largest-block[OF False l] have lb: (l-start,l) \in set blocks
        and llarge: }\bigwedgei\mathrm{ -begin i-end. (i-begin,i-end) }\in\mathrm{ set blocks #l-l-start }
i-end - i-begin by auto
    from lb have x0:x\not=0 unfolding blocks x by simp
    {
        fix i-start i-end
        assume (i-start,i-end) \in set blocks
            hence (i-start,i-end) \in set (identify-blocks B k) unfolding blocks ab by
simp
            from identify-blocks[OF this]
            have i-end <k by simp
    } note block-bound = this
    from block-bound[OF lb]
    have lk:l<k.
    from False have F:F=E unfolding E D Cx Fl Let-def by simp
    from Bn have Cn: C carrier-mat n n unfolding C carrier-mat-def by
simp
    have CB: similar-mat C B unfolding C
    proof (rule step-3-c-similar[OF Bn lk kn])
        fix i-begin i-end
        assume i:(i-begin, i-end) \in set blocks
        from llarge [OF i] block-bound[OF i]
        show i-end }\leqk\wedgei\mathrm{ -end - i-begin }\leql\mathrm{ by auto
    qed
    from x0 have inverse x\not=0 by simp
    from mult-col-div-row-similar[OF Cn kn this]
    have DC: similar-mat DC unfolding D .
    from Cn have Dn: D \in carrier-mat n n unfolding D carrier-mat-def by
simp
    from lk have Sucl l sk by auto
    from swap-cols-rows-block-similar[OF Dn kn this]
    have ED: similar-mat E D unfolding E .
    from similar-mat-trans[OF ED similar-mat-trans[OF DC
            similar-mat-trans[OF CB similar-mat-refl[OF Bn]]]]
        show ?thesis unfolding F .
    qed
```

```
    have 0<Suc k by simp
    note IH = IH[OF Fn this]
    have id: step-3-main n k A = step-3-main n (Suc k)F using kn
    by (simp add: F Let-def blocks ab B)
    show ?thesis unfolding id
        by (rule similar-mat-trans[OF IH similar-mat-trans[OF FB BA]])
    qed
qed
lemma step-1-similar: A \in carrier-mat n n\Longrightarrow similar-mat (step-1 A) A
    unfolding step-1-def by (rule step-1-main-similar, auto)
lemma step-2-similar: A \in carrier-mat n n\Longrightarrow similar-mat (step-2 A) A
    unfolding step-2-def by (rule step-2-main-similar, auto)
lemma step-3-similar: A \in carrier-mat n n\Longrightarrow similar-mat (step-3 A) A
    unfolding step-3-def by (rule step-3-main-similar, auto)
end
```


### 18.6 Invariants for Proving that Result is in JNF

 context    fixes \(n\) :: nat
    and \(t y::\) ' \(a\) :: field itself
    begin
definition uppert $::$ ' $a$ mat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ bool where
uppert $A i j \equiv j<i \longrightarrow A \$ \$(i, j)=0$
definition diff-ev :: ' $a$ mat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ bool where
diff-ev $A i j \equiv i<j \longrightarrow A \$ \$(i, i) \neq A \$ \$(j, j) \longrightarrow A \$ \$(i, j)=0$
definition ev-blocks-part :: nat $\Rightarrow$ 'a mat $\Rightarrow$ bool where
ev-blocks-part $m A \equiv \forall i j k . i<j \longrightarrow j<k \longrightarrow k<m \longrightarrow A \$ \$(k, k)=A \$ \$$
$(i, i) \longrightarrow A \$ \$(j, j)=A \$ \$(i, i)$
definition ev-blocks :: 'a mat $\Rightarrow$ bool where
ev-blocks $\equiv$ ev-blocks-part $n$

In step 3, there is a separation at which iteration we are. The columns left of $k$ will be in JNF, the columns right of $k$ or equal to $k$ will satisfy local.uppert, local.diff-ev, and local.ev-blocks, and the column at $k$ will have one of the following properties, which are ensured in the different phases of step 3.
private definition one-zero :: 'a mat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ bool where

```
one-zero A ij\equiv
    (Suc i<j\longrightarrowA$$(i,Suc i)=1\longrightarrowA$$(i,j)=0)^
    (j<i\longrightarrowA$$ (i,j)=0)^
    (i<j\longrightarrowA$$(i,i)\not=A$$(j,j)\longrightarrowA$$(i,j)=0)
```

private definition single-non-zero :: nat $\Rightarrow$ nat $\Rightarrow{ }^{\prime} a \Rightarrow{ }^{\prime} a$ mat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ bool where
single-non-zero $\equiv \lambda l k x .(\lambda A i j .(i \notin\{k, l\} \longrightarrow A \$ \$(i, k)=0) \wedge A \$ \$(l, k)$ $=x)$
private definition single-one $::$ nat $\Rightarrow n a t \Rightarrow{ }^{\prime} a$ mat $\Rightarrow n a t \Rightarrow n a t \Rightarrow$ bool where single-one $\equiv \lambda l k .(\lambda A i j .(i \notin\{k, l\} \longrightarrow A \$ \$(i, k)=0) \wedge A \$ \$(l, k)=1)$
private definition lower-one $::$ nat $\Rightarrow{ }^{\prime} a$ mat $\Rightarrow n a t \Rightarrow n a t \Rightarrow$ bool where lower-one $k$ A $i j=(j=k \longrightarrow$
$(A \$ \$(i, j)=0 \vee i=j \vee(A \$ \$(i, j)=1 \wedge j=S u c i \wedge A \$ \$(i, i)=A \$ \$$ $(j, j)))$ )
definition $j b::$ 'a mat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ bool where

$$
\begin{aligned}
& j b A i j \equiv(\text { Suc } i=j \longrightarrow A \$ \$(i, j) \in\{0,1\}) \\
& \wedge(i \neq j \longrightarrow(S u c i \neq j \vee A \$ \$(i, i) \neq A \$ \$(j, j)) \longrightarrow A \$ \$(i, j)=0)
\end{aligned}
$$

The following properties are useful to easily ensure the above invariants just from invariants of other matrices. The properties are essential in showing that the blocks identified in step 3 b are the same as one would identify for the matrices in the upcoming steps 3 c and 3 d .

```
definition same-diag :: 'a mat }=>\mathrm{ 'a mat }=>\mathrm{ bool where
```

$$
\text { same-diag } A B \equiv \forall i<n . A \$ \$(i, i)=B \$ \$(i, i)
$$

private definition same-upto :: nat $\Rightarrow{ }^{\prime}$ a mat $\Rightarrow{ }^{\prime}$ a mat $\Rightarrow$ bool where same-upto $j A B \equiv \forall i^{\prime} j^{\prime} . i^{\prime}<n \longrightarrow j^{\prime}<j \longrightarrow A \$ \$\left(i^{\prime}, j^{\prime}\right)=B \$ \$\left(i^{\prime}, j^{\prime}\right)$

Definitions stating where the properties hold
definition inv-all $::\left({ }^{\prime}\right.$ a mat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ bool $) \Rightarrow{ }^{\prime}$ a mat $\Rightarrow$ bool where inv-all $p A \equiv \forall i j . i<n \longrightarrow j<n \longrightarrow p A i j$
private definition inv-part :: ('a mat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ bool $) \Rightarrow{ }^{\prime}$ a mat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ bool where
inv-part p $A m-i m-j \equiv \forall i j . i<n \longrightarrow j<n \longrightarrow j<m-j \vee j=m-j \wedge i \geq m-i$ $\longrightarrow p A i j$
private definition inv-upto $::$ ('a mat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ bool) $\Rightarrow$ 'a mat $\Rightarrow$ nat $\Rightarrow$ bool where
inv-upto $p A m \equiv \forall i j . i<n \longrightarrow j<n \longrightarrow j<m \longrightarrow p A i j$
private definition inv-from $::\left({ }^{\prime}\right.$ a mat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ bool $) \Rightarrow{ }^{\prime}$ a mat $\Rightarrow$ nat $\Rightarrow$ bool where
inv-from $p A m \equiv \forall i j . i<n \longrightarrow j<n \longrightarrow j>m \longrightarrow p A i j$
private definition inv-at $::($ 'a mat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ bool $) \Rightarrow$ 'a mat $\Rightarrow$ nat $\Rightarrow$ bool where
inv-at $p A m \equiv \forall i . i<n \longrightarrow p A i m$
private definition inv-from-bot $::\left({ }^{\prime}\right.$ a mat $\Rightarrow$ nat $\Rightarrow$ bool) $\Rightarrow{ }^{\prime}$ a mat $\Rightarrow$ nat $\Rightarrow$ bool where
inv-from-bot p $A \mathrm{mi} \equiv \forall i . i \geq m i \longrightarrow i<n \longrightarrow p A i$
Auxiliary Lemmas on Handling, Comparing, and Accessing Invariants

```
lemma jb-imp-uppert: \(j b A i j \Longrightarrow\) uppert \(A i j\)
    unfolding \(j b\)-def uppert-def by auto
private lemma ev-blocks-partD:
    ev-blocks-part \(m A \Longrightarrow i<j \Longrightarrow j<k \Longrightarrow k<m \Longrightarrow A \$ \$(k, k)=A \$ \$(i, i)\)
\(\Longrightarrow A \$ \$(j, j)=A \$ \$(i, i)\)
    unfolding ev-blocks-part-def by auto
private lemma ev-blocks-part-leD:
    assumes ev-blocks-part m A
    \(i \leq j j \leq k k<m A \$ \$(k, k)=A \$ \$(i, i)\)
    shows \(A \$ \$(j, j)=A \$ \$(i, i)\)
proof -
    show ?thesis
    proof (cases \(i=j \vee j=k\) )
        case False
        with assms(2-3) have \(i<j j<k\) by auto
        from ev-blocks-partD \([O F\) assms(1) this assms(4-)] show ?thesis .
    next
        case True
        thus ?thesis using assms(5) by auto
    qed
qed
private lemma ev-blocks-partI:
    assumes \(\bigwedge i j k . i<j \Longrightarrow j<k \Longrightarrow k<m \Longrightarrow A \$ \$(k, k)=A \$ \$(i, i) \Longrightarrow\)
\(A \$ \$(j, j)=A \$ \$(i, i)\)
    shows ev-blocks-part m A
    using assms unfolding ev-blocks-part-def by blast
private lemma ev-blocksD:
    ev-blocks \(A \Longrightarrow i<j \Longrightarrow j<k \Longrightarrow k<n \Longrightarrow A \$ \$(k, k)=A \$ \$(i, i) \Longrightarrow A\)
\(\$ \$(j, j)=A \$ \$(i, i)\)
    unfolding ev-blocks-def by (rule ev-blocks-partD)
private lemma ev-blocks-leD:
```

```
    ev-blocks A\Longrightarrowi\leqj\Longrightarrowj\leqk\Longrightarrowk<n\LongrightarrowA$$(k,k)=A$$(i,i)\LongrightarrowA
$$ (j,j)=A $$ (i,i)
    unfolding ev-blocks-def by (rule ev-blocks-part-leD)
lemma inv-allD: inv-all pA\Longrightarrowi<n\Longrightarrowj<n\LongrightarrowpAij
    unfolding inv-all-def by auto
private lemma inv-allI: assumes }\bigwedgeij.i<n\Longrightarrowj<n\LongrightarrowpAi
    shows inv-all p A
    using assms unfolding inv-all-def by blast
private lemma inv-partI: assumes }\ij.i<n\Longrightarrowj<n\Longrightarrowj<m-j\veej
m-j^i\geqm-i\LongrightarrowpA ij
    shows inv-part p A m-i m-j
    using assms unfolding inv-part-def by auto
private lemma inv-partD: assumes inv-part p A m-i m-j i<nj<n
    shows j<m-j\LongrightarrowpAij
    and}j=m-j\Longrightarrowi\geqm-i\LongrightarrowpAi
    and}j<m-j\veej=m-j\wedgei\geqm-i\LongrightarrowpAi
    using assms unfolding inv-part-def by auto
private lemma inv-uptoI: assumes }\ij.i<n\Longrightarrowj<n\Longrightarrowj<m\Longrightarrowp
ij
    shows inv-upto p A m
    using assms unfolding inv-upto-def by auto
private lemma inv-uptoD: assumes inv-upto p A mi<nj<nj<m
    shows pA ij
    using assms unfolding inv-upto-def by auto
private lemma inv-upto-Suc: assumes inv-upto p A m
    and }\bigwedgei.i<n\LongrightarrowpAi
    shows inv-upto p A (Suc m)
proof (intro inv-uptoI)
    fix ij
    assume i<nj<nj<Suc m
    thus p A i j using inv-uptoD[OF assms(1), of i j] assms(2)[of i] by (cases j =
m, auto)
qed
private lemma inv-upto-mono: assumes }\ij.i<n\Longrightarrowj<k\LongrightarrowpAij
q A ij
    shows inv-upto p A k\Longrightarrow inv-upto q A k
    using assms unfolding inv-upto-def by auto
private lemma inv-fromI: assumes }\ij.i<n\Longrightarrowj<n\Longrightarrowj>m\Longrightarrowp
i j
    shows inv-from p A m
```

using assms unfolding inv-from-def by auto
private lemma inv-fromD: assumes inv-from $p A m i<n j<n j>m$ shows $p A i j$
using assms unfolding inv-from-def by auto
private lemma inv-atI[intro]: assumes $\bigwedge i . i<n \Longrightarrow p A i m$ shows inv-at $p A m$
using assms unfolding inv-at-def by auto
private lemma inv-atD: assumes inv-at pAmi<n
shows $p A i m$
using assms unfolding inv-at-def by auto
private lemma inv-all-imp-inv-part: $m i \leq n \Longrightarrow m-j \leq n \Longrightarrow$ inv-all $p A \Longrightarrow$ inv-part p A m-i m-j
unfolding inv-all-def inv-part-def by auto
private lemma inv-all-eq-inv-part: inv-all $p A=$ inv-part p $A n n$ unfolding inv-all-def inv-part-def by auto
private lemma inv-part- $0-S u c: m-j<n \Longrightarrow$ inv-part p A $0 m-j=$ inv-part $p$ A $n$ (Suc m-j)
unfolding inv-part-def by (auto, case-tac $j=m-j$, auto)
private lemma inv-all-uppertD: inv-all uppert $A \Longrightarrow j<i \Longrightarrow i<n \Longrightarrow A \$ \$$ $(i, j)=0$
unfolding inv-all-def uppert-def by auto
private lemma inv-all-diff-evD: inv-all diff-ev $A \Longrightarrow i<j \Longrightarrow j<n$
$\Longrightarrow A \$ \$(i, i) \neq A \$ \$(j, j) \Longrightarrow A \$ \$(i, j)=0$
unfolding inv-all-def diff-ev-def by auto
private lemma inv-all-diff-ev-uppertD: assumes inv-all diff-ev $A$
inv-all uppert $A$
$i<n j<n$
and neg: $A \$ \$(i, i) \neq A \$ \$(j, j)$
shows $A \$ \$(i, j)=0$
proof -
from neg have $i \neq j$ by auto
hence $i<j \vee j<i$ by arith
thus ?thesis
proof
assume $i<j$
from inv-all-diff-evD[OF assms(1) this $\langle j<n\rangle$ neg $]$ show ?thesis.
next
assume $j<i$
from inv-all-uppertD $[$ OF assms(2) this $\langle i<n\rangle]$ show ?thesis .
qed

## qed

private lemma inv-from-bot-step: $p A i \Longrightarrow$ inv-from-bot $p A(S u c i) \Longrightarrow$ inv-from-bot p A i
unfolding inv-from-bot-def by (auto, case-tac ia $=i$, auto)
private lemma same-diag-refl[simp]: same-diag $A$ A unfolding same-diag-def by auto
private lemma same-diag-trans: same-diag $A B \Longrightarrow$ same-diag $B C$ same-diag A C
unfolding same-diag-def by auto
private lemma same-diag-ev-blocks: same-diag $A B \Longrightarrow$ ev-blocks $A \Longrightarrow$ ev-blocks B
unfolding same-diag-def ev-blocks-def ev-blocks-part-def by auto
private lemma same-uptoI[intro]: assumes $\bigwedge i^{\prime} j^{\prime} \cdot i^{\prime}<n \Longrightarrow j^{\prime}<j \Longrightarrow A \$ \$$ $\left(i^{\prime}, j^{\prime}\right)=B \$ \$\left(i^{\prime}, j^{\prime}\right)$
shows same-upto $j A B$
using assms unfolding same-upto-def by blast
private lemma same-uptoD[dest]: assumes same-upto jAB $i^{\prime}<n j^{\prime}<j$
shows $A \$ \$\left(i^{\prime}, j^{\prime}\right)=B \$ \$\left(i^{\prime}, j^{\prime}\right)$
using assms unfolding same-upto-def by blast
private lemma same-upto-refl[simp]: same-upto j $A$ A unfolding same-upto-def by auto
private lemma same-upto-trans: same-upto j $A B \Longrightarrow$ same-upto j $B C \Longrightarrow$ same-upto j $A C$
unfolding same-upto-def by auto
private lemma same-upto-inv-upto-jb: same-upto j $A B \Longrightarrow$ inv-upto jb $A j \Longrightarrow$ inv-upto $j b B j$
unfolding inv-upto-def same-upto-def jb-def by auto
lemma jb-imp-diff-ev: $j b A i j \Longrightarrow d i f f-e v A i j$
unfolding $j b$-def diff-ev-def by auto
private lemma ev-blocks-diag:
same-diag $A B \Longrightarrow$ ev-blocks $B \Longrightarrow$ ev-blocks $A$
unfolding ev-blocks-def ev-blocks-part-def same-diag-def by auto
private lemma inv-all-imp-inv-from: inv-all $p A \Longrightarrow$ inv-from p $A k$
unfolding inv-all-def inv-from-def by auto
private lemma inv-all-imp-inv-at: inv-all $p A \Longrightarrow k<n \Longrightarrow$ inv-at $p A k$
unfolding inv-all-def inv-at-def by auto
private lemma inv-from-upto-at-all:
assumes inv-upto $j b A k$ inv-from diff-ev $A k$ inv-from uppert $A k$ inv-at p $A k$ and $\wedge i . i<n \Longrightarrow p A i k \Longrightarrow$ diff-ev $A i k \wedge$ uppert $A i k$
shows inv-all diff-ev $A$ inv-all uppert $A$

```
proof -
```

    \{
        fix \(i j\)
        assume \(i j: i<n j<n\)
        have diff-ev \(A i j \wedge\) uppert \(A i j\)
        proof (cases \(j<k\) )
            case True
            with assms(1) ij have \(j b A i j\) unfolding inv-upto-def by auto
            thus ?thesis using ij unfolding jb-def diff-ev-def uppert-def by auto
        next
            case False note \(g e=\) this
            show ?thesis
            proof (cases \(j=k\) )
                case True
                with \(\operatorname{assms}(4-)\) ij show ?thesis unfolding inv-at-def by auto
            next
                case False
                with \(g e\) have \(j>k\) by auto
                with \(\operatorname{assms}(2-3) i j\) show ?thesis unfolding inv-from-def by auto
            qed
        qed
    \}
    thus inv-all diff-ev \(A\) inv-all uppert \(A\) unfolding inv-all-def by auto
    qed
private lemma lower-one-diff-uppert:
$i<n \Longrightarrow$ lower-one $k B i k \Longrightarrow$ diff-ev B $i k \wedge$ uppert Bik
unfolding lower-one-def diff-ev-def uppert-def by auto
definition ev-block :: nat $\Rightarrow{ }^{\prime}$ a mat $\Rightarrow$ bool where
^n. ev-block $n A=(\forall i j . i<n \longrightarrow j<n \longrightarrow A \$ \$(i, i)=A \$ \$(j, j))$
lemma ev-blockD: $\bigwedge n$. ev-block $n A \Longrightarrow i<n \Longrightarrow j<n \Longrightarrow A \$ \$(i, i)=A \$ \$$
$(j, j)$
unfolding ev-block-def carrier-mat-def by blast
lemma same-diag-ev-block: same-diag $A B \Longrightarrow$ ev-block $n A \Longrightarrow$ ev-block $n B$
unfolding ev-block-def carrier-mat-def same-diag-def by metis
18.7 Alternative Characterization of identify-blocks in Presence of local.ev-block
private lemma identify-blocks-main-iff: assumes $*$ : $k \leq k^{\prime}$ $k^{\prime} \neq k \longrightarrow k>0 \longrightarrow A \$ \$(k-1, k) \neq 1$ and $k^{\prime}<n$ shows set (identify-blocks-main A $k$ list) $=$

```
    set list \cup{(i,j)| ij.i\leqj^j<k\wedge(\foralll.i\leql\longrightarrowl<j\longrightarrowA$$(l,Suc l)
= 1)
    \wedge(Suc j\not=k'l\longrightarrowA$$(j, Suc j)\not=1)^(i>0\longrightarrowA$$(i-1,i)\not=1)}(is
- = - U?ss A k)
    using *
proof (induct A k list rule: identify-blocks-main.induct)
    case 1
    show ?case unfolding identify-blocks-main.simps by auto
next
    case (2 A i-e list)
    let ?s = ?ss A
    obtain i-b where id: identify-block A i-e = i-b by force
    note IH = 2(1)[OF id[symmetric]]
    let ?res = identify-blocks-main A (Suc i-e) list
    let ?rec = identify-blocks-main A i-b ((i-b,i-e) # list)
    note idb = identify-block[OF id]
    hence res: ?res = ?rec using id by simp
    from 2(2-) have iek: i-e< k' by simp
    from identify-block-le}[{OFid] have ibe:i-b\leqi-e 
    from ibe iek have i-b\leqk' by simp
    have }\mp@subsup{k}{}{\prime}\not=i-b\longrightarrow0<i-b\longrightarrowA$$(i-b-1,i-b)\not=
    using idb(2) by auto
note IH = IH[OF<i-b\leq k'>this]
have cong: \ a bcd. insert a c=d\Longrightarrow set (a#b)\cupc= set b\cupd by auto
show ?case unfolding res IH
proof (rule cong)
    from ibe have ?s i-b \subseteq?s (Suc i-e) by auto
    moreover
    have inter: \l. i-b\leql\Longrightarrowl<i-e\LongrightarrowA$$(l,Suc l)=1 using idb by blast
    have last: Suc i-e }\not=\mp@subsup{k}{}{\prime}\LongrightarrowA$$(i-e, Suc i-e) \not=1 using 2(3) by aut
    have (i-b,i-e)\in?s (Suc i-e) using ibe idb(2) inter last by blast
    ultimately have insert (i-b,i-e) (?s i-b)\subseteq?s (Suc i-e) by auto
    moreover
    {
        fix ij
        assume ij: (i,j)\in?s (Suc i-e)
        hence (i,j) \in insert (i-b,i-e) (?s i-b)
        proof (cases j<i-b)
            case True
            with ij show ?thesis by blast
        next
            case False
            with ij have i-b\leqjj\leqi-e by auto
            {
                assume j: j<i-e
                from idb(3)[OF\langlei-b\leqj\ranglethis] have 1:A $$ (j,Suc j)=1.
                from j\langleSuc i-e\leq k'〉 have Suc j}\not=\mp@subsup{k}{}{\prime}\mathrm{ by auto
                with ij 1 have False by auto
            }
```

```
        with <j\leqi-e〉 have j: j=i-e by (cases j =i-e, auto)
        {
            assume i:i<i-b\veei> i-b
        hence False
        proof
            assume i<i-b
            hence i-b>0 by auto
            with idb(2) have *: A $$ (i-b - 1,i-b) =1 by auto
            from }\langlei<i-b\rangle\langlei-b\leqi-e\rangle\langlei-e<k'\rangle have i\leqi-b-1i-b-1\leqk' by
auto
        from <i< i-b\rangle\langlei-b\leqi-e\ranglej have **:i\leqi-b - 1 i-b - 1<j Suc (i-b -
1) = i-b by auto
            from ij have }\wedgel.l\geqi\Longrightarrowl<j\LongrightarrowA$$(l,Sucl)=1 by aut
            from this[OF **(1-2)]**(3)* show False by auto
                next
                    assume i> i-b
                    with ij j have A $$ (i-1,i)\not=1 and
                *:i-1\geqi-b i-1\leqi-e i-1<i-e Suc (i-1)=i by auto
                    with idb(3)[OF*(1,3)] show False by auto
                qed
        }
        hence i: i= i-b by arith
        show ?thesis unfolding ij by simp
        qed
    }
    hence ?s (Suc i-e)\subseteq insert (i-b,i-e) (?s i-b) by blast
    ultimately
    show insert (i-b,i-e) (?s i-b) = ?s (Suc i-e) by blast
    qed
qed
```

private lemma identify-blocks-iff: assumes $k<n$
shows set (identify-blocks $A k$ ) $=$
$\{(i, j) \mid i j . i \leq j \wedge j<k \wedge(\forall l . i \leq l \longrightarrow l<j \longrightarrow A \$ \$(l, S u c l)=1)$
$\wedge($ Suc $j \neq k \longrightarrow A \$ \$(j$, Suc $j) \neq 1) \wedge(i>0 \longrightarrow A \$ \$(i-1, i) \neq 1)\}$
unfolding identify-blocks-def using identify-blocks-main-iff [OF le-refl $-\langle k<n\rangle$ ]
by auto
private lemma identify-blocksD: assumes $k<n$ and $(i, j) \in$ set (identify-blocks A k)
shows $i \leq j j<k$
$\wedge l . i \leq l \Longrightarrow l<j \Longrightarrow A \$ \$(l$, Suc $l)=1$
Suc $j \neq k \Longrightarrow A \$ \$(j$, Suc $j) \neq 1$
$i>0 \Longrightarrow A \$ \$(i-1, i-1) \neq A \$ \$(k, k) \vee A \$ \$(i-1, i) \neq 1$
using assms unfolding identify-blocks-iff [OF assms(1)] by auto
private lemma identify-blocksI: assumes inv: $k<n$

$$
i \leq j j<k \wedge l . i \leq l \Longrightarrow l<j \Longrightarrow A \$ \$(l, \text { Suc } l)=1
$$

```
Suc \(j \neq k \Longrightarrow A \$ \$(j\), Suc \(j) \neq 1 i>0 \Longrightarrow A \$ \$(i-1, i) \neq 1\)
```

shows $(i, j) \in \operatorname{set}$ (identify-blocks A $k$ )
unfolding identify-blocks-iff [OF inv(1)] using inv by blast
private lemma identify-blocks-rev: assumes $A \$ \$(i$, Suc $i)=0 \wedge$ Suc $i<k \vee$
Suc $i=k$
and inv: $k<n$
shows (identify-block A $i, i) \in \operatorname{set}$ (identify-blocks $A k$ )
proof -
obtain $j$ where $i d$ : identify-block $A i=j$ by force
note $i d b=$ identify-block[OF this]
show ?thesis unfolding id
by (rule identify-blocksI[OF inv], insert idb assms, auto)
qed

### 18.8 Proving the Invariants

private lemma add-col-sub-row-diag: assumes $A: A \in$ carrier-mat $n n$ and ut: inv-all uppert $A$
and $i j k: i<j j<n k<n$
shows add-col-sub-row a ij A $\$ \$(k, k)=A \$ \$(k, k)$
proof -
from inv-all-uppert $D[$ OF ut]
show ?thesis
by (subst add-col-sub-index-row, insert A ijk, auto)
qed
private lemma add-col-sub-row-diff-ev-part-old: assumes $A: A \in$ carrier-mat $n$ n
and $i j: i \leq j i \neq 0 i<n j<n i^{\prime}<n j^{\prime}<n$
and choice: $j^{\prime}<j \vee j^{\prime}=j \wedge i^{\prime} \geq i$
and old: inv-part diff-ev $A$ i $j$
and ut: inv-all uppert $A$
shows diff-ev (add-col-sub-row a $(i-1) j A) i^{\prime} j^{\prime}$
unfolding diff-ev-def
proof (intro impI)
assume $i j^{\prime}: i^{\prime}<j^{\prime}$
let $? A=$ add-col-sub-row $a(i-1) j A$
assume neq: ? $A \$ \$\left(i^{\prime}, i^{\prime}\right) \neq ? A \$ \$\left(j^{\prime}, j^{\prime}\right)$
from $A$ have dim: dim-row $A=n \operatorname{dim}-c o l A=n$ by auto
note $u t d=$ inv-all-uppertD $[$ OF ut]
let ? $i=i-1$
have ? $i<j$ using $\langle i \leq j\rangle\langle i \neq 0\rangle\langle i<n\rangle$ by auto
from utd[OF this $\langle j<n\rangle]$ have $A j i: A \$ \$(j, ? i)=0$ by simp
from add-col-sub-row-diag[OF A ut $\langle ? i<j\rangle\langle j<n\rangle]$
have diag: $\wedge k . k<n \Longrightarrow ? A \$ \$(k, k)=A \$ \$(k, k)$.
from neq[unfolded $\left.\operatorname{diag}\left[O F\left\langle i^{\prime}<n\right\rangle\right] \operatorname{diag}\left[O F\left\langle j^{\prime}<n\right\rangle\right]\right]$
have neq: $A \$ \$\left(i^{\prime}, i^{\prime}\right) \neq A \$ \$\left(j^{\prime}, j^{\prime}\right)$ by auto
\{

```
    from inv-partD(3)[OF old < 'i}<<n\rangle\langle\mp@subsup{j}{}{\prime}<n\rangle\mathrm{ choice]
    have diff-ev A i' j' by auto
    with neq ij' have A $$ ( }\mp@subsup{i}{}{\prime},\mp@subsup{j}{}{\prime})=0\mathrm{ unfolding diff-ev-def by auto
    } note zero = this
    {
    assume i' = ?i j' = j
    with ij' ij(1) choice have i'}\mp@subsup{i}{}{\prime}>\mathrm{ ?i by auto
    from utd[OF this] ij
    have A$$(i',?i)=0 by auto
    } note 1 = this
    {
    assume j' 
    with ij' ij(1) choice have j> j' by auto
    from utd[OF this] ij
    have }A$$(j,\mp@subsup{j}{}{\prime})=0\mathrm{ by auto
    } note 2 = this
    from ij' ij choice have ( }\mp@subsup{i}{}{\prime}=
    note id =add-col-sub-index-row[of i' A j' j a ?i, unfolded dim this if-False,
```



```
    show ?A $$ (i',}\mp@subsup{j}{}{\prime})=0\mathrm{ unfolding id zero using 12 by auto
qed
private lemma add-col-sub-row-uppert: assumes A \in carrier-mat n n
    and i<j
    and}j<
    and inv: inv-all uppert (A :: 'a mat)
    shows inv-all uppert (add-col-sub-row a i j A)
    unfolding inv-all-def uppert-def
proof (intro allI impI)
    fix }\mp@subsup{i}{}{\prime}\mp@subsup{j}{}{\prime
    assume *: i'<n j'<n j'< i'
    note inv = inv-allD[OF inv, unfolded uppert-def]
    show add-col-sub-row a i j A $$ (i', j')=0
        by (subst add-col-sub-index-row, insert assms * inv, auto)
qed
private lemma step-1-main-inv: i\leqj
    A E carrier-mat n n
    \Longrightarrow \text { inv-all uppert A}
    \Longrightarrow ~ i n v - p a r t ~ d i f f - e v ~ A ~ i ~ j ~
    \Longrightarrow ~ i n v - a l l ~ u p p e r t ~ ( s t e p - 1 - m a i n ~ n ~ i ~ j ~ A ) ~ \wedge ~ i n v - a l l ~ d i f f - e v ~ ( s t e p - 1 - m a i n ~ n ~ i ~ j ~ A ) ~
proof (induct i j A taking: n rule: step-1-main.induct)
    case (1 i j A)
    let ? i = i-1
    note [simp]= step-1-main.simps[of n i j A]
    from 1(3-) have ij:i\leqj and A:A\in carrier-mat n n and inv: inv-all uppert
A
    inv-part diff-ev A i j by auto
    show ?case
```

```
proof (cases \(j \geq n\) )
    case True
    thus ? thesis using inv by (simp add: inv-all-eq-inv-part, auto simp: inv-part-def)
next
    case False
    hence \(j n: j<n\) by simp
    note \(I H=1(1-2)[\) OF False \(]\)
    show ?thesis
    proof (cases \(i=0\) )
        case True
        from inv[unfolded True inv-part-0-Suc[OF jn]]
        have inv2: inv-part diff-ev \(A n(j+1)\) by simp
        have inv-part diff-ev \(A(j+1)(j+1)\)
        proof (intro inv-partI)
            fix \(i^{\prime} j^{\prime}\)
            assume \(i j: i^{\prime}<n j^{\prime}<n\) and choice: \(j^{\prime}<j+1 \vee j^{\prime}=j+1 \wedge j+1 \leq i^{\prime}\)
            from inv-partD[OF inv2 ij] choice
            show diff-ev \(A i^{\prime} j^{\prime}\) using \(j n\) unfolding diff-ev-def by auto
        qed
        from \(\operatorname{IH}(1)[O F \operatorname{True}-A \operatorname{inv(1)this]}\)
        show ?thesis using \(j n\) by (simp, simp add: True)
    next
        case False
        let ?evi \(=A \$ \$(? i, ? i)\)
        let ? evj \(=A \$ \$(j, j)\)
        let ? choice \(=\) ? evi \(\neq ?\) ? evj \(\wedge A \$ \$(? i, j) \neq 0\)
        let ? \(A=\) add-col-sub-row \((A \$ \$(? i, j) /(? e v j-? e v i)) ? i j A\)
        let ? \(B=\) if ?choice then ?A else \(A\)
        obtain \(B\) where \(B: B=? B\) by auto
        have \(B n: B \in\) carrier-mat \(n n\) unfolding \(B\) using \(A\) by simp
        from False \(i j\) jn have \(*: ~ ? ~ i<j j<n ? i<n\) by auto
    have inv1: inv-all uppert \(B\) unfolding \(B\) using inv add-col-sub-row-uppert \([O F\)
\(A *(1-2) \operatorname{inv}(1)]\)
        by auto
        note inv2 \(=\) inv-partD[OF inv(2)]
        have inv2: inv-part diff-ev \(B\) ? \(i j\)
        proof (cases ?choice)
            case False
            hence \(B: B=A\) unfolding \(B\) by auto
            show ?thesis unfolding \(B\)
            proof (rule inv-partI)
                fix \(i^{\prime} j^{\prime}\)
            assume \(i j: i^{\prime}<n j^{\prime}<n\) and \(j^{\prime}<j \vee j^{\prime}=j \wedge ? i \leq i^{\prime}\)
            hence choice: \(\left(j^{\prime}<j \vee j^{\prime}=j \wedge i \leq i^{\prime}\right) \vee j^{\prime}=j \wedge i^{\prime}=\) ? \(i\) by auto
            note inv2 \(=\) inv2 \([O F i j]\)
            from choice
            show diff-ev \(A i^{\prime} j^{\prime}\)
            proof
                assume \(j^{\prime}<j \vee j^{\prime}=j \wedge i \leq i^{\prime}\)
```

```
                    from inv2(3)[OF this] show ?thesis.
            next
                    assume \(j^{\prime}=j \wedge i^{\prime}=? i\)
                    thus ?thesis using False unfolding diff-ev-def by auto
            qed
        qed
    next
        case True
        hence \(B: B=\) ? \(A\) unfolding \(B\) by auto
        from * True have \(i<n\) by auto
        note old \(=\) add-col-sub-row-diff-ev-part-old \([O F A\langle i \leq j\rangle\langle i \neq 0\rangle\langle i<n\rangle\langle j\)
\(<n\)
        - - \(\operatorname{inv(2)} \operatorname{inv(1)]}\)
    show ?thesis unfolding \(B\)
    proof (rule inv-partI)
        fix \(i^{\prime} j^{\prime}\)
        assume \(i j: i^{\prime}<n j^{\prime}<n\) and \(j^{\prime}<j \vee j^{\prime}=j \wedge ? ~ i \leq i^{\prime}\)
        hence choice: \(\left(j^{\prime}<j \vee j^{\prime}=j \wedge i \leq i^{\prime}\right) \vee j^{\prime}=j \wedge i^{\prime}=\) ? \(i\) by auto
        note inv2 \(=\) inv2 \([O F i j]\)
        from choice
        show diff-ev ? A \(i^{\prime} j^{\prime}\)
        proof
            assume \(j^{\prime}<j \vee j^{\prime}=j \wedge i \leq i^{\prime}\)
            from old \([O F\) ij this] show ?thesis.
        next
            assume \(j^{\prime}=j \wedge i^{\prime}=? i\)
            hence \(i j^{\prime}: j^{\prime}=j i^{\prime}=\) ? \(i\) by auto
            note \(\operatorname{diag}=a d d\)-col-sub-row- \(\operatorname{diag}[O F A \operatorname{inv}(1)\langle ? i<j\rangle\langle j<n\rangle]\)
            show ?thesis unfolding \(i j^{\prime} \operatorname{diff}-\) ev-def \(\left.\operatorname{diag}[O F<j<n\rangle\right] \operatorname{diag}[O F<? i<\)
\(n\rangle\) ]
            proof (intro impI)
                from True have neq: ?evi \(\neq\) ? evj by simp
                note \(u t=\) inv-all-uppert \(D[O F \operatorname{inv}(1)]\)
                obtain \(i^{\prime}\) where \(i^{\prime}: i^{\prime}=i-\) Suc 0 by auto
                obtain diff where diff: diff \(=\) ? evj \(-A \$ \$\left(i^{\prime}, i^{\prime}\right)\) by auto
                from neq have \([\) simp \(]\) : diff \(\neq 0\) unfolding diff \(i^{\prime}\) by auto
                from \(u t[O F\langle ? i<j\rangle\langle j<n\rangle]\) have \([\operatorname{simp}]: A \$ \$\left(j, i^{\prime}\right)=0\) unfolding
diff \(i^{\prime}\) by simp
                have ? \(A \$ \$(? i, j)=\)
                    \(A \$ \$\left(i^{\prime}, j\right)+\left(A \$ \$\left(i^{\prime}, j\right) * A \$ \$\left(i^{\prime}, i^{\prime}\right)-\right.\)
                    \(\left.A \$ \$\left(i^{\prime}, j\right) * A \$ \$(j, j)\right) /\) diff
                            by (subst add-col-sub-index-row, insert \(A *\), auto simp: diff [symmetric]
\(i^{\prime}[\) symmetric \(]\) field-simps)
            also have \(A \$ \$\left(i^{\prime}, j\right) * A \$ \$\left(i^{\prime}, i^{\prime}\right)-A \$ \$\left(i^{\prime}, j\right) * A \$ \$(j, j)\)
                    \(=-A \$ \$\left(i^{\prime}, j\right) *\) diff \(\mathbf{b y}\left(\right.\) simp add: diff \(i^{\prime}\) field-simps \()\)
            also have \(\ldots /\) diff \(=-A \$ \$\left(i^{\prime}, j\right)\) by simp
            finally show ?A \(\$ \$(? i, j)=0\) by simp
        qed
        qed
```

```
            qed
            qed
            from ij have i-1\leqj by simp
            note IH=IH(2)[OF False refl refl refl refl B this Bn inv1 inv2]
            from False jn have id: step-1-main n i j A = step-1-main n (i-1) j B
            unfolding B by (simp add: Let-def)
            show ?thesis unfolding id by (rule IH)
    qed
    qed
qed
private lemma step-2-main-inv: A \in carrier-mat n n
    \Longrightarrow \text { inv-all uppert A}
    \Longrightarrow \text { inv-all diff-ev A}
    \Longrightarrow \text { ev-blocks-part j A}
    \Longrightarrow ~ i n v - a l l ~ u p p e r t ~ ( s t e p - 2 - m a i n ~ n ~ j ~ A ) ~ \wedge ~ i n v - a l l ~ d i f f - e v ~ ( s t e p - 2 - m a i n ~ n ~ j ~ A ) ~
    ^ ev-blocks(step-2-main n j A)
proof (induct j A taking: n rule: step-2-main.induct)
    case (1 j A)
    note [simp] = step-2-main.simps[of n j A]
    from 1(2-) have A:A\in carrier-mat n n
        and inv: inv-all uppert A inv-all diff-ev A ev-blocks-part j A by auto
    show ?case
    proof (cases j\geqn)
        case True
            with inv(3) have ev-blocks A unfolding ev-blocks-def ev-blocks-part-def by
auto
    thus ?thesis using True inv(1-2) by auto
    next
        case False
    hence jn: j<n by simp
    note intro = ev-blocks-partI
    note dest = ev-blocks-partD
    note IH = 1(1)[OF False]
    let ?look= lookup-ev (A $$ (j,j)) j A
    let ?B= case ?look of
                None }=>\mathrm{ A
            | Some i => swap-cols-rows-block (Suc i) j A
    obtain B where B: B=?B by auto
    have id: step-2-main n j A = step-2-main n (Suc j) B unfolding B using
False by simp
    have Bn: B \in carrier-mat n n unfolding B using A by (auto split:op-
tion.splits)
    have inv-all uppert B ^ inv-all diff-ev B ^ ev-blocks-part (Suc j) B
    proof (cases ?look)
        case None
        have ev-blocks-part (Suc j) A
        proof (intro intro)
            fix }\mp@subsup{i}{}{\prime}\mp@subsup{j}{}{\prime}\mp@subsup{k}{}{\prime
```

```
        assume \(*: i^{\prime}<j^{\prime} j^{\prime}<k^{\prime} k^{\prime}<S u c j A \$ \$\left(k^{\prime}, k^{\prime}\right)=A \$ \$\left(i^{\prime}, i^{\prime}\right)\)
        show \(A \$ \$\left(j^{\prime}, j^{\prime}\right)=A \$ \$\left(i^{\prime}, i^{\prime}\right)\)
        proof (cases \(j=k^{\prime}\) )
            case False
            with \(*\) have \(k^{\prime}<j\) by auto
            from dest[OF \(\operatorname{inv}(3) *(1-2)\) this \(*(4)]\)
            show ?thesis.
        next
            case True
            with lookup-ev-None[OF None, of \(i\) i] * have False by simp
            thus ?thesis ..
        qed
    qed
    with None show ?thesis unfolding \(B\) using inv by auto
    next
    case (Some \(i\) )
    from lookup-ev-Some[OF Some]
    have \(i j: i<j\) and \(i d: A \$ \$(i, i)=A \$ \$(j, j)\)
        and neq: \(\wedge k . i<k \Longrightarrow k<j \Longrightarrow A \$ \$(k, k) \neq A \$ \$(j, j)\) by auto
    let ? \(A=\) swap-cols-rows-block (Suc i) \(j A\)
    let ?perm \(=\lambda i^{\prime}\). if \(i^{\prime}=\) Suc \(i\) then \(j\) else if Suc \(i<i^{\prime} \wedge i^{\prime} \leq j\) then \(i^{\prime}-1\)
else \(i^{\prime}\)
    from Some have \(B: B=\) ? A unfolding \(B\) by simp
    have Aind: \(\bigwedge i^{\prime} j^{\prime} . i^{\prime}<n \Longrightarrow j^{\prime}<n \Longrightarrow ? A \$ \$\left(i^{\prime}, j^{\prime}\right)=A \$ \$\left(? p e r m i^{\prime}\right.\),
?perm \(j^{\prime}\) )
            by (subst swap-cols-rows-block-index, insert False A ij, auto)
    have inv-ev: ev-blocks-part (Suc j) ?A
    proof (intro intro)
        fix \(i^{\prime} j^{\prime} k\)
        assume \(*: i^{\prime}<j^{\prime} j^{\prime}<k k<S u c j\) and \(k i: ? A \$ \$(k, k)=? A \$ \$\left(i^{\prime}, i^{\prime}\right)\)
            from \(* j n\) have \(j^{\prime}<n i^{\prime}<n k<n\) by auto
    note \(i d^{\prime}=\operatorname{Aind}\left[O F\left\langle j^{\prime}<n\right\rangle\left\langle j^{\prime}<n\right\rangle\right] \operatorname{Aind}\left[O F\left\langle i^{\prime}<n\right\rangle\left\langle i^{\prime}<n\right\rangle\right] \operatorname{Aind}[O F\)
\(\langle k<n\rangle\langle k<n\rangle\) ]
    note \(\operatorname{inv}-e v=\operatorname{dest}[O F \operatorname{inv}(3)]\)
    show ? A \(\$ \$\left(j^{\prime}, j^{\prime}\right)=\) ? \(A \$ \$\left(i^{\prime}, i^{\prime}\right)\)
    proof (cases \(i^{\prime}<\) Suc \(i\) )
    case True note \(i^{\prime}=\) this
    hence \(p i\) : ?perm \(i^{\prime}=i^{\prime}\) by simp
    show ?thesis
    proof (cases \(j^{\prime}<S u c i\) )
        case True note \(j^{\prime}=\) this
        hence \(p j\) : ?perm \(j^{\prime}=j^{\prime}\) by simp
        show ?thesis
        proof (cases \(k<\) Suc \(i\) )
            case True note \(k=\) this
            hence \(p k\) : ?perm \(k=k\) by simp
            from True \(i j\) have \(k<j\) by simp
            from inv-ev[OF *(1-2) this] ki
            show ?thesis unfolding \(i d^{\prime}\) pi pj pk by auto
```

```
    next
        case False note kf1 = this
        show ?thesis
        proof (cases k=Suc i)
        case True note k= this
        hence pk: ?perm k=j by simp
        from ki id have }i\mp@subsup{i}{}{\prime}:A$$(i,i)=A$$(\mp@subsup{i}{}{\prime},\mp@subsup{i}{}{\prime})\mathrm{ unfolding id ' pi pj
pk by simp
            have ji:A$$ (j',j')=A$$(\mp@subsup{i}{}{\prime},\mp@subsup{i}{}{\prime})
            proof (cases j' = i)
                    case True
                    with ii' show ?thesis by simp
            next
                    case False
                    with «j}<\mp@code{Suc i> have j'<i by auto
                    from ki id inv-ev[OF <i'< j'> this ij] show ?thesis
                    unfolding id' pi pj pk by simp
            qed
            thus ?thesis unfolding id' pi pj pk.
        next
            case False note kf2 = this
            with kf1 have k: k> Suc i by auto
            hence pk:?perm k=k-1 and kj: k-1<j
                using * <k< Suc j> by auto
            from k j' have j' <k-1 by auto
            from inv-ev[OF *(1) this kj] ki
            show ?thesis unfolding id' pi pj pk by simp
        qed
        qed
    next
        case False note j'f1 = this
        show ?thesis
        proof (cases j' = Suc i)
            case True note j' = this
            hence pj: ?perm j'=j by simp
            from j}\mp@subsup{j}{}{*}\mathrm{ have k: k>Suc i by auto
            hence pk:?perm k=k-1 and kj:k-1<j
                using * <k< <Suc j> by auto
            from ki[unfolded id' pi pj pk] have eq: A $$ (k-1,k-1)=A$$
(i',}\mp@subsup{i}{}{\prime})
            from * i'k have le: }\mp@subsup{i}{}{\prime}\leqi\mathrm{ and lt:i<k-1k-1<j by auto
            from inv-ev[OF - lt eq] le have A $$ (i,i)=A$$(i', i')
                by (cases i= i', auto)
    with id show ?thesis unfolding id' pi pj pk by simp
    next
        case False note j'f2 = this
        with j'f1 have }\mp@subsup{j}{}{\prime}>Suci\mathrm{ by auto
        hence pj: ?perm j' = j' - 1 and pk: ?perm k=k-1
            and kj: i'< j' - 1 j' - 1<k-1k-1<j
```

```
                    using \(* i^{\prime}\langle k<S u c j\rangle\) by auto
            from inv-ev[OF \(k j] k i\)
            show ?thesis unfolding \(i d^{\prime}\) pi pj pk by simp
        qed
        qed
    next
        case False note \(i^{\prime} f 1=\) this
        show ?thesis
        proof (cases \(i^{\prime}=\) Suc \(\left.i\right)\)
        case True note \(i^{\prime}=\) this
        with * have \(g t: i<k-1 k-1<j\)
            and perm: ?perm \(i^{\prime}=j\) ?perm \(k=k-1\) by auto
        from ki[unfolded id' perm] neq[OF gt] have False by auto
        thus ?thesis ..
    next
        case False note \(i^{\prime} f 2=\) this
        with \(i^{\prime} f 1\) have \(i^{\prime}>S u c i\) by auto
        with \(*\) have \(g t: i^{\prime}-1<j^{\prime}-1 j^{\prime}-1<k-1 k-1<j\)
                and perm: ?perm \(i^{\prime}=i^{\prime}-1\) ?perm \(j^{\prime}=j^{\prime}-1\) ?perm \(k=k-1\) by
auto
            show ?thesis using inv-ev[OF gt] ki
                unfolding \(i d^{\prime}\) perm by simp
    qed
    qed
qed
let ?both \(=\lambda A i j\). uppert \(A i j \wedge \operatorname{diff-ev~} A i j\)
have inv-all ?both ?A
proof (intro inv-allI)
    fix \(i i j j\)
    assume \(i i: i i<n\) and \(j j: j j<n\)
    note \(i d=\) Aind \([\) OF ii ii] Aind \([O F j j j j]\) Aind \([O F ~ i i ~ j j]\)
    note \(u t=\) inv-all-uppert \(D[O F \operatorname{inv}(1)]\)
    note diff \(=\) inv-all-diff-evD[OF inv(2)]
    have upper: uppert ? A ii jj unfolding uppert-def
    proof
    assume \(j i\) : \(j j<i i\)
    show ? \(A \$ \$(i i, j j)=0\)
    proof (cases \(i i<\) Suc \(i\) )
        case True note \(i=\) this
        with \(j i\) have perm: ?perm \(i i=i i\) ?perm \(j j=j j\) by auto
        show ?thesis unfolding id perm using ut [OF ji ii].
    next
        case False note if1 = this
        show ?thesis
        proof (cases \(i i=\) Suc \(i\) )
            case True note \(i=\) this
        with \(j i\) ij have perm: ?perm \(i i=j\) ? perm \(j j=j j\) and \(j j\) : \(j j<j\) by auto
                show ?thesis unfolding id perm
                by (rule ut [OF jj jn])
```

```
    next
        case False
        with if1 have if1: ii > Suc i by auto
        show ?thesis
        proof (cases ii < j)
            case True note i= this
            with if1 have pi:?perm ii= ii - 1 by auto
            show ?thesis
            proof (cases jj = Suc i)
            case True note j= this
            hence pj: ?perm jj = j by simp
            from i ji if1 ii j have ij: ii - 1<j and ii: i< ii-1 by auto
            show ?thesis unfolding id pi pj
                    by (rule diff[OF ij jn neq[OF ii ij]])
            next
                case False
                with i ji if1 ii have ?perm jj< ii - 1 ii - 1<n by auto
                from ut[OF this]
                show ?thesis unfolding id pi .
            qed
        next
            case False
            hence i: ii > j by auto
            with if1 have pi: ?perm ii = ii by simp
            from iji if1 ii have ?perm jj < ii by auto
            from ut[OF this ii]
            show ?thesis unfolding id pi.
        qed
        qed
        qed
qed
have diff: diff-ev ?A ii jj unfolding diff-ev-def
proof (intro impI)
    assume ij': ii<jj and neq: ?A $$ (ii,ii) \not=?A $$ (jj,jj)
    show ?A $$ (ii,jj) = 0
    proof (cases jj < Suc i)
        case True note j= this
        with ij' have perm: ?perm ii = ii ?perm jj = jj by auto
    show ?thesis using neq unfolding id perm using diff [OF ij' jj] by simp
next
    case False note jf1 = this
    show ?thesis
    proof (cases jj = Suc i)
        case True note j= this
            with ij' ij have perm: ?perm jj = j ?perm ii = ii and ii: ii<j by
    show ?thesis using neq unfolding id perm
            by (intro diff[OF ii jn])
    next
```

auto

```
                    case False
                    with \(j f 1\) have \(j f 1\) : \(j j>S u c i\) by auto
                        show ?thesis
                proof (cases \(j j \leq j\) )
                    case True note \(j=\) this
                        with \(j f 1\) have \(p j\) : ?perm \(j j=j j-1\) by auto
                    show ?thesis
                    proof (cases \(i i=\) Suc \(i\) )
                    case True note \(i=\) this
                    hence pi: ?perm \(i i=j\) by simp
                    from \(i{ }^{i}{ }^{\prime}\) jf1 \(j j j\) have \(i j: j j-1<j\) by auto
                    show ?thesis unfolding id pi pj
                    by (rule \(u t[O F ~ i j ~ j n])\)
                    next
                    case False
                    with \(j i j^{\prime} j f 1 j j\) have ?perm \(i i<j j-1 j j-1<n\) by auto
                    from diff[OF this] neq
                    show ?thesis unfolding id pj .
                    qed
                    next
                        case False
                            hence \(j: j j>j\) by auto
                    with \(j f 1\) have \(p j\) : ?perm \(j j=j j\) by simp
                    from \(j i j^{\prime} j f 1 j j\) have ?perm \(i i<j j\) by auto
                    from diff[OF this jj] neq
                    show ?thesis unfolding id \(p j\).
                    qed
                qed
            qed
            qed
            from upper diff
            show ?both ? A ii jj ..
            qed
            hence inv-all diff-ev?A inv-all uppert ?A
            unfolding inv-all-def by blast+
            with inv-ev show ?thesis unfolding \(B\) by auto
        qed
        with \(I H[O F\) refl \(B\) Bn]
        show ?thesis unfolding id by auto
    qed
qed
```

private lemma add-col-sub-row-same-upto: assumes $i<j j<n A \in$ carrier-mat $n$ n inv-upto uppert $A j$
shows same-upto $j A($ add-col-sub-row vij $A)$
by (intro same-uptoI, subst add-col-sub-index-row, insert assms, auto simp: up-pert-def inv-upto-def)
private lemma add-col-sub-row-inv-from-uppert: assumes $*$ : inv-from uppert $A j$
and $* *: A \in$ carrier-mat $n n i<n i<j j<n$
shows inv-from uppert (add-col-sub-row vijA) $j$
proof -
note $*=* * *$
let $? A=a d d$-col-sub-row $v i j A$
show inv-from uppert ? A $j$ unfolding inv-from-def
proof (intro allI impI)
fix $i^{\prime} j^{\prime}$
assume $* *: i^{\prime}<n j^{\prime}<n j<j^{\prime}$
from $* * *$ have $i^{\prime}<\operatorname{dim}$-row $A i^{\prime}<\operatorname{dim}-c o l A j^{\prime}<d i m-r o w A j^{\prime}<\operatorname{dim}$-col
$A j<$ dim-row $A$ by auto
note $i d 2=$ add-col-sub-index-row[OF this]
show uppert ? A $i^{\prime} j^{\prime}$ unfolding uppert-def
proof (intro conjI impI)
assume $j^{\prime}<i^{\prime}$
with inv-fromD $\left[O F\left\langle i n v-f r o m ~ u p p e r t ~ A ~ j 〉\right.\right.$, unfolded uppert-def, of $\left.i^{\prime} j^{\prime}\right] * * *$ show ?A $\$ \$\left(i^{\prime}, j^{\prime}\right)=0$ unfolding $i d 2$ using $* * *\left\langle j^{\prime}<i^{\prime}\right\rangle$ by simp qed
qed
qed
private lemma step-3-a-inv: $A \in$ carrier-mat $n n$
$\Longrightarrow i<j \Longrightarrow j<n$
$\Longrightarrow$ inv-upto $j b A j$
$\Longrightarrow$ inv-from uppert $A j$
$\Longrightarrow$ inv-from-bot ( $\lambda$ A i. one-zero $A i j$ ) Ai
$\Longrightarrow$ ev-block $n A$
$\Longrightarrow$ inv-from uppert (step-3-a $i j A) j$
$\wedge$ inv-upto $j b$ (step-3-a $i j A) j$
$\wedge$ inv-at one-zero (step-3-a ij A) $j \wedge$ same-diag A (step-3-a ij A)
proof (induct ij A rule: step-3-a.induct)
case (1 j A)
thus ?case by (simp add: inv-from-bot-def inv-at-def)
next
case (2ijA)
from 2(2-) have $A: A \in$ carrier-mat $n n$ and $i j:$ Suc $i<j i<j$ and $j: j<n$
by auto
let ?cond $=A \$ \$(i, i+1)=1 \wedge A \$ \$(i, j) \neq 0$
let ? $B=$ add-col-sub-row $(-A \$ \$(i, j))(S u c$ i) $j A$
obtain $B$ where $B: B=($ if ?cond then ? $B$ else $A)$ by auto
from $A$ have $B n: B \in$ carrier-mat $n n$ unfolding $B$ by simp
note $I H=2(1)[O F$ refl $B$ Bn ij(2) $j]$
have id: step-3-a (Suc i) j $A=$ step-3-a i j $B$ unfolding $B$ by (simp add: Let-def)
from $i j j$ have $*$ : Suc $i<n j<n$ Suc $i \neq j$ by auto
from 2(2-) have inv: inv-upto jb $A j$ inv-from uppert $A j$ ev-block $n A$
inv-from-bot ( $\lambda A$ i. one-zero $A i j$ ) $A(S u c i)$ by auto
note $e v b A=e v$-blockD $[$ OF inv(3)]

```
show ?case
proof (cases ?cond)
    case False
    hence }B:B=A\mathrm{ unfolding }B\mathrm{ by auto
    have inv2: inv-from-bot ( }\lambdaA\mathrm{ i. one-zero A i j) A i
        by (rule inv-from-bot-step[OF - inv(4)],
        insert False ij evbA[of i j] *, auto simp: one-zero-def)
    show ?thesis unfolding id B
        by (rule IH[unfolded B], insert inv inv2, auto)
next
    case True
    hence }B:B=?B\mathrm{ unfolding }B\mathrm{ by auto
    let ?C = step-3-a i j B
    from inv-uptoD[OF inv(1) j*(1) ij(1), unfolded jb-def] ij
    have Aji: A $$ (j,Suc i)=0 by auto
    have diag: same-diag A B unfolding same-diag-def
        by (intro allI impI, insert ij j A Aji B, auto)
    have upto: same-upto j A B unfolding B
    by (rule add-col-sub-row-same-upto[OF \langleSuc i<j\rangle<j<n\rangleA inv-upto-mono[OF
jb-imp-uppert inv(1)]])
    from add-col-sub-row-inv-from-uppert[OF inv(2) A <Suc i<n`\langleSuc i<j\rangle\langlej
< n>]
    have from-j: inv-from uppert B j unfolding B by blast
    have ev:A $$(Suc i,Suc i)=A $$ (j,j) using evbA[of Suc i j] ij j by auto
    have evb-B: ev-block n B
        by (rule same-diag-ev-block[OF diag inv(3)])
    note evbB = ev-blockD[OF evb-B]
    {
        fix }
        assume k<n
        with A* have k: k< dim-row A k<dim-col A j<dim-row A j<dim-col
A j< dim-row A by auto
    note id = B add-col-sub-index-row[OF k]
    have B $$ (k,j) = (if k=i then 0 else A $$ (k,j)) unfolding id
        using inv-uptoD[OF inv(1), of k Suc i, unfolded jb-def]
        by (insert * Aji True ij 〈k<n`, auto simp: ev)
    } note id2 = this
    have inv-from-bot ( }\lambdaA\mathrm{ i. one-zero A i j) B i unfolding inv-from-bot-def
    proof (intro allI impI)
        fix }
        assume i\leqkk<n
        thus one-zero B kjusing inv(4)[unfolded inv-from-bot-def]
            upto[unfolded same-upto-def] evbB[OF <k< n><j< n>]
            unfolding one-zero-def id2[OF <k<n`] by auto
    qed
    from IH[OF same-upto-inv-upto-jb[OF upto inv(1)] from-j this evb-B]
        same-diag-trans[OF diag]
    show ?thesis unfolding id by blast
qed
```


## qed

private lemma identify-block-cong: assumes su: same-upto $k A B$ and $k n: k<$ $n$
shows $i<k \Longrightarrow$ identify-block $A i=$ identify-block $B i$
proof (induct $i$ )
case (Suc i)
hence $i<k$ by auto
note $I H=\operatorname{Suc}(1)[$ OF this]
let $? c=\lambda A . A \$ \$(i, S u c i)=1$
from same-uptoD[OF su, of $i$ Suc $i]$ kn Suc(2) have 1: $A \$ \$(i, S u c i)=B \$ \$$ ( $i$, Suc $i$ ) by auto
from 1 have $i d$ : ?c $A=$ ?c $B$ by $\operatorname{simp}$
show ?case
proof (cases ?c A)
case True
with True[unfolded $i d]$ IH show ?thesis by simp
next
case False
with False[unfolded id] show ?thesis by auto
qed
qed $\operatorname{simp}$
private lemma identify-blocks-main-cong:
$k<n \Longrightarrow$ same-upto $k A B \Longrightarrow$ identify-blocks-main $A k x=$ identify-blocks-main
$B k x s$
proof (induct $k$ arbitrary: xs rule: less-induct)
case (less $k$ list)
show ?case
proof (cases $k=0$ )
case False
then obtain $i$-e where $k: k=S u c i-e$ by (cases $k$, auto)
obtain $i$ - $b$ where idA: identify-block $A$ i-e $=i$-b by force
from identify-block-le' $[O F i d A]$ have $i b e: i-b \leq i-e$.
have idB: identify-block $B i$ - $e=i$ - $b$ unfolding $i d A[s y m m e t r i c]$
by (rule sym, rule identify-block-cong, insert $k$ less(2-3), auto)
let ? $I=$ identify-blocks-main
let ? res $A=$ ? I $A($ Suc $i$-e) list
let ? $r e c A=$ ? $I A$ i-b ((i-b, i-e) \# list)
let ? res $B=$ ? I $B$ (Suc i-e) list
let ? $r e c B=$ ? $I B i-b((i-b, i-e) \#$ list $)$
have res: ?res $A=$ ? rec $A$ ?res $B=$ ?recB using $i d A$ idB by auto
from $k$ ibe have $i b k: i-b<k$ by simp
with less(3) have same-upto i-b A B unfolding same-upto-def by auto
from less (1)[OF ibk-this] ibk $\langle k<n\rangle$ have ?rec $A=$ ? recB by auto
thus ?thesis unfolding $k$ res by simp
qed simp
qed
private lemma identify-blocks-cong:
$k<n \Longrightarrow$ same-diag $A B \Longrightarrow$ same-upto $k A B \Longrightarrow$ identify-blocks $A k=$ identify-blocks $B k$
unfolding identify-blocks-def
by (intro identify-blocks-main-cong, auto simp: same-diag-def)
private lemma inv-from-upto-at-all-ev-block:
assumes $j b$ : inv-upto $j b A k$ and ut: inv-from uppert $A k$ and at: inv-at p $A k$ and evb: ev-block $n A$
and $p: \bigwedge i . i<n \Longrightarrow p A i k \Longrightarrow$ uppert $A i k$
and $k: k<n$
shows inv-all uppert $A$
proof (rule inv-from-upto-at-all $[O F j b-u t ~ a t])$
from ev-block $D[O F$ evb $]$
show inv-from diff-ev $A k$ unfolding inv-from-def diff-ev-def by blast
fix $i$
assume $i<n p A i k$
with ev-blockD[OF evb $k$, of $i] p[$ OF this] $k$
show diff-ev $A$ i $k \wedge$ uppert $A i k$
unfolding diff-ev-def by auto
qed
For step 3c, during the inner loop, the invariants are NOT preserved. However, at the end of the inner loop, the invariants are again preserved. Therefore, for the inner loop we prove how the resulting matrix looks like in each iteration.
private lemma step-3-c-inner-result: assumes inv:

```
    inv-upto jb A \(k\)
    inv-from uppert \(A k\)
    inv-at one-zero \(A k\)
    ev-block \(n\) A
    and \(k: k<n\)
    and \(A: A \in\) carrier-mat \(n n\)
    and \(l b l:(l b, l) \in \operatorname{set}\) (identify-blocks \(A k)\)
    and ib-block: \((i\)-begin,i-end \() \in \operatorname{set}(i d e n t i f y\)-blocks \(A k)\)
    and \(i l: i\)-end \(\neq l\)
    and large: \(l-l b \geq i\)-end \(-i\)-begin
    and Alk: \(A \$ \$(l, k) \neq 0\)
    shows step-3-c-inner-loop ( \(A \$ \$(i\)-end, \(k\) ) / A \$\$ (l,k)) li-end (Suc i-end -
\(i\)-begin) \(A=\)
    mat \(n\) n
        \((\lambda(i, j)\). if \((i, j)=(i\)-end, \(k)\) then 0
            else if \(i\)-begin \(\leq i \wedge i \leq i\)-end \(\wedge k<j\) then \(A \$ \$(i, j)-A \$ \$\) (i-end,
\(k) / A \$ \$(l, k) * A \$ \$(l+i-i\)-end, \(j)\)
                else \(A \$ \$(i, j))(\) is \(? L=? R)\)
proof -
    let \(? A l k=A \$ \$(l, k)\)
    let ?Aik \(=A \$ \$(i\)-end,\(k)\)
    define quot where quot \(=\) ? Aik / ?Alk
```

let ?idiff $=i$-end $-i$-begin
let ? $m=\lambda$ iter diff $i j$. if $(i, j)=(i$-end, $k)$ then if diff $=($ Suc ?idiff $)$ then ?Aik else 0
else if $i \geq i$-begin + diff $\wedge i \leq i$-end $\wedge k<j$ then $A \$ \$(i, j)-q u o t * A \$ \$(l$ $+i-i$-end, $j$ )
else if $(i, j)=(i$-end - iter, Suc $l-$ iter $) \wedge$ iter $\notin\{0$, Suc ?idiff $\}$ then quot else $A \$ \$(i, j)$
let ? $m m=\lambda$ iter diff $i j$. if $(i, j)=(i$-end,$k)$ then 0
else if $i \geq i$-begin + diff $\wedge i \leq i$-end $\wedge k<j$ then $A \$ \$(i, j)-q u o t * A \$ \$(l$ $+i-i$-end, $j$ )
else if $(i, j)=(i$-end - Suc iter, $l-$ iter $) \wedge$ iter $\neq$ ?idiff then quot
else $A \$ \$(i, j)$
let ?mat $=\lambda$ iter diff. mat $n n(\lambda(i, j)$. ? $m$ iter diff $i j)$
from identify-blocks $[O F$ ib-block $]$ have $i b: i$-begin $\leq i$-end $i$-end $<k$ by auto
from identify-blocks[OF lbl] have $l b: l b \leq l l<k$ by auto
have mend: ?mat 0 (Suc ?idiff) $=A$
by (rule eq-matI, insert $A$ ib, auto)
\{
fix $l l$ ii diff iter
assume diff $\neq 0 \Longrightarrow i i+$ iter $=$ i-end diff $\neq 0 \Longrightarrow l l+$ iter $=l$ diff + iter
$=$ Suc ? idiff
hence step-3-c-inner-loop quot ll ii diff (?mat iter diff) $=$ ?R
proof (induct diff arbitrary: ii ll iter)
case 0
hence iter: iter $=$ Suc ?idiff by auto
have step-3-c-inner-loop quot ll ii 0 (?mat iter 0) $=$ ?mat (Suc ?idiff) 0
unfolding iter step-3-c-inner-loop.simps ..
also have $\ldots=$ ? $R$
by (rule eq-matI, insert ib, auto simp: quot-def)
finally show ?case .
next
case (Suc diff ii ll)
note prems $=\operatorname{Suc}(2-)$
let ? $B=$ ? mat iter (Suc diff)
have step-3-c-inner-loop quot ll ii (Suc diff) ?B
$=$ step-3-c-inner-loop quot $(l l-1)(i i-1)$ diff $(a d d-c o l-s u b-r o w ~ q u o t ~ i i ~ l l ~$
?B)
by $\operatorname{simp}$
also have add-col-sub-row quot ii ll ?B
$=$ ?mat (Suc iter) diff (is ?C = ? D)
proof (rule eq-matI, unfold dim-row-mat dim-col-mat)
fix $i j$
assume $i: i<n$ and $j: j<n$
have $l l$ : $l l<n$ using prems $l b k$ by auto
from prems ib $k$ have $i i: i i \geq i$-begin $i i<n i i<k i i \leq i$-end
and eqs: $i i+$ iter $=i$-end $l l+$ iter $=l$ Suc diff + iter $=$ Suc ? idiff by
auto
from eqs have diff: diff $<$ Suc ?idiff by auto
from eqs $l b\langle k<n\rangle$ have $l l<k l<n$ by auto
note index $=i b l b k i j$ ll il large $i i$ this
let ? $A i j=A \$ \$(i, j)$
have $D$ : ? $D \$(i, j)=$ ? mm iter diff $i j$ using diff $i j$ by (auto split: if-splits)
define $B$ where $B=$ ? $B$
have $B B$ : $\bigwedge i j . i<n \Longrightarrow j<n \Longrightarrow B \$ \$(i, j)=$ ? $m$ iter (Suc diff) $i j$
unfolding $B$-def by auto
have $B$ : $B \$ \$(i, j)=$ ? $m$ iter (Suc diff) $i j$ by (rule $B B[O F i j])$
have $C$ : ? $C \$ \$(i, j)=$
(if $i=i i \wedge j=l l$ then $B \$ \$(i, j)+q u o t * B \$ \$(i, i)-q u o t * q u o t * B$
$\$ \$(j, i)-q u o t * B \$ \$(j, j)$
else if $i=i i \wedge j \neq l l$ then $B \$ \$(i, j)-q u o t * B \$ \$(l l, j)$
else if $i \neq i i \wedge j=l l$ then $B \$ \$(i, j)+q u o t * B \$ \$(i, i i)$
else $B \$ \$(i, j))$ unfolding $B$-def
by (rule add-col-sub-index-row(1), insert i $j$ ll, auto)
from inv-from-upto-at-all-ev-block[OF $\operatorname{inv}(1-4)-\langle k<n\rangle]$
have invA: inv-all uppert $A$
unfolding one-zero-def uppert-def by auto
note $u t=$ inv-all-uppert $D[O F$ invA $]$
note $j b=$ inv-upto $D[O F \operatorname{inv}(1)$, unfolded $j b-d e f]$
note $o z=\operatorname{inv}-a t D[O F \operatorname{inv}(3)$, unfolded one-zero-def $]$
note $e v b=e v$-block $D[O F \operatorname{inv}(4)]$
note iblock $=$ identify-blocks $D[O F\langle k<n\rangle]$
note $i b e=i b l o c k[O F$ ib-block]
let ? $e v=\lambda i . A \$ \$(i, i)$

## \{

fix $i$ ib $i e$
assume $(i b, i e) \in \operatorname{set}(i d e n t i f y$-blocks $A k)$ and $i: i b \leq i i<i e$
note ibe $=$ iblock[OF this(1)]
from ibe (3)[OF i] have id: A $\$ \$(i, S u c i)=1$ by auto
from $i$ ibe $\langle k<n\rangle$ have $i<n$ Suc $i<k$ by auto
with oz[OF this(1)] id
have $A \$ \$(i, k)=0$ by auto
$\}$ note $A-i k=t h i s$

## \{

fix $i$
assume $i: i<n$ and $\neg(i \geq i$-begin $\wedge i \leq i$-end $)$
hence choice: $i>i$-end $\vee i<i$-begin by auto
note index $=$ index $i$
from index eqs choice have $i \neq i i$ by auto
\{
assume 0: $A \$ \$(i, i i) \neq 0$
from $0 u t[$ of $i i, O F-i]\langle i \neq i i\rangle$ have $i<i i$ by force
from choice index eqs this have $i<i$-begin by auto
with index have $i<k$ by auto
from $j b[O F i\langle i i<n\rangle\langle i i<k\rangle] 0\langle i \neq i i\rangle$
have $*$ : Suc $i=$ ii $A \$ \$(i, i i)=1$ ?ev $i=$ ?ev $i i$ by auto
with index $\langle i<i$-begin $\rangle$ have $i i=i$-begin by auto
with $\operatorname{evb[OF}\langle i<n\rangle\langle k<n\rangle]$ ibe(5) * have False by auto \} hence $A i i$ : $A \$(i, i i)=0$ by auto \{
fix $j$ assume $j: j<n$
have $B: B \$(i, j)=$ ? $m$ iter (Suc diff) $i j$ using $i j$ unfolding $B$-def by $\operatorname{simp}$
from choice have id: $((i, j)=(i$-end - iter, Suc $l-$ iter $) \wedge$ iter $\notin\{0$, Suc ?idiff \}) $=$ False
using ib index eqs by auto
have $B \$ \$(i, j)=A \$ \$(i, j)$ unfolding $B$ id using choice ib index by auto
\}
note Aii this
\}
hence $A$-outside-ii: $\wedge i . i<n \Longrightarrow \neg(i$-begin $\leq i \wedge i \leq i$-end $) \Longrightarrow A \$ \$$ $(i, i i)=0$
and B-outside: $\bigwedge i j . i<n \Longrightarrow j<n \Longrightarrow \neg(i$-begin $\leq i \wedge i \leq i$-end $) \Longrightarrow$ $B \$ \$(i, j)=A \$ \$(i, j)$ by auto
from diff eqs have iter: iter $\neq$ Suc ?idiff by auto \{
fix $i b i e j b j e$
assume $i:(i b, i e) \in \operatorname{set}(i d e n t i f y-b l o c k s A k)$ and
$j:(j b, j e) \in \operatorname{set}($ identify-blocks $A k)$ and $l t: i e<j e$
note $i=$ iblock $[$ OF $i]$
note $j=$ iblock[OF $j]$
from $i j$ lt have Suc ie $<k$ by auto
with $i$ have Aie: $A \$ \$$ (ie, Suc ie) $\neq 1$ by auto
have $i e<j b$
proof (rule ccontr)
assume $\neg i e<j b$
hence $i e \geq j b$ by auto
from $j(3)[$ OF this lt $]$ Aie show False by auto
qed
\} note block-bounds $=$ this
\{
assume $i$-end $<l$
from block-bounds[OF ib-block lbl this]
have $i$-end $<l b$.
\} note $i$-less-l $=$ this
\{
assume $l<i$-end
from block-bounds[OF lbl ib-block this]
have $l<i$-begin .
$\}$ note $l$-less- $i=$ this
\{
assume $i$-end - iter $=S u c l-i t e r$
with iter large eqs have $i$-end $=S u c l$ by auto
with l-less- $i$ have $l<i$-begin by auto
with index $\langle i$-end $=S u c l\rangle$ have $i$-begin $=i$-end by auto
\} note block $=$ this
have Alie: A $\$ \$(l, i$-end $)=0$
proof (cases $l<i$-end)
case True
\{
assume $n z: A \$ \$(l, i$-end $) \neq 0$
from l-less-i[OF True] index have $0<i$-begin $l<i$-begin $i$-end $<n$
$i$-end $<k$ by auto
from $j b[O F\langle l<n\rangle$ this $(3-4)]$ il $n z$
have $i$-end $=$ Suc l A $\$ \$(l$, Suc $l)=1$ by auto
with iblock[OF lbl] have $k=S u c l$ by auto
with $\langle i$-end $=S u c l\rangle\langle i$-end $<k\rangle$ have False by auto
\}
thus ?thesis by auto
next
case False
with $i l$ have $i$-end $<l$ by auto
from $u t[O F$ this $\langle l<n\rangle]$ show ?thesis .
qed
show ? $C$ \$ $(i, j)=? D \$ \$(i, j)$
proof (cases $i \geq i$-begin $\wedge i \leq i$-end $)$
case False
hence choice: $i>i$-end $\vee i<i$-begin by auto
from choice have $i d:((i, j)=(i$-end $-S u c$ iter, $l-$ iter $) \wedge$ iter $\neq$ ? idiff $)$ $=$ False
using $i b$ index eqs by auto
have $D$ : ? $D \$ \$(i, j)=$ ? Aij unfolding $D$ id using choice ib index by auto have $B$ : $B \$ \$(i, j)=$ ? Aij unfolding B-outside[OF ij False] ..
from index eqs False have $i \neq i i$ by auto
have Bii: $B \$(i, i i)=A \$ \$(i, i i)$ unfolding $B$-outside $[O F i\langle i i<n 〉$
False] ..
hence $C$ : ? $C \$ \$(i, j)=$ ? Aij unfolding $C$ B Bii using $\langle i \neq i i\rangle$ A-outside-ii $[O F i$ False $]$ by auto
show ?thesis unfolding $D C$..
next
case True
with index have $i$-begin $\leq i i \leq i$-end $i<k$ by auto
note index $=$ index this
show ?thesis
proof (cases $j>k$ )
case True
note index $=$ index this
have $D: ? D \$(i, j)=($ if $i$-begin + diff $\leq i$ then ? Aij $-q u o t * A \$ \$(l$ $+i-i$-end, $j$ ) else ? Aij) unfolding $D$
using index by auto
have $B: B \$ \$(i, j)=($ if $i$-begin + Suc diff $\leq i$ then ?Aij $-q u o t * A \$ \$$
$(l+i-i$-end,$j)$ else ?Aij) unfolding $B$
using index by auto
from index eqs have $j>l l$ by auto
hence $C$ : ? $C \$ \$(i, j)=($ if $i=$ ii then $B \$ \$(i, j)-q u o t * B \$ \$(l l, j)$
else $B \$(i, j)$ ) unfolding $C$
using index by auto
show ?thesis
proof (cases $i$-begin + Suc diff $\leq i \vee \neg(i$-begin + diff $\leq i))$ case True
from True eqs index have $i \neq i i$ by auto
from True have ? $D \$ \$(i, j)=B \$ \$(i, j)$ unfolding $D B$ by auto also have $B \$ \$(i, j)=? C \$ \$(i, j)$ unfolding $C$ using $\langle i \neq i i\rangle$ by
finally show ?thesis ..
next case False
hence $i: i=i$-begin + diff by simp
with eqs index have $i i$ : $i i=i$ by auto
from index eqs $i$ ii have $l l$ : $l l=l+i-i$-end by auto
have not: $\neg(i$-begin + Suc diff $\leq l l \wedge l l \leq i$-end $)$
proof
from eqs have $l l \leq l$ by auto
assume $i$-begin + Suc diff $\leq l l \wedge l l \leq i$-end
hence $i$-begin $<l l l l \leq i$-end by auto
with $\langle l l \leq l\rangle$ have $i$-begin $<l$ by auto
with $l$-less- $i$ have $\neg l<i$-end by auto
hence $l \geq i$-end by simp
with $i l$-less-l have $i$-end $<l b$ by auto
from index large eqs have $l b \leq l l$ by auto
with $\langle i$-end $<l b\rangle$ have $i$-end $<l l$ by auto
with 〈ll $\leq i$-end〉
show False by auto
qed
have $D:$ ? $D \$ \$(i, j)=$ ?Aij $-q u o t * A \$ \$(l l, j)$ unfolding $D$
unfolding $i l l$ by simp
have $C$ : ? $C \$ \$(i, j)=$ ? Aij $-q u o t * B \$ \$(l l, j)$ unfolding $C B$
unfolding ii $i$ by simp
have $B: B \$ \$(l l, j)=A \$ \$(l l, j)$ unfolding $B B[O F\langle l l<n\rangle j]$ using
index not by auto
show ?thesis unfolding $C D B$ unfolding ii $i$ by (simp split: if-splits)
qed
next
case False
hence $j<k \vee j=k$ by auto
thus ?thesis
proof
assume $j k: j=k$
hence $j \neq$ Suc $l-S u c$ iter using index by auto
hence ? $D \$(i, j)=(i f i=i$-end then 0 else ?Aij) unfolding $D$ using $j k$ by auto
also have $\ldots=0$ using $A$-ik[OF ib-block $\langle i$-begin $\leq i\rangle]\langle i \leq i$-end $\rangle$ unfolding $j k$ by auto
finally have $D$ : ? $D \$ \$(i, j)=0$.
from $j k$ index have $j \neq l l$ by auto
hence $C$ : ? $C \$ \$(i, j)=($ if $i=$ ii then $B \$ \$(i, j)-q u o t * B \$ \$(l l$,
j) else $B \$ \$(i, j))$
unfolding $C$ unfolding $j k$ by simp
have $C$ : ? $C \$ \$(i, j)=0$
proof (cases $i=i$-end)
case False
with index ii $j k$ have $i$ : $i$-begin $\leq i i<i$-end by auto
from $A$-ik[OF ib-block this] have $A i j: A \$ \$(i, j)=0$ unfolding $j k$.
from index $i j k$ have $\neg((i, j)=(i$-end - iter, Suc $l-i t e r) \wedge$ iter $\notin\{0$, Suc ?idiff $\}$ ) by auto
hence $B i j$ : $B \$(i, j)=0$
unfolding $B$ Aij using $i j k$ by auto
hence $C$ : ? $C \$ \$(i, j)=($ if $i=$ ii then $-q u o t * B \$ \$(l l, j)$ else 0$)$
unfolding $C$ by auto
let $? l=l-$ iter
from index eqs have $l l: l l=? l$ by auto
show ? $C \$ \$(i, j)=0$
proof (cases $i=i i$ )
case True
with index eqs $i$ have $l: l b \leq ? l ? l<l$ and diff: Suc diff $\neq$ Suc ?idiff by auto
from $A$-ik[OF $l b l l]$ have $A l j: A \$ \$(l l, j)=0$ unfolding $j k l l$.
from index $l j k$ eqs have $\neg((l l, j)=(i$-end - iter, Suc $l-i t e r) \wedge$
iter $\notin\{0$, Suc ? idiff $\}$ ) by auto
hence $B i j: B \$(l l, j)=0$ unfolding $B B[O F\langle l l<n\rangle j] A l j$ using $l j k$ diff by auto
thus ?thesis unfolding $C$ by simp
next
case False
thus ?thesis unfolding $C$ by simp
qed
next
case True note $i=$ this
hence Bij: $B \$ \$(i, j)=($ if diff $=$ ? idiff then $A \$ \$(i$-end,$k)$ else 0$)$
unfolding $B$ unfolding $j k$ by auto
show ?thesis
proof (cases $i=i i$ )
case True
with $i$ eqs have diff $=$ ? idiff $l l=l$ iter $=0$ by auto
hence $B$ : $B \$ \$(i, j)=A \$ \$$ ( $i$-end, $k$ ) unfolding Bij by auto
have $C$ : ? $C \$ \$(i, j)=A \$ \$(i$-end,$k)-q u o t * B \$ \$(l, k)$
unfolding $C B$ unfolding True $\langle l l=l\rangle j k$ by simp
also have $B \$(l, k)=A \$ \$(l, k)$
unfolding $B B[O F\langle l<n\rangle\langle k<n\rangle]$ using $i l\langle i t e r=0\rangle$ by auto also have $A \$ \$(i$-end, $k)-q u o t * \ldots=0$ unfolding quot-def
using Alk by auto
finally show ?thesis. next
case False
with $i$ eqs have diff $\neq$ ? idiff by auto
thus ?thesis unfolding $C$ Bij using False by auto
qed
qed
show ?thesis unfolding $C D$..
next
assume $j k: j<k$
from eqs il have $i i \neq l l$ by auto
show ?thesis
proof (cases diff $=0 \vee(i, j) \neq(i i-1, l l))$
case False
with eqs have $*^{*}$ : $i=i$-end - Suc iter $j=l-$ iter iter $\neq$ ? idiff and $*$ : diff $\neq 0 i=i i-1 j=l l i i \neq 0 i \neq i i$ by auto
hence $D$ : ? $D \$(i, j)=$ quot unfolding $D$ using $j k$ index by auto
from $*$ index eqs False $j k$ have $i$ : ii $=$ Suc $i i<i$-end by auto
from iblock(3)[OF ib-block 〈i-begin $\leq i\rangle\langle i<i$-end $\rangle$ ]
have $A i: A \$ \$(i, i i)=1$ unfolding $i$.
have $i i<k i \neq i$-end - iter using index $* * *$ eqs by (blast, force)
hence $B i$ : $B \$ \$(i, i i)=1$ unfolding $B B[O F\langle i<n\rangle\langle i i<n\rangle] A i$
by auto
have $B \$ \$(i, l l)=A \$ \$(i, l l)$ unfolding $B B[O F\langle i<n\rangle\langle l l<n\rangle]$
using $\langle i \neq i$-end - iter $\rangle\langle l l<k\rangle$ by auto
also have $A \$ \$(i, l l)=0$
proof (rule ccontr)
assume $n z: A \$ \$(i, l l) \neq 0$
from $i$ eqs $i l$ have neq: Suc $i \neq l l$ by auto
from $j b[O F\langle i<n\rangle\langle l l<n\rangle\langle l l<k\rangle] n z n e q$
have $i=l l$ by auto
with $i$ have $i i=S u c l l$ by simp
hence $i$-end - iter $=$ Suc $l-$ iter using eqs by auto
from block $[O F$ this $]$ have $i$-begin $=i$-end by auto
with large $i b l b$ index have $i=i i$ by auto
with * show False by auto
qed
finally have $C$ : ? $C \$ \$(i, j)=q u o t$ unfolding $C$ using $* B i$ by auto show ?thesis unfolding $C D$..
next
case True
with eqs have $\neg((i, j)=(i$-end - Suc iter, $l-$ iter $) \wedge$ iter $\neq$ ? idiff $)$
and not: $\neg(i=i i-1 \wedge j=l l \wedge$ iter $\neq$ ? idiff $)$ by auto
from this (1) have $D$ : ? $D \$ \$(i, j)=$ ? Aij unfolding $D$ using $j k$ index
by auto
\{
fix $i$
assume $i<n$
with index have id: $((i, i)=(i$-end,$k))=$ False $(i$-begin + Suc diff $\leq i \wedge i \leq i$-end $\wedge k<i)=$ False by auto
\{
assume $*:(i, i)=(i$-end - iter, Suc $l-$ iter $) \wedge$ iter $\notin\{0$, Suc
?idiff $\}$
hence $i$-end - iter $=$ Suc $l-$ iter by auto
from block[OF this] * index large eqs have False by auto
\}
hence $B \$ \$(i, i)=$ ?ev $i$ unfolding $B B[O F\langle i<n\rangle\langle i<n\rangle]$ id if-False by auto
\} note Bdiag $=$ this
from eqs have $i i: i i=i$-end - iter Suc $l-i$ ter $=S u c l l$ by auto have $B: B \$ \$(i, j)=$
(if $(i, j)=(i i$, Suc $l l) \wedge$ iter $\neq 0$ then quot else $A \$ \$(i, j))$
unfolding $B$ using $i i j k$ iter by auto
have $l l-i: l l \neq i$-end - iter using $\langle i i \neq l l\rangle$ eqs by auto
have $B \$ \$(l l, i i)=A \$ \$(l l, i i)$ unfolding $B B[O F\langle l l<n\rangle\langle i i<n\rangle]$
using 〈ii<k〉ll-i by auto
also have $\ldots=0$
proof (rule ccontr)
assume $n z: A \$(l l, i i) \neq 0$
with $j b[O F\langle l l<n\rangle\langle i i<n\rangle\langle i i<k\rangle]\langle i i \neq l l\rangle$ have $S u c l l=i i$
by auto
with eqs have $i$-end - iter $=S u c l-$ iter by auto
from block $[O F$ this $]$ index eqs have iter $=0$ by auto
with $i i$ have $l l=l i i=i$-end by auto
with Alie nz show False by auto
qed
finally have $B l i: B \$(l l, i i)=0$.
have $C$ : ? $C \$ \$(i, j)=$ ? Aij
proof (cases $i=j$ )
case True
show ?thesis unfolding $C$ unfolding $\operatorname{Bdiag}[O F \prec j<n\rangle]$ True
using $\langle i i \neq l l\rangle B l i$
by auto
next
case False
from $l b$ eqs index large have $l b \leq l l l l \leq l$ by auto
note $C=C[$ unfolded Bdiag $[O F<i<n\rangle] \operatorname{Bdiag}[O F<j<n\rangle]]$
show ?thesis
proof (cases $(i, j)=(i i$, Suc ll $) \wedge$ iter $\neq 0)$
case True
hence $*: i=i i j=$ Suc ll iter $\neq 0$ by auto
from $*$ eqs index have $l l<l$ Suc $l l<k S u c l l<n$ by auto
have $B: B \$(i, j)=$ quot unfolding $B$ using $*$ by auto
have $\neg((l l, j)=(i$-end - iter, Suc $l-$ iter $) \wedge$ iter $\notin\{0$, Suc

```
            using * index eqs by auto
    hence }\mp@subsup{B}{}{\prime}:B$$(ll,j)=A$$(ll,j
                            unfolding BB[OF<ll< n><j<n>] using jk by auto
                            have ?C $$ (i,j) = quot - quot * A $$ (ll, Suc ll) unfolding C
B using * B' by auto
                            with iblock(3)[OF lbl«lb\leqll\rangle\langlell<l\rangle] have C:?C $$ (i,j)=0
by simp
    {
    assume A $$ (ii, Suc ll)}\not=
    with jb[OF〈ii<n\rangle\langleSuc ll < n\rangle\langleSuc ll<k\rangle]\langleii\not=ll\rangle
    have ii=Suc ll by auto
    with eqs have i-end - iter = Suc l - iter by auto
    from block[OF this]\langleiter }\not=0\rangle\langleiter \not= Suc ?idiff\rangle eqs large hav
False by auto
            }
            hence A $$ (ii,Suc ll) = 0 by auto
    thus ?thesis unfolding C unfolding * by simp
    next
    case False
    hence B: B $$ (i,j) = ?Aij unfolding B by auto
    from eqs index have lb\leqll ll \leql by auto
    note index = index this ll-i
    from evb[of ll k] index have evl: ?ev ll = ?ev k by auto
    from evb[of ik] index have evi: ?ev i=? ?v k by auto
    from not have not: i\not=ii-1\veej\not=ll\vee iter= ?idiff by auto
    from False have not2: i\not=ii\veej\not=Sucll\vee iter = 0 by auto
    show ?thesis
    proof (cases i= ii)
        case True
            let ?diff = if j = ll then 0 else - quot *A $$ (ll, j)
            have Bli: B $$ (ll,i)=0 using True Bli by simp
            have Blj: B $$ (ll, j)=A$$ (ll,j) unfolding BB[OF<ll<n>
<j<n>]
            using index jk by auto
            from True have C: ?C $$ (i,j) = ?Aij + ?diff
            unfolding C B evi using Bli Blj evl by auto
            also have ? diff =0
            proof (rule ccontr)
                        assume ?diff \not=0
                            hence jl: j\not=ll and Alj: A $$ (ll,j) \not=0 by (auto split: if-splits)
        with jb[OF\langlell<n\rangle\langlej<n>jk] have j=Sucll ?ev ll=? ?ev j
by auto
            with not2 True have iter = 0 by auto
            with eqs index jk have id: A $$ (ll, j)=A $$ (l, Suc l) and
                    j=Sucl Suc l<kll=l
                    unfolding <j = Suc ll` by auto
                            from iblock[OF lbl]〈Suc l<k\rangle have A $$ (l,Suc l)\not=1 by
auto
```

    from \(j b[O F\langle l<n\rangle\langle j<n\rangle j k]\) Alj this show False unfolding
    ```
<j=Suc l`\langlell = l` by auto
                qed
                finally show ?thesis by simp
                next
                    case False
                            let ?diff = if j = ll then quot * B $$(i, ii) else 0
                            from False have C: ?C $$ (i,j) = ?Aij + ?diff
                    unfolding C B by auto
                            also have ?diff = 0
                    proof (rule ccontr)
                    assume ?diff \not=0
                            hence j:j=ll and Bi:B $$(i,ii)\not=0 by (auto split: if-splits)
                        from eqs have ii: i-end - iter = ii by auto
                    have Bii: B$$(i,ii)=A$$(i,ii)
                    unfolding BB[OF〈i<n\rangle\langleii<n\rangle] using <ii<k\rangle iter ii
False by auto
                                from Bi Bii have Ai: A $$ (i,ii)\not=0 by auto
                        from jb[OF <i<n\rangle\langleii<n\rangle\langleii<k\rangle] False Ai have ii: ii=
Suc i
                    and Ai:A $$ (i,ii)=1 by auto
                    from not ii j have iter: iter = ?idiff by auto
                            with eqs index have ii = i-begin by auto
                            with ii <i\geq i-begin>
                            show False by auto
                            qed
                            finally show ?thesis by simp
                        qed
                        qed
                        qed
                        show ?thesis unfolding D C ..
                qed
                qed
            qed
        qed
        qed auto
        also have step-3-c-inner-loop quot (ll - 1) (ii - 1) diff ... = ?R
            by (rule Suc(1), insert prems large, auto)
            finally show ?case.
        qed
}
note main = this[of Suc ?idiff i-end 0 l]
from ib have suc: Suc i-end - i-begin = Suc ?idiff by simp
have step-3-c-inner-loop (A $$ (i-end, k) / A $$ (l,k)) l i-end (Suc ?idiff) A
    = step-3-c-inner-loop quot l i-end (Suc ?idiff) (?mat 0 (Suc ?idiff))
    unfolding mend unfolding quot-def ..
    also have \ldots.. = ?R by (rule main, auto)
    finally show ?thesis unfolding suc.
qed
```

private lemma step-3-c-inv: $A \in$ carrier-mat $n$ n
$\Longrightarrow k<n$
$\Longrightarrow(l b, l) \in \operatorname{set}($ identify-blocks $A k)$
$\Longrightarrow$ inv-upto jb Ak
$\Longrightarrow$ inv-from uppert $A k$
$\Longrightarrow$ inv-at one-zero $A k$
$\Longrightarrow$ ev-block $n A$
$\Longrightarrow$ set bs $\subseteq$ set (identify-blocks $A k$ )
$\Longrightarrow(\bigwedge$ be. be $\notin$ snd'set $b s \Longrightarrow b e \notin\{l, k\} \Longrightarrow b e<n \Longrightarrow A \$ \$(b e, k)=0)$
$\Longrightarrow(\bigwedge b b b e .(b b, b e) \in$ set $b s \Longrightarrow b e-b b \leq l-l b)$ - largest block
$\Longrightarrow x=A \$ \$(l, k)$
$\Longrightarrow x \neq 0$
$\Longrightarrow$ inv-all uppert (step-3-c x lkbsA)
$\wedge$ same-diag $A($ step-3-c $x l k$ bs $A)$
$\wedge$ same-upto $k A($ step-3-c x $l k b s A)$
$\wedge$ inv-at (single-non-zero lkx) (step-3-c x lkbs A) $k$
proof (induct bs arbitrary: A)
case (Nil A)
note $\operatorname{inv}=\operatorname{Nil}(4-7)$
from inv-from-upto-at-all-ev-block[OF $\operatorname{inv}(1-4)-\langle k<n\rangle]$
have inv-all uppert $A$ unfolding one-zero-def diff-ev-def uppert-def by auto moreover
have inv-at (single-non-zero $l k x$ ) A $k$ unfolding single-non-zero-def inv-at-def by (intro allI impI conjI, insert Nil, auto)
ultimately show ?case by auto

## next

case (Cons $p$ bs A)
obtain $i$-begin $i$-end where $p: p=(i$-begin, $i$-end $)$ by force
note Cons $=$ Cons[unfolded $p$ ]
note $I H=\operatorname{Cons}(1)$
note $A=\operatorname{Cons}(2)$
note $k n=\operatorname{Cons}(3)$
note $l b l=\operatorname{Cons}(4)$
note $\operatorname{inv}=\operatorname{Cons}(5-8)$
note blocks $=\operatorname{Cons}(9-11)$
note $x=\operatorname{Cons}(12-13)$
from identify-blocks[OF lbl] $k n$ have $l k: l<k$ and $l n: l<n$ and $l b \leq l$ by auto
define $B$ where $B=$ step-3-c-inner-loop $(A \$ \$(i$-end,$k) / x)$ li-end (Suc $i$-end

- i-begin) $A$
show ?case
proof (cases $i$-end $=l$ )
case True
hence id: step-3-c x lk( $p$ \# b) $A=$ step-3-c x $l k b s A$ unfolding $p$ by simp show ?thesis unfolding id
by (rule IH[OF A kn lbl inv-blocks(2-3) x], insert blocks(1), auto simp: p True)
next
case False note $i l=$ this
hence id: step-3-c x lk( $\quad \# \operatorname{bs}) A=$ step-3-c x $l k b s B$ unfolding $B$-def $p$


## by $\operatorname{simp}$

from blocks[unfolded $p$ ] have
ib-block: $(i$-begin, $i$-end $) \in$ set (identify-blocks $A k)$ and large: $i$-end - $i$-begin $\leq l-l b$ by auto
from identify-blocks[OF this(1)]
have ibe: $i$-begin $\leq i$-end $i$-end $<k$ by auto
have $B: B=$ mat $n n(\lambda(i, j)$. if $(i, j)=(i$-end,$k)$ then 0 else
if $i$-begin $\leq i \wedge i \leq i$-end $\wedge j>k$ then $A \$ \$(i, j)-A \$ \$(i$-end,$k) / x * A$
$\$ \$(l+i-i$-end,$j)$ else $A \$ \$(i, j))$
unfolding $B$-def $x$
by (rule step-3-c-inner-result[OF inv kn A lbl ib-block il large], insert $x$, auto)
have $B n: B \in$ carrier-mat $n n$ unfolding $B$ by auto
have sdAB: same-diag $A B$ unfolding $B$ same-diag-def using ibe by auto
have suAB: same-upto $k A B$ using $A$ kn unfolding $B$ same-upto-def by auto
have inv-ev: ev-block $n B$ using same-diag-ev-block[OF sdAB inv(4)].
have inv-jb: inv-upto jb Bk using same-upto-inv-upto-jb[OF suAB inv(1)].
have ib: identify-blocks $A k=$ identify-blocks $B k$ using identify-blocks-cong[OF $k n s d A B$ su $A B]$.
have inv-ut: inv-from uppert $B k$ using inv(2) ibe unfolding $B$ inv-from-def uppert-def by auto
from $x$ il ibe $k n l k$ have $x B: x=B \$ \$(l, k)$ by (auto simp: $B$ )
\{
fix be
assume $*$ : be $\notin$ snd' set bs be $\notin\{l, k\}$ be $<n$ hence $B \$ \$(b e, k)=0$ using kn blocks(2)[of be] unfolding $B$
by (cases be $=i$-end, auto)
\} note blocksB $=$ this
have bs: set bs $\subseteq$ set (identify-blocks $A k$ ) using blocks(1) by auto
have inv-oz: inv-at one-zero $B k$ using $\operatorname{inv(3)}$ ibe $k n$ unfolding $B$ inv-at-def one-zero-def by simp
show ?thesis unfolding id using $I H[O F$ Bn kn, folded ib, OF lbl inv-jb inv-ut inv-oz inv-ev bs blocksB
blocks(3) xB x(2)]
using same-diag-trans[OF sdAB] same-upto-trans $[O F$ suAB]
by auto
qed
qed
lemma step-3-main-inv: $A \in$ carrier-mat $n n$
$\Longrightarrow k>0$
$\Longrightarrow$ inv-all uppert $A$
$\Longrightarrow$ ev-block $n A$
$\Longrightarrow$ inv-upto $j b A k$
$\Longrightarrow$ inv-all jb (step-3-main $n k A) \wedge$ same-diag $A($ step-3-main $n k A)$
proof (induct $k$ A taking: $n$ rule: step-3-main.induct)
case ( $1 k A$ )
from 1 (2-) have $A: A \in$ carrier-mat $n n$ and $k: k>0$ and
inv: inv-all uppert $A$ ev-block $n A$ inv-upto $j b A k$ by auto
note $[$ simp $]=$ step-3-main.simps $[$ of $n k A]$

```
show ?case
proof (cases k}\geqn
    case True
    thus ?thesis using inv-uptoD[OF inv(3)]
        by (intro conjI inv-allI, auto)
next
    case False
    hence kn: k<n by simp
    obtain B where B: B= step-3-a (k-1) kA by auto
    note IH=1(1)[OF False B]
    from A have Bn: B\incarrier-mat n n unfolding B carrier-mat-def by simp
    from k have k-1<k by simp
    {
        fix }
        assume i<k
        with ev-blockD[OF inv(2) - <k<n\rangle, of i] <k<n\rangle have A $$ (i,i)=A$$
(k,k) by auto
    }
    hence inv-from-bot ( }\lambdaA\mathrm{ i. one-zero A ik) A (k-1)
        using inv-all-uppertD[OF inv(1), of k]
        unfolding inv-from-bot-def one-zero-def by auto
    from step-3-a-inv[OF A<k-1<k\rangle\langlek<n> inv(3) inv-all-imp-inv-from[OF
inv(1)]
            this inv(2)] same-diag-ev-block[OF - inv(2)]
    have inv: inv-from uppert B k ev-block n B inv-upto jb B k
        inv-at one-zero B k and sd: same-diag A B unfolding B by auto
    note evb = ev-blockD[OF inv(2)]
    obtain all-blocks where ab: all-blocks = identify-blocks B k by simp
    obtain blocks where blocks: blocks = filter ( }\lambda\mathrm{ block. B $$ (snd block,k)}=0\mathrm{ )
all-blocks by simp
    obtain F where F:F=(if blocks = [] then B
        else let (l-begin,l) = find-largest-block (hd blocks) (tl blocks); x=B $$ (l, k);
C = step-3-c x l k blocks B;
                    D=mult-col-div-row (inverse x) k C; E= swap-cols-rows-block (Suc l)
k D
            in E) by simp
    note IH = IH[OF ab blocks F]
    have Fn: F G carrier-mat n n unfolding F Let-def carrier-mat-def using Bn
        by (simp split: prod.splits)
    have inv: inv-all uppert F}\wedge\mathrm{ same-diag A F ^ inv-upto jb F (Suc k)
    proof (cases blocks=[])
        case True
        hence F:F=B unfolding F by simp
        have lo: inv-at (lower-one k) Bk
        proof
            fix }
            assume i:i<n
            note lower-one-def[simp]
            show lower-one k B i k
```

```
        proof (cases \(B \$ \$(i, k)=0)\)
            case False note \(n z=\) this
            note \(o z=\operatorname{inv}-a t D[O F \operatorname{inv}(4) i\), unfolded one-zero-def \(]\)
    from \(n z\) oz have \(i \leq k\) by auto
    show ?thesis
    proof (cases \(i=k\) )
            case False
            with \(\langle i \leq k\rangle\) have \(i<k\) by auto
            with \(n z\) oz have ev: \(B \$ \$(i, i)=B \$ \$(k, k)\) unfolding diff-ev-def by
auto
            have (identify-block \(B i, i) \in\) set all-blocks unfolding \(a b\)
            proof (rule identify-blocks-rev[OF-〈k<n〉])
                show \(B \$ \$(i\), Suc \(i)=0 \wedge\) Suc \(i<k \vee\) Suc \(i=k\)
                proof (cases Suc \(i=k\) )
                    case False
                        with \(\langle i<k\rangle\langle k<n\rangle\) have Suc \(i<k\) Suc \(i<n\) by simp-all
                    with \(n z\) oz have \(B \$ \$(i\), Suc \(i) \neq 1\) by simp
                    with inv-uptoD[OF \(\operatorname{inv}(3)\langle i<n\rangle\langle S u c i<n\rangle\langle S u c i<k\rangle\), unfolded
\(j b-d e f]\)
                    have \(B \$ \$(i\), Suc \(i)=0\) by simp
                    thus ?thesis using \(\langle S u c i<k\rangle\) by simp
                    qed \(\operatorname{simp}\)
            qed
            with arg-cong \([O F 〈 b l o c k s=[]\rangle[\) unfolded blocks \(]\), of set \(]\) have \(B \$ \$(i, k)\)
\(=0\) by auto
            with \(n z\) show ?thesis by auto
            qed auto
        qed auto
    qed
    have inv-jb: inv-upto \(j b B\) (Suc \(k\) )
    proof (rule inv-upto-Suc[OF inv(3)])
        fix \(i\)
        assume \(i<n\)
        from inv-atD[OF lo \(\langle i<n\rangle\), unfolded lower-one-def]
        show \(j b B i k\) unfolding \(j b\)-def by auto
        qed
    from inv-from-upto-at-all-ev-block[OF inv(3,1) lo inv(2) - \(\langle k<n\rangle]\) lower-one-diff-uppert
        have inv-all uppert \(B\) by auto
        with inv inv-jb sd
        show ?thesis unfolding \(F\) by simp
    next
        case False
        obtain l-start \(l\) where \(l\) : find-largest-block (hd blocks) \((t l\) blocks \()=(l\)-start,
l) by force
    obtain \(x\) where \(x: x=B \$ \$(l, k)\) by simp
    obtain \(C\) where \(C: C=\) step-3-c x lk blocks \(B\) by simp
    obtain \(D\) where \(D: D=\) mult-col-div-row (inverse \(x\) ) \(k C\) by auto
    obtain \(E\) where \(E: E=\) swap-cols-rows-block (Suc l) \(k D\) by auto
    from find-largest-block[OF False l] have lb: \((l\)-start,\(l) \in\) set blocks
```

and llarge: $\bigwedge i$-begin $i$-end. $(i$-begin, $i$-end $) \in$ set blocks $\Longrightarrow l-l$-start $\geq$ $i$-end $-i$-begin by auto
from $l b$ have $x 0: x \neq 0$ unfolding blocks $x$ by simp
\{
fix $i$-start $i$-end
assume $(i$-start,$i$-end $) \in$ set blocks
hence $(i$-start, $i$-end) $\in$ set (identify-blocks $B k$ ) unfolding blocks $a b$ by simp
with identify-blocks[OF this]
have $i$-end $<k$ ( $i$-start,, -end $) \in$ set (identify-blocks $B k$ ) by auto
$\}$ note block-bound $=$ this
from block-bound[OF lb]
have $l k: l<k$ and lblock: (l-start, $l$ ) $\in$ set (identify-blocks $B k$ ) by auto
from $l k\langle k<n\rangle$ have $l n: l<n$ by simp
from $e v b[O F\langle l<n\rangle\langle k<n\rangle]$
have $B l l: B \$(l, l)=B \$ \$(k, k)$.
from False have $F: F=E$ unfolding ED C x F l Let-def by simp from $B n$ have $C n: C \in$ carrier-mat $n n$ unfolding $C$ carrier-mat-def by simp
$\{$
fix be
assume nmem: be $\notin$ snd' set blocks and belk: be $\notin\{l, k\}$ and $b e: b e<n$
have $B \$ \$(b e, k)=0$
proof (rule ccontr)
assume $n z: \neg$ ?thesis
note $o z=$ inv-atD $[O F \operatorname{inv(4)~be,~unfolded~one-zero-def]~}$
from belk oz be nz have be $<k$ by auto
obtain $b b$ where $i b$ : identify-block $B b e=b b$ by force
note $i b-i n v=$ identify-block[OF ib]
have $B \$ \$(b e$, Suc be $)=0 \wedge$ Suc be $<k \vee$ Suc be $=k$
proof (cases Suc be $=k$ )
case False
with 〈be $<k$ 〉 have sbek: Suc be $<k$ by auto
from inv-uptoD $[O F \operatorname{inv}(3)\langle b e<n\rangle-s b e k]$ sbek kn have $j b B$ be (Suc
be) by auto
from this[unfolded jb-def] have $01: B \$ \$$ (be, Suc be) $\in\{0,1\}$ by auto from 01 oz sbek $n z$ have $B \$ \$(b e$, Suc be) $=0$ by auto
with sbek show ?thesis by auto
qed auto
from identify-blocks-rev[OF this kn]
nz nmem show False unfolding ab blocks by force
qed
\}
note $\operatorname{inv} 3=$ step-3-c-inv[OF Bn $\langle k<n\rangle \operatorname{lblock} \operatorname{inv}(3,1,4,2)-$ this llarge $x$ x0, of blocks, folded C,
unfolded ab blocks]
from inv3 have sdC: same-diag $B C$ and suC: same-upto $k B C$ by auto note $s d=$ same-diag-trans $[O F$ sd $s d C]$
from Bll sdC ln $\langle k<n\rangle$

```
        have Cll:C $$ (l,l) = C $$ (k,k) unfolding same-diag-def by auto
    from same-diag-ev-block[OF sdC inv(2)] same-upto-inv-upto-jb[OF suC inv(3)]
inv3
    have inv: inv-all uppert C ev-block n C
    inv-upto jb Ck inv-at (single-non-zero lk x) Ck by auto
    from x0 have inverse x\not=0 by simp
    from Cn have Dn: D \in carrier-mat n n unfolding D carrier-mat-def by
simp
    {
        fix ij
        assume i:i<n and j:j<n
        with Cn have dC:i<dim-row C i<dim-col C j<dim-row C j<dim-col
C by auto
    let ?c =C $$ (i,j)
    let ?x = inverse x
    have D$$(i,j)=(if i=l^j=k then 1 else if i=k\wedgej\not=k then x*?c
else ?c)
    unfolding D
    proof (subst mult-col-div-index-row[OF dC〈inverse x}\not=0\rangle]\mathrm{ , unfold in-
verse-inverse-eq)
    note at = inv-atD[OF inv(4)<i<n>, unfolded single-non-zero-def]
    show (if i=k\wedgej\not=i then x*?c
        else if j=k\wedgej\not=i then ?? * ?c else ?c) =
        (if i=l\wedgej=k then 1 else if i=k\wedgej\not=k then x* ?c else ?c) (is ?l
=?r)
            proof (cases (i,j)=(l,k))
            case True
            with lk have ?l = ? x * ?c by auto
            also have ...=1 using at True <inverse x }\not=0\mathrm{ ` by auto
            finally show ?thesis using True by simp
            next
            case False note neq = this
            have ?l = (if i=k^j\not=k then x* ?c else ?c)
            proof (cases i=k\wedgej\not=k\veej=k^i\not=k)
                case True
                thus ?thesis
                proof
                assume *: i=k^j\not=k
                hence l: ?l = x*?c by simp
                show ?thesis using * neq unfolding l by simp
                next
                    assume *: j=k^i\not=k
                hence ?l = ? x * ?c using lk by auto
                from * neq have }i\not=l\mathrm{ and **: }\neg(i=k\wedgej\not=k) by aut
                from at }\langlei\not=l\rangle*\mathrm{ have ?c = 0 by auto
                with〈?l = ? x * ?c〉** show ?thesis by auto
                qed
            qed auto
            also have ... = ?r using False by auto
```

```
            finally show ?thesis.
            qed
            qed
\} note \(D=\) this
                            have \(s D[\operatorname{simp}]: \bigwedge i . i<n \Longrightarrow D \$ \$(i, i)=C \$ \$(i, i)\) using \(l k\) by (auto
```

simp: $D$ )
from $\langle C \$ \$(l, l)=C \$ \$(k, k)\rangle\langle l<n\rangle\langle k<n\rangle$
have $D l l: D \$(l, l)=D \$ \$(k, k)$ by simp
have sdD: same-diag $C D$ unfolding same-diag-def by simp
note $s d=$ same-diag-trans $[O F$ sd $s d D]$
from same-diag-ev-block[OF sdD inv(2)] have invD: ev-block n D.
note inv $=$ inv-upto $D[O F \operatorname{inv}(3)$, unfolded $j b-$ def $]$ inv-all-uppert $D[$ OF inv(1) $]$
inv-atD $[$ OF $\operatorname{inv}(4)$, unfolded single-non-zero-def $]$
moreover have inv-all uppert $D$
by (intro inv-allI, insert inv(2) lk, auto simp: uppert-def D)
moreover have suD: same-upto $k C D$
proof
fix $i j$
assume $i: i<n$ and $j: j<k$
with $k n$ have $j n: j<n$ by simp
show $C \$ \$(i, j)=D \$ \$(i, j)$
unfolding $D[O F i j n]$ using $j k$

qed
from same-upto-inv-upto-jb[OF suD〈inv-upto jb C $k>]$
have inv-upto jb $D k$.
moreover
let ?single-one $=$ single-one $l k$
have inv-at ?single-one $D k$
by (intro inv-atI, insert $\operatorname{inv}(3) D[O F-\langle k<n\rangle] \ln$, auto simp: single-one-def)
ultimately
have inv: inv-all uppert $D$ ev-block $n D$
inv-upto jb $D k$ inv-at ?single-one $D k$ using invD by blast+
note $\operatorname{inv}=\operatorname{inv}$-upto $D[\operatorname{OF} \operatorname{inv(3),~unfolded~jb-def}]$
inv-all-uppert $D[$ OF $\operatorname{inv}(1)]$
inv-atD[OF inv(4), unfolded single-one-def]
ev-block $D[O F \operatorname{inv}(2)]$
from suC suD have suD: same-upto $k B D$ unfolding same-upto-def by auto
let ? $I=\lambda j$. if $j=$ Suc $l$ then $k$ else if Suc $l<j \wedge j \leq k$ then $j-1$ else $j$
let ? $I^{\prime}=\lambda j$. if $j=$ Suc $l$ then $k$ else $j-1$
\{
fix $i j$
assume $i: i<n$ and $j: j<n$
with $D n l k<k<n\rangle$
have dims: $i<$ dim-row $D i<d i m-c o l ~ D j<d i m-r o w ~ D j<d i m-c o l ~ D ~$
Suc $l \leq k k<$ dim-row $D k<$ dim-col $D$ by auto
have $E \$ \$(i, j)=D \$ \$(? I ~ i, ? I j)$
unfolding $E$ by (rule subst swap-cols-rows-block-index[OF dims])

```
    } note E = this
    {
    fix i
    assume i:i<n
    from <l < k> have l\leqSuc l Suc l \leqk by auto
    have E$$(i,i)=D$$(i,i) unfolding E[OF i i]
        by(rule inv(4), insert i < < < n`, auto)
} note Ed= this
from Ed have ed: same-diag D E unfolding same-diag-def by auto
note sd = same-diag-trans[OF sd ed]
have ev-block n E using same-diag-ev-block[OF ed «ev-block n D\] by auto
moreover have Eut: inv-all uppert E
proof (intro inv-allI, unfold uppert-def, intro impI)
    fix ij
    assume i: i< n and j:j<n and ji: j<i
    have ?I i<n using i<k<n> by auto
    show E $$ (i,j) = 0
    proof (cases ?I j < ?I i)
        case True
        from inv(2)[OF this «?I i < n`] show ?thesis unfolding E[OF i j].
    next
        case False
        have ?I i\not= ?I j using ji lk by (auto split: if-splits)
        with False have ij: ?I i< ?I j by simp
        from ij ji have jl: j=Suc l using lk by (auto split: if-splits)
        with ji ij have il:i>Suc li\leqk by (auto split: if-splits)
        from jl il have Eij: E$$ (i,j)=D $$ (i-1,k) unfolding E[OF i j] by
            have i-1<ni-1\not\in{k,l} using i il by auto
            with inv(3)[of i-1] have D: D $$ (i-1,k)=0 by auto
            show ?thesis unfolding Eij D by simp
        qed
    qed
    moreover
    from same-diag-trans[OF〈same-diag B C`<same-diag C D`] have same-diag
    from identify-blocks-cong[OF <k<n\rangle this suD]
    have idb: identify-blocks B k = identify-blocks D k .
    have inv-upto jb E (Suc k)
proof (intro inv-uptoI)
    fix ij
    assume i:i<n and j:j<n and j<Suc k
    hence jk: j\leqk by simp
    show jb E ij
    proof (cases E$$ (i,j)=0\veej=i)
        case True
        thus ?thesis unfolding jb-def by auto
        next
        case False note enz = this
```

simp
$B D$.


```
    note inv2 \(=\) inv-all-uppertD \([\) OF Eut \(-i\), of \(j]\)
    from False inv2 have \(\neg j<i\) by auto
    with False have \(j i: j>i\) by auto
    have \(E \$ \$(i, j) \in\{0,1\} \wedge(j \neq S u c i \longrightarrow E \$ \$(i, j)=0)\)
    proof (cases \(j \leq l\) )
        case True note \(j l=\) this
        with \(j i l k\) have \(i l: i \leq l\) and \(j k: j<k\) by auto
    have \(i d\) : \(E \$ \$(i, j)=D \$ \$(i, j)\) unfolding \(E[O F i j]\) using \(j l i l\) by simp
    from \(\operatorname{inv}(1)[O F i j j k] j i\)
    show ?thesis unfolding id by auto
next
    case False note \(j l=\) this
    show ?thesis
    proof (cases \(j=\) Suc \(l\) )
        case True note \(j l=\) this
        with \(j i l k\) have \(i l: i \leq l i \neq k\) by auto
        have \(i d\) : \(E \$ \$(i, j)=D \$ \$(i, k)\) unfolding \(E[O F i j]\) using \(j l i l\) by
auto
        from \(\operatorname{inv(3)[OF~i]~jl~il~}\)
        show ?thesis unfolding id by (cases \(i=l\), auto)
    next
        case False
        with \(j l j k k n\) have \(j l: j>S u c l\) and \(j k: j-1<k\) and \(j n: j-1<n\)
by auto
        with \(j k\) have \(i d\) : ? \(1 j=j-1\) by auto
        note \(j b=\operatorname{inv}(1)[O F-j n j k]\)
        show ?thesis
        proof (cases \(i<S u c l\) )
            case True note \(i l=\) this
            with id have \(i d\) : \(E \$ \$(i, j)=D \$ \$(i, j-1)\) unfolding \(E[O F i j]\)
by auto
            show ?thesis
            proof (cases \(i=j-2)\)
            case False
            thus ?thesis unfolding id using \(j b[O F i] i l j l\) by auto
            next
                    case True
                    with \(i l j l\) have \(*: j=S u c\) (Suc \(l\) ) \(i=l\) by auto
                    with id have id: \(E \$ \$(i, j)=D \$ \$(l, S u c l)\) by auto
                    from \(* j l j k\) have neq: Suc \(l \neq k\) by auto
                    from lblock[unfolded idb] have (l-start, l) \(\in\) set (identify-blocks \(D\)
k).
                    from this[unfolded identify-blocks-iff[OF kn]] neq
                    have \(D \$ \$(l\), Suc \(l) \neq 1\) by auto
                    with \(j b[O F i] i l j l j i *\) have \(D \$ \$(l\), Suc \(l)=0\) by auto
                    thus ?thesis unfolding id by simp
        qed
        next
```

```
case False note \(i l=\) this
show ?thesis
proof (cases \(i=\) Suc \(l\) )
    case True
    with id have \(i d\) : \(E \$ \$(i, j)=D \$ \$(k, j-1)\) unfolding \(E[O F i j]\)
```

by auto
from $\operatorname{inv}(2)[O F j k k n]$ show ?thesis unfolding id by simp
next
case False
with $i l j l j i j k$ kn have $i l: i>S u c l$ and $i k: i<k$ and $i-n: i-1$
$<n$ by auto
with $i d$ have $i d: E \$(i, j)=D \$ \$(i-1, j-1)$ unfolding $E[O F$
$i j$ ] by auto
show ?thesis unfolding id using $j b[O F i-n] i l j l j i$ by auto
qed
qed
qed
qed
thus $j b E i j$ unfolding $j b$-def $E d[O F i] E d[O F j]$ same-ev by auto
qed
qed
ultimately show ?thesis using sd unfolding $F$ by simp
qed
hence inv: inv-all uppert $F$ ev-block $n F$ inv-upto jb $F$ (Suc $k$ )
and sd: same-diag A F using same-diag-ev-block[OF - 〈ev-block $n A\rangle]$ by
auto
have $0<$ Suc $k$ by simp
note $I H=I H[O F$ Fn this inv $(1-3)]$
have id: step-3-main $n k A=$ step-3-main $n$ (Suc k) $F$ using $k n$
by (simp add: F Let-def blocks ab B)
from same-diag-trans[OF sd] IH
show ?thesis unfolding id by auto
qed
qed
lemma step-1-2-inv:
assumes $A: A \in$ carrier-mat $n n$
and upper-t: upper-triangular $A$
and Bid: $B=$ step-2 (step-1 A)
shows inv-all uppert $B$ inv-all diff-ev $B$ ev-blocks $B$
proof -
from $A$ have $d$ : dim-row $A=n$ by simp
let $? B=$ step-2 $($ step-1 $A)$
from upper-triangularD $[$ OF upper- $t]$ have inv: inv-all uppert $A$
unfolding inv-all-def uppert-def using $A$ by auto
from upper-t have inv2: inv-part diff-ev A 00
unfolding inv-part-def diff-ev-def by auto
have inv3: ev-blocks-part 0 (step-1 A)
by (rule ev-blocks-partI, auto)
have A1: step-1 $A \in$ carrier-mat $n n$ using $A$ unfolding carrier-mat-def by auto
from $A 1$ have d1: dim-row (step-1 $A$ ) $=n$ unfolding carrier-mat-def by simp have $B: ? B \in$ carrier-mat $n n$ using $A$ unfolding carrier-mat-def by auto from $B$ have d2: dim-row ? $B=n$ unfolding carrier-mat-def by simp have inv-all uppert (step-1 A) $\wedge$ inv-all diff-ev (step-1 A) unfolding step-1-def $d$
by (rule step-1-main-inv[OF - A inv inv2], simp)
hence inv-all uppert (step-1 A) and inv-all diff-ev (step-1 A) by auto
from step-2-main-inv[OF A1 this inv3]
show inv-all uppert $B$ inv-all diff-ev $B$ ev-blocks $B$ unfolding step-2-def $d$ d1 Bid by auto
qed
definition inv-all' $::\left({ }^{\prime}\right.$ a mat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow$ bool $) \Rightarrow$ 'a mat $\Rightarrow$ bool where inv-all' $p A \equiv \forall i j . i<$ dim-row $A \longrightarrow j<$ dim-row $A \longrightarrow p A i j$
private lemma lookup-other-ev-None: assumes lookup-other-ev ev $k A=$ None and $i<k$
shows $A \$ \$(i, i)=e v$
using assms by (induct ev $k$ A rule: lookup-other-ev.induct, auto split: if-splits) (insert less-antisym, blast)
private lemma lookup-other-ev-Some: assumes lookup-other-ev ev $k A=$ Some $i$ shows $i<k \wedge A \$ \$(i, i) \neq e v \wedge(\forall j . i<j \wedge j<k \longrightarrow A \$ \$(j, j)=e v)$
using assms by (induct ev $k$ A rule: lookup-other-ev.induct, auto split: if-splits) (insert less-SucE, blast)
lemma partition-jb: assumes $A:\left(A::{ }^{\prime} a\right.$ mat $) \in$ carrier-mat $n n$ and inv: inv-all uppert $A$ inv-all diff-ev $A$ ev-blocks $A$ and part: partition-ev-blocks $A[]=b s$
shows $A=$ diag-block-mat bs $\wedge B . B \in$ set $b s \Longrightarrow$ inv-all' uppert $B \wedge$ ev-block
(dim-col $B) B \wedge$ dim-row $B=$ dim-col $B$
proof -
have diag: diag-block-mat $[A]=A$ using $A$ by auto
\{
fix $c s$
assume $*: \wedge C . C \in$ set cs $\Longrightarrow$ dim-row $C=$ dim-col $C \wedge$ inv-all' uppert $C$ $\wedge$ ev-block (dim-col C) C partition-ev-blocks $A$ cs $=b s$
from inv have inv: inv-all' uppert $A$ inv-all' diff-ev $A$ ev-blocks-part $n A$
unfolding inv-all-def inv-all'-def ev-blocks-def using $A$ by auto
hence diag-block-mat $(A \# c s)=$ diag-block-mat bs $\wedge(\forall B \in$ set bs. inv-all'
uppert $B \wedge$ ev-block (dim-col $B) B \wedge$ dim-row $B=\operatorname{dim}$-col $B$ )
using $A *$
proof (induct $n$ arbitrary: $A$ cs bs rule: less-induct)
case (less $n A$ cs bs)
from less(5) have $A$ : $A \in$ carrier-mat $n n$ by auto
hence dim: dim-row $A=n \operatorname{dim}-\operatorname{col} A=n$ by auto

```
    let ? dim \(=\) sum-list (map dim-col cs)
    let ? \(C=\) diag-block-mat cs
    define \(C\) where \(C=? C\)
    from less \((6)\) have \(c s: \wedge C . C \in\) set \(c s \Longrightarrow\) inv-all' uppert \(C \wedge\) ev-block
(dim-col \(C) C \wedge\) dim-row \(C=\operatorname{dim}-c o l C\) by auto
    hence dimcs[simp]: sum-list (map dim-row cs) = ?dim by (induct cs, auto)
    from dim-diag-block-mat[of cs, unfolded dimcs] obtain \(n c\) where \(C\) : ?C \(\in\)
carrier-mat nc nc unfolding carrier-mat-def by auto
    hence \(\operatorname{dim} C\) : dim-row \(C=n c\) dim-col \(C=n c\) unfolding \(C\)-def by auto
            note \(b s=\operatorname{less}(7)[\) unfolded partition-ev-blocks.simps[of A cs] Let-def dim,
symmetric]
    show ?case
    proof (cases \(n=0\) )
        case True
        hence \(b s\) : \(b s=c s\) unfolding \(b s\) by simp
        thus ?thesis using cs A by (auto simp: Let-def True)
    next
        case False
        let \(? n 1=n-1\)
        let ?look = lookup-other-ev ( \(A\) \$\$ (?n1, ?n1)) ?n1 A
        show ?thesis
        proof (cases ?look)
            case None
            from False None have bs: bs \(=A \# c s\) unfolding bs by auto
            have ut: inv-all' uppert \(A\) using less(2) by auto
            from lookup-other-ev-None[OF None] have \(\wedge i . i<n \Longrightarrow A \$ \$(i, i)=\)
A \$\$ (?n1, ?n1)
                by (case-tac \(i=\) ?n1, auto)
            hence evb: ev-block \(n A\) unfolding ev-block-def dim by metis
            from cs \(A\) ut evb show ?thesis unfolding bs by auto
            next
                case (Some i)
            let ? \(s i=\) Suc \(i\)
                    from lookup-other-ev-Some[OF Some] have \(i: i<? n 1\) and neq: \(A \$ \$\)
\((i, i) \neq A \$ \$\) (?n1, ? n1)
            and between: \(\bigwedge j . i<j \Longrightarrow j<? n 1 \Longrightarrow A \$ \$(j, j)=A \$ \$(? n 1, ? n 1)\)
by auto
            define \(m\) where \(m=n-\) ? \(s i\)
            from \(i\) False have si: ?si \(<n\) by auto
            from False \(i\) have nsi: \(n=\) ? si \(+m\) unfolding \(m\)-def by auto
            obtain \(U L U R L L L R\) where split: split-block A ? si ?si \(=(U L, U R, L L\),
                \(L R)\) by (rule prod-cases4)
            from split-block[OF split dim[unfolded nsi]]
                    have carr: UL \(\in\) carrier-mat ?si ?si UR \(\in\) carrier-mat ?si \(m L L \in\)
carrier-mat \(m\) ?si \(L R \in\) carrier-mat \(m m\)
            and Ablock: \(A=\) four-block-mat \(U L U R L L L R\) by auto
            hence \(\operatorname{dim} L R\) : dim-row \(L R=m \operatorname{dim}\)-col \(L R=m\) and dimUL: dim-col
\(U L=\) ?si dim-row \(U L=\) ? si by auto
            from less(3)[unfolded inv-all'-def diff-ev-def] dim
```

have diff: $\wedge i j . i<n \Longrightarrow j<n \Longrightarrow i<j \Longrightarrow A \$ \$(i, i) \neq A \$ \$(j, j)$ $\Longrightarrow A \$ \$(i, j)=0$ by auto
from less(2)[unfolded inv-all'-def uppert-def] dim
have ut: $\bigwedge i j . i<n \Longrightarrow j<n \Longrightarrow j<i \Longrightarrow A \$ \$(i, j)=0$ by auto
let ? $U R=O_{m}$ ?si $m$
have $U R: U R=$ ? $U R$
proof (rule eq-matI)
fix $i a j$
assume $i j$ : ia $<$ dim-row $\left(O_{m}(\right.$ Suc $\left.i) m\right) j<\operatorname{dim}-c o l\left(O_{m}\right.$ (Suc $\left.\left.i\right) m\right)$
let $? j=$ ? $s i+j$
have $U L \$ \$(i a, i a)=A \$ \$(i a, i a)$ using ij carr unfolding Ablock by
auto
also have $\ldots \neq A \$ \$(? j, ? j)$
proof
assume eq: $A \$ \$(i a, i a)=A \$ \$(? j, ? j)$
from $i j$ have rel: $i a \leq i i \leq ? j ? j<n$ using nsi $i$ by auto
from ev-blocks-part-leD[OF less(4) this eq[symmetric]] eq
have eq: $A \$ \$(i, i)=A \$ \$(? j, ? j)$ by auto
also have $\ldots=A \$ \$(? n 1, ? n 1)$ using between[of ?j] rel by (cases ?j
= ?n1, auto)
finally show False using neq by auto
qed
also have $A \$ \$(? s i+j$, ?si $+j)=L R \$ \$(j, j)$ using $i j$ carr unfolding
Ablock by auto
finally show $U R \$ \$(i a, j)=O_{m}$ (Suc i) $m \$ \$(i a, j)$
using diff [of ia ?si $+j$, unfolded Ablock] ij nsi carr by auto
qed (insert carr, auto)
let ? $L L=0_{m} m$ ?si
have $L L: L L=$ ? $L L$
proof (rule eq-matI)
fix $i a j$
show $i a<$ dim-row $\left(0_{m} m(\right.$ Suc $\left.i)\right) \Longrightarrow j<\operatorname{dim-col}\left(0_{m} m(\right.$ Suc $\left.i)\right)$
$\Longrightarrow L L \$ \$(i a, j)=0_{m} m($ Suc $i) \$ \$(i a, j)$
using ut[of ?si $+i a j$, unfolded Ablock] nsi carr by auto
qed (insert carr, auto)
have utUL: inv-all' uppert ULunfolding inv-all'-def uppert-def proof (intro allI impI) fix $i j$
show $i<$ dim-row $U L \Longrightarrow j<$ dim-row $U L \Longrightarrow j<i \Longrightarrow U L \$ \$(i, j)$
$=0$
using ut[of ij, unfolded Ablock] using nsi carr by auto
qed
have diffUL: inv-all' diff-ev ULunfolding inv-all'-def diff-ev-def
proof (intro allI impI)
fix $i j$
show $i<$ dim-row $U L \Longrightarrow j<$ dim-row $U L \Longrightarrow i<j \Longrightarrow U L \$ \$(i, i)$
$\neq U L \$ \$(j, j) \Longrightarrow U L \$ \$(i, j)=0$
using diff[of $i j$, unfolded Ablock] using nsi carr by auto
qed
have evbUL: ev-blocks-part ?si ULunfolding ev-blocks-part-def proof (intro allI impI)
fix $i a j k$
show $i a<j \Longrightarrow j<k \Longrightarrow k<S u c i \Longrightarrow U L \$ \$(k, k)=U L \$ \$(i a$, $i a) \Longrightarrow U L \$ \$(j, j)=U L \$ \$(i a, i a)$
using less(4)[unfolded Ablock ev-blocks-part-def, rule-format, of ia $j k]$
using nsi carr by auto

## qed

have utLR: inv-all' uppert $L R$ unfolding inv-all'-def uppert-def
proof (intro allI impI)
fix $i j$
show $i<$ dim-row $L R \Longrightarrow j<$ dim-row $L R \Longrightarrow j<i \Longrightarrow L R \$ \$(i, j)$
$=0$
using ut $[o f$ ?si $+i$ ?si $+j$, unfolded Ablock $]$ nsi carr by auto
qed
have evbLR: ev-block (dim-row LR) LR unfolding ev-block-def
proof (intro allI impI)
fix $i j$
show $i<$ dim-row $L R \Longrightarrow j<$ dim-row $L R \Longrightarrow L R \$ \$(i, i)=L R \$ \$$
$(j, j)$
using between $[$ of ? si $+i$ ] between $[$ of ? si $+j$ ] carr nsi
unfolding Ablock by auto (metis One-nat-def Suc-lessI diff-Suc-1)
qed
from False Some split have bs: partition-ev-blocks $U L(L R \# c s)=b s$ unfolding bs by auto
have IH: diag-block-mat ( $U L \# L R \# c s$ ) $=$ diag-block-mat bs $\wedge(\forall B \in$ set bs. inv-all' uppert $B \wedge$ ev-block (dim-col $B) B \wedge$ dim-row $B=$ dim-col $B$ )
by (rule less(1)[OF si utUL diffUL evbUL carr(1) - bs], insert dimLR evbLR utLR cs, auto)
have diag-block-mat $(A \# c s)=$ diag-block-mat $(U L \# L R \# c s)$
unfolding diag-block-mat.simps $\operatorname{dim} C$-def[symmetric] $\operatorname{dimC} \operatorname{dimLR}$
$\operatorname{dimUL}$ Let-def
index-mat-four-block(2-3) Ablock UR LL
using assoc-four-block-mat[of UL LR C] dimC carr by simp
with $I H$ show ?thesis by auto
qed
qed
qed
\}
from this [of Nil, OF - part] show $A=$ diag-block-mat bs $\wedge B . B \in$ set $b s \Longrightarrow$ inv-all' uppert $B \wedge$ ev-block (dim-col $B) B \wedge$ dim-row $B=$ dim-col $B$
unfolding diag by fastforce+
qed
lemma uppert-to-jb: assumes $u t$ : inv-all uppert $A$ and $A \in$ carrier-mat $n n$
shows inv-upto jb A 1
proof (rule inv-uptoI)
fix $i j$
assume $i<n j<n$ and $j<1$

```
    hence j:j=0 and jn: 0<n by auto
    show jb A i j unfolding jb-def j using inv-all-uppertD[OF ut - <i<n>, of 0]
    by auto
qed
lemma jnf-vector: assumes A:A\incarrier-mat n n
    and jb: \bigwedgeij.i<n\Longrightarrowj<n\LongrightarrowjbAij
    and evb: ev-block n A
shows jordan-matrix (jnf-vector A) = (A :: 'a mat)
    0 &fst'set (jnf-vector A)
proof -
    from }A\mathrm{ have dim-row }A=n\mathrm{ by simp
    hence id: jnf-vector A = jnf-vector-main n A unfolding jnf-vector-def by auto
    let ?map = map ( }\lambda(n,a)\mathrm{ . jordan-block n (a ::' 'a))
    let ?B = \lambdak. diag-block-mat (?map (jnf-vector-main k A))
    {
        fix }
        assume k\leqn
        hence (\forallij. i<k\longrightarrowj<k\longrightarrow?B k$$ (i,j)=A $$ (i,j))
            ^diag-block-mat (?map (jnf-vector-main k A)) \in carrier-mat k k
            ^0 &fst'set (jnf-vector-main k A)
            proof (induct k rule: less-induct)
            case (less sk)
            show ?case
            proof (cases sk)
                case (Suc k)
                            obtain b}\mathrm{ where ib: identify-block A k=b by force
                            let ?ev = A $$ (b,b)
                            from ib have id: jnf-vector-main sk A=jnf-vector-main b A @ [(Suc k-
b, ?ev)] unfolding Suc by simp
            let ?c = Suc k-b
            define }B\mathrm{ where }B=?,B
            define C where C=jordan-block ?c ?ev
                    have C:C\incarrier-mat ?c ?c unfolding C-def by auto
            let ?FB=\lambda Bb Cc.four-block-mat Bb (0m (dim-row Bb)(dim-col Cc)) (0
(dim-row Cc) (dim-col Bb)) Cc
            from identify-block-le'[OF ib] have bk: b\leqk.
            with Suc less(2) have b<sk b\leqn by auto
            note IH = less(1)[OF this, folded B-def]
            have B: B\incarrier-mat b b using IH by simp
            from bk Suc have sk:sk=b+?c by auto
            show ?thesis unfolding id map-append list.simps diag-block-mat-last split
B-def[symmetric] C-def[symmetric] Let-def
            proof (intro allI conjI impI)
            show ?FB B C \in carrier-mat sk sk unfolding sk using four-block-carrier-mat[OF
B C] .
            fix ij
            assume i:i<sk and j:j<sk
            with jb <sk \leq n`
```

have $j b$ : $j b A i j$ by auto
have ut: uppert $A i j$ by (rule jb-imp-uppert $[O F j b]$ )
have de: diff-ev A ij by (rule jb-imp-diff-ev[OF jb])
from $B C$ have dim: dim-row $B=b$ dim-col $B=b$ dim-col $C=$ ? $c$ dim-row $C=? c$ by auto
from sk $B C i j$ have $i<$ dim-row $B+$ dim-row $C j<\operatorname{dim}$-col $B+$ dim-col $C$ by auto
note $i d=$ index-mat-four-block(1)[OF this, unfolded dim]
have $i d$ : ? $F B B C \$ \$(i, j)=$
(if $i<b$ then if $j<b$ then $B \$ \$(i, j)$ else 0
else if $j<b$ then 0 else $C \$ \$(i-b, j-b))$
unfolding id dim using $i j s k$ by auto
show ? FB B C $\$ \$(i, j)=A \$ \$(i, j)$
proof (cases $i<b \wedge j<b$ )
case True
hence ? $F B B C \$ \$(i, j)=B \$ \$(i, j)$ unfolding $i d$ by auto
with IH True show ?thesis by auto
next
case False note not-ul $=$ this
note $i b=$ identify-block[OF $i b]$
show ?thesis
proof (cases $\neg i<b \wedge j<b \vee i<b \wedge \neg j<b$ )
case True
hence $i d$ : ? $F B B C \$ \$(i, j)=0$ unfolding $i d$ by auto
show ?thesis
proof (cases $j<i$ )
case True
with ut show ?thesis unfolding id uppert-def by auto
next
case False
with True have $*: j \geq b i<b j>i$ by auto
have $A \$ \$(i, j)=0$
proof (rule ccontr)
assume $A \$ \$(i, j) \neq 0$
with $j b[$ unfolded $j b-$ def] *
have $j i: j=b i=b-1 b>0$ and no-border: $A \$ \$(i, i)=A \$ \$$
$(j, j) A \$ \$(i, j)=1$ by auto
from no-border[unfolded $j i] i b(2)\langle b>0\rangle$ show False by auto
qed
thus ?thesis unfolding id by simp
qed
next
case False
with not-ul have $*: \neg i<b \neg j<b$ by auto
hence $i d$ : ? $F B B C \$ \$(i, j)=C \$ \$(i-b, j-b)$ unfolding $i d$ by
auto
from $* i j$ have $i j c: i-b<? c j-b<? c$ unfolding $s k$ by auto have $i d$ : ?FB B $C \$ \$(i, j)=($ if $i-b=j-b$ then ?ev else if Suc $(i$

```
- b) = j - b then 1 else 0)
            unfolding id unfolding C-def jordan-block-index(1)[OF ijc] ..
                show ?thesis
                proof (cases i-b=j-b)
                case True
                hence id: ?FB B C $$ (i,j) = ?ev unfolding id by simp
                from True * have ij:j=i by auto
                have i-n:i<n using i<sk\leqn\rangle by auto
                have b-n: b<n using <b<sk\rangle\langlesk\leqn> by auto
                from ib(3)[of i] True * ij Suc ev-blockD[OF evb i-n b-n] have A $$
(i,j) = ?ev unfolding ij by auto
                    with id show ?thesis by simp
                    next
                    case False note neq = this
                        show ?thesis
                        proof (cases j - b = Suc (i-b))
                        case True
                    hence id: ?FB B C $$ (i,j)=1 unfolding id by simp
                    from True * have ij: j=Suc i by auto
                    from ib(3)[of i] True * ij Suc have A $$ (i,j)=1 unfolding ij
by auto
                    with id show ?thesis by simp
                    next
                    case False
                    with neq have id:?FB B C $$ (i,j)=0 unfolding id by simp
                    from * neq False have i\not=jSuc i\not=j by auto
                    with jb[unfolded jb-def] have }A$$(i,j)=0 by aut
                    with id show ?thesis by simp
                    qed
                qed
                qed
            qed
            qed (insert bk IH, auto)
        qed auto
    qed
    }
    from this[OF le-refl] A
    show jordan-matrix (jnf-vector A) = A 0 &fst'set (jnf-vector A)
        unfolding id jordan-matrix-def by auto
qed
end
lemma triangular-to-jnf-vector:
    assumes A:A\incarrier-mat n n
    and upper-t: upper-triangular A
    shows jordan-nf A (triangular-to-jnf-vector A)
proof -
```

from $A$ have $d$ : dim-row $A=n$ by simp
let $? B=$ step-2 $($ step $-1 A)$
let ? $J=$ triangular-to-jnf-vector $A$
have A1: step-1 $A \in$ carrier-mat $n n$ using $A$ unfolding carrier-mat-def by simp
from similar-mat-trans[OF step-2-similar step-1-similar, OF A1 A]
have sim: similar-mat ? $B A$.
have A1: step-1 $A \in$ carrier-mat $n n$ using $A$ unfolding carrier-mat-def by auto
from $A 1$ have d1: dim-row (step-1 $A$ ) $=n$ unfolding carrier-mat-def by simp
have $B: ? B \in$ carrier-mat $n n$ using $A$ unfolding carrier-mat-def by auto
from $B$ have $d 2$ : dim-row ? $B=n$ unfolding carrier-mat-def by simp
define $C s$ where $C s=$ partition-ev-blocks ? $B$ []
from step-1-2-inv[OF A upper-t refl]
have inv: inv-all $n$ uppert ?B inv-all $n$ diff-ev ?B ev-blocks $n$ ? $B$ by auto
from partition-jb[OF B inv, of Cs] have BC: ?B = diag-block-mat Cs
and $C s: \wedge C . C \in$ set $C s \Longrightarrow$ inv-all' uppert $C \wedge$ ev-block (dim-col $C$ ) $C \wedge$
dim-row $C=$ dim-col $C$ unfolding $C s$-def by auto
define $D$ where $D=$ map step-3 Cs
let $? D=$ diag-block-mat $D$
let ? $C D=\operatorname{map}(\lambda C .(C,($ jnf-vector o step-3) $C)) C s$
\{
fix $C D$
assume mem: $(C, D) \in$ set ? $C D$
hence $D C: D=j n f$-vector (step-3 $C$ ) and $C: C \in$ set $C s$ by auto
let ? $D=$ step-3 $C$
define $n$ where $n=d i m$-col $C$
from $C s[O F \quad C]$ have $C$ : inv-all $n$ uppert $C$ ev-block n $C C \in$ carrier-mat $n n$ unfolding inv-all'-def inv-all-def $n$-def carrier-mat-def by auto
from step-3-similar $[O F C(3)]$ have sim: similar-mat $C$ ? $D$ by (rule simi-lar-mat-sym)
from similar-matD[OF sim] $C$ have $D: ? D \in$ carrier-mat $n n$ unfolding carrier-mat-def by auto
from $C(3)$ have $\operatorname{dim} C$ : dim-row $C=n$ by auto
from step-3-main-inv[OF C(3)-C(1,2) uppert-to-jb[OF C(1) C(3)]]
have inv-all $n j b$ (step-3 $C$ ) and sd: same-diag $n C$ (step-3 $C$ ) unfolding step-3-def $\operatorname{dimC}$ by auto
hence $j b D: ~ \bigwedge i j . i<n \Longrightarrow j<n \Longrightarrow j b$ ? $D i j$ unfolding inv-all-def $D C$ by auto
from same-diag-ev-block[OF sd C(2)] have ev-block $n$ (step-3 C) by auto
from jnf-vector $[O F D j b D$ this] have jordan-matrix $D=? D 0 \notin f s t$ 'set $D$ unfolding $D C$ by auto
with sim have jordan-nf C D unfolding jordan-nf-def by simp
\} note jnf-blocks $=$ this
have id: map fst ? $C D=C s$ by (induct Cs, auto)
have id2: map snd ?CD = map (jnf-vector o step-3) Cs by (induct Cs, auto)
have $J:$ ? $J=$ concat (map (jnf-vector $\circ$ step-3) Cs) unfolding
triangular-to-jnf-vector-def Let-def Cs-def ..
from jordan-nf-diag-block-mat[of ?CD, OF jnf-blocks, unfolded id id2]

```
    have jnf: jordan-nf (diag-block-mat Cs) ?J unfolding J .
    hence similar-mat (diag-block-mat Cs) (jordan-matrix ?J)
    unfolding jordan-nf-def by auto
from similar-mat-sym[OF similar-mat-trans[OF similar-mat-sym[OF this] sim[unfolded
BC]]] jnf
    show ?thesis unfolding jordan-nf-def by auto
qed
```


## hide-const

lookup-ev
find-largest-block
swap-cols-rows-block
identify-block
identify-blocks-main
identify-blocks
inv-all inv-all' same-diag
jb uppert diff-ev ev-blocks ev-block
step-1-main step-1
step-2-main step-2
step-3-a step-3-c step-3-c-inner-loop step-3
jnf-vector-main

### 18.9 Combination with Schur-decomposition

definition jordan-nf-via-factored-charpoly :: ' $a$ :: conjugatable-ordered-field mat $\Rightarrow$ 'a list $\Rightarrow($ nat $\times$ 'a) list
where jordan-nf-via-factored-charpoly $A$ es $=$
triangular-to-jnf-vector (schur-upper-triangular A es)
lemma jordan-nf-via-factored-charpoly: assumes $A: A \in$ carrier-mat $n n$ and linear: char-poly $A=\left(\prod a \leftarrow e s .[:-a, 1:]\right)$
shows jordan-nf A (jordan-nf-via-factored-charpoly A es)
proof -
let $? B=$ schur-upper-triangular $A$ es
let ? $J=j o r d a n-n f$-via-factored-charpoly $A$ es
from schur-upper-triangular[OF A linear]
have $B: ? B \in$ carrier-mat $n$ n upper-triangular ? $B$ and $A B$ : similar-mat $A$ ? $B$ by auto
from triangular-to-jnf-vector $[O F B]$ have jordan-nf ?B ?J
unfolding jordan-nf-via-factored-charpoly-def .
with similar-mat-trans $[O F A B]$ show jordan-nf $A$ ?J unfolding jordan-nf-def
by blast
qed
lemma jordan-nf-exists: assumes $A: A \in$ carrier-mat $n n$ and linear: char-poly $A=\left(\prod(a:: ' a\right.$ :: conjugatable-ordered-field $) \leftarrow a s .[:-a$, 1:])

```
    shows \existsn-as. jordan-nf A n-as
    using jordan-nf-via-factored-charpoly[OF A linear] by blast
    lemma jordan-nf-iff-linear-factorization: fixes }A\mathrm{ :: 'a :: conjugatable-ordered-field
mat
    assumes A: A \in carrier-mat n n
    shows (\exists n-as. jordan-nf A n-as)}=(\exists\mathrm{ as. char-poly }A=(\Pia\leftarrowas. [:- a,
1:]))
    (is ?l = ?r)
proof
    assume ?r
    thus ?l using jordan-nf-exists[OF A] by auto
next
    assume ?l
    then obtain n-as where jnf: jordan-nf A n-as by auto
    show ?r unfolding jordan-nf-char-poly[OF jnf] expand-powers[of \lambda a. [:-a, 1:]
n-as] by blast
qed
lemma similar-iff-same-jordan-nf: fixes A :: complex mat
    assumes A:A\incarrier-mat n n and B:B\incarrier-mat n n
    shows similar-mat A B = (jordan-nf A = jordan-nf B)
proof
    show similar-mat }AB\Longrightarrow\mathrm{ jordan-nf }A=\mathrm{ jordan-nf }
    by (intro ext, auto simp: jordan-nf-def, insert similar-mat-trans similar-mat-sym,
blast+)
    assume id: jordan-nf A = jordan-nf B
    from char-poly-factorized[OF A] obtain as where char-poly A = (\proda\leftarrowas. [:-
a, 1:]) by auto
    from jordan-nf-exists[OF A this] obtain n-as where jnfA: jordan-nf A n-as ..
    with id have jnfB: jordan-nf B n-as by simp
    from jnfA jnfB show similar-mat A B
        unfolding jordan-nf-def using similar-mat-trans similar-mat-sym by blast
qed
lemma order-char-poly-smult: fixes A :: complex mat
    assumes A:A\incarrier-mat n n
    and k: k\not=0
shows order x (char-poly (k\cdotm A)) = order (x/k) (char-poly A)
proof -
    from char-poly-factorized[OF A] obtain as where char-poly A = (\proda\leftarrowas. [:-
a,1:]) by auto
    from jordan-nf-exists[OF A this] obtain n-as where jnf: jordan-nf A n-as ..
    show ?thesis unfolding jordan-nf-order[OF jnf] jordan-nf-order[OF jordan-nf-smult[OF
jnf k]]
    by (induct n-as, auto simp: k)
qed
```


### 18.10 Application for Complexity

We can estimate the complexity via the multiplicity of the eigenvalues with norm 1.
lemma factored-char-poly-norm-bound-cof: assumes $A: A \in$ carrier-mat $n n$
and linear-factors: char-poly $A=\left(\Pi \quad\left(a::{ }^{\prime} a\right.\right.$ :: \{conjugatable-ordered-field, real-normed-field $\}) \leftarrow a s$. [:-a, 1:])
and $l e-1: \wedge a . a \in$ set as $\Longrightarrow$ norm $a \leq 1$
and $l e-N: \bigwedge a . a \in$ set $a s \Longrightarrow$ norm $a=1 \Longrightarrow$ length (filter $((=) a) a s) \leq N$ shows $\exists c 1 c 2 . \forall k$. norm-bound $\left(A{ }_{m} k\right)(c 1+c 2 *$ of-nat $k \wedge(N-1))$ by (rule factored-char-poly-norm-bound[OF A linear-factors jordan-nf-exists[OF A linear-factors] le-1 le-N])

If we have an upper triangular matrix, then EVs are exactly the entries on the diagonal. So then we don't need to explicitly compute the characteristic polynomial.

```
lemma counting-ones-complexity:
    fixes \(A\) :: ' \(a\) :: real-normed-field mat
    assumes \(A: A \in\) carrier-mat \(n n\)
    and upper-t: upper-triangular \(A\)
    and le-1: \(\bigwedge a . a \in \operatorname{set}(\operatorname{diag}-m a t A) \Longrightarrow\) norm \(a \leq 1\)
    and le-N: \(\bigwedge a . a \in \operatorname{set}(\) diag-mat \(A) \Longrightarrow\) norm \(a=1 \Longrightarrow\) length (filter \(((=) a)\)
\((\operatorname{diag}-m a t A)) \leq N\)
    shows \(\exists c 1 c 2 . \forall k\). norm-bound \((A \widehat{m} k)\left(c 1+c 2 *\right.\) of-nat \(\left.k{ }^{\wedge}(N-1)\right)\)
proof -
    from triangular-to-jnf-vector \([O F A\) upper-t \(]\) have jnf: \(\exists n\)-as. jordan-nf \(A\) n-as
    show ?thesis
            by (rule factored-char-poly-norm-bound[OF A char-poly-upper-triangular[OF A
upper-t] jnf le-1 le-N])
qed
```

If we have an upper triangular matrix $A$ then we can compute a JNFvector of it. If this vector does not contain entries ( $n, e v$ ) with $e v$ being larger 1, then the growth rate of $A^{k}$ can be restricted by $\mathcal{O}\left(k^{N-1}\right)$ where $N$ is the maximal value for $n$, where $(n,|e v|=1)$ occurs in the vector, i.e., the size of the largest Jordan Block with Eigenvalue of norm 1. This method gives a precise complexity bound.

```
lemma compute-jnf-complexity:
    assumes \(A: A \in\) carrier-mat \(n n\)
    and upper-t: upper-triangular ( \(A\) :: 'a :: real-normed-field mat)
    and \(l e-1: \bigwedge n a .(n, a) \in\) set (triangular-to-jnf-vector \(A) \Longrightarrow\) norm \(a \leq 1\)
    and \(l e-N: \wedge n a .(n, a) \in \operatorname{set}(\) triangular-to-jnf-vector \(A) \Longrightarrow\) norm \(a=1 \Longrightarrow\)
\(n \leq N\)
    shows \(\exists c 1 c \mathcal{2} . \forall k\). norm-bound \(\left(A{ }_{m} k\right)\left(c 1+c \mathcal{L} *\right.\) of-nat \(\left.k^{\wedge}(N-1)\right)\)
proof -
    let \(? j n f=\) triangular-to-jnf-vector \(A\)
    from triangular-to-jnf-vector[OF A upper-t \(]\) have \(j n f: ~ j o r d a n-n f ~ A ? j n f\).
```

```
    show ?thesis
    by (rule jordan-nf-matrix-poly-bound[OF A le-1 le-N jnf])
qed
end
```


## 19 Code Equations for All Algorithms

In this theory we load all executable algorithms, i.e., Gauss-Jordan, determinants, Jordan normal form computation, etc., and perform some basic tests.

```
theory Matrix-Impl
imports
    Matrix-IArray-Impl
    Gauss-Jordan-IArray-Impl
    Determinant-Impl
    Show-Matrix
    Jordan-Normal-Form-Existence
    Show.Show-Instances
begin
```

For determinants we require class idom-divide, so integers, rationals, etc. can be used.
value $[$ code $] \operatorname{det}$ (mat-of-rows-list $4[[1::$ int, 4, $9,-1],[-3,-1,5,4]$, [4, 2,
0,2], [8,-9, 5,7]])
value[code] det (mat-of-rows-list 4 [ $[1$ :: rat, 4, 9, -1$],[-3,-1,5,4]$, [4, 2,
0,2], $[8,-9,5,7]]$ )

Since polynomials require field elements to be in class idom-divide, the implementation of characteristic polynomials is not applicable for integer matrices, but it is for rational and real matrices.
value[code] char-poly (mat-of-rows-list 4 [ $[1$ :: real, 4, $9,-1],[-3,-1,5,4]$, [4, 2, 0,2$],[8,-9,5,7]])$

Also Jordan normal form computation requires matrices over field entries.
value[code] triangular-to-jnf-vector (mat-of-rows-list 6 [

$$
\begin{aligned}
& {[3,4,1,4,7,18],} \\
& {[0,3,0,8,9,4],} \\
& {[0,0,3,2,0,4],} \\
& {[0,0,0,5,17,7],} \\
& {[0,0,0,0,5,3],} \\
& [0,0,0,0,0,3:: \text { rat }]])
\end{aligned}
$$

value[code] show (mat-of-rows-list $3[[1,4,5],[3,6,8]] *$ mat $34(\lambda(i, j) . i+$ $2 * j$ )

Inverses can only be computed for matrices over fields.
value[code] show (mat-inverse (mat-of-rows-list 4 [ $[1$ :: rat, $4,9,-1],[-3,-1$, $5,4],[4,2,0,2],[8,-9,5,7]]))$
value $[$ code $]$ show (mat-inverse (mat-of-rows-list $4[[1$ :: rat, 4, 9, -1$],[-3,-1$, $5,4],[-2,3,14,3],[8,-9,5,7]]))$
end

## 20 Strassen's algorithm for matrix multiplication.

We define the algorithm for arbitrary matrices over rings, where an alignment of the dimensions to even numbers will be performed throughout the algorithm.

```
theory Strassen-Algorithm
imports
    Matrix
begin
```

With four-block-mat and split-block we can define Strassen's multiplication algorithm.

We start with a simple heuristic on when to switch to the basic algorithm.

```
definition strassen-constant :: nat where
```

    [code-unfold]: strassen-constant \(=20\)
    definition strassen-too-small $A B \equiv$
dim-row $A<$ strassen-constant $\vee$
dim-col $A<$ strassen-constant $\vee$
dim-col $B<$ strassen-constant

We have to make a case analysis on whether all dimensions are even.
definition strassen-even $A B \equiv$ even $($ dim-row $A) \wedge$ even $($ dim-col $A) \wedge$ even (dim-col B)

And then we can define the algorithm.

```
function strassen-mat-mult :: 'a :: ring mat }=>\mp@subsup{}{}{\prime}'a mat => ' 'a mat where
    strassen-mat-mult A B = (let nr = dim-row A; n= dim-col A;nc=dim-col B
in
    if strassen-too-small }A\mathrm{ B then A*B else
    if strassen-even A B then let
        nr2 = nr div 2;
        n2 = n div 2;
        nc2 = nc div 2;
        (A1,A2,A3,A4) = split-block A nr2 n2;
        (B1,B2,B3,B4) = split-block B n2 nc2;
        M1 = strassen-mat-mult (A1 + A4) (B1 + B4);
        M2 = strassen-mat-mult (A3 + A4) B1;
        M3 = strassen-mat-mult A1 (B2 - B4);
```

For termination, we use the following measure.
definition strassen-measure $\equiv \lambda(A, B)$. (dim-row $A+$ dim-col $A+$ dim-col $B)$
$+($ dim-row $A+\operatorname{dim}-c o l A+d i m-c o l B)+($ if strassen-even $A B$ then 0 else 1$)$
lemma strassen-measure-add[simp]:
strassen-measure $(A+B, C)=$ strassen-measure $(B, C)$
strassen-measure $(A, B+C)=$ strassen-measure $(A, C)$
strassen-measure $(A-B, C)=$ strassen-measure $(B, C)$
strassen-measure $(A, B-C)=$ strassen-measure $(A, C)$
strassen-measure $(-A, B)=$ strassen-measure $(A, B)$
strassen-measure $(A,-B)=$ strassen-measure $(A, B)$
unfolding strassen-measure-def strassen-even-def by auto
lemma strassen-measure-div-2: assumes $(A 1, A 2, A 3, ~ A 4)=$ split-block $A($ dim-row A div 2) (dim-col $A$ div 2)
$($ B1, B2, B3, B4) $=$ split-block $B($ dim-col $A$ div 2) $($ dim-col $B$ div 2)
and large: $\neg$ strassen-too-small $A B$
shows
strassen-measure $(A 1, B 4)<$ strassen-measure $(A, B)$
strassen-measure $(A 1, B 2)<$ strassen-measure $(A, B)$
strassen-measure $(A 2, B 4)<$ strassen-measure $(A, B)$
strassen-measure $(A 3, B 2)<$ strassen-measure $(A, B)$
strassen-measure $(A 4, B 1)<$ strassen-measure $(A, B)$
strassen-measure $(A 4, B 3)<$ strassen-measure $(A, B)$
strassen-measure $(A 4, B 4)<$ strassen-measure $(A, B)$

## proof -

```
    fix Ai Bi
    assume Ai:Ai\in{A1,A2,A3,A4} and Bi: Bi }{{B1,B2,B3,B4
    from large[unfolded strassen-too-small-def strassen-constant-def]
    have ᄀ dim-row A<2 by auto
    with assms Ai Bi have Ar:
        dim-row Ai < dim-row A
        dim-col Ai\leqdim-col A
        dim-col Bi\leqdim-col B
        unfolding split-block-def Let-def by auto
    hence strassen-measure (Ai,Bi)< strassen-measure (A,B)
        unfolding strassen-measure-def split by auto
    }
    thus
    strassen-measure (A1,B2)< strassen-measure (A,B)
    strassen-measure (A1,B4)< strassen-measure (A,B)
    strassen-measure (A2,B4)< strassen-measure (A,B)
    strassen-measure (A3,B2) < strassen-measure (A,B)
    strassen-measure (A4,B1)< strassen-measure (A,B)
    strassen-measure (A4,B3)< strassen-measure (A,B)
    strassen-measure (A4,B4)< strassen-measure (A,B)
    by auto
qed
lemma strassen-measure-odd: assumes \((\) A1, A2, \(A 3, ~ A 4)=\) split-block \(A((\) dim-row \(A \operatorname{div} 2) *\) 2) \(((\operatorname{dim}-c o l A \operatorname{div} 2) * 2)\)
    and (B1, B2, B3, B4) = split-block B ((dim-col A div 2) * 2) ((dim-col B div
2) * 2)
    and odd: ᄀ strassen-even A B
    shows strassen-measure (A1,B1) < strassen-measure (A,B)
proof -
    from assms have Ar:
        dim-row A1 < dim-row A v dim-row A1 = dim-row A ^even (dim-row A)
        unfolding split-block-def Let-def by auto presburger
    from assms have Ac:
        dim-col A1 < dim-col A\veedim-col A1 = dim-col A ^even (dim-col A)
    unfolding split-block-def Let-def by auto presburger
    from assms have Bc:
        dim-col B1 < dim-col B \vee dim-col B1 = dim-col B ^ even (dim-col B)
        unfolding split-block-def Let-def by auto presburger
    from Ar Ac Bc odd show ?thesis unfolding strassen-measure-def strassen-even-def
split
    by (auto split: if-splits)
qed
termination by (relation measure strassen-measure,
auto elim: strassen-measure-div-2 strassen-measure-odd)
```

lemma strassen-mat-mult:

```
    dim-col }A=\mathrm{ dim-row }B\Longrightarrow\mathrm{ strassen-mat-mult }AB=A*
proof (induct A B rule: strassen-mat-mult.induct)
    case (1 A B)
    let ?nr = dim-row A
    let ?nc = dim-col B
    let ?n = dim-col A
    show ?case
    proof (cases strassen-too-small A B)
    case False note large = this
    let ?smm = strassen-mat-mult
    note IH = 1(1-8)[OF refl refl refl False - refl refl refl - refl refl refl - refl refl
refl]
    show ?thesis
    proof (cases strassen-even A B)
        case True
        note even = True[unfolded strassen-even-def]
        let ?nr2 = ?nr div 2
        let ? n2 = ? n div 2
        let ?nc2 = ?nc div 2
        from even have nr:?nr = ?nr2 + ?nr2 by presburger
        from even have n:?n =?n2 + ?n2 by presburger
        from even have nc: ?nc = ?nc2 + ?nc2 by presburger
        from 1(9) even have n': dim-row B =?n2 + ?n2
            by auto
        obtain A1 A2 A3 A4 where splitA:
        split-block A ?nr2 ?n2 = (A1,A2,A3,A4) by (rule prod-cases4)
        obtain B1 B2 B3 B4 where splitB:
        split-block B ?n2 ?nc2 = (B1,B2,B3,B4) by (rule prod-cases4)
    note IH =IH(1-7)[OF True splitA[symmetric] splitB[symmetric]]
    from split-block[OF splitA nr n]
    have blockA: A = four-block-mat A1 A2 A3 A4
        and A1: A1 \in carrier-mat ?nr2 ?n2
        and A2: A2 \in carrier-mat ?nr2 ?n2
        and A3:A3 \in carrier-mat ?nr2 ?n2
        and A4:A4 \in carrier-mat ?nr2 ?n2
        by blast+
    from split-block[OF splitB n' nc]
    have blockB: B = four-block-mat B1 B2 B3 B4
        and B1: B1 \in carrier-mat ?n2 ?nc2
        and B2: B2 \in carrier-mat ?n2 ?nc2
        and B3:B3 G carrier-mat ?n2 ?nc2
        and B4: B4 G carrier-mat ?n2 ?nc2
        by blast+
    note carr = A1 A2 A3 A4 B1 B2 B3 B4
    let ?M11 = A1 + A4 let ?M12 = B1 + B4
    let ?M21 = A3 + A4 let ?M22 = B1
    let ?M31 = A1 let ?M32 = B2 - B4
    let ?M41 = A4 let ?M42 = B3 - B1
    let ?M51 = A1 + A2 let ?M52 = B4
```

```
let ?M61 = A3 - A1 let ?M62 = B1 + B2
let ?M71 = A2 - A4 let ?M72 = B3 + B4
let ?M1 = ?smm ?M11 ?M12
let ?M2 = ?smm ?M21 ?M22
let ?M3 = ?smm ?M31 ?M32
let ?M4 = ?smm ?M41 ?M42
let ?M5 = ?smm ?M51 ?M52
let ?M6 = ?smm ?M61 ?M62
let ?M7 = ?smm ?M71 ?M72
let ?C1 =?M1 + ?M4 - ?M5 + ?M7
let ?C2 = ?M3 + ?M5
let ?C3 = ?M2 + ?M4
let ?C4 = ?M1 - ?M2 + ?M3 + ?M6
have res: ?smm A B = four-block-mat ?C1 ?C2 ?C3 ?C4
    using large True
    unfolding strassen-mat-mult.simps[of A B] Let-def splitA splitB split
    by auto
have M1:?M1 = ?M11 * ?M12
    by (rule IH(1), insert carr, auto)
note IH=IH(2-)[OF refl]
have M2:?M2 = ?M21 * ?M22
    by (rule IH(1), insert carr, auto)
note IH = IH(2-)[OF refl]
have M3: ?M3 = ?M31 * ?M32
    by (rule IH(1), insert carr, auto)
note IH=IH(2-)[OF refl]
have M4:?M4 = ?M41 * ?M42
    by (rule IH(1), insert carr, auto)
note IH=IH(2-)[OF refl]
have M5: ?M5 = ?M51 * ?M52
    by (rule IH(1), insert carr, auto)
note IH = IH(2-)[OF refl]
have M6: ?M6 = ?M61 * ?M62
    by (rule IH(1), insert carr, auto)
note IH=IH(2-)[OF refl]
have M7: ?M7 = ?M71 * ?M72
    by (rule IH(1), insert carr, auto)
note distr =
    add-mult-distrib-mat[of - ?nr2 ?n2 - - ?nc2]
    minus-mult-distrib-mat[of - ?nr2 ?n2 - - ?nc2]
    mult-add-distrib-mat[of - ?nr2 ?n2 - ?nc2]
    mult-minus-distrib-mat[of - ?nr2 ?n2 - ?nc2]
note closed = add-carrier-mat[of - ?nr2 ?nc2]
    uminus-carrier-iff-mat[of - ?nr2 ?nc2]
note ac=assoc-add-mat[of - ?nr2 ?nc2] comm-add-mat[of - ?nr2 ?nc2]
show ?thesis unfolding res M1 M2 M3 M4 M5 M6 M7
    unfolding blockA blockB
        mult-four-block-mat[OF carr]
    by (rule cong-four-block-mat)
```

```
            (insert carr, auto simp: distr ac closed)
    next
        case False
        let ?nr2 = ?nr div 2 * 2 let ?nr2' = ?nr - ?nr2
    let ?n2 = ?n div 2 * 2 let ?n2' = ?n - ?n2
    let ?nc2 = ?nc div 2 * 2 let ?nc2' = ?nc - ?nc2
    have nr: ?nr = ?nr2 + ?nr2' by presburger
    have n:?n = ?n2 + ?n2' by presburger
    have nc: ?nc=? nc2 + ?ncQ' by presburger
    from 1(9) have n': dim-row B = ?n2 + ?n2' by auto
    obtain A1 A2 A3 A4 where splitA:
        split-block A ?nr2 ?n2 = (A1,A2,A3,A4) by (rule prod-cases4)
    obtain B1 B2 B3 B4 where splitB:
        split-block B ?n2 ?nc2 = (B1,B2,B3,B4) by (rule prod-cases4)
    note IH = IH(8)[OF False splitA[symmetric] splitB[symmetric]]
    from split-block[OF splitA nr n]
    have blockA: A = four-block-mat A1 A2 A3 A4
        and A1:A1 \in carrier-mat ?nr2 ?n2
        and A2: A2 \in carrier-mat ?nr2 ?n2'
        and A3:A3 \in carrier-mat ?nr2' ?n2
        and A4:A4 \in carrier-mat ?nr2' ?n2'
        by blast+
    from split-block[OF splitB n' nc]
    have blockB: B = four-block-mat B1 B2 B3 B4
        and B1: B1 \in carrier-mat ?n2 ?nc2
        and B2: B2 \in carrier-mat ?n2 ?nc2'
        and B3: B3 \in carrier-mat ?n2' ?nc2
        and B4: B4 \in carrier-mat ?n2' ?nc2'
        by blast+
    note carr = A1 A2 A3 A4 B1 B2 B3 B4
    from carr have dim-col A1 = dim-row B1 by simp
    note IH = IH[OF this]
    have ?smm A B = four-block-mat
        (A1*B1 + A2 * B3)
        (A1*B2+A2*B4)
        (A3*B1+A4*B3)
        (A3 * B2 + A4 * B4)
        unfolding strassen-mat-mult.simps[of A B] Let-def
            splitA splitB split IH using large False by auto
    also have ... = A*B
        unfolding blockA blockB
            mult-four-block-mat[OF carr] by simp
            finally show ?thesis by simp
        qed
    qed simp
qed
end
```


## 21 Strassen's Algorithm as Code Equation

We replace the code-equations for matrix-multiplication by Strassen's algorithm. Note that this will strengthen the class-constraint for matrix multiplication from semirings to rings!

```
theory Strassen-Algorithm-Code
imports
    Strassen-Algorithm
begin
```

The aim is to replace the implementation of ? $A * ? B \equiv$ mat (dim-row ?A) (dim-col ? $B)(\lambda(i, j)$. row ? $A$ i col ? $B j)$ by strassen-mat-mult.

We first need a copy of standard matrix multiplication to execute the base case.
definition basic-mat-mult $=(*)$
lemma basic-mat-mult-code[code]: basic-mat-mult A B=mat (dim-row A) (dim-col B) $(\lambda(i, j)$. row $A i \cdot \operatorname{col} B j)$
unfolding basic-mat-mult-def by auto
Next use this new matrix multiplication code within Strassen's algorithm.
lemmas strassen-mat-mult-code $[$ code $]=$ strassen-mat-mult.simps[folded basic-mat-mult-def]
And finally use Strassen's algorithm for implementing matrix-multiplication.
lemma mat-mult-code[code]: $A * B=$ (if dim-col $A=$ dim-row $B$ then strassen-mat-mult A B else basic-mat-mult $A B$ )
using strassen-mat-mult[of A B] unfolding basic-mat-mult-def by auto
end

## 22 Comparison of Matrices

We use matrices over ordered semirings to again define ordered semirings. There are two instances, one for ordinary semirings (where addition is monotone w.r.t. the strict ordering in a single argument); and one for semirings like the arctic one, where addition is interpreted as maximum, and therefore monotonicity of the strict ordering in a single argument is no longer provided.

Both ordered semirings are used for checking termination proofs, where at the moment only the ordinary semirings is supported for checking complexity proofs.

```
theory Matrix-Comparison
imports
    Matrix
    Matrix.Ordered-Semiring
```


## begin

```
context ord
begin
definition mat-ge :: 'a mat => 'a mat }=>\mathrm{ bool (infix }\mp@subsup{\geq}{m}{}50)\mathrm{ where
    A\geqm}B=(\foralli<dim-row A. \forallj<dim-col A. A $$ (i,j)\geqB $$ (i,j)
```

lemma mat-geI[intro]: assumes $A \in$ carrier-mat nr nc
$\bigwedge i j . i<n r \Longrightarrow j<n c \Longrightarrow A \$ \$(i, j) \geq B \$ \$(i, j)$
shows $A \geq_{m} B$
using assms unfolding mat-ge-def by auto
lemma mat-ge $D[$ dest $]$ : assumes $A \geq_{m} B$ and $i<\operatorname{dim}$-row $A j<\operatorname{dim}$-col $A$
shows $A \$ \$(i, j) \geq B \$ \$(i, j)$
using assms unfolding mat-ge-def by auto
definition mat-gt $::\left({ }^{\prime} a \Rightarrow{ }^{\prime} a \Rightarrow\right.$ bool $) \Rightarrow$ nat $\Rightarrow$ ' $a$ mat $\Rightarrow$ 'a mat $\Rightarrow$ bool where
mat-gt gt sd $A B=\left(A \geq_{m} B \wedge(\exists i<s d . \exists j<\operatorname{sd}\right.$.gt $\left.(A \$ \$(i, j))(B \$ \$(i, j)))\right)$
lemma mat-gtI[intro]: assumes $A \geq_{m} B$
and $i<s d j<s d g t(A \$ \$(i, j))(B \$ \$(i, j))$
shows mat-gt gt sd A B
using assms unfolding mat-gt-def by auto
lemma mat-gt $D[d e s t]$ : assumes mat-gt gt sd $A B$
shows $A \geq_{m} B \exists i<s d . \exists j<s d$.gt $(A \$ \$(i, j))(B \$ \$(i, j))$
using assms unfolding mat-gt-def by auto
definition mat-max :: 'a mat $\Rightarrow$ 'a mat $\Rightarrow$ 'a mat $\left(\max _{m}\right)$ where
$\max _{m} A B=\operatorname{mat}($ dim-row $A)($ dim-col $A)(\lambda i j . \max (A \$ \$ i j)(B \$ \$ i j))$
lemma mat-max-carrier[simp]:
$\max _{m} A B \in$ carrier-mat (dim-row $\left.A\right)(d i m-c o l ~ A)$
unfolding mat-max-def by auto
lemma mat-max-closed[intro]:
$A \in$ carrier-mat $n r n c \Longrightarrow B \in$ carrier-mat $n r n c \Longrightarrow \max _{m} A B \in$ carrier-mat
$n r n c$
unfolding mat-max-def by auto
lemma mat-max-index:
assumes $i<\operatorname{dim}$-row $A j<d i m-\operatorname{col} A$
shows $($ mat-max $A B) \$ \$(i, j)=\max (A \$ \$(i, j))(B \$ \$(i, j))$
unfolding mat-max-def using index-mat assms by auto
definition (in zero) mat-default :: ' $a \Rightarrow n a t \Rightarrow$ 'a mat (default ${ }_{m}$ ) where
default $_{m} d n=$ mat $n n(\lambda(i, j)$. if $i=j$ then $d$ else 0$)$
lemma mat-default-carrier[simp]:default ${ }_{m} d n \in$ carrier-mat $n n$
unfolding mat-default-def by auto end
definition mat-mono $::\left({ }^{\prime} a \Rightarrow\right.$ bool $) \Rightarrow$ nat $\Rightarrow{ }^{\prime}$ a mat $\Rightarrow$ bool
where mat-mono $P$ sd $A=(\forall j<s d . \exists i<s d$. $P(A \$ \$(i, j)))$
context non-strict-order
begin
lemma mat-ge-trans: assumes $A \geq_{m} B B \geq_{m} C$
and $A \in$ carrier-mat nr nc $B \in$ carrier-mat nr nc
shows $A \geq{ }_{m} C$
using assms ge-trans[of $B \$ \$(i, j) A \$ \$(i, j)$ for $i j]$
unfolding mat-ge-def carrier-mat-def by auto
lemma mat-ge-refl: $A \geq_{m} A$
unfolding mat-ge-def by (auto simp: ge-refl)
lemma mat-max-comm: $A \in$ carrier-mat $n r n c \Longrightarrow B \in$ carrier-mat $n r n c \Longrightarrow$ $\max _{m} A B=\max _{m} B A$
unfolding mat-max-def by (intro eq-matI, auto simp: max-comm)
lemma mat-max-ge: $\max _{m} A B \geq_{m} A$
unfolding mat-max-def by (intro mat-geI[of-dim-row A dim-col A], auto)
lemma mat-max-ge- $0: A \in$ carrier-mat $n r n c \Longrightarrow B \in$ carrier-mat $n r n c \Longrightarrow A$ $\geq_{m} B \Longrightarrow \max _{m} A B=A$
unfolding mat-max-def by (intro eq-matI, auto simp: max-id)
lemma mat-max-mono: $A \geq{ }_{m} B \Longrightarrow$
$A \in$ carrier-mat $n r n c \Longrightarrow B \in$ carrier-mat $n r n c \Longrightarrow C \in$ carrier-mat $n r n c$

$$
\Longrightarrow
$$

$\max _{m} C A \geq_{m} \max _{m} C B$
by (intro mat-geI[of-nr nc], auto simp: max-mono mat-max-def)
end
lemma mat-plus-left-mono: $A \geq_{m}$ ( $B::$ ' $a$ :: ordered-ab-semigroup mat)
$\Longrightarrow A \in$ carrier-mat $n r n c \Longrightarrow B \in$ carrier-mat $n r n c \Longrightarrow C \in$ carrier-mat $n r$ $n c$
$\Longrightarrow A+C \geq_{m} B+C$
by (intro mat-geI[of-nr nc], auto simp: plus-left-mono)
lemma mat-plus-right-mono: $B \geq_{m}\left(C::{ }^{\prime} a\right.$ :: ordered-ab-semigroup mat)
$\Longrightarrow A \in$ carrier-mat $n r n c \Longrightarrow B \in$ carrier-mat $n r n c \Longrightarrow C \in$ carrier-mat $n r$ $n c$
$\Longrightarrow A+B \geq_{m} A+C$
by (intro mat-geI[of -nr nc], auto simp: plus-right-mono)
lemma plus-mono: $x_{1} \geq\left(x_{2}::\right.$ ' $a::$ ordered-ab-semigroup $) \Longrightarrow$
$y_{1} \geq y_{2} \Longrightarrow x_{1}+y_{1} \geq x_{2}+y_{2}$
using ge-trans[OF plus-left-mono[of $x_{2} x_{1}$ ] plus-right-mono[of $\left.\left.y_{2} y_{1}\right]\right]$.
Since one cannot use $(\bigwedge i . i \in ? K \Longrightarrow$ ?f $i \leq$ ? $g i) \Longrightarrow$ sum ?f ? $K \leq$ sum ? $g$ ?K (it requires other class constraints like order), we make our own copy of this fact.

```
lemma sum-mono-ge:
    assumes \(g e: \bigwedge i . i \in K \Longrightarrow f\left(i::^{\prime} a\right) \geq((g i)::(' b::\) ordered-semiring-0 \())\)
    shows \(\left(\sum i \in K . f i\right) \geq\left(\sum i \in K . g i\right)\)
proof (cases finite \(K\) )
    case True
    thus ?thesis using ge
    proof induct
        case empty
        show ?case by (simp add: ge-refl)
    next
        case insert
        thus ?case using plus-mono by fastforce
    qed
next
    case False then show ?thesis by (simp add: ge-refl)
qed
lemma (in one-mono-ordered-semiring-1) sum-mono-gt:
    assumes \(l e: \bigwedge i\). \(i \in K \Longrightarrow f\left(i::^{\prime} b\right) \geq\left((g i)::^{\prime} a\right)\)
    and \(i: i \in K\)
    and \(g t: f i \succ g i\)
    and \(K\) : finite \(K\)
    shows \(\left(\sum i \in K . f i\right) \succ\left(\sum i \in K . g i\right)\)
proof -
    have \(i d: \wedge f .\left(\sum i \in K . f i\right)=f i+\left(\sum i \in K-\{i\} . f i\right)\)
        by (rule sum.remove \([\) OF \(K i]\) )
    have \(g e:\left(\sum i \in K-\{i\} . f i\right) \geq\left(\sum i \in K-\{i\} . g i\right)\)
    by (rule sum-mono-ge [OF le], auto)
    show ?thesis unfolding id using compat[OF plus-right-mono[OF ge] plus-gt-left-mono[OF
\(g t]\).
qed
lemma scalar-left-mono: assumes
    \(u \in\) carrier-vec \(n v \in\) carrier-vec \(n w \in\) carrier-vec \(n\)
    and \(\bigwedge i . i<n \Longrightarrow u \$ i \geq v \$ i\)
    and \(\bigwedge i . i<n \Longrightarrow w \$ i \geq(0:: ' a::\) ordered-semiring- 0\()\)
    shows \(u \cdot w \geq v \cdot w\) unfolding scalar-prod-def
    by (intro sum-mono-ge times-left-mono, insert assms, auto)
lemma scalar-right-mono: assumes
    \(u \in\) carrier-vec \(n v \in\) carrier-vec \(n w \in\) carrier-vec \(n\)
    and \(\bigwedge i . i<n \Longrightarrow v \$ i \geq w \$ i\)
    and \(\bigwedge i . i<n \Longrightarrow u \$ i \geq(0:: ' a::\) ordered-semiring-0 \()\)
```

```
    shows u\cdotv\gequ\cdotw
proof -
    have dim: dim-vec v= dim-vec w using assms by auto
    show ?thesis unfolding scalar-prod-def dim
        by (intro sum-mono-ge times-right-mono, insert assms, auto)
qed
lemma mat-mult-left-mono: assumes C0:C \geqmm Om}n
    and AB:A \geqm
    and carr: A \in carrier-mat n n B carrier-mat n n C \in carrier-mat n n
    shows A*C \geqm}B*
proof -
    {
        fix ij
        assume i: i<nj<n
        have row A i col C j\geq row B i col Col j
        by (rule scalar-left-mono[of-n], insert C0 AB carr i, auto)
    }
    thus ?thesis
    by (intro mat-geI[of-n n], insert carr, auto)
qed
lemma mat-mult-right-mono: assumes A0: A \geqmm 0m n n
    and BC:B \geqm
    and carr: A carrier-mat n n B carrier-mat n n C E carrier-mat n n
    shows }A*B\mp@subsup{\geq}{m}{}A*
proof -
    {
        fix ij
        assume i: i<nj<n
        have row A i . col B j\geq row A i • col C j
            by (rule scalar-right-mono[of - n], insert A0 BC carr i, auto)
    }
    thus ?thesis
        by (intro mat-geI[of - n n], insert carr, auto)
qed
lemma one-mat-ge-zero: (1 1m n :: 'a :: ordered-semiring-1 mat) \geqm 0 0m n n
    by (intro mat-geI[of - n n], auto simp: one-ge-zero ge-refl)
context order-pair
begin
lemma mat-ge-gt-trans: assumes sd:sd \leqn and AB:A \geqm B and BC:mat-gt
gt sd B C
    and A:A\incarrier-mat n n and B:B\incarrier-mat n n
shows mat-gt gt sd A C
proof -
    from mat-gtD[OF BC] obtain ij where ij: i< sd j< sd and gt: B $$ (i,j)
\succC$$(i,j)
```

and $B C: B \geq_{m} C$ by auto
from mat-ge-trans $[O F A B B C A B]$ have $A C: A \geq_{m} C$.
from mat-ge $D[O F A B$, of $i j] A$ sd $i j$ have $g e: A \$ \$(i, j) \geq B \$ \$(i, j)$ by auto from compat[OF ge gt] have $g t: A \$ \$(i, j) \succ C \$ \$(i, j)$.
with ij $A C$ show ?thesis by auto
qed
lemma mat-gt-ge-trans: assumes $s d: s d \leq n$ and $A B$ : mat-gt gt $s d A B$ and $B C$ :
$B \geq_{m} C$
and $A: A \in$ carrier-mat $n n$ and $B: B \in$ carrier-mat $n n$
shows mat-gt gt sd A C
proof -
from mat-gtD[OF $A B]$ obtain $i j$ where $i j: i<s d j<s d$ and $g t: A \$ \$(i, j)$
$\succ B \$ \$(i, j)$
and $A B: A \geq_{m} B$ by auto
from mat-ge-trans $[O F A B B C A B]$ have $A C: A \geq_{m} C$.
from mat-geD[OF BC, of $i j] B$ sd $i j$ have ge: $B \$ \$(i, j) \geq C \$ \$(i, j)$ by auto
from compat2[OF gt ge] have $g t: A \$ \$(i, j) \succ C \$ \$(i, j)$.
with ij $A C$ show ?thesis by auto
qed
lemma mat-gt-imp-mat-ge: mat-gt gt sd $A B \Longrightarrow A \geq{ }_{m} B$
by (rule mat-gtD)
lemma mat-gt-trans: assumes $s d: s d \leq n$ and $A B$ : mat-gt gt $s d A B$ and $B C$ : mat-gt gt sd BC
and $A: A \in$ carrier-mat $n n$ and $B: B \in$ carrier-mat $n n$
shows mat-gt gt sd A C
using mat-ge-gt-trans[OF sd mat-gt-imp-mat-ge $[O F A B] B C A B]$.
lemma mat-default-ge-0: default ${ }_{m}$ default $n \geq_{m} 0_{m} n n$
by (intro mat-geI [of - $n$ n], auto simp: mat-default-def default-ge-zero ge-refl)
end
definition mat-ordered-semiring :: nat $\Rightarrow$ nat $\Rightarrow$ (' $a$ :: ordered-semiring-1 $\Rightarrow{ }^{\prime} a$ $\Rightarrow$ bool $) \Rightarrow{ }^{\prime} b \Rightarrow\left({ }^{\prime}\right.$ a mat, $\left.{ }^{\prime} b\right)$ ordered-semiring-scheme where
mat-ordered-semiring $n$ sd gt $b \equiv$ ring-mat TYPE ('a) n 0
ordered-semiring.geq $=\left(\geq_{m}\right)$,
$g t=m a t-g t ~ g t s d$,
$\max =\max _{m}$,
$\ldots=b$ )
lemma (in one-mono-ordered-semiring-1) mat-ordered-semiring: assumes sd-n: $s d \leq n$
shows ordered-semiring
(mat-ordered-semiring $n$ sd $(\succ) b::$ ('a mat,'b) ordered-semiring-scheme)
(is ordered-semiring ? $R$ )
proof -
interpret semiring ?R unfolding mat-ordered-semiring-def by (rule semir-
show ?thesis
by (unfold-locales, unfold ring-mat-def mat-ordered-semiring-def ordered-semiring-record-simps,
auto intro: mat-ge-trans mat-ge-refl mat-ge-gt-trans[OF sd-n] mat-gt-ge-trans[OF
sd-n] mat-max-comm
mat-max-ge mat-max-ge-0 mat-max-mono one-mat-ge-zero mat-gt-trans[OF
sd-n] mat-gt-imp-mat-ge
mat-plus-left-mono mat-mult-left-mono mat-mult-right-mono)
qed
context weak-SN-strict-mono-ordered-semiring-1
begin
lemma weak-mat-gt-mono: assumes $s d-n: s d \leq n$ and
orient: $\bigwedge A B . A \in$ carrier-mat $n n \Longrightarrow B \in$ carrier-mat $n n \Longrightarrow(A, B) \in$ set
ABs $\Longrightarrow$ mat-gt weak-gt sd A B
shows $\exists$ gt. SN-strict-mono-ordered-semiring-1 default gt mono $\wedge$
$(\forall A B . A \in$ carrier-mat $n n \longrightarrow B \in$ carrier-mat $n n \longrightarrow(A, B) \in$ set $A B s$
$\longrightarrow$ mat-gt gt sd AB)
proof -
let ? $n=[0 . .<n]$
let ? $m 1 x=[A \$ \$(i, j) . A<-\operatorname{map} f s t A B s, i<-$ ? $n, j<-$ ? $n]$
let ?m2y $=[B \$ \$(i, j) . B<-$ map snd $A B s, i<-$ ? $n, j<-$ ? $n]$
let ?pairs = concat $(\operatorname{map}(\lambda x . \operatorname{map}(\lambda y .(x, y))$ ?m2y) ?m1x)
let ?strict $=$ filter $(\lambda(x, y)$. weak-gt $x y)$ ?pairs
have $\forall x y .(x, y) \in$ set ?strict $\longrightarrow$ weak-gt $x y$ by auto
from weak-gt-mono[OF this] obtain gt where order: SN-strict-mono-ordered-semiring-1
default gt mono
and orient2: $\bigwedge x y .(x, y) \in$ set ?strict $\Longrightarrow g t x y$ by auto
show ?thesis
proof (intro exI allI conjI impI, rule order)
fix $A B$
assume $A: A \in$ carrier-mat $n n$ and $B: B \in$ carrier-mat $n n$
and $A B:(A, B) \in$ set $A B s$
from orient $[O F$ this $]$ have mat-gt weak-gt sd $A B$ by auto
from mat-gtD[OF this] obtain $i j$ where
ge: $A \geq_{m} B$ and $i j: i<s d j<s d$ and wgt: weak-gt $(A \$ \$(i, j))(B \$ \$(i, j))$
by auto
from $i j\langle s d \leq n\rangle$ have $i j^{\prime}: i<n j<n$ by auto
have gt: gt $(A \$ \$(i, j))(B \$ \$(i, j))$
by (rule orient2, insert $i j^{\prime} A B$ wgt, force)
show mat-gt gt sd $A B$ using ij gt ge by auto
qed
qed
end
lemma sum-mat-mono:
assumes $A: A \in$ carrier-mat nr nc and $B: B \in$ carrier-mat nr nc
and $A B: A \geq_{m}(B:: ' a::$ ordered-semiring-0 mat)

```
    shows sum-mat }A\geq\mathrm{ sum-mat }
proof -
```

    from \(A B\) have \(i d\) : dim-row \(B=\operatorname{dim}\)-row \(A \operatorname{dim}\)-col \(B=\operatorname{dim}\)-col \(A\) by auto
    show ?thesis unfolding sum-mat-def id
            by (rule sum-mono-ge, insert mat-geD \([O F A B] i d\), auto)
    qed
context one-mono-ordered-semiring-1
begin
lemma sum-mat-mono-gt:
assumes $s d \leq n$
and $A: A \in$ carrier-mat $n n$ and $B: B \in$ carrier-mat $n n$
and $A B$ : mat-gt $(\succ)$ sd $A\left(B::{ }^{\prime} a\right.$ mat $)$
shows sum-mat $A \succ$ sum-mat $B$
proof -
from $A B$ have $i d$ : dim-row $B=\operatorname{dim}$-row $A$ dim-col $B=\operatorname{dim}$-col $A$ by auto
from mat-gtD[OF $A B]$ obtain $i j$ where $A B: A \geq_{m} B$ and
$i j: i<s d j<s d$ and $g t: A \$ \$(i, j) \succ B \$ \$(i, j)$ by auto
show ?thesis unfolding sum-mat-def id
by (rule sum-mono-gt[of -- $(i, j)]$, insert ij gt mat-geD $[O F A B] A B<s d \leq$
$n$, auto)
qed
lemma mat-plus-gt-left-mono: assumes $s d-n: s d \leq n$ and $g t:$ mat-gt $(\succ)$ sd $A B$
and $A: A \in$ carrier-mat $n n$ and $B: B \in$ carrier-mat $n n$ and $C: C \in$ carrier-mat
n $n$
shows mat-gt $(\succ) s d(A+C)(B+C)$
proof -
note $w f=A B C$
from mat-gtD[OF gt] obtain $i j$
where $A B: A \geq_{m} B$ and $i j: i<s d j<s d$ and $g t: A \$ \$(i, j) \succ B \$ \$(i, j)$ by
auto
from plus-gt-left-mono[OF gt, of C $\$ \$(i, j)$ ]
show ?thesis
by (intro mat-gtI[OF mat-geI[of -n n] ij], insert mat-ge $D[O F A B]$ wf ij sd-n,
auto intro: plus-left-mono)
qed
lemma mat-gt-ge-mono: sd $\leq n \Longrightarrow$ mat-gt gt sd $A B \Longrightarrow$
mat-gt gt sd C D $\Longrightarrow$
$A \in$ carrier-mat $n n \Longrightarrow$
$B \in$ carrier-mat $n n \Longrightarrow$
$C \in$ carrier-mat $n n \Longrightarrow$
$D \in$ carrier-mat $n n \Longrightarrow$
mat-gt gt sd $(A+C)(B+D)$
by (rule mat-gt-ge-trans[OF - mat-plus-gt-left-mono mat-plus-right-mono[OF mat-gt-imp-mat-ge]],
auto)

```
lemma mat-default-gt-mat0: assumes sd-pos: sd > 0 and sd-n: sd \leqn
    shows mat-gt }(\succ)\mathrm{ sd (default m default n) (0 ( }\mp@subsup{|}{m}{}nn
proof -
    from assms have n: n>0 by auto
    show ?thesis
        by (intro mat-gtI[OF mat-default-ge-0 sd-pos sd-pos], insert sd-pos sd-n, auto
simp: mat-default-def default-gt-zero)
qed
end
```

context SN-one-mono-ordered-semiring-1
begin
abbreviation mat-s :: 'a mat $\Rightarrow$ nat $\Rightarrow$ nat $\Rightarrow{ }^{\prime}$ 'a mat $\Rightarrow$ bool $\left(\left(-\succ_{m}---\right)\right.$
[51,51,51,51] 50)
where $A \succ_{m} n$ sd $B \equiv\left(A \in\right.$ carrier-mat $n n \wedge B \in$ carrier-mat $n n \wedge B \geq_{m} 0_{m}$
$n n \wedge$ mat-gt $(\succ) s d A B)$
lemma mat-gt-SN: assumes $s d-n: s d \leq n$ shows $S N\left\{(m 1, m 2) . m 1 \succ_{m} n\right.$ sd
m2 $\}$
proof
fix $A$
assume $\forall i$. $(A i, A($ Suc $i)) \in\left\{(m 1, m 2) . m 1 \succ_{m} n\right.$ sd $\left.m 2\right\}$
hence $\wedge i .(A i, A(S u c i)) \in\left\{(m 1, m 2) . m 1 \succ_{m} n\right.$ sd m2 $\}$ by blast
hence $A: \wedge i$. $A i \in$ carrier-mat $n n$
and ge: $\bigwedge i$. $A($ Suc $i) \geq_{m} O_{m} n n$
and $g t: \bigwedge i$. mat-gt $(\succ)$ sd $(A i)(A(S u c i))$ by auto
define $s$ where $s=(\lambda i$. sum-mat $(A i))$
\{
fix $i$
from sum-mat-mono-gt[OF sd-n A A gt[of i]]
have $g t$ : $s i \succ s$ (Suc $i$ ) unfolding $s$-def.
from sum-mat-mono[OF A - ge[of $i]]$
have ge: $s$ (Suc $i$ ) $\geq 0$ unfolding $s$-def by auto
note $g e g t$
\}
with $S N$ show False by auto
qed
end
context $S N$-strict-mono-ordered-semiring-1
begin
lemma mat-mono: assumes $s d-n: s d \leq n$ and $A: A \in$ carrier-mat $n n$ and $B: B$ $\in$ carrier-mat $n n$ and $C: C \in$ carrier-mat $n n$
and gt: mat-gt $(\succ)$ sd $B C$ and gez: $A \geq_{m} O_{m} n n$ and mmono: mat-mono mono sd $A$
shows mat-gt $(\succ)$ sd $(A * B)(A * C)$ (is mat-gt - - ?AB ?AC)

```
proof -
    from mat-gtD[OF gt] obtain ij where
        i:i<sd and j:j<sd and gt:B$$ (i,j)\succC$$(i,j) and BC:B \geqm}C\mathrm{ by
auto
    from mat-mult-right-mono[OF gez BC A B C] have ge: ?AB \geqm ? ?AC .
    from mmono[unfolded mat-mono-def] i obtain k where k: k<sd and mon:
mono (A $$ (k,i)) by auto
    from mat-geD[OF gez] ki sd-n A have A $$ (k,i)\geq0 by auto
    note mono = mono[OF mon gt this]
    have id: dim-vec (col B j) = n dim-vec (col C j) =n using j sd-n B C by auto
    {
        fix }
        assume i<n
        hence row A k$i* col Bj$i\geq row A k$ i* col C j$i
        by (intro times-right-mono, insert jk sd-n A B C mat-geD[OF gez] mat-geD[OF
BC], auto)
    } note sge = this
    have gt: row A k col Bj\succ row A k col Col unfolding scalar-prod-def id
        by (rule sum-mono-gt[of-- - , OF sge], insert mono k ij A B C sd-n, auto)
    show ?thesis
    by (rule mat-gtI[OF ge k j], insert k j sd-n A B C gt, auto)
qed
end
definition mat-comp-all :: ('a > ' }a=>\mathrm{ bool ) = 'a mat }=>\mp@subsup{}{}{\prime}'a mat =>b boo
where mat-comp-all r A B =
    (\foralli<dim-row A.}\forallj<dim-col A.r (A$$ (i,j))(B$$(i,j))
lemma mat-comp-allI:
    assumes A\incarrier-mat nr nc B\incarrier-mat nr nc
    and }\bigwedgeij.i<nr\Longrightarrowj<nc\Longrightarrowr(A$$(i,j))(B$$(i,j)
    shows mat-comp-all r A B
    unfolding mat-comp-all-def using assms by simp
lemma mat-comp-allE:
    assumes mat-comp-all r A B
    and A\in carrier-mat nr nc B \in carrier-mat nr nc
    shows }\bigwedgeij.i<nr\Longrightarrowj<nc\Longrightarrowr(A$$(i,j))(B$$(i,j)
    using assms unfolding mat-comp-all-def by auto
context weak-SN-both-mono-ordered-semiring-1
begin
abbreviation weak-mat-gt-arc :: 'a mat }=>\mathrm{ 'a mat }=>\mathrm{ bool
where weak-mat-gt-arc \equivmat-comp-all weak-gt
lemma weak-mat-gt-both-mono:
    assumes ABs: set ABs\subseteqcarrier-mat n n x carrier-mat n n
    and orient: }\forall(A,B)\in\mathrm{ set ABs.weak-mat-gt-arc A B
```

shows $\exists$ gt. SN-both-mono-ordered-semiring-1 default gt arc-pos $\wedge$ $(\forall(A, B) \in$ set $A B$. mat-comp-all gt $A B)$

## proof -

let ?pairs $=[(f s t A B \$ \$(i, j)$, snd $A B \$ \$(i, j)) . A B<-A B s, i<-[0 . .<n]$,
$j<-[0 . .<n]]$
let ?strict $=$ filter $(\lambda(x, y)$. weak-gt $x y)$ ?pairs
have $\forall x y .(x, y) \in$ set ?strict $\longrightarrow$ weak-gt $x y$ by auto
from weak-gt-both-mono[OF this]
obtain $g t$
where order: SN-both-mono-ordered-semiring-1 default gt arc-pos
and orient2: $\bigwedge x y .(x, y) \in$ set ? strict $\Longrightarrow g t x y$
by auto
\{
fix $A B$ assume $A B:(A, B) \in$ set $A B s$
hence $A: A \in$ carrier-mat $n n$ and $B: B \in$ carrier-mat $n n$
using $A B A B s$ by auto
have mat-comp-all gt $A B$
proof (rule mat-comp-allI[ $\left[\begin{array}{ll}\text { F } A B\end{array}\right]$ )
fix $i j$
assume $i: i<n$ and $j: j<n$
from mat-comp-allE[OF - A B this] orient $A B$
have weak-gt: weak-gt $(A \$ \$(i, j))(B \$ \$(i, j))$ (is weak-gt ? $x$ ? $y$ ) by auto
have $(? x, ? y) \in$ set ?pairs using $A A B i j$ by force
with weak-gt
have $g t:(? x, ? y) \in$ set ?strict by simp
show $g t$ ? $x$ ? y by (rule orient2 [OF gt])
qed
\}
hence $\forall(A, B) \in$ set $A B$. mat-comp-all gt $A B$ by auto
thus ?thesis using order by auto
qed
end
definition mat-both-ordered-semiring :: nat $\Rightarrow\left({ }^{\prime} a\right.$ :: ordered-semiring-1 $\Rightarrow{ }^{\prime} a \Rightarrow$ bool $) \Rightarrow{ }^{\prime} b \Rightarrow\left({ }^{\prime} a\right.$ mat, $\left.{ }^{\prime} b\right)$ ordered-semiring-scheme where
mat-both-ordered-semiring $n$ gt $b \equiv$ ring-mat TYPE ('a) n 0
ordered-semiring.geq $=$ mat-ge,
$g t=$ mat-comp-all gt,
$\max =$ mat-max,

$$
\ldots=b \mid
$$

definition mat-arc-posI $::\left({ }^{\prime} a \Rightarrow\right.$ bool $) \Rightarrow{ }^{\prime} a$ mat $\Rightarrow$ bool
where mat-arc-posI ap $A \equiv \operatorname{ap}(A \$ \$(0,0))$
context both-mono-ordered-semiring-1
begin
abbreviation mat-gt-arc :: 'a mat $\Rightarrow{ }^{\prime}$ 'a mat $\Rightarrow$ bool
where mat-gt-arc $\equiv$ mat-comp-all gt
abbreviation mat-arc-pos :: 'a mat $\Rightarrow$ bool
where mat-arc-pos $\equiv$ mat-arc-posI arc-pos
lemma mat-max-id: fixes $A$ :: 'a mat
assumes ge: mat-ge $A B$
and $A: A \in$ carrier-mat $n r n c$
and $B: B \in$ carrier-mat nr nc
shows mat-max $A B=A$
using mat-max-ge- $0\left[\begin{array}{llll}O F & A & B & g e\end{array}\right]$.
lemma mat-gt-arc-trans:
assumes $A$-B: mat-gt-arc $A B$
and $B$-C: mat-gt-arc $B C$
and $A: A \in$ carrier-mat nr nc
and $B: B \in$ carrier-mat nr nc
and $C: C \in$ carrier-mat $n r n c$
shows mat-gt-arc A C
proof (rule mat-comp-allI[OF A C $]$ )
fix $i j$
assume $i: i<n r$ and $j: j<n c$
from mat-comp-allE[OF A-B A B i $j$ ] mat-comp-alle[OF B-C B Cij] show $A \$ \$(i, j) \succ C \$ \$(i, j)$ by (rule gt-trans)
qed
lemma mat-gt-arc-compat:
assumes ge: mat-ge $A B$
and gt: mat-gt-arc $B C$
and $A: A \in$ carrier-mat $n r n c$
and $B: B \in$ carrier-mat nr nc
and $C: C \in$ carrier-mat $n r n c$
shows mat-gt-arc A C
proof (rule mat-comp-allI[OF A C])
fix $i j$ assume $i: i<n r$ and $j: j<n c$
have $A \$ \$(i, j) \geq B \$ \$(i, j)$ using ge $A i j$ by auto
also have $B \$ \$(i, j) \succ C \$ \$(i, j)$
using mat-comp-allE[OF gt B Cij] by auto
finally show $A \$ \$(i, j) \succ C \$ \$(i, j)$ by auto
qed
lemma mat-gt-arc-compat2:
assumes gt: mat-gt-arc A B
and ge: mat-ge $B C$
and $A: A \in$ carrier-mat $n r n c$
and $B: B \in$ carrier-mat $n r n c$
and $C: C \in$ carrier-mat $n r n c$
shows mat-gt-arc A C
proof (rule mat-comp-allI[OFA $\left.\begin{array}{ll}\text { OF }\end{array}\right]$

```
    fix ij assume i:i<nr and j:j<nc
    have A $$ (i,j)\succB$$(i,j)
    using mat-comp-allE[OF gt] A B i j by auto
    also have B$$(i,j)\geqC$$(i,j)
    using ge B i j by auto
    finally show A $$(i,j)\succC$$(i,j) by auto
qed
lemma mat-gt-arc-imp-mat-ge:
    assumes gt: mat-gt-arc A B
    and A:A\incarrier-mat nr nc
    and B:B\incarrier-mat nr nc
    shows mat-ge A B
    using subst mat-geI[OF A]
    using mat-comp-allE[OF gt A B] gt-imp-ge by auto
lemma (in both-mono-ordered-semiring-1) mat-both-ordered-semiring: assumes
n: n>0
    shows ordered-semiring
        (mat-both-ordered-semiring n (\succ) b :: ('a mat,'b) ordered-semiring-scheme)
    (is ordered-semiring ?R)
proof -
    interpret semiring ?R unfolding mat-both-ordered-semiring-def by (rule semir-
ing-mat)
    show ?thesis
        apply (unfold-locales)
        unfolding ring-mat-def mat-both-ordered-semiring-def ordered-semiring-record-simps
        apply(
            auto intro: mat-max-comm mat-ge-trans
            mat-plus-left-mono mat-mult-left-mono mat-mult-right-mono mat-ge-refl
            one-mat-ge-zero mat-max-mono mat-max-ge mat-max-id
            mat-gt-arc-trans mat-gt-arc-imp-mat-ge
            mat-gt-arc-compat mat-gt-arc-compat2)
        done
qed
lemma mat0-leastI:
    assumes A:A\in carrier-mat nr nc
    shows mat-gt-arc A (0m nr nc)
proof (rule mat-comp-allI[OF A])
    fix ij
    assume i: i<nr and j:j<nc
    thus A$$(i,j)\succ\mp@subsup{0}{m}{}nrnc$$(i,j) by (auto simp: zero-leastI)
qed auto
lemma mat0-leastII:
    assumes gt: mat-gt-arc (0m nr nc) A
    and A:A\in carrier-mat nr nc
```

```
    shows A= 0m nr nc
    apply (rule eq-matI)
    unfolding index-zero-mat
    using }
proof -
    fix ij
    assume i:i<nr and j:j<nc
    show A $$ (i,j)=0
    using zero-leastII mat-comp-allE[OF gt - A] ij by auto
qed auto
lemma mat0-leastIII:
    assumes A: A \in carrier-mat nr nc
    shows mat-ge A ((0m nr nc) :: 'a mat)
proof (rule mat-geI[OF A]; unfold index-zero-mat)
    fix ij
    assume i: i<nr and j:j<nc
    show A $$ (i,j)\geq0 using zero-leastIII by simp
qed
lemma mat-max-0-id: fixes A :: 'a mat
    assumes A: A carrier-mat nr nc
    shows mat-max (0m nr nc) A = A
    unfolding mat-max-comm[OF zero-carrier-mat A]
    by (rule mat-max-id[OF mat0-leastIII[OF A] A], simp)
lemma mat-arc-pos-one:
    assumes n0: n>0
    shows mat-arc-posI arc-pos (1m n)
    unfolding mat-arc-posI-def
    unfolding arc-pos-one index-one-mat(1)[OF n0 n0]
    using arc-pos-one by simp
lemma mat-arc-pos-zero:
    assumes n0: n>0
    shows \neg mat-arc-posI arc-pos ( }0m=n\mp@code{n
    unfolding mat-arc-posI-def
    unfolding index-zero-mat(1)[OF n0 n0] using arc-pos-zero by simp
lemma mat-gt-arc-plus-mono:
    assumes gt1: mat-gt-arc A B
    and gt2: mat-gt-arc C D
    and A:(A::'a mat) \in carrier-mat nr nc
    and B:(B::'a mat) \in carrier-mat nr nc
    and C:(C::'a mat) \in carrier-mat nr nc
    and D:(D::'a mat) \in carrier-mat nr nc
    shows mat-gt-arc (A+C) (B+D) (is mat-gt-arc ?AC ?BD)
proof -
    show ?thesis
```

```
    proof (rule mat-comp-allI)
    fix i
    assume i:i<nr and j:j<nc
    hence ijC: i< dim-row C j<dim-col C
        and ijD:i<dim-row Dj<dim-col D
        using C D by auto
    show ?AC $$ (i,j)\succ ?BD $$ (i,j)
        unfolding index-add-mat(1)[OF ijC]
        unfolding index-add-mat(1)[OF ijD]
        using plus-gt-both-mono
        using mat-comp-allE[OF gt1 A B] mat-comp-allE[OF gt2 C D] ij by auto
    qed (insert A B C D, auto)
qed
definition vec-comp-all :: ('a = 'a # bool) = 'a vec => 'a vec = bool
    where vec-comp-all r v w\equiv\foralli< dim-vec v.r (v$i) (w$i)
lemma vec-comp-allI:
    assumes }\bigwedgei.i<dim-vec v\Longrightarrowr(v$i)(w$i
    shows vec-comp-all r v w
    unfolding vec-comp-all-def using assms by auto
lemma vec-comp-allE:
    vec-comp-all r v w\Longrightarrowi< dim-vec v\Longrightarrowr(v$i)(w$i)
    unfolding vec-comp-all-def by auto
lemma scalar-prod-left-mono:
    assumes u: u\in carrier-vec n
    and v:v\in carrier-vec n
    and w:w\incarrier-vec n
    and uv: vec-comp-all gt uv
    shows scalar-prod u w\succ scalar-prod v w
proof -
    { fix m assume m}\leq
        hence }(\sumi<m.(u$i)*(w$i))\succ(\sumi<m.(v$i)*(w$i)
        proof (induct m)
            case 0 show ?case using zero-leastI by simp next
            case (Suc m)
            hence uv: u$ m\succv$ m
                    using vec-comp-allE[OF uv] u by auto
                    show ?case
                        unfolding sum.lessThan-Suc
                        apply (subst plus-gt-both-mono)
                            using times-gt-left-mono Suc times-gt-left-mono[OF uv] by auto
        qed
}
from this[OF order.refl]
show ?thesis
    unfolding scalar-prod-def atLeastOLessThan
```

```
    using w by auto
qed
lemma scalar-prod-right-mono:
    assumes u:u\in carrier-vec n
    and v:v\in carrier-vec n
    and w:w\in carrier-vec n
    and vw: vec-comp-all gt v w
    shows scalar-prod uv\succ scalar-prod u w
proof -
    { fix m assume m\leqn
        hence}(\sumi<m.(u$i)*(v$i))\succ(\sumi<m.(u$i)*(w$i)
        proof (induct m)
            case 0 show ?case using zero-leastI by simp next
            case (Suc m)
                    hence vw:v$m\succw$ m
                    using vec-comp-allE[OF vw]v by auto
                    show ?case
                    unfolding sum.lessThan-Suc
                    apply (subst plus-gt-both-mono)
                    using times-gt-left-mono Suc times-gt-right-mono[OF vw] by auto
        qed
    }
    from this[OF order.refl]
    show ?thesis
        unfolding scalar-prod-def atLeast0LessThan
        using v w by auto
qed
lemma mat-gt-arc-mult-left-mono:
    assumes gt1: mat-gt-arc A B
    and A:(A::'a mat) \in carrier-mat nr n
    and B:(B::'a mat) \in carrier-mat nr n
    and C:(C::'a mat) \in carrier-mat n nc
    shows mat-gt-arc (A*C)(B*C) (is mat-gt-arc ?AC ?BC)
proof (rule mat-comp-allI)
    fix ij assume i:i<nr and j:j<nc
    hence iA: i< dim-row }
        and iB:i<dim-row B
        and jC:j<dim-col C
        using A B C by auto
    show ?AC $$ (i,j)\succ? ?C $$ (i,j)
        unfolding index-mult-mat(1)[OF iA jC]
        unfolding index-mult-mat(1)[OF iB jC]
    proof(rule scalar-prod-left-mono)
        show row A i\in carrier-vec n using A by auto
        show row B i carrier-vec n using B by auto
        show col C j\incarrier-vec n using C by auto
        show rowAB: vec-comp-all (\succ) (row A i) (row B i)
```

```
    proof (intro vec-comp-allI)
        fix j assume j:j<dim-vec (row A i)
        have A$$(i,j)\succB$$(i,j)
        using mat-comp-allE[OF gt1 A B i] j A by simp
    thus row A i$j\succ row Bi$j
        using A B Cij by simp
        qed
    qed
qed (insert A B C, auto)
lemma mat-gt-arc-mult-right-mono:
    assumes gt1: mat-gt-arc B C
    and A: (A::'a mat) \in carrier-mat nr n
    and B:(B::'a mat ) \incarrier-mat n nc
    and C:(C::'a mat) \in carrier-mat n nc
    shows mat-gt-arc (A*B)(A*C) (is mat-gt-arc ?AB ?AC)
proof (rule mat-comp-allI)
    fix ij assume i:i<nr and j:j<nc
    hence iA: i<dim-row A
    and jB: j<dim-col B
    and jC: j<dim-col C
    using A B C by auto
show ?AB $$ (i,j)\succ?AC $$ (i,j)
    unfolding index-mult-mat(1)[OF iA jB]
    unfolding index-mult-mat(1)[OF iA jC]
proof(rule scalar-prod-right-mono)
    show row A i carrier-vec n using A by auto
    show col B j carrier-vec n using B by auto
    show col C j carrier-vec n using C by auto
    show rowAB: vec-comp-all (\succ) (col B j) (col C j)
    proof (intro vec-comp-allI)
        fix i assume i: i<dim-vec (col B j)
        have B$$(i,j)\succC$$(i,j)
            using mat-comp-allE[OF gt1 B C] i j B by simp
            thus col Bj$ i\succ col Cj$i
            using A B Cij by simp
        qed
    qed
qed (insert A B C, auto)
lemma mat-arc-pos-plus:
    assumes n0: n>0
    and A:A C carrier-mat nn
    and B:B\in carrier-mat n n
    and arc-pos: mat-arc-pos A
    shows mat-arc-pos ( }A+B\mathrm{ )
    unfolding mat-arc-posI-def
    apply (subst index-add-mat(1))
    using arc-pos-plus[OF arc-pos[unfolded mat-arc-posI-def]]
```

assms by auto
lemma scalar-prod-split-head: assumes
$A \in$ carrier-mat $n$ n $B \in$ carrier-mat $n$ n $n>0$
shows row $A 0 \cdot \operatorname{col} B 0=A \$ \$(0,0) * B \$ \$(0,0)+\left(\sum i=1 . .<n . A \$ \$(0\right.$,
$i) * B \$ \$(i, 0))$
unfolding scalar-prod-def
using assms sum.atLeast-Suc-lessThan by auto
lemma mat-arc-pos-mult:
assumes n0: $n>0$
and $A: A \in$ carrier-mat $n n$
and $B: B \in$ carrier-mat $n n$
and apA: mat-arc-pos $A$
and apB: mat-arc-pos $B$
shows mat-arc-pos $(A * B)$
unfolding mat-arc-posI-def
apply(subst index-mult-mat(1))
proof -
let ?prod $=$ row $A 0 \cdot \operatorname{col} B 0$
let ?head $=A \$ \$(0,0) * B \$ \$(0,0)$
let ?rest $=\sum i=1 . .<n . A \$ \$(0, i) * B \$ \$(i, 0)$
have ap: arc-pos ?head
using $a p A$ ap $B$
unfolding mat-arc-posI-def
using arc-pos-mult by auto
have split: ? prod $=$ ? head + ? rest
by (rule scalar-prod-split-head $[O F A B n 0])$
show arc-pos (row A $0 \cdot \operatorname{col} B 0$ )
unfolding split
using ap arc-pos-plus by auto
qed (insert $A B$ n0, auto)
lemma mat-arc-pos-mat-default:
assumes $n 0: n>0$ shows mat-arc-pos (mat-default default $n$ )
unfolding mat-arc-posI-def
unfolding mat-default-def
unfolding index-mat (1)[OF n0 n0]
using arc-pos-default by simp
lemma mat-not-all-ge:
assumes n-pos: $n>0$
and $A: A \in$ carrier-mat $n n$
and $B: B \in$ carrier-mat $n n$
and apB: mat-arc-pos $B$
shows $\exists C . C \in$ carrier-mat $n n \wedge$ mat-ge $C\left(O_{m} n n\right) \wedge$ mat-arc-pos $C \wedge \neg$ mat-ge $A(B * C)$
proof -
define $c$ where $c=A \$ \$(0,0)$
from $\operatorname{ap} B$ have arc-pos ( $B \$ \$(0,0)$ ) unfolding mat-arc-posI-def.
from not-all-ge $[O F$ this, of $c]$ obtain $e$ where $e 0: e \geq 0$ and ae: arc-pos $e$
and $n c: \neg c \geq B \$ \$(0,0) * e$ by auto
let ?f $=\lambda i j$. if $i=0 \wedge j=0$ then $e$ else 0
let ? $C=$ mat $n n(\lambda(i, j)$. ?f $i j)$
have $C: ? C \in$ carrier-mat $n n$ by auto
have C00: ? $C \$ \$(0,0)=e$ using $n$-pos by auto
show ?thesis
proof (intro exI conjI)
show ? $C \geq_{m} 0_{m} n n$
by (rule mat-geI[of-n n], auto simp: ge-refl e0)
show mat-arc-pos? $C$
unfolding mat-arc-posI-def
unfolding C00 by (rule ae)
let ? mult $=B *$ ? $C$
from n-pos obtain $n n$ where $n$ : $n=S u c n n$ by (cases $n$, auto)
have col: col ?C $0=$ vec $n$ (?f 0 ) using $n$-pos by auto
let ?prod $=$ row $B 0 \cdot \mathrm{col}$ ? $C 0$
let ?head $=B \$ \$(0,0) *$ ? $C \$ \$(0,0)$
let ?rest $=\sum i=1 . .<n . B \$ \$(0, i) * ? C \$ \$(i, 0)$
from $n$-pos $B$ have ?mult $\$ \$(0,0)=$ ?prod by auto
also have $\ldots=$ ? head + ?rest
by (rule scalar-prod-split-head $[$ OF B $C$ n-pos])
also have ? rest $=0$
by (rule sum.neutral, auto)
finally have ? mult $\$ \$(0,0)=B \$ \$(0,0) * e$ using $n$-pos by simp
with nc c-def have not-ge: $\neg A \$ \$(0,0) \geq$ ? mult $\$ \$(0,0)$ by simp
show $\neg A \geq_{m}$ ? mult
proof
assume $A \geq_{m}$ ?mult
from mat-geD[OF this, of 00$]$ A $B$ not-ge n-pos show False by auto
qed
qed auto
qed
end
context SN-both-mono-ordered-semiring-1
begin
lemma mat-gt-arc-SN:
assumes $n$-pos: $n>0$
shows $S N\{(A, B) \in$ carrier-mat $n n \times$ carrier-mat $n$ n. mat-arc-pos $B \wedge$
mat-gt-arc $A B\}$
(is $S N$ ? rel)
proof (rule ccontr)
assume $\neg S N$ ? rel

```
    then obtain f A where f(0 :: nat) = A and steps: }\foralli.(fi,f(Suc i))\in?re
unfolding SN-defs by blast
    hence pos: }\forall\mathrm{ i. arc-pos ( }f(\mathrm{ Suc i) $$ (0,0)) unfolding mat-arc-posI-def by blast
    have gt: }\forall\mathrm{ i.fi $$(0,0)}\succf(Suc i)$$(0,0
    proof
        fix }
        from steps
    have wf1: fi\incarrier-mat n n
        and wf2: f (Suc i) \in carrier-mat n n
        and gt: mat-gt-arc (f i) (f (Suc i)) by auto
    show fi$$(0,0)\succ}f(\mathrm{ Suc i) $$ (0,0)
        using mat-comp-allE[OF gt wf1 wf2]
        using index-zero-mat n-pos by force
    qed
    from pos gt SN show False unfolding SN-defs by force
qed
end
end
```


## 23 Matrix Conversions

Essentially, the idea is to use the JNF results to estimate the growth rates of matrices. Since the results in JNF are only applicable for real normed fields, we cannot directly use them for matrices over the integers or the rational numbers. To this end, we define a homomorphism which allows us to first convert all numbers to real numbers, and then do the analysis.

```
theory Ring-Hom-Matrix
imports
    Matrix
    Polynomial-Interpolation.Ring-Hom
begin
```

locale ord-ring-hom $=$ idom-hom hom for
hom :: 'a :: linordered-idom $\Rightarrow$ ' $b$ :: floor-ceiling +
assumes hom-le: hom $x \leq z \Longrightarrow x \leq$ of-int $\lceil z\rceil$

Now a class based variant especially for homomorphisms into the reals.

```
class real-embedding \(=\) linordered-idom +
    fixes real-of :: ' \(a \Rightarrow\) real
    assumes
        real-add: real-of \(((x:: ' a)+y)=\) real-of \(x+\) real-of \(y\) and
        real-mult: real-of \((x * y)=\) real-of \(x *\) real-of \(y\) and
        real-zero: real-of \(0=0\) and
        real-one: real-of \(1=1\) and
        real-le: real-of \(x \leq z \Longrightarrow x \leq\) of-int \(\lceil z\rceil\)
```

```
interpretation real-embedding: ord-ring-hom (real-of :: 'a :: real-embedding \(\Rightarrow\)
real)
    by (unfold-locales; fact real-add real-mult real-zero real-one real-le)
instantiation real :: real-embedding
begin
definition real-of-real :: real \(\Rightarrow\) real where
    real-of-real \(x=x\)
instance
    by (intro-classes, auto simp: real-of-real-def, linarith)
end
instantiation int :: real-embedding
begin
definition real-of-int :: int \(\Rightarrow\) real where
    real-of-int \(x=x\)
instance
    by (intro-classes, auto simp: real-of-int-def, linarith)
end
lemma real-of-rat-ineq: assumes real-of-rat \(x \leq z\)
    shows \(x \leq\) of-int \(\lceil z\rceil\)
proof -
    have \(z \leq\) of-int \(\lceil z\rceil\) by linarith
    from order-trans[OF assms this]
    have real-of-rat \(x \leq\) real-of-rat (of-int \(\lceil z\rceil\) ) by auto
    thus \(x \leq\) of-int \(\lceil z\rceil\) using of-rat-less-eq by blast
qed
instantiation rat :: real-embedding
begin
definition real-of-rat :: rat \(\Rightarrow\) real where
    real-of-rat \(x=\) of-rat \(x\)
instance
    by (intro-classes, auto simp: real-of-rat-def of-rat-add of-rat-mult real-of-rat-ineq)
end
abbreviation mat-real \(\left(\right.\) mat \(\left._{\mathrm{R}}\right)\) where mat \(_{\mathbb{R}} \equiv\) map-mat (real-of :: 'a :: real-embedding
\(\Rightarrow\) real)
end
```


## 24 Derivation Bounds

Starting from this point onwards we apply the results on matrices to derive complexity bounds in IsaFoR. So, here begins the connection to the definitions and prerequisites that have originally been defined within IsaFoR.

This theory contains the notion of a derivation bound.

```
theory Derivation-Bound
imports
    Abstract-Rewriting.Abstract-Rewriting
begin
definition deriv-bound :: 'a rel }=>\mp@subsup{}{}{\prime}a=>\mathrm{ nat }=>\mathrm{ bool
where
    deriv-bound r a n \longleftrightarrow ᄀ(\exists b. (a,b) \inr^^Suc n)
lemma deriv-boundI [intro?]:
    (\bigwedgeb m. n<m\Longrightarrow (a,b) \inr^m m False) \Longrightarrow deriv-bound r a n
    by (auto simp: deriv-bound-def) (metis lessI relpow-Suc-I)
lemma deriv-boundE:
    assumes deriv-bound r a n
        and (\bigwedgeb m.n<m\Longrightarrow(a,b) 的^m\Longrightarrow False) \LongrightarrowP
    shows P
    using assms(1)
    by (intro assms)
        (auto simp:deriv-bound-def relpow-add relcomp.simps dest!: less-imp-Suc-add,
metis relpow-E2)
lemma deriv-bound-iff:
    deriv-bound r a n \longleftrightarrow(\forallbm.n<m\longrightarrow(a,b)\not\inr^m
    by (auto elim: deriv-boundE intro:deriv-boundI)
lemma deriv-bound-empty [simp]:
    deriv-bound {} a n
    by (simp add: deriv-bound-def)
lemma deriv-bound-mono:
    assumes m\leqn and deriv-bound r a m
    shows deriv-bound r a n
    using assms by (auto simp: deriv-bound-iff)
lemma deriv-bound-image:
    assumes b: deriv-bound r'}\mp@subsup{r}{}{\prime}fa)
        and step: \ab. (a,b) \inr\Longrightarrow(fa,fb)\in\mp@subsup{r}{}{\prime+}
    shows deriv-bound r a n
proof
    fix b m
    assume (a,b) \inr^m
    from relpow-image [OF step this] have (fa,fb)\in r'+ ~~ m.
```

```
    from trancl-steps-relpow [OF subset-refl this]
        obtain k}\mathrm{ where }k\geqm\mathrm{ and (fa,fb) G r'~}~~\mathrm{ by auto
    moreover assume n<m
    moreover with deriv-bound-mono [OF - b, of m - 1]
    have deriv-bound r' (f a) (m-1) by simp
    ultimately show False using b by (simp add: deriv-bound-iff)
qed
lemma deriv-bound-subset:
    assumes r\subseteq\mp@subsup{r}{}{\prime+}
    and b: deriv-bound r' a n
    shows deriv-bound r a n
    using assms by (intro deriv-bound-image [of - \lambdax. x, OF b]) auto
lemma deriv-bound-SN-on:
    assumes deriv-bound r a n
    shows SN-on r {a}
proof
    fix f
    assume steps: }\foralli.(fi,f(Suc i))\inr and f0\in{a
    with assms have (f 0,f(Suc n)) #r^^ Suc n by (blast elim: deriv-boundE)
    moreover have (f 0,f(Suc n)) \inr^~ Suc n
        using steps unfolding relpow-fun-conv by (intro exI [of - f]) auto
    ultimately show False ..
qed
lemma deriv-bound-steps:
    assumes (a,b) \inr^n
    and deriv-bound r a m
    shows n\leqm
    using assms by (auto iff: not-less deriv-bound-iff)
end
```


## 25 Complexity Carrier

We define which properties a carrier of matrices must exhibit, so that it can be used for checking complexity proofs.

```
theory Complexity-Carrier
imports
    Abstract-Rewriting.SN-Order-Carrier
    Ring-Hom-Matrix
    Derivation-Bound
    HOL.Real
begin
class large-real-ordered-semiring-1 = large-ordered-semiring-1 + real-embedding
instance real :: large-real-ordered-semiring-1 ..
```

```
instance int :: large-real-ordered-semiring-1 ..
```

instance rat :: large-real-ordered-semiring-1 ..

For complexity analysis, we need a bounding function which tells us how often one can strictly decrease a value. To this end, $\delta$-orderings are usually applied when working with the reals or rational numbers.
locale complexity-one-mono-ordered-semiring-1 = one-mono-ordered-semiring-1 default gt
for $g t::$ ' $a$ :: large-ordered-semiring-1 $\Rightarrow{ }^{\prime} a \Rightarrow$ bool (infix $\succ 50$ ) and default :: ' $a+$
fixes bound :: ' $a \Rightarrow$ nat
assumes bound-mono: $\wedge a b . a \geq b \Longrightarrow$ bound $a \geq$ bound $b$
and bound-plus: $\bigwedge a b$. bound $(a+b) \leq$ bound $a+$ bound $b$
and bound-plus-of-nat: $\wedge$ a n. $a \geq 0 \Longrightarrow$ bound $(a+$ of-nat $n)=$ bound $a+$ bound (of-nat n)
and bound-zero $[$ simp $]$ : bound $0=0$
and bound-one: bound $1 \geq 1$
and bound: $\wedge a$. deriv-bound $\{(a, b) . b \geq 0 \wedge a \succ b\} a$ (bound a)
begin

```
lemma bound-linear: \(\exists c . \forall n\). bound (of-nat \(n\) ) \(\leq c * n\)
proof (rule exI[of - bound 1], intro allI)
    fix \(n\)
    show bound (of-nat \(n\) ) \(\leq\) bound \(1 * n\)
    proof (induct \(n\) )
        case (Suc n)
        have bound (of-nat (Suc n)) = bound \((1+\) of-nat \(n)\) by simp
        also have \(\ldots \leq\) bound \(1+\) bound (of-nat \(n\) )
            by (rule bound-plus)
        also have \(\ldots \leq\) bound \(1+\) bound \(1 * n\)
            using Suc by auto
    finally show ?case by auto
    qed simp
qed
lemma bound-of-nat-times: bound (of-nat \(n * v\) ) \(\leq n *\) bound \(v\)
proof (induct \(n\) )
    case (Suc n)
    have bound (of-nat (Suc n) *v) bound ( \(v+\) of-nat \(n * v\) ) by (simp add:
field-simps)
    also have \(\ldots \leq\) bound \(v+\) bound (of-nat \(n * v\) ) by (rule bound-plus)
    also have \(\ldots \leq\) bound \(v+n *\) bound \(v\) using Suc by auto
    finally show? case by simp
qed \(\operatorname{simp}\)
lemma bound-mult-of-nat: bound \((a *\) of-nat \(n) \leq\) bound \(a *\) bound (of-nat \(n\) )
proof (induct \(n\) )
    case (Suc n)
```

```
    have bound (a* of-nat (Suc n)) = bound (a+a* of-nat n) by (simp add:
field-simps)
    also have .. \leq bound a + bound (a* of-nat n)
    by (rule bound-plus)
    also have .. \leq bound a + bound a* bound (of-nat n) using Suc by auto
    also have ... = bound a* (1 + bound (of-nat n)) by (simp add: field-simps)
    also have .. \leq bound a*(bound (1 + of-nat n))
    proof (rule mult-le-mono2)
        show 1 + bound(of-nat n) \leqbound (1 + of-nat n) using bound-one
        using bound-plus
            unfolding bound-plus-of-nat[OF one-ge-zero] by simp
    qed
    finally show ?case by simp
qed simp
lemma bound-pow-of-nat: bound (a* of-nat n^deg) \leqbound a* of-nat n^deg
proof (induct deg)
    case (Suc deg)
    have bound (a*of-nat n^ Suc deg) = bound (of-nat n * (a*of-nat n^^deg))
    by (simp add: field-simps)
    also have ...\leqn* bound (a* of-nat n^ deg)
    by (rule bound-of-nat-times)
    also have ...\leqn* (bound a*of-nat n^ deg)
    using Suc by auto
    finally show ?case by (simp add: field-simps)
qed simp
end
end
```


## 26 Converting Arctic Numbers to Strings

We just instantiate arctic numbers in the show-class.

```
theory Show-Arctic
imports
    Abstract-Rewriting.SN-Order-Carrier
    Show.Show-Instances
begin
instantiation arctic :: show
begin
fun shows-arctic :: arctic \(\Rightarrow\) shows
where
    shows-arctic (Num-arc \(i)=\) shows \(i \mid\)
    shows-arctic \((\) MinInfty \()=\) shows \({ }^{\prime \prime}-\) inf \({ }^{\prime \prime}\)
definition shows-prec ( \(p\) :: nat) ai = shows-arctic ai
```

```
lemma shows-prec-artic-append [show-law-simps]:
    shows-prec p (a :: arctic) (r@ s)= shows-prec p ar @ s
    by (cases a) (auto simp: shows-prec-arctic-def show-law-simps)
definition shows-list (as :: arctic list) = showsp-list shows-prec 0 as
instance
    by standard (simp-all add: shows-list-arctic-def show-law-simps)
end
instantiation arctic-delta :: (show) show
begin
fun shows-arctic-delta :: 'a arctic-delta }=>\mathrm{ shows
where
    shows-arctic-delta (Num-arc-delta i) = shows i
    shows-arctic-delta (MinInfty-delta) = shows "'-inf"
definition shows-prec (d :: nat) ari = shows-arctic-delta ari
lemma shows-prec-arctic-delta-append [show-law-simps]:
    shows-prec d (a :: 'a arctic-delta) (r @ s)= shows-prec d a r @ s
    by (cases a) (auto simp: shows-prec-arctic-delta-def show-law-simps)
definition shows-list (ps :: 'a arctic-delta list) = showsp-list shows-prec 0 ps
instance
    by standard (simp-all add: shows-list-arctic-delta-def show-law-simps)
end
end
```


## 27 Application: Complexity of Matrix Orderings

In this theory we provide various carriers which can be used for matrix interpretations.

```
theory Matrix-Complexity
imports
    Matrix-Comparison
    Complexity-Carrier
    Show-Arctic
```

begin

### 27.1 Locales for Carriers of Matrix Interpretations and Polynomial Orders

locale matrix-carrier $=S N$-one-mono-ordered-semiring-1 $d g t$
for $g t::{ }^{\prime} a::\{$ show,ordered-semiring-1 $\} \Rightarrow{ }^{\prime} a \Rightarrow$ bool (infix $\succ 50$ ) and $d::^{\prime} a$
locale mono-matrix-carrier $=$ complexity-one-mono-ordered-semiring-1 gt d bound for $g t$ :: ' $a$ :: \{show,large-real-ordered-semiring-1\} $\Rightarrow^{\prime} a \Rightarrow$ bool (infix $\succ 50$ ) and $d::{ }^{\prime} a$
and bound $::$ ' $a \Rightarrow$ nat

+ fixes mono :: ' $a \Rightarrow$ bool
assumes mono: $\bigwedge x y z$. mono $x \Longrightarrow y \succ z \Longrightarrow x \geq 0 \Longrightarrow x * y \succ x * z$
The weak version make comparison with $>$ and then synthesize a suitable $\delta$-ordering by choosing the least difference in the finite set of comparisons.

```
locale weak-complexity-linear-poly-order-carrier \(=\)
    fixes weak-gt :: 'a :: \{large-real-ordered-semiring-1,show \(\} \Rightarrow{ }^{\prime} a \Rightarrow\) bool
        and default :: ' \(a\)
        and mono :: ' \(a \Rightarrow\) bool
    assumes weak-gt-mono: \(\forall x y .(x, y) \in\) set xys \(\longrightarrow\) weak-gt \(x y\)
    \(\Longrightarrow \exists\) gt bound. mono-matrix-carrier gt default bound mono \(\wedge(\forall x y .(x, y) \in\)
set \(x y s \longrightarrow g t x y\) )
begin
```

abbreviation weak-mat-gt :: nat $\Rightarrow$ 'a mat $\Rightarrow$ 'a mat $\Rightarrow$ bool
where weak-mat-gt $\equiv$ mat-gt weak-gt
lemma weak-mat-gt-mono: assumes $s d-n: s d \leq n$ and
orient: $\bigwedge A B . A \in$ carrier-mat $n n \Longrightarrow B \in$ carrier-mat $n n \Longrightarrow(A, B) \in$ set
$A B s \Longrightarrow$ weak-mat-gt sd $A B$
shows $\exists$ gt bound. mono-matrix-carrier gt default bound mono
$\wedge(\forall A B . A \in$ carrier-mat $n n \longrightarrow B \in$ carrier-mat $n n \longrightarrow(A, B) \in$ set $A B s$
$\longrightarrow$ mat-gt gt sd $A B$ )
proof -
let ? $n=[0$.. $<n]$
let ? $m 1 x=[A \$ \$(i, j) . A<-\operatorname{map} f s t A B s, i<-? n, j<-? n]$
let ?m2y $=[B \$ \$(i, j) . B<-$ map snd $A B s, i<-$ ? $n, j<-$ ? $n]$
let ?pairs $=$ concat $(\operatorname{map}(\lambda x \cdot \operatorname{map}(\lambda y .(x, y))$ ?m2y) ?m1x $)$
let ?strict $=$ filter $(\lambda(x, y)$. weak-gt $x y)$ ?pairs
have $\forall x y .(x, y) \in$ set ?strict $\longrightarrow$ weak-gt $x y$ by auto
from weak-gt-mono[OF this] obtain gt bound where order: mono-matrix-carrier
gt default bound mono
and orient2: $\bigwedge x y .(x, y) \in$ set ? strict $\Longrightarrow g t x y$ by auto
show ?thesis
proof (intro exI allI conjI impI, rule order)
fix $A B$
assume $A: A \in$ carrier-mat $n n$ and $B: B \in$ carrier-mat $n n$
and $A B:(A, B) \in$ set $A B s$
from orient $[O F$ this $]$ have mat-gt weak-gt sd $A B$ by auto
from mat-gtD[OF this] obtain $i j$ where
ge: $A \geq_{m} B$ and $i j: i<s d j<s d$ and wgt: weak-gt $(A \$ \$(i, j))(B \$ \$(i, j))$ by auto
from $i j\langle s d \leq n\rangle$ have $i j^{\prime}: i<n j<n$ by auto
have gt: gt $(A \$ \$(i, j))(B \$ \$(i, j))$
by (rule orient2, insert $i j^{\prime} A B$ wgt, force)
show mat-gt gt sd $A B$ using ij gt ge by auto
qed
qed
end
sublocale mono-matrix-carrier $\subseteq$ SN-strict-mono-ordered-semiring-1 d gt mono proof
show $S N\{(x, y) . y \geq 0 \wedge x \succ y\}$
unfolding $S N$-def
by (intro allI deriv-bound-SN-on[OF bound $]$ )
qed (rule mono)
sublocale mono-matrix-carrier $\subseteq$ matrix-carrier ..

### 27.2 The Integers as Carrier

```
lemma int-complexity:
    mono-matrix-carrier ((>) :: int }=>\mathrm{ int }=>\mathrm{ bool) 1 nat int-mono
proof (unfold-locales)
    fix }
    let ?R = {(x,y). 0 \leq (y :: int) ^ y<x}
    show deriv-bound ?R x (nat x)
        unfolding deriv-bound-def
    proof
    assume (\existsy.(x,y)\in?R ^~Suc (nat x))
    then obtain }y\mathrm{ where xy:(x,y) &?R^ Suc (nat x) ..
    from }xy\mathrm{ have y:0 { y by auto
    obtain n where n: n=Suc (nat x) by auto
    from xy[unfolded n[symmetric]]
    have }x\geqy+\mathrm{ int n
    proof (induct n arbitrary: x y)
        case 0 thus ?case by auto
    next
        case (Suc n)
        from Suc(2) obtain z where xz:(x,z)\in?R ^n n and zy: (z,y)\in?R
            by auto
            from Suc(1)[OF xz] have le:z+int n\leqx.
            from zy have le2:y+1\leqz by simp
            with le show ?case by auto
    qed
    with }y\mathrm{ have nx: int n Sx by simp
    from nx have x0: x\geq0 by simp
    with nx n
    show False by simp
```


## qed

qed (insert int-SN.mono, auto)
lemma int-weak-complexity:
weak-complexity-linear-poly-order-carrier (>) 1 int-mono
by (unfold-locales, intro exI[of - (>)] exI[of - nat] conjI, rule int-complexity,
auto)

### 27.3 The Rational and Real Numbers as Carrier

```
definition delta-bound :: 'a :: floor-ceiling \(\Rightarrow{ }^{\prime} a \Rightarrow\) nat
where
    delta-bound \(d x=\operatorname{nat}(\operatorname{ceiling}(x *\) of-int \((\operatorname{ceiling}(1 / d))))\)
lemma delta-complexity:
    assumes \(d 0: d>0\) and \(d 1: d \leq d e f\)
    shows mono-matrix-carrier (delta-gt d) def (delta-bound d) delta-mono
proof -
    from \(d 0\) have \(d 00: 0 \leq d\) by simp
    define \(N\) where \(N=\) ceiling \((1 / d)\)
    let ? \(N=o f\)-int \(N::{ }^{\prime} a\)
    from \(d 0\) have \(1 / d>0\) by (auto simp: field-simps)
    with ceiling-correct[of \(1 / d]\) have \(N d: 1 / d \leq ? N\) and \(N: N>0\) unfolding
\(N\)-def by auto
    let ?nat \(=\lambda x\).nat \((\) ceiling \((x * ? N))\)
    let \(? g t=d e l t a-g t d\)
    have nn: delta-bound \(d=\) ?nat unfolding fun-eq-iff \(N\)-def by (simp add: delta-bound-def)
    from delta-interpretation \([O F\) d0 d1]
    interpret \(S N\)-strict-mono-ordered-semiring-1 def ?gt delta-mono.
    show ?thesis unfolding nn
    proof (unfold-locales)
    show ?nat \(0=0\) by auto
next
    fix \(x y\) :: 'a
    assume \(x y: x \geq y\)
    show ?nat \(x \geq\) ?nat \(y\)
            by (rule nat-mono, rule ceiling-mono, insert xy \(N\), auto simp: field-simps)
    next
    have \(1 \leq\) nat 1 by simp
    also have ... \(\leq\) ? nat 1
    proof (rule nat-mono)
            have 1 = ceiling (1 :: rat) by simp
            also have \(\ldots \leq\) ceiling \((1 * ? N)\) using \(N\) by simp
            finally show \(1 \leq\) ceiling \((1 * ? N)\).
    qed
    finally show \(1 \leq\) ?nat 1 .
    next
    fix \(x y\) :: ' \(a\)
            have ceiling \(((x+y) * ? N)=\) ceiling \((x * ? N+y * ? N)\) by (simp add:
```

field-simps)
also have $\ldots \leq$ ceiling $(x *$ ? $N)+$ ceiling $(y *$ ? $N)$ by (rule ceiling-add-le)
finally show ?nat $(x+y) \leq$ ?nat $x+$ ?nat $y$ by auto
next
fix $x::{ }^{\prime} a$ and $n::$ nat
assume $x: 0 \leq x$
interpret mono-matrix-carrier ( $>$ ) 1 nat int-mono by (rule int-complexity)
have ?nat $(x+$ of-nat $n)=$ nat $($ ceiling $(x *$ ? $N+$ of-nat $n * ? N))$
by (simp add: field-simps)
also have id: of-nat $n * ? N=o f-i n t(o f-n a t(n * n a t N))$ using $N$ by (simp add: field-simps)
also have ceiling $(x * ? N+o f-i n t(o f-n a t(n *$ nat $N)))=$ ceiling $(x * ? N)+$ of-nat ( $n *$ nat $N$ ) unfolding ceiling-add-of-int ..
also have nat $($ ceiling $(x * ? N)+$ of-nat $(n *$ nat $N))=$ ?nat $x+$ nat (int ( $n$ * nat $N$ ))
proof (rule bound-plus-of-nat)
have $x *$ ? $N \geq 0$
by (rule mult-nonneg-nonneg, insert $x N$, auto)
thus ceiling $(x * ? N) \geq 0$ by auto
qed
also have $($ nat $(\operatorname{int}(n *$ nat $N)))=n *$ nat $N$ by presburger
also have $n *$ nat $N=$ ?nat (of-nat n) using $N$ by (metis id ceiling-of-int nat-int)
finally
show ?nat $(x+$ of-nat $n)=$ ?nat $x+$ ?nat (of-nat $n)$.
next
fix $x y z::$ ' $a$
assume $*$ : delta-mono $x$ delta-gt $d y z$ and $x: 0 \leq x$
from mono $[O F * x]$
show delta-gt $d(x * y)(x * z)$.
next
fix $x::^{\prime} a$
let $? R=\{(x, y) .0 \leq y \wedge$ ? $g t x y\}$
show deriv-bound ?R $x$ (?nat $x$ ) unfolding deriv-bound-def
proof
assume $\left(\exists y .(x, y) \in ? R{ }^{\wedge}\right.$ Suc (?nat $\left.\left.x\right)\right)$
then obtain $y$ where $x y:(x, y) \in ? R \leadsto$ Suc (?nat $x$ ) ..
from $x y$ have $y: 0 \leq y$ by auto
obtain $n$ where $n: n=$ Suc (?nat $x$ ) by auto
from $x y[$ unfolded $n[$ symmetric $]]$
have $x \geq y+d *$ of-nat $n$
proof (induct $n$ arbitrary: $x y$ )
case 0 thus ? case by auto
next
case (Suc n)
from $S u c(2)$ obtain $z$ where $x z:(x, z) \in ? R \sim n$ and $z y:(z, y) \in ? R$
by auto
from $\operatorname{Suc}(1)[O F x z]$ have $l e: z+d *$ of-nat $n \leq x$.
from zy[unfolded delta-gt-def] have le2: $y+d \leq z$ by simp

```
            with le show ?case by (auto simp: field-simps)
        qed
        with \(y\) have \(n x: d *\) of-nat \(n \leq x\) by simp
        have \(0 \leq d *\) of-nat \(n\) by (rule mult-nonneg-nonneg, insert d00, auto)
        with \(n x\) have \(x 0: x \geq 0\) by auto
        have \(x d 0: 0 \leq x / d\)
        by (rule divide-nonneg-pos[OF x0 d0])
    from \(n x[\) unfolded \(n\) ]
    have \(d+d *\) of-nat (?nat \(x\) ) \(\leq x\) by (simp add: field-simps)
    with \(d 0\) have less: \(d *\) of-nat (?nat \(x\) ) \(<x\) by simp
    from \(N d d 0\) have \(1 \leq d * ? N\) by (auto simp: field-simps)
    from mult-left-mono[OF this x0]
    have \(x \leq d *(x *\) ? \(N\) ) by (simp add: ac-simps)
    also have \(\ldots \leq d *\) of-nat (?nat \(x\) )
    proof (rule mult-left-mono[OF - d00])
        show \(x *\) ? \(N \leq\) of-nat (nat 「 \(x *\) ? \(N\rceil\) ) using \(x 0\) ceiling-correct \([o f x *\) ? \(N]\)
            by (metis int-nat-eq le-cases of-int-0-le-iff of-int-of-nat-eq order-trans)
        qed
        also have \(\ldots<x\) using less.
        finally show False by simp
    qed
    qed
qed
lemma delta-weak-complexity-carrier:
    assumes d0: def >0
    shows weak-complexity-linear-poly-order-carrier ( \(>\) ) def delta-mono
proof
    fix xys :: ( \(\left.{ }^{\prime} a \times{ }^{\prime} a\right)\) list
    assume ass: \(\forall x y .(x, y) \in\) set \(x y s \longrightarrow y<x\)
    let ?cs \(=\operatorname{map}(\lambda(x, y) . x-y)\) xys
    let ? \(d s=\operatorname{def} \#\) ?cs
    define \(d\) where \(d=\operatorname{Min}(\) set ? \(d s)\)
    have \(d: d \leq\) def and dcs: \(\bigwedge x . x \in\) set \(? c s \Longrightarrow d \leq x\) unfolding \(d\)-def by auto
    have \(d \in\) set ? ds unfolding \(d\)-def by (rule Min-in, auto)
    hence \(d=\operatorname{def} \vee d \in\) set ?cs by auto
    hence \(d 0: d>0\)
        by (cases, insert d0 ass, auto simp: field-simps)
    show \(\exists\) gt bound. mono-matrix-carrier gt def bound delta-mono \(\wedge(\forall x y .(x, y)\)
\(\in\) set \(x y s \longrightarrow g t x y\) )
    by (intro exI conjI, rule delta-complexity \([O F d 0 d]\), insert dcs, force simp:
delta-gt-def)
qed
```


### 27.4 The Arctic Numbers as Carrier

lemma arctic-delta-weak-carrier:
weak-SN-both-mono-ordered-semiring-1 weak-gt-arctic-delta 1 pos-arctic-delta ..

```
lemma arctic-weak-carrier:
    weak-SN-both-mono-ordered-semiring-1 (>) 1 pos-arctic
proof -
    have SN:SN-both-mono-ordered-semiring-1 1 (>) pos-arctic ..
    show ?thesis
    by (unfold-locales, intro conjI exI, rule SN, auto)
qed
end
```


## 28 Matrix Kernel

We define the kernel of a matrix $A$ and prove the following properties.

- The kernel stays invariant when multiplying $A$ with an invertible matrix from the left.
- The dimension of the kernel stays invariant when multiplying $A$ with an invertible matrix from the right.
- The function find-base-vectors returns a basis of the kernel if $A$ is in row-echelon form.
- The dimension of the kernel of a block-diagonal matrix is the sum of the dimensions of the kernels of the blocks.
- There is an executable algorithm which computes the dimension of the kernel of a matrix (which just invokes Gauss-Jordan and then counts the number of pivot elements).

```
theory Matrix-Kernel
imports
    VS-Connect
    Missing-VectorSpace
    Determinant
begin
hide-const real-vector.span
hide-const (open) Real-Vector-Spaces.span
hide-const real-vector.dim
hide-const (open) Real-Vector-Spaces.dim
definition mat-kernel :: ' a :: comm-ring-1 mat => ' }a\mathrm{ vec set where
    mat-kernel A}={v.v\in\operatorname{carrier-vec}(\operatorname{dim}-col A)\wedgeA*vv=0v(dim-row A)
lemma mat-kernelI: assumes A\incarrier-mat nr nc v\incarrier-vec nc A *vv =
Ov nr
```

```
    shows v\in mat-kernel A
    using assms unfolding mat-kernel-def by auto
lemma mat-kernelD: assumes A\incarrier-mat nr nc v \in mat-kernel A
    shows v\incarrier-vec nc A *vv=0 0v nr
    using assms unfolding mat-kernel-def by auto
lemma mat-kernel: assumes A\incarrier-mat nr nc
    shows mat-kernel }A={v.v\incarrier-vec nc ^A*vv=\mp@subsup{O}{v}{}nr
    unfolding mat-kernel-def using assms by auto
lemma mat-kernel-carrier:
    assumes A\incarrier-mat nr nc shows mat-kernel A\subseteqcarrier-vec nc
    using assms mat-kernel by auto
lemma mat-kernel-mult-subset: assumes A:A \in carrier-mat nr nc
    and B: B\in carrier-mat n nr
    shows mat-kernel }A\subseteq\mathrm{ mat-kernel ( }B*A
proof -
    from A B have BA: B*A\in carrier-mat n nc by auto
    show ?thesis unfolding mat-kernel[OF BA] mat-kernel[OF A] using A B by
auto
qed
lemma mat-kernel-smult: assumes A:A carrier-mat nr nc
    and v:v\in mat-kernel A
    shows }a\cdotvv\in mat-kernel 
proof -
    from mat-kernelD[OFA v] have v:v\incarrier-vec nc
        and z:A*vv= Ov nr by auto
    from arg-cong[OF z, of \lambda v.a\cdotv}v]
    have }a\cdotv(A\mp@subsup{*}{v}{}v)=\mp@subsup{0}{v}{}\mathrm{ nr by auto
    also have }a\cdotvv(A\mp@subsup{*}{v}{}v)=A\mp@subsup{*}{v}{}(a\cdotvv) using Av by aut
    finally show ?thesis using v A
        by (intro mat-kernelI, auto)
qed
lemma mat-kernel-mult-eq: assumes A: A \in carrier-mat nr nc
    and B:B\in carrier-mat nr nr
    and C:C\incarrier-mat nr nr
    and inv:}C*B=1mn
    shows mat-kernel ( }B*A)=\mathrm{ mat-kernel }
proof
    from B A have BA: B*A\in carrier-mat nr nc by auto
    show mat-kernel }A\subseteq\mathrm{ mat-kernel ( }B*A\mathrm{ ) by (rule mat-kernel-mult-subset[OF A
B])
    {
        fix v
        assume v:v\in mat-kernel ( }B*A
```

from mat-kernelD[OF BA this] have $v: v \in$ carrier-vec nc and $z: B * A *_{v} v$ $=O_{v} n r$ by auto
from $\arg -c o n g\left[O F z\right.$, of $\left.\lambda v . C *_{v} v\right]$
have $C *_{v}\left(B * A *_{v} v\right)=O_{v} n r$ using $C v$ by auto
also have $C *_{v}\left(B * A *_{v} v\right)=((C * B) * A) *_{v} v$
unfolding assoc-mult-mat-vec[symmetric, OF C BA v]
unfolding assoc-mult-mat $[O F C B A]$ by simp
also have $\ldots=A *_{v} v$ unfolding inv using $A v$ by auto
finally have $v \in$ mat-kernel $A$
by (intro mat-kernelI[ $\left[\begin{array}{lll}O F & A & v\end{array}\right]$ )
\}
thus mat-kernel $(B * A) \subseteq$ mat-kernel $A$ by auto
qed
locale kernel $=$
fixes $n r$ :: nat
and $n c::$ nat
and $A$ :: 'a :: field mat
assumes $A: A \in$ carrier-mat $n r n c$
begin
sublocale $N C$ : vec-space $\operatorname{TYPE}\left({ }^{\prime} a\right) n c$.
abbreviation $V K \equiv$ NC. $V($ carrier $:=$ mat-kernel $A)$
sublocale Ker: vectorspace class-ring VK
rewrites carrier $V K=$ mat-kernel $A$
and $[$ simp]: add $V K=(+)$
and [simp]: zero $V K=0_{v} n c$
and [simp]: module.smult VK $=\left(\cdot{ }_{v}\right)$
and carrier class-ring $=U N I V$
and monoid.mult class-ring $=(*)$
and add class-ring $=(+)$
and one class-ring $=1$
and zero class-ring $=0$
and $a$-inv (class-ring $::{ }^{\prime}$ a ring $)=$ uminus
and $a$-minus (class-ring $::$ 'a ring) $=$ minus
and pow (class-ring :: 'a ring) $=(\mathcal{)}$
and finsum (class-ring $::$ 'a ring) $=$ sum
and finprod (class-ring :: 'a ring) $=$ prod
and $m$-inv (class-ring $::$ 'a ring) $x=($ if $x=0$ then div0 else inverse $x)$
apply (intro vectorspace.intro)
apply (rule NC.submodule-is-module)
apply (unfold-locales)
by (insert A mult-add-distrib-mat-vec [OF A] mult-mat-vec [OF A] mat-kernel[OF
A], auto simp: class-ring-simps)
abbreviation basis $\equiv$ Ker.basis
abbreviation span $\equiv$ Ker.span

```
abbreviation lincomb \equiv Ker.lincomb
abbreviation dim \equiv Ker.dim
abbreviation lin-dep \equiv Ker.lin-dep
abbreviation lin-indpt \equiv Ker.lin-indpt
abbreviation gen-set }\equiv\mathrm{ Ker.gen-set
lemma finsum-same:
    assumes }f:S->\mathrm{ mat-kernel A
    shows finsum VKfS = finsum NC.VfS
    using assms
proof (induct S rule: infinite-finite-induct)
    case (insert s S)
        hence base: finite S s\not\inS
            and f-VK:f:S-> mat-kernel A f s: mat-kernel A by auto
            hence f-NC:f:S->carrier-vec nc f s : carrier-vec nc using mat-kernel[OF
A] by auto
    have IH: finsum VKfS=finsum NC.V f S using insert f-VK by auto
    thus ?case
            unfolding NC.M.finsum-insert[OF base f-NC]
            unfolding Ker.finsum-insert[OF base f-VK]
            by simp
qed auto
lemma lincomb-same:
    assumes S-kernel: S\subseteq mat-kernel A
    shows lincomb a S = NC.lincomb a S
    unfolding Ker.lincomb-def
    unfolding NC.lincomb-def
    apply(subst finsum-same)
    using S-kernel Ker.smult-closed[unfolded module-vec-simps class-ring-simps] by
auto
lemma span-same:
    assumes S-kernel: S\subseteqmat-kernel A
    shows span S = NC.span S
proof (rule;rule)
    fix v assume L: v: span S show v:NC.span S
    proof -
        obtain a U where know: finite U U\subseteqS a:U->UNIV v = lincomb a U
            using L unfolding Ker.span-def by auto
            hence v:v=NC.lincomb a U using lincomb-same S-kernel by auto
            show ?thesis
                unfolding NC.span-def by (rule,intro exI conjI;fact)
    qed
    next fix v assume R:v:NC.span S show v:span S
    proof -
            obtain a U where know: finite U U\subseteqS v = NC.lincomb a U
                using R unfolding NC.span-def by auto
            hence v:v= lincomb a U using lincomb-same S-kernel by auto
```

```
    show ?thesis unfolding Ker.span-def by (rule, intro exI conjI, insert v know,
auto)
    qed
qed
lemma lindep-same:
    assumes S-kernel: S\subseteq mat-kernel A
    shows Ker.lin-dep S=NC.lin-dep S
proof
    note [simp] = module-vec-simps class-ring-simps
    { assume L: Ker.lin-dep S
        then obtain va U
    where finU}\mathrm{ : finite }U\mathrm{ and }US:U\subseteq
        and lc: lincomb a U = O v nc
        and vU:v\inU
        and av0:av\not=0
        unfolding Ker.lin-dep-def by auto
    have lc': NC.lincomb a U = O v nc
        using lc lincomb-same US S-kernel by auto
    show NC.lin-dep S unfolding NC.lin-dep-def
        by (intro exI conjI, insert finU US lc'vU av0, auto)
    }
    assume R: NC.lin-dep S
    then obtain va U
    where finU: finite U and US:U\subseteqS
        and lc:NC.lincomb a }U=\mp@subsup{O}{v}{}n
        and vU:v:U
        and av0: a v\not=0
        unfolding NC.lin-dep-def by auto
    have lc': lincomb a U = zero VK
    using lc lincomb-same US S-kernel by auto
    show Ker.lin-dep S unfolding Ker.lin-dep-def
    by (intro exI conjI,insert finU US lc'vU av0, auto)
qed
lemma lincomb-index:
    assumes i: i<nc
        and Xk:X\subseteqmat-kernel A
    shows lincomb a X $ i= sum( }\lambdax.ax*x$ i)
proof -
    have X:X\subseteqcarrier-vec nc using Xk mat-kernel-def A by auto
    show ?thesis
        using vec-space.lincomb-index[OF i X]
        using lincomb-same[OF Xk] by auto
qed
end
```

lemma find-base-vectors: assumes ref: row-echelon-form $A$

```
and A:A carrier-mat nr nc shows
set (find-base-vectors A)\subseteq mat-kernel A
Ov nc & set (find-base-vectors A)
kernel.basis nc A (set (find-base-vectors A))
card (set (find-base-vectors A)) = nc - card { i. i<nr ^ row A i\not= 0v nc}
length (pivot-positions A) = card { i.i<nr\wedge row A i\not= 0v nc}
kernel.dim nc A = nc - card { i. i<nr^ row A i\not= Ov nc}
proof -
note non-pivot-base = non-pivot-base[OF ref A]
let ?B = set (find-base-vectors A)
let ?pp = set (pivot-positions A)
from A have dim: dim-row }A=nr dim-col A=nc by aut
from ref[unfolded row-echelon-form-def] obtain p
where pivot: pivot-fun A p nc using dim by auto
note piv = pivot-funD[OF dim(1) pivot]
{
    fix v
    assume v}\in\mathrm{ ? }
    from this[unfolded find-base-vectors-def Let-def dim]
        obtain c where c: c<nc c\not\in snd'?pp
        and res: v = non-pivot-base A (pivot-positions A) c by auto
    from non-pivot-base[OF c, folded res] c
    have v\in mat-kernel Av\not=0 0}n
        by (intro mat-kernelI[OF A], auto)
}
thus sub:?B}\subseteq\mathrm{ mat-kernel }A\mathrm{ and
    Ov nc & ?B by auto
{
    fix }j\mp@subsup{j}{}{\prime
    assume j: j<nc j\not\insnd'??pp and j': j'<nc j'\not\insnd'?pp and neq: j'\not=j
    from non-pivot-base(2)[OF j] non-pivot-base(4)[OF j' j neq]
    have non-pivot-base A (pivot-positions A) j\not= non-pivot-base A (pivot-positions
A) j' by auto
}
hence inj: inj-on (non-pivot-base A (pivot-positions A))
    (set [j\leftarrow[0..<nc].j\not\in snd'? ?pp]) unfolding inj-on-def by auto
    note pp= pivot-positions[OF A pivot]
have lc: length (pivot-positions A) = card (snd' ?pp)
    using distinct-card[OF pp(3)] by auto
show card: card ?B=nc - card { i. i<nr ^ row A i\not= Ov nc}
    length (pivot-positions A)= card { i. i<nr\wedge row A i\not= Ov nc}
    unfolding find-base-vectors-def Let-def dim set-map card-image[OF inj] pp(4)[symmetric]
    unfolding pp(1) lc
proof -
    have nc - card (snd'{(i,pi)|i.i<nr\wedge pi\not=nc})
        = card {0..<nc} - card (snd'{(i,pi) |i.i<nr\wedge pi\not=nc}) by auto
    also have ... = card ({0..<nc} - snd'{(i,pi) |i.i<nr\wedgepi\not=nc})
    by (rule card-Diff-subset[symmetric], insert piv(1), force+)
    also have {0..<nc} - snd'{(i,pi) |i.i<nr\wedge pi\not=nc} = (set [j\leftarrow[0..<nc]
```

```
. \(j \notin\) snd ' \(\{(i, p i) \mid i . i<n r \wedge p i \neq n c\}])\)
        by auto
    finally show \(\operatorname{card}\left(\operatorname{set}\left[j \leftarrow[0 . .<n c] . j \notin \operatorname{snd}{ }^{\prime}\{(i, p i) \mid i . i<n r \wedge p i \neq n c\}\right]\right)\)
\(=\)
    \(n c-\operatorname{card}(\operatorname{snd} \cdot\{(i, p i) \mid i . i<n r \wedge p i \neq n c\})\) by \(\operatorname{simp}\)
    qed auto
    interpret kernel \(n r n c A\) by (unfold-locales, rule \(A\) )
    show basis: basis ?B
    unfolding Ker.basis-def
    proof (intro conjI)
    show span ? \(B=\) mat-kernel \(A\)
    proof
        show span ? \(B \subseteq\) mat-kernel \(A\)
            using sub by (rule Ker.span-is-subset2)
        show mat-kernel \(A \subseteq\) Ker.span ?B
        proof
            fix \(v\)
            assume \(v \in\) mat-kernel \(A\)
            from mat-kernelD[OF A this]
            have \(v: v \in\) carrier-vec \(n c\) and \(A v: A *_{v} v=O_{v} n r\) by auto
            let ?bi \(=\) non-pivot-base \(A\) (pivot-positions \(A)\)
            let ?ran \(=\) set \([j \leftarrow[0 . .<n c] . j \notin\) snd ' ? \(p p]\)
            let \({ }^{2} \mathrm{ran}^{\prime}=\) set \([j \leftarrow[0 . .<n c] . j \in\) snd ' ? \(p p]\)
            have dimv: dim-vec \(v=n c\) using \(v\) by auto
            define \(I\) where \(I=(\lambda b . S O M E\) i. \(i \in\) ? \(r a n \wedge\) ?bi \(i=b)\)
                \{
                    fix \(j\)
                            assume \(j: j \in\) ? ran
            hence \(\exists i . i \in\) ?ran \(\wedge\) ? bi \(i=\) ? bi \(j\) unfolding find-base-vectors-def Let-def
dim by auto
            from someI-ex[OF this] have \(I: I(? b i j) \in ? r a n\) and \(i d: ? b i(I(? b i j))\)
\(=\) ? \(b i j\) unfolding \(I-d e f\) by blast +
            from inj-onD[OF inj id \(I j\) ] have \(I(? b i j)=j\).
            \} note \(I=\) this
            define \(a\) where \(a=(\lambda b . v \$(I b))\)
            from Ker.lincomb-closed \([\) OF sub] have diml: dim-vec (lincomb a ?B) \(=n c\)
            unfolding mat-kernel-def using dim lincomb-same by auto
                have \(v=\) lincomb \(a\) ? \(B\)
                proof (rule eq-vecI; unfold diml dimv)
            fix \(j\)
            assume \(j: j<n c\)
                    have Ker.lincomb \(a\) ? \(B \$ j=\left(\sum b \in\right.\) ? \(\left.B . a b * b \$ j\right)\) by (rule lin-
comb-index[OF j sub])
            also have \(\ldots=\left(\sum i \in\right.\) ? ran. \(v \$ i *\) ? bi \(\left.i \$ j\right)\)
            proof (subst sum.reindex-cong[OF inj])
                show ? \(B=\) ? bi' ?ran unfolding find-base-vectors-def Let-def \(\operatorname{dim}\) by
auto
                    fix \(i\)
                    assume \(i \in\) ?ran
```

hence $I($ ? $b i i)=i$ by (rule $I)$
hence $a($ ?bi $i)=v \$ i$ unfolding $a$-def by simp
thus $a(? b i i) * ? b i i \$ j=v \$ i * ? b i i \$ j$ by simp
qed auto
also have $\ldots=v \$ j$
proof (cases $j \in$ ? ran)
case True
hence nmem: $j \notin$ snd'set (pivot-positions $A$ ) by auto
note $n p b=$ non-pivot-base[OF j nmem]
have $\left(\sum i \in\right.$ ? ran. $\left.v \$ i *(? b i i) \$ j\right)=$
$v \$ j * ? b i j \$ j+\left(\sum i \in ? r a n-\{j\} . v \$ i * ? b i i \$ j\right)$
by (subst sum.remove[OF - True], auto)
also have ?bi $j \$ j=1$ using npb by simp
also have $\left(\sum i \in\right.$ ? $\left.\mathrm{ran}-\{j\} . v \$ i * ? b i i \$ j\right)=0$
using insert non-pivot-base(4)[OF--jnmem] by (intro sum.neutral, auto)
finally show ?thesis by simp
next
case False
with $j$ have $j p p: j \in$ snd'? $p p$ by auto
with $j p p$ obtain $i$ where $i: i<n r$ and $j i: j=p i$ and $p i: p i<n c$
by auto
from arg-cong[OF Av, of $\lambda u . u \$ i] i A$
have $v \$ j=v \$ j-$ row $A i \cdot v$ by auto
also have row $A i \cdot v=\left(\sum j=0 . .<n c . A \$ \$(i, j) * v \$ j\right)$ unfolding scalar-prod-def using $v A i$ by auto
also have $\ldots=\left(\sum j \in\right.$ ?ran. $\left.A \$ \$(i, j) * v \$ j\right)+\left(\sum j \in\right.$ ? $r^{r a n}$. $A$ $\$ \$(i, j) * v \$ j)$
by (subst sum.union-disjoint[symmetric], auto intro: sum.cong)
also have $\left(\sum j \in\right.$ ? $\left.\mathrm{ran}^{\prime} . A \$ \$(i, j) * v \$ j\right)=$
$A \$ \$(i, p i) * v \$ j+\left(\sum j \in ?{ }^{2} a n^{\prime}-\{p i\} . A \$ \$(i, j) * v \$ j\right)$
using jpp by (subst sum.remove, auto simp: ji i pi)
also have $A \$ \$(i, p i)=1$ using $\operatorname{piv}(4)[O F i]$ pi $j i$ by auto
also have $\left(\sum j \in\right.$ ? $\left.\mathrm{ran}^{\prime}-\{p i\} . A \$ \$(i, j) * v \$ j\right)=0$
proof (rule sum.neutral, intro ballI)
fix $j^{\prime}$
assume $j^{\prime} \in ?{ }^{r} a n^{\prime}-\{p i\}$
then obtain $i^{\prime}$ where $i^{\prime}: i^{\prime}<n r$ and $j^{\prime}: j^{\prime}=p i^{\prime}$ and $p i^{\prime}: p i^{\prime} \neq n c$
and neq: $p i^{\prime} \neq p i$
unfolding $p p$ by auto
from $p i^{\prime} \operatorname{piv}[O F i]$ have $p i^{\prime}: p i^{\prime}<n c$ by auto
from $p p p i^{\prime}$ neq $j i^{\prime} i$ have $i \neq i^{\prime}$ by auto
from $\operatorname{piv}(5)\left[O F i^{\prime} p i^{\prime} i\right.$ this]
show $A \$ \$\left(i, j^{\prime}\right) * v \$ j^{\prime}=0$ unfolding $j^{\prime}$ by simp
qed
also have $\left(\sum j \in\right.$ ? ran. $\left.A \$ \$(i, j) * v \$ j\right)=-\left(\sum j \in\right.$ ? ran. $v \$ j *$ - $A \$ \$(i, j))$
unfolding sum-negf[symmetric] by (rule sum.cong, auto)
finally have $v j: v \$ j=\left(\sum j \in\right.$ ? $\left.\operatorname{ran} . v \$ j *-A \$ \$(i, j)\right)$ by simp

```
show ?thesis unfolding vj j
proof (rule sum.cong[OF refl])
    fix j}\mp@subsup{j}{}{\prime
    assume j': j' }\in\mathrm{ ?ran
    from jpp j' have }j\mp@subsup{j}{}{\prime}:j\not=\mp@subsup{j}{}{\prime}\mathrm{ by auto
    let ?map = map prod.swap (pivot-positions A)
    from ji i j have (i,j) \in set (pivot-positions A) unfolding pp by auto
    hence mem: (j,i)\in set ?map by auto
        from pp have distinct (map fst ?map) unfolding map-map o-def
prod.swap-def fst-conv by auto
                    from map-of-is-SomeI[OF this mem] have map-of ?map j = Some i
by auto
            hence ?bi j' $ j= - A $$ (i, j')
                            unfolding non-pivot-base-def Let-def dim using j jj' by auto
                            thus v$ j'*?bi j'$ j=v$ j'*-A$$(i,j) by simp
                qed
            qed
            finally show v $ j = lincomb a ?B $ j ..
            qed auto
            thus v\in span ?B unfolding Ker.span-def by auto
        qed
    qed
    show ?B \subseteq mat-kernel A by (rule sub)
    {
        fix av
        assume lc: lincomb a ?B = Ov nc and vB:v\in?B
        from vB[unfolded find-base-vectors-def Let-def dim]
            obtain j where j:j<nc j\not\in snd' ?pp and v: v= non-pivot-base A
(pivot-positions A) j
            by auto
    from arg-cong[OF lc, of \lambda v.v $ j] j
    have 0 = lincomb a ?B $ j by auto
    also have \ldots=(\sumv\in?B. a v*v$j)
        by (subst lincomb-index[OF j(1) sub], simp)
    also have ... =av*v$j+(\sumw\in?B-{v}.aw*w$j)
        by (subst sum.remove[OF - vB],auto)
    also have av*v$j=av using non-pivot-base[OF j, folded v] by simp
    also have (\sumw\in?B-{v}.aw*w$j)=0
    proof (rule sum.neutral, intro ballI)
        fix }
        assume wB:w\in?B-{v}
        from this[unfolded find-base-vectors-def Let-def dim]
        obtain j' where j': j'<nc j' & snd '?pp and w: w = non-pivot-base A
(pivot-positions A) j'
            by auto
        with wBv have j'}=j\mathrm{ by auto
        from non-pivot-base(4)[OF j' j this]
        show a w*w$j=0 unfolding w by simp
    qed
```

```
        finally have av=0 by simp
    }
    thus \neg lin-dep ?B
        by (intro Ker.finite-lin-indpt2[OF finite-set sub], auto simp: class-field-def)
    qed
    show dim = nc - card { i. i<nr^ row A i\not= Ov nc}
    using Ker.dim-basis[OF finite-set basis] card by simp
qed
```

definition kernel-dim :: ' $a$ :: field mat $\Rightarrow$ nat where [code del]: kernel-dim $A=$ kernel.dim (dim-col $A) A$
lemma (in kernel) kernel-dim [simp]: kernel-dim $A=\operatorname{dim}$ unfolding kernel-dim-def using $A$ by simp
lemma kernel-dim-code[code]:
kernel-dim $A=\operatorname{dim}-c o l A-$ length (pivot-positions (gauss-jordan-single A))
proof -
define $n r$ where $n r=$ dim-row $A$
define $n c$ where $n c=\operatorname{dim}$-col $A$
let $? B=$ gauss-jordan-single $A$
have $A: A \in$ carrier-mat $n r n c$ unfolding $n r-d e f n c$-def by auto
from gauss-jordan-single[OF A refl]
obtain $P Q$ where $A B: ? B=P * A$ and $Q P: Q * P=1_{m} n r$ and
$P: P \in$ carrier-mat $n r n r$ and $Q: Q \in$ carrier-mat $n r n r$ and $B: ? B \in$ car-
rier-mat $n r n c$
and row: row-echelon-form ?B by auto
interpret $K$ : kernel nr nc ? $B$
by (unfold-locales, rule $B$ )
from mat-kernel-mult-eq[OF A P Q QP, folded $A B]$
have kernel-dim $A=K$.dim unfolding kernel-dim-def using $A$ by simp
also have $\ldots=n c-$ length (pivot-positions ?B) using find-base-vectors[OF row
$B]$ by auto
also have $\ldots=\operatorname{dim}$-col $A-$ length (pivot-positions ? $B$ )
unfolding $n c$-def by simp
finally show ?thesis .
qed
lemma kernel-one-mat: fixes $A$ :: ' $a$ :: field mat and $n::$ nat
defines $A: A \equiv 1_{m} n$
shows
kernel.dim $n A=0$
kernel.basis $n$ A \{\}
proof -
have $A c: A \in$ carrier-mat $n n$ unfolding $A$ by auto
have pivot-fun $A$ id $n$ unfolding $A$ by (rule pivot-funI, auto)

```
    hence row: row-echelon-form A unfolding row-echelon-form-def A by auto
    have}{i.i<n\wedge row A i\not=\mp@subsup{O}{v}{}n}={0..<n} unfolding A by aut
    hence id: card {i.i<n\wedge row A i\not= Ov n} = n by auto
    interpret kernel n n A by (unfold-locales, rule Ac)
    from find-base-vectors[OF row Ac, unfolded id]
    show dim = 0 basis {} by auto
qed
lemma kernel-upper-triangular: assumes A:A\incarrier-mat n n
    and ut: upper-triangular A and 0:0 & set (diag-mat A)
    shows kernel.dim n A = 0 kernel.basis n A {}
proof -
    define ma where ma= diag-mat A
    from det-upper-triangular[OF ut A] have det A = prod-list (diag-mat A).
    also have .. }\not=0\mathrm{ using 0 unfolding ma-def[symmetric]
        by (induct ma, auto)
    finally have }\operatorname{det}A\not=0\mathrm{ .
    from det-non-zero-imp-unit[OF A this, unfolded Units-def, of ()]
        obtain B where B:B\incarrier-mat n n and BA: B*A=1m}n\mathrm{ and AB:A
* B=1 m}
            by (auto simp: ring-mat-def)
    from mat-kernel-mult-eq[OF A B A AB, unfolded BA]
    have id: mat-kernel A = mat-kernel (1m n) ..
    show kernel.dim n A = 0 kernel.basis n A {}
        unfolding id by (rule kernel-one-mat)+
qed
lemma kernel-basis-exists: assumes A: A \in carrier-mat nr nc
    shows \exists B. finite }B\wedge\mathrm{ kernel.basis nc A B
proof -
    obtain C where gj: gauss-jordan-single A =C by auto
    from gauss-jordan-single[OF A gj]
    obtain PQ where CPA:C=P*A and QP:Q*P=1m nr
            and P:P\in carrier-mat nr nr and Q:Q\in carrier-mat nr nr
            and C:C\incarrier-mat nr nc and row: row-echelon-form C
            by auto
    from find-base-vectors[OF row C] have \exists B. finite B ^ kernel.basis nc C B by
blast
    also have mat-kernel C = mat-kernel A unfolding CPA
        by (rule mat-kernel-mult-eq[OF A P Q QP])
    finally show ?thesis .
qed
```

lemma mat-kernel-mult-right-gen-set: assumes $A: A \in$ carrier-mat nr nc and $B: B \in$ carrier-mat nc nc
and $C: C \in$ carrier-mat nc nc
and $i n v: B * C=1_{m} n c$
and gen-set: kernel.gen-set $n c(A * B)$ gen and gen: gen $\subseteq$ mat-kernel $(A * B)$
shows kernel.gen-set nc $A\left(\left(\left(*_{v}\right) B\right)\right.$ 'gen $)\left(*_{v}\right) B$ 'gen $\subseteq$ mat-kernel $A$ card $\left(\left(\left(*_{v}\right) B\right)\right.$ 'gen $)=$ card gen proof -
let ? $A B=A * B$
let ? gen $=\left(\left(*_{v}\right) B\right)$ 'gen
from $A B$ have $A B: A * B \in$ carrier-mat $n r$ nc by auto
from $B$ have $\operatorname{dim} B$ : dim-row $B=n c$ by auto
from inv $B C$ have $C B: C * B=1_{m} n c$ by (metis mat-mult-left-right-inverse)
interpret $A B$ : kernel $n r n c$ ? $A B$
by (unfold-locales, rule $A B$ )
interpret $A$ : kernel nr nc $A$
by (unfold-locales, rule A)
\{
fix $w$
assume $w \in$ ?gen
then obtain $v$ where $w: w=B *_{v} v$ and $v: v \in$ gen by auto
from $v$ have $v \in$ mat-kernel ? AB using gen by auto
hence $v: v \in$ carrier-vec $n c$ and $0: ? A B *_{v} v=0_{v} n r$ unfolding mat-kernel [OF $A B]$ by auto
have ? $A B *_{v} v=A *_{v} w$ unfolding $w$ using $v A B$ by simp
with 0 have $0: A *_{v} w=O_{v} n r$ by auto
from $w B v$ have $w: w \in$ carrier-vec $n c$ by auto
from $0 w$ have $w \in$ mat-kernel $A$ unfolding mat-kernel $[O F A]$ by auto \}
thus genn: ?gen $\subseteq$ mat-kernel $A$ by auto
hence one-dir: A.span ?gen $\subseteq$ mat-kernel $A$ by fastforce
\{
fix $v v^{\prime}$
assume $v: v \in g e n$ and $v^{\prime}: v^{\prime} \in g e n$ and $i d: B *_{v} v=B *_{v} v^{\prime}$
from $v v^{\prime}$ have $v: v \in$ carrier-vec $n c$ and $v^{\prime}: v^{\prime} \in$ carrier-vec nc
using gen unfolding mat-kernel $[O F A B]$ by auto
from arg-cong[OF id, of $\left.\lambda v . C *_{v} v\right]$
have $v=v^{\prime}$ using $v v^{\prime}$
unfolding assoc-mult-mat-vec[symmetric, OF C B v] assoc-mult-mat-vec[symmetric, OF C B v〕 $C B$
by auto
\} note $i n j=t h i s$
hence inj-gen: inj-on $\left(\left(*_{v}\right)\right.$ B) gen unfolding inj-on-def by auto
show card ?gen $=$ card gen using inj-gen by (rule card-image)
\{
fix $v$
let ? $C v=C *_{v} v$
assume $v \in$ mat-kernel $A$
from mat-kernelD[OF A this] have $v: v \in$ carrier-vec nc and $0: A *_{v} v=0_{v}$ $n r$ by auto
have ? $A B *_{v}$ ? $C v=(A *(B * C)) *_{v} v$ using $A B C v$
by (subst assoc-mult-mat-vec [symmetric, OF AB C v], subst assoc-mult-mat $[O F$ A B C], simp)
also have $\ldots=O_{v} n r$ unfolding $i n v$ using 0 A $v$ by simp
finally have $0: ? A B *_{v} ? C v=O_{v} n r$ and $C v: ? C v \in$ carrier-vec nc using $C$ $v$ by auto
hence ? $C v \in$ mat-kernel ?AB unfolding mat-kernel $[O F A B]$ by auto
with gen-set have ? $C v \in A B$.span gen by auto
from this[unfolded AB.Ker.span-def] obtain $a \mathrm{gen}^{\prime}$ where
$C v: ? C v=A B . l i n c o m b$ a gen' and sub: gen' $\subseteq$ gen and fin: finite gen' by auto
let ${ }^{\text {g } g e n ' ~}=\left(\left(*_{v}\right) B\right)^{\prime} \mathrm{gen}^{\prime}$
from sub gen have $g e n^{\prime}:$ gen' $\subseteq$ mat-kernel ? AB by auto
have lin1: AB.lincomb a gen' $\in$ carrier-vec nc
using AB.Ker.lincomb-closed $[$ OF gen', of a]
unfolding mat-kernel $[O F A B]$ by (auto simp: class-field-def)
hence dim1: dim-vec (AB.lincomb a gen') $=n c$ by auto
hence dim1b: dim-vec $\left(B *_{v}\left(A B\right.\right.$. Ker.lincomb a gen $\left.\left.{ }^{\prime}\right)\right)=n c$ using $B$ by auto
from genn sub have genn': ?gen' $\subseteq$ mat-kernel $A$ by auto
from gen sub have gen'nc: gen' $\subseteq$ carrier-vec nc unfolding mat-kernel $[O F$ $A B]$ by auto
define $a^{\prime}$ where $a^{\prime}=\left(\lambda b, a\left(C *_{v} b\right)\right)$
from A.Ker.lincomb-closed $[$ OF genn $]$
have lin2: A.Ker.lincomb $a^{\prime}$ ?gen' $\in$ carrier-vec nc
unfolding mat-kernel[OF A] by (auto simp: class-field-def)
hence dim2: dim-vec (A.Ker.lincomb $a^{\prime}$ ?gen') $=n c$ by auto
have $v=B *_{v}$ ? $C v$
by (unfold assoc-mult-mat-vec[symmetric, OF B C v] inv, insert v, simp)
hence $v=B *_{v} A B$. Ker.lincomb a gen' unfolding $C v$ by simp
also have $\ldots=$ A.Ker.lincomb $a^{\prime}$ ?gen'
proof (rule eq-vecI; unfold dim1 dim1b dim2)
fix $i$
assume $i: i<n c$
with $\operatorname{dim} B$ have $i i: i<\operatorname{dim}$-row $B$ by auto
from sub inj have inj: inj-on $\left(\left(*_{v}\right) B\right.$ ) gen' unfolding inj-on-def by auto \{
fix $v$
assume $v \in g e n^{\prime}$
with gen'nc have $v: v \in$ carrier-vec nc by auto
hence $a^{\prime}\left(B *_{v} v\right)=a v$ unfolding $a^{\prime}$-def assoc-mult-mat-vec[symmetric,
OF CBv] CB by auto
$\}$ note $a^{\prime}=$ this
have A.Ker.lincomb $a^{\prime}$ ?gen' $\$ i=\left(\sum v \in\left(*_{v}\right) B\right.$ 'gen'. $\left.a^{\prime} v * v \$ i\right)$
unfolding A.lincomb-index[OF i genn $]$ by simp
also have $\ldots=\left(\sum v \in g e n^{\prime} . a v *\left(\left(B *_{v} v\right) \$ i\right)\right)$
by (rule sum.reindex-cong[OF inj refl], auto simp: $a^{\prime}$ )
also have $\ldots=\left(\sum v \in g e n^{\prime} .\left(\sum j=0 . .<n c . a v *\right.\right.$ row $\left.\left.B i \$ j * v \$ j\right)\right)$ unfolding mult-mat-vec-def $\operatorname{dimB}$ scalar-prod-def index-vec[OF $i]$
by (rule sum.cong, insert gen'nc, auto simp: sum-distrib-left ac-simps)
also have $\ldots=\left(\sum j=0 . .<n c .\left(\sum v \in\right.\right.$ gen $^{\prime} . a v *$ row $\left.\left.B i \$ j * v \$ j\right)\right)$ by (rule sum.swap)
also have $\ldots=\left(\sum j=0 . .<n c\right.$. row $\left.B i \$ j *\left(\sum v \in g e n^{\prime} . a v * v \$ j\right)\right)$ by (rule sum.cong, auto simp: sum-distrib-left ac-simps)

```
        also have ... = ( B *v AB.Ker.lincomb a gen') $ i
            unfolding index-mult-mat-vec[OF ii]
            unfolding scalar-prod-def dim1
            by (rule sum.cong[OF refl], subst AB.lincomb-index[OF - gen '], auto)
            finally show ( }B\mp@subsup{*}{v}{}\mathrm{ AB.Ker.lincomb a gen') $ i = A.Ker.lincomb a' ?gen'$$
i ..
    qed auto
    finally have v\inA.Ker.span ?gen using sub fin
        unfolding A.Ker.span-def by (auto simp: class-field-def intro!: exI[of-a]
exI[of - ?gen ])
    }
    hence other-dir:A.Ker.span ?gen \supseteq mat-kernel A by fastforce
    from one-dir other-dir show kernel.gen-set nc A (((*v)B)'gen) by auto
qed
lemma mat-kernel-mult-right-basis: assumes A: A carrier-mat nr nc
    and B:B\incarrier-mat nc nc
    and C:C\incarrier-mat nc nc
    and inv: }B*C=1mn
    and fin: finite gen
    and basis: kernel.basis nc (A*B) gen
    shows kernel.basis nc A (((*v) B)'gen)
    card (((*v) B)'gen) = card gen
proof -
    let ?AB = A*B
    let ?gen = ((*v) B)`gen
    from }AB\mathrm{ have AB:?AB E carrier-mat nr nc by auto
    from }B\mathrm{ have dimB: dim-row }B=nc\mathrm{ by auto
    from inv B C have CB:C*B=1 m nc by (metis mat-mult-left-right-inverse)
    interpret AB: kernel nr nc ? AB
    by (unfold-locales, rule AB)
    interpret A: kernel nr nc A
    by (unfold-locales, rule A)
    from basis[unfolded AB.Ker.basis-def] have gen-set: AB.gen-set gen and genAB:
gen \subseteqmat-kernel ?AB by auto
    from mat-kernel-mult-right-gen-set[OF A B C inv gen-set genAB]
    have gen: A.gen-set ?gen and sub:?gen \subseteqmat-kernel A and card:card ?gen =
card gen .
    from card show card ?gen = card gen .
    from fin have fing: finite?gen by auto
    from gen have gen: A.Ker.span ?gen = mat-kernel A by auto
    have ABC:A*B*C=A using A B C inv by simp
    from kernel-basis-exists[OF A] obtain bas where finb: finite bas and bas: A.basis
bas by auto
    from bas have bas': A.gen-set bas bas \subseteq mat-kernel A unfolding A.Ker.basis-def
by auto
    let ?bas = (*v)C`bas
    from mat-kernel-mult-right-gen-set[OF AB C B CB, unfolded ABC,OF bas']
    have bas': ?bas \subseteq mat-kernel ?AB AB.Ker.span ?bas = mat-kernel ?AB card
```

```
?bas = card bas by auto
    from finb bas have cardb: A.dim = card bas by (rule A.Ker.dim-basis)
    from fin basis have cardg: AB.dim = card gen by (rule AB.Ker.dim-basis)
    from AB.Ker.gen-ge-dim[OF - bas'(1-2)] finb bas'(3) cardb cardg
    have ineq1: card gen \leqA.dim by auto
    from A.Ker.dim-gen-is-basis[OF fing sub gen, unfolded card, OF this]
    show A.basis ?gen .
qed
```

lemma mat-kernel-dim-mult-eq-right: assumes $A: A \in$ carrier-mat nr nc
and $B: B \in$ carrier-mat nc nc
and $C: C \in$ carrier-mat nc nc
and $B C: B * C=1_{m} n c$
shows kernel.dim nc $(A * B)=$ kernel. $\operatorname{dim} n c A$
proof -
let ? $A B=A * B$
from $A B$ have $A B: ? A B \in$ carrier-mat $n r n c$ by auto
interpret $A B$ : kernel $n r n c$ ? $A B$
by (unfold-locales, rule $A B$ )
interpret $A$ : kernel $n r n c A$
by (unfold-locales, rule A)
from kernel-basis-exists $[O F A B]$ obtain bas where finb: finite bas and bas:
AB.basis bas by auto
let ?bas $=\left(\left(*_{v}\right) B\right)^{\prime}$ bas
from mat-kernel-mult-right-basis[OF A B C BC finb bas] finb
have bas': A.basis ?bas and finb': finite ?bas and card: card ?bas = card bas by
auto
show $A B$.dim $=$ A.dim unfolding $A$.Ker.dim-basis $\left[O F\right.$ finb ${ }^{\prime}$ bas $]$ AB.Ker.dim-basis $[O F$
finb bas] card ..
qed
locale vardim $=$
fixes $f$-ty :: 'a :: field itself
begin
abbreviation $M==\lambda k$. module-vec $\operatorname{TYPE}\left({ }^{\prime} a\right) k$
abbreviation span $==\lambda k$. LinearCombinations.module.span class-ring ( $M k$ )
abbreviation lincomb $==\lambda k$. module.lincomb ( $M k$ )
abbreviation lin-dep $==\lambda k$. module.lin-dep class-ring ( $M k$ )
abbreviation padr $m v==v @_{v} 0_{v} m$
definition unpadr $m v==\operatorname{vec}(\operatorname{dim-vec} v-m)(\lambda i . v \$ i)$
abbreviation padl $m v==0_{v} m @_{v} v$
definition unpadl $m v==\operatorname{vec}(\operatorname{dim}-v e c \quad v-m)(\lambda i . v \$(m+i))$
lemma unpadr-padr[simp]: unpadr $m(p a d r m v)=v$ unfolding unpadr-def by
auto
lemma unpadl-padl $[$ simp $]$ : unpadl $m(\operatorname{padl} m v)=v$ unfolding unpadl-def by auto
lemma padr-unpadr[simp]: $v:$ padr $m{ }^{\prime} U \Longrightarrow$ padr $m$ (unpadr $m v$ ) $=v$ by auto
lemma padl-unpadl $[\mathrm{simp}]: v:$ padl $m{ }^{\prime} U \Longrightarrow$ padl $m$ (unpadl $m v$ ) $=v$ by auto

```
lemma padr-image:
    assumes \(U \subseteq\) carrier-vec \(n\) shows padr \(m^{\prime} U \subseteq\) carrier-vec \((n+m)\)
proof (rule subsetI)
    fix \(v\) assume \(v\) : padr \(m\) ' \(U\)
    then obtain \(u\) where \(u: U\) and \(v m u: v=p a d r m u\) by auto
    hence \(u\) : carrier-vec \(n\) using assms by auto
    thus \(v\) : carrier-vec \((n+m)\)
        unfolding \(v m u\)
        using zero-carrier-vec \([\) of \(m]\) append-carrier-vec by metis
qed
lemma padl-image:
    assumes \(U \subseteq\) carrier-vec \(n\) shows padl \(m\) ' \(U \subseteq\) carrier-vec \((m+n)\)
proof (rule subsetI)
    fix \(v\) assume \(v\) : padl \(m\) ' \(U\)
    then obtain \(u\) where \(u: U\) and \(v m u: v=p a d l m u\) by auto
    hence \(u\) : carrier-vec \(n\) using assms by auto
    thus \(v:\) carrier-vec \((m+n)\)
        unfolding vmu
        using zero-carrier-vec [of m] append-carrier-vec by metis
qed
lemma padr-inj:
    shows inj-on (padr m) (carrier-vec \(n\) :: 'a vec set)
    apply (intro inj-onI) using append-vec-eq by auto
lemma padl-inj:
    shows inj-on (padl m) (carrier-vec \(n\) :: 'a vec set)
    apply (intro inj-onI)
    using append-vec-eq[OF zero-carrier-vec zero-carrier-vec] by auto
lemma lincomb-pad:
    fixes \(m n a\)
    assumes \(U:(U::\) 'a vec set \() \subseteq\) carrier-vec \(n\)
            and fin \(U\) : finite \(U\)
        defines goal pad unpad \(W==\) pad \(m\) (lincomb na \(W\) ) \(=\operatorname{lincomb}(n+m)(a o\)
unpad \(m\) ) (pad \(m\) ‘ \(W\) )
    shows goal padr unpadr \(U\) (is ?R) and goal padl unpadl \(U\) (is ? \(L\) )
proof -
    interpret \(N\) : vectorspace class-ring \(M n\) using vec-vs.
    interpret \(N M\) : vectorspace class-ring \(M(n+m)\) using vec-vs.
    note \([\) simp \(]=\) module-vec-simps class-ring-simps
    have ? \(R \wedge\) ? \(L\) using fin \(U U\)
```

proof (induct set:finite)
case empty thus?case
unfolding goal-def unfolding N.lincomb-def NM.lincomb-def by auto next
case (insert u $U$ )
hence fin $U$ : finite $U$
and $U: U \subseteq$ carrier-vec $n$
and $u[$ simp $]: u:$ carrier-vec $n$
and $u U: u \notin U$
and $a u U: a$ : insert $u U \rightarrow U N I V$
and $a U: a: U \rightarrow U N I V$
and au: a $u$ : UNIV
by auto
have $I H r$ : goal padr unpadr $U$ and $I H l$ : goal padl unpadl $U$ using insert(3) $U$ a $U$ by auto
note $N$-lci $=$ N.lincomb-insert2[unfolded module-vec-simps]
note $N M$-lci $=$ NM.lincomb-insert2[unfolded module-vec-simps]
have auu[simp]: a $u \cdot v u$ : carrier-vec $n$ using $a u u$ by simp
have laU[simp]: lincomb n a $U$ : carrier-vec $n$
using $N$.lincomb-closed[unfolded module-vec-simps class-ring-simps, OF $U$ $a U]$.
let ? $m 0=0 v m$ :: 'a vec
have m0: ?m0 : carrier-vec $m$ by auto
have ins: lincomb $n$ a (insert $u U)=a u \cdot{ }_{v} u+$ lincomb n a $U$ using $N$-lci[OF finU $U$ ] auU $u U u$ by auto
show ?case
proof
have padr m $\left(a u \cdot{ }_{v} u+\right.$ lincomb $n$ a $\left.U\right)=$
( $a u \cdot{ }_{v} u+$ lincomb $\left.n a U\right) @_{v}(? m 0+? m 0)$ by auto also have $\ldots=$ padr $m(a u \cdot v u)+p a d r m($ lincomb $n$ a $U)$
using append-vec-add[symmetric, OF auu laU]
using zero-carrier-vec $[$ of $m]$ by metis
also have padr $m$ (lincomb $n$ a $U$ ) $=$ lincomb $(n+m)$ ( $a$ o unpadr $m$ ) (padr $m^{\prime} U$ )
using $I H r$ unfolding goal-def.
also have padr $m\left(a u \cdot{ }_{v} u\right)=a u \cdot{ }_{v} p a d r m u$ by auto
also have $\ldots=($ a o unpadr $m)(p a d r m u) \cdot{ }_{v}$ padr $m u$ by auto
also have $\ldots+$ lincomb $(n+m)$ ( $a$ o unpadr $m$ ) (padr $m$ ' $U$ ) $=$
lincomb $(n+m)$ ( $a$ o unpadr $m$ ) (insert (padr mu) (padr m‘U))
apply(subst NM-lci[symmetric])
using fin $U u U U$ append-vec-eq $[O F u]$ by auto
also have insert (padr mu) (padr m' $U$ ) $=$ padr $m{ }^{\prime}$ insert $u U$ by auto
finally show goal padr unpadr (insert $u \quad U$ ) unfolding goal-def ins.
have $[$ simp $]: n+m=m+n$ by auto
have padl $m\left(a u \cdot_{v} u+\right.$ lincomb $n$ a $\left.U\right)=$
$(? m 0+? m 0) @_{v}(a u \cdot v u+$ lincomb $n a U)$ by auto
also have $\ldots=$ padl $m(a u \cdot v u)+$ padl $m($ lincomb $n$ a $U)$
using append-vec-add[symmetric, OF - auu laU]
using zero-carrier-vec [of m] by metis

```
        also have padl m(lincomb n a U)= lincomb (n+m) (a o unpadl m) (padl
m`}U
            using IHl unfolding goal-def.
            also have padl m (a u v}vu)=au\cdot\mp@subsup{}{v}{
            also have ... = (a o unpadl m) (padl mu) vv padl mu by auto
            also have ... + lincomb ( }n+m\mathrm{ ) (a o unpadl m) (padl m'U)=
            lincomb (n+m) (a o unpadl m) (insert (padl m u) (padl m'U))
            apply(subst NM-lci[symmetric])
            using finU uU U append-vec-eq[OF m0] by auto
            also have insert (padl mu) (padl m' U) = padl m'insert u U
            by auto
            finally show goal padl unpadl (insert u U) unfolding goal-def ins.
        qed
    qed
    thus ?R ?L by auto
qed
lemma span-pad:
    assumes U:(U::'a vec set)\subseteqcarrier-vec n
    defines goal pad m== pad m'span n U = span ( }n+m\mathrm{ ) (pad m'U)
    shows goal padr m goal padl m
proof -
    interpret N: vectorspace class-ring M n using vec-vs.
    interpret NM: vectorspace class-ring M (n+m) using vec-vs.
    { fix pad :: 'a vec }=>\mathrm{ ' 'a vec and unpad :: 'a vec }=>\mp@subsup{|}{}{\prime}a\mathrm{ vec
        assume main: \Aa.A\subseteqU\Longrightarrow finite }A
            pad (lincomb n a A) = lincomb ( }n+m\mathrm{ ) (a o unpad) (pad' A)
    assume [simp]: \v. unpad (pad v)=v
    assume pU: pad'}U\subseteq\mathrm{ carrier-vec ( }n+m
    have pad'(span n U)= span (n+m) (pad'U)
    proof (intro Set.equalityI subsetI)
        fix x assume x : pad'(span n U)
        then obtain v}\mathrm{ where v:span n U and xv:x=pad v by auto
        then obtain a A
            where AU:A\subseteqU and finA: finite A and a: a:A }->\mathrm{ UNIV
                    and vaA:v = lincomb n a A
                    unfolding N.span-def by auto
            hence A:A\subseteqcarrier-vec n using U by auto
            show }x\mathrm{ : span ( }n+m\mathrm{ ) (pad' U) unfolding NM.span-def
            proof (intro CollectI exI conjI)
                    show }x=lincomb (n+m)(a o unpad) (pad`A
                    using xv vaA main[OF AU finA] by auto
                    show pad ' }A\subseteqpad' U using AU by aut
            qed (insert finA, auto simp: class-ring-simps)
            next
            fix }x\mathrm{ assume }x\mathrm{ : span (n+m) (pad'U)
            then obtain }\mp@subsup{a}{}{\prime}\mp@subsup{A}{}{\prime
                    where }\mp@subsup{A}{}{\prime}U:\mp@subsup{A}{}{\prime}\subseteqpad' U and finA': finite A' and 的: a' : A' ' O UNIV
                        and }x\mp@subsup{a}{}{\prime}\mp@subsup{A}{}{\prime}:x=lincomb (n+m) a' A'
```

```
            unfolding NM.span-def by auto
            then obtain }A\mathrm{ where finA: finite }A\mathrm{ and }AU:A\subseteqU\mathrm{ and }\mp@subsup{A}{}{\prime}A:\mp@subsup{A}{}{\prime}=pad
A
            using finite-subset-image[OF finA' }\mp@subsup{A}{}{\prime}U]\mathrm{ by auto
            hence A:A\subseteqcarrier-vec n using U by auto
            have }\mp@subsup{A}{}{\prime}:\mp@subsup{A}{}{\prime}\subseteq\mathrm{ carrier-vec ( }n+m\mathrm{ ) using }\mp@subsup{A}{}{\prime}UpU\mathrm{ by auto
            define a where a= a' o pad
            define }\mp@subsup{a}{}{\prime\prime}\mathrm{ where }\mp@subsup{a}{}{\prime\prime}=(\mp@subsup{a}{}{\prime}\mathrm{ o pad) o unpad
            have a:a:A UNIV by auto
            have restr: restrict a' A' = restrict a'\prime A'
            proof(rule restrict-ext)
            fix }\mp@subsup{u}{}{\prime}\mathrm{ assume }\mp@subsup{u}{}{\prime}:\mp@subsup{A}{}{\prime
            then obtain u where u:A and }\mp@subsup{u}{}{\prime}=pad u unfolding A'A by aut
            thus }\mp@subsup{a}{}{\prime}\mp@subsup{u}{}{\prime}=\mp@subsup{a}{}{\prime\prime}\mp@subsup{u}{}{\prime}\mathrm{ unfolding }\mp@subsup{a}{}{\prime\prime}\mathrm{ -def a-def by auto
    qed
    have }x=lincomb (n+m) a' A' using xa'A' unfolding A'A
    also have ... = lincomb (n+m) a' }\mp@subsup{a}{}{\prime
        apply (subst NM.lincomb-restrict)
        using finA' }\mp@subsup{A}{}{\prime}\mathrm{ restr by (auto simp: module-vec-simps class-ring-simps)
    also have ... = lincomb (n+m) a' (pad'A) unfolding A'A..
    also have ... = pad (lincomb n a A)
        unfolding a't-def using main[OF AU finA] unfolding a-def by auto
        finally show x : pad'(span n U) unfolding N.span-def
        apply(rule image-eqI, intro CollectI exI conjI)
            using finA AU by (auto simp: class-ring-simps)
        qed
}
note main = this
have AUC: \bigwedgeA.A\subseteqU\LongrightarrowA\subseteqcarrier-vec n using U by simp
have [simp]: }n+m=m+n\mathrm{ by auto
show goal padr m unfolding goal-def
    apply (subst main[OF - - padr-image[OF U]])
    using lincomb-pad[OF AUC] unpadr-padr by auto
show goal padl m unfolding goal-def
    apply (subst main)
    using lincomb-pad[OF AUC] unpadl-padl padl-image[OF U] by auto
qed
lemma kernel-padr:
    assumes aA: a : mat-kernel ( }A\mathrm{ :: ' }a\mathrm{ :: field mat)
        and A:A : carrier-mat nr1 nc1
        and B: B: carrier-mat nr1 nc\mathcal{L}
        and D:D : carrier-mat nr2 nc2
    shows padr nc2 a : mat-kernel (four-block-mat A B (0m nr2 nc1) D) (is - :
mat-kernel ?ABCD)
    unfolding mat-kernel-def
proof (rule, intro conjI)
    have [simp]: dim-row A = nr1 dim-row D = nr2 dim-row ? ABCD = nr1 + nr2
using A D by auto
```

```
    have a: a : carrier-vec nc1 using mat-kernel-carrier[OF A] aA by auto
    show ?ABCD *v padr nc2 }a=
    proof
    fix i assume i:i< dim-vec ?r
    hence ?l $ i = row ?ABCD i • padr nc2 a by auto
    also have ... = 0
    proof (cases i<nr1)
        case True
            hence rows: row A i : carrier-vec nc1 row B i: carrier-vec nc2
                using A B by auto
            have row?ABCD i = row A i @ vow B i
                using row-four-block-mat(1)[OF A B - D True] by auto
            also have ... • padr nc2 a = row A i • a + row B i | 0 v nc2
                using scalar-prod-append[OF rows] a by auto
            also have row A i \cdot a= (A *vva)$i using True A by auto
            also have .. = 0 using mat-kernelD[OF A aA] True by auto
            also have row B i ( Ov nc2 = 0 using True rows by auto
            finally show ?thesis by simp
    next case False
            let ?C = 0 m nr2 nc1
            let ?i=i - nr1
            have rows:
                    row ?C ?i : carrier-vec nc1 row D ?i : carrier-vec nc\mathcal{Z}
                    using D i False A by auto
            have row ?ABCD i = row ?C ?i @ }\mp@subsup{}{v}{}\mathrm{ row D ?i
                using row-four-block-mat(2)[OF A B - D False] i A D by auto
            also have ... • padr nc2 a = row ?C ? i • a + row D ?i • 0v nc2
                    using scalar-prod-append[OF rows] a by auto
            also have row ?C ? i \cdot a = O vnc1 • a using False A i by auto
            also have \ldots=0 using a by auto
            also have row D ? i | Ov nc2 = 0 using False rows by auto
            finally show ?thesis by simp
    qed
    finally show ?l $ i=?r $ i using i by auto
    qed auto
    show padr nc2 a : carrier-vec (dim-col ?ABCD) using a A D by auto
qed
lemma kernel-padl:
    assumes dD:d \in mat-kernel ( D :: ' }a::\mathrm{ field mat)
        and A:A\in carrier-mat nr1 nc1
        and C:C\in carrier-mat nr2 nc1
        and D:D\in carrier-mat nr2 nc2
    shows padl nc1 d \in mat-kernel (four-block-mat A (0m nr1 nc2) C D) (is - \epsilon
mat-kernel ?ABCD)
    unfolding mat-kernel-def
proof (rule, intro conjI)
    have [simp]: dim-row A = nr1 dim-row D = nr2 dim-row ? ABCD = nr1 + nr2
using A D by auto
```

```
    have \(d: d\) : carrier-vec nc2 using mat-kernel-carrier \([O F D] d D\) by auto
    show ? \(A B C D *_{v}\) padl nc1 \(d=0_{v}(\) dim-row ? \(A B C D)(\) is ?l \(=? r)\)
    proof
    fix \(i\) assume \(i: i<\) dim-vec ?r
    hence ?l \(\$ i=\) row ? \(A B C D i \cdot p a d l n c 1 d\) by auto
    also have \(\ldots=0\)
    proof (cases \(i<n r 1\) )
        case True
            let ? \(B=0_{m} n r 1 n c 2\)
            have rows: row \(A\) : carrier-vec nc1 row ?B \(i\) : carrier-vec nc2
                using \(A\) True by auto
            have row ? \(A B C D i=\) row \(A i @_{v}\) row? \(B i\)
                using row-four-block-mat(1)[OF A-CD True \(]\) by auto
            also have \(\ldots \cdot\) padl nc1 \(d=\) row \(A i \cdot O_{v} n c 1+\) row ?B \(i \cdot d\)
                using scalar-prod-append[OF rows] \(d\) by auto
            also have row \(A i \cdot 0_{v} n c 1=0\) using \(A\) True by auto
            also have row ? \(B i \cdot d=0\) using True \(d\) by auto
            finally show ?thesis by simp
    next case False
            let ? \(i=i-n r 1\)
            have rows:
                    row \(C\) ? \(i\) : carrier-vec nc1 row \(D\) ? \(i:\) carrier-vec nc2
                    using \(C D i\) False \(A\) by auto
            have row ? \(A B C D i=\) row \(C\) ? \(i @_{v}\) row \(D\) ? \(i\)
                    using row-four-block-mat(2)[OF A-CD False] iA D by auto
            also have \(\ldots \cdot\) padl nc1 \(d=\) row \(C\) ? \(i \cdot 0_{v} n c 1+\) row \(D ? i \cdot d\)
                using scalar-prod-append[OF rows] \(d\) by auto
            also have row \(C\) ? \(i \cdot 0_{v} n c 1=0\) using False \(A C i\) by auto
            also have row \(D\) ? \(i \cdot d=\left(D *_{v} d\right) \$\) ? \(i\) using \(D d\) False \(i\) by auto
            also have \(\ldots=0\) using mat-kernelD \([O F D d D]\) using False \(i\) by auto
            finally show ?thesis by simp
    qed
    finally show ?l \(\$ i=? r \$ i\) using \(i\) by auto
qed auto
    show padl nc1 \(d\) : carrier-vec (dim-col ? \(A B C D\) ) using \(d A D\) by auto
qed
lemma mat-kernel-split:
    assumes \(A: A \in\) carrier-mat \(n n\)
        and \(D: D \in\) carrier-mat \(m m\)
        and \(k A D: k \in\) mat-kernel (four-block-mat \(\left.A\left(O_{m} n m\right)\left(O_{m} m n\right) D\right)\)
            (is - \(\in\) mat-kernel ?A00D)
    shows vec-first \(k n \in\) mat-kernel \(A\) (is ?a \(\in-\) )
    and vec-last \(k m \in\) mat-kernel \(D\) (is ?d \(\in\)-)
proof -
    have \(0_{v} n @_{v} 0_{v} m=O_{v}(n+m)\) by auto
    also
        have A00D: ?A00D : carrier-mat \((n+m)(n+m)\) using four-block-carrier-mat[OF
\(A D]\).
```

hence $k$ : $k$ : carrier-vec $(n+m)$ using $k A D$ mat-kernel-carrier by auto
hence ?a $@_{v}$ ? $d=k$ by simp
hence $0_{v}(n+m)=? A 00 D *_{v}\left(? a @_{v}\right.$ ?d) using mat-kernelD $[O F$ A00D] kAD
by auto
also have $\ldots=A *_{v}$ ? $a @_{v} D *_{v}$ ? d
using mult-mat-vec-split[OF A D] by auto
finally have $0_{v} n @_{v} 0_{v} m=A *_{v}$ ? $a @_{v} D *_{v}$ ? $d$.
hence $0_{v} n=A *_{v} ? a \wedge 0_{v} m=D *_{v}$ ? $d$
apply (subst append-vec-eq[of-n, symmetric]) using $A D$ by auto
thus ? $a$ : mat-kernel $A$ ? d : mat-kernel $D$ unfolding mat-kernel-def using $A D$ by auto
qed
lemma padr-padl-eq:
assumes $v: v:$ carrier-vec $n$
shows padr $m v=$ padl $n u \longleftrightarrow v=O_{v} n \wedge u=O_{v} m$
apply (subst append-vec-eq) using $v$ by auto
lemma pad-disjoint:
assumes $A: A \subseteq$ carrier-vec $n$ and $A 0: 0_{v} n \notin A$ and $B: B \subseteq$ carrier-vec $m$ shows padr $m$ ' $A \cap$ padl $n$ ' $B=\{ \}$ (is ? $A \cap ? B=-$ )
proof (intro equals0I)
fix $a b$ assume $a b: ? A \cap ? B$
then obtain $a b$
where $a b=$ padr $m a a b=p a d l n b$ and $\operatorname{dim}: a: A b: B$ by force
hence padr ma=padl $n b$ by auto
hence $a=0_{v} n$ using $\operatorname{dim} A B$ by auto
thus False using $\operatorname{dim} A 0$ by auto
qed
lemma padr-padl-lindep:
assumes $A: A \subseteq$ carrier-vec $n$ and liA: $\sim \operatorname{lin}$-dep $n A$
and $B: B \subseteq$ carrier-vec $m$ and liB: $\sim$ lin-dep $m B$
shows $\sim \operatorname{lin-dep}(n+m)(p a d r m ‘ A \cup$ padl $n ‘ B)($ is $\sim \operatorname{lin}-d e p-(? A \cup ? B))$
proof -
interpret $N$ : vectorspace class-ring $M n$ using vec-vs.
interpret $M$ : vectorspace class-ring $M m$ using vec-vs.
interpret $N M$ : vectorspace class-ring $M(n+m)$ using vec-vs.
note $[$ simp $]=$ module-vec-simps class-ring-simps
have $A B: ? A \cup ? B \subseteq$ carrier-vec $(n+m)$
using padr-image $[O F A]$ padl-image $[O F B]$ by auto
show ?thesis
unfolding NM.lin-dep-def
unfolding not-ex not-imp[symmetric] not-not
proof (intro allI impI)
fix $U f u$
assume fin $U$ : finite $U$
and $U A B: U \subseteq ? A \cup ? B$
and $f: f: U \rightarrow$ carrier class-ring
and 0 : lincomb $(n+m) f U=\mathbf{0}_{M(n+m)}$
and $u U: u: U$
let ? $U A=U \cap$ ? $A$ and ? $U B=U \cap$ ? $B$
have ? $U A \subseteq ? A ? U B \subseteq ? B$ by auto
then obtain $A^{\prime} B^{\prime}$
where $A^{\prime} A: A^{\prime} \subseteq A$ and $B^{\prime} B: B^{\prime} \subseteq B$
and $U A A^{\prime}: ? U A=p a d r m{ }^{\prime} A^{\prime}$ and $U B B^{\prime}: ? U B=$ padl $n ' B^{\prime}$
unfolding subset-image-iff by auto
hence $A^{\prime}: A^{\prime} \subseteq$ carrier-vec $n$ and $B^{\prime}: B^{\prime} \subseteq$ carrier-vec $m$ using $A B$ by auto
have fin $A^{\prime}$ : finite $A^{\prime}$ and $\operatorname{fin} B^{\prime}$ : finite $B^{\prime}$
proof -
have padr $m$ ' $A^{\prime} \subseteq U$ padl $n$ ' $B^{\prime} \subseteq U$ using $U A A^{\prime} U B B^{\prime}$ by auto
hence pre: finite (padr m' $A^{\prime}$ ) finite (padl n' $B^{\prime}$ )
using finite-subset $[O F-f i n U]$ by auto
show finite $A^{\prime}$
apply (rule finite-imageD) using subset-inj-on[OF padr-inj A] pre by auto show finite $B^{\prime}$
apply (rule finite-imageD) using subset-inj-on[OF padl-inj B] pre by auto qed
have $O_{v} n \notin A$ using $N$.zero-nin-lin-indpt $[O F-l i A] A$ class-semiring.one-zeroI by auto
hence ? $A \cap ? B=\{ \}$ using pad-disjoint $A B$ by auto
hence disj: ? $U A \cap$ ? $U B=\{ \}$ by auto
have split: $U=p a d r m$ ' $A^{\prime} \cup$ padl $n$ ' $B^{\prime}$
unfolding $U A A^{\prime}[$ symmetric $] U B B^{\prime}[$ symmetric $]$ using $U A B$ by auto
show $f u=\mathbf{0}_{\text {(class-ring::'a ring) }}$
proof -
let $? a=f \circ p a d r m$
let $? b=f \circ$ padl $n$
have $l c A^{\prime}$ : lincomb $n$ ? a $A^{\prime}$ : carrier-vec $n$ using $N$.lincomb-closed $A^{\prime}$ by auto have $l c B^{\prime}$ : lincomb $m$ ?b $B^{\prime}$ : carrier-vec $m$ using $M$.lincomb-closed $B^{\prime}$ by
auto
have $0_{v} n @_{v} 0_{v} m=O_{v}(n+m)$ by auto
also have $\ldots=$ lincomb $(n+m) f U$ using 0 by auto
also have $U=$ ? $U A \cup$ ? $U B$ using $U A B$ by auto
also have lincomb $(n+m) f \ldots=\operatorname{lincomb}(n+m) f ? U A+\operatorname{lincomb}(n+m) f$ ?UB
apply(subst NM.lincomb-union) using $A B$ fin $U$ disj by auto
also have lincomb $(n+m) f$ ? UA $=$ lincomb $(n+m)($ restrict $f$ ? $U A)$ ? UA
apply (subst NM.lincomb-restrict) using $A$ fin $U$ by auto
also have restrict $f$ ? $U A=$ restrict $(? a \circ$ unpadr $m)$ ? $U A$
apply (rule restrict-ext) by auto
also have lincomb $(n+m)$... ? $U A=\operatorname{lincomb}(n+m)(? a \circ$ unpadr $m)$ ? $U A$
apply (subst NM.lincomb-restrict) using $A$ fin $U$ by auto
also have ? $U A=p a d r m$ ' $A^{\prime}$ using $U A A^{\prime}$.
also have lincomb $(n+m)(? a \circ$ unpadr $m) \ldots=$
padr m (lincomb $n$ ?a $A^{\prime}$ )

```
        using lincomb-pad(1)[OF A' finA',symmetric].
        also have lincomb (n+m) f ?UB = lincomb (n+m) (restrict f ?UB) ?UB
        apply (subst NM.lincomb-restrict) using B finU by auto
        also have restrict f ?UB = restrict (?b ○ unpadl n) ?UB
        apply(rule restrict-ext) by auto
    also have lincomb ( }n+m)\ldots\mathrm{ ?UB = lincomb ( }n+m\mathrm{ ) (?b ○ unpadl n) ?UB
        apply(subst NM.lincomb-restrict) using B finU by auto
    also have }n+m=m+n\mathrm{ by auto
    also have ? UB = padl n ' B' using UBB'.
    also have lincomb (m+n)(?b ○ unpadl n) ... =
        padl n (lincomb m ?b B')
        using lincomb-pad(2)[OF B' finB',symmetric].
    also have padr m (lincomb n ?a A') + .. =
            (lincomb n ?a A' }+\mp@subsup{O}{v}{}n)\mp@subsup{@}{v}{}(\mp@subsup{O}{v}{}m+\mathrm{ lincomb m ?b B')
        apply (rule append-vec-add) using lcA' lcB' by auto
    also have ... = lincomb n?a A' @ }\mp@subsup{|}{v}{}\mathrm{ lincomb m?b B' using lcA' lcB' by auto
    finally have }\mp@subsup{0}{v}{}n\mp@subsup{@}{v}{}\mp@subsup{O}{v}{}m=\mathrm{ lincomb n?a A' @ lincomb m?b B'.
    hence }\mp@subsup{O}{v}{}n=\mathrm{ lincomb }n\mathrm{ ?a a A'^ }\mp@subsup{O}{v}{}m=\mathrm{ lincomb m ?b B'
            apply(subst append-vec-eq[symmetric]) using lc\mp@subsup{A}{}{\prime}lc\mp@subsup{B}{}{\prime}}\mathrm{ by auto
    from conjunct1[OF this] conjunct2[OF this]
    have ?a: : A'}->{0} ?b: 洼->{0
        using N.not-lindepD[OF liA finA'}\mp@subsup{A}{}{\prime}A
            using M.not-lindepD[OF liB finB' B'B] by auto
        hence f: padr m' 'A'->{0} f: padl n' ' B' }->{0}\mathrm{ by auto
        hence f : padr m' 'A'\cup padl n' ' ''->{0} by auto
        hence f:U->{0} using split by auto
        hence f u=0 using uU by auto
        thus ?thesis by simp
    qed
    qed
qed
end
lemma kernel-four-block-0-mat:
    assumes Adef:(A :: 'a::field mat) = four-block-mat B (0m n m) (0m m n) D
    and B:B\incarrier-mat n n
    and D:D carrier-mat m m
    shows kernel.dim (n+m)A=kernel.dim n B + kernel.dim m D
proof -
    have [simp]: n +m=m+n by auto
    have A:A\in carrier-mat (n+m) ( n+m)
        using Adef four-block-carrier-mat[OF B D] by auto
    interpret vardim TYPE('a).
    interpret MN: vectorspace class-ring M (n+m) using vec-vs.
    interpret KA: kernel n+m n+m A by (unfold-locales, rule A)
    interpret KB: kernel n n B by (unfold-locales, rule B)
    interpret KD: kernel m m D by (unfold-locales, rule D)
```

note $[$ simp $]=$ module-vec-simps
from kernel-basis-exists $[$ OF $B]$
obtain base $B$ where fin-bB: finite baseB and bB: KB.basis baseB by blast
hence bBkB: base $B \subseteq$ mat-kernel $B$ unfolding KB.Ker.basis-def by auto hence $b B c$ : base $B \subseteq$ carrier-vec $n$ using mat-kernel-carrier $[O F B]$ by auto have $b B 0: O_{v} \quad n \notin$ base $B$
using $b B$ unfolding KB.Ker.basis-def
using $K B$.Ker.vs-zero-lin-dep $[O F b B k B]$ by auto
have bBkA: padr m'baseB $\subseteq$ mat-kernel $A$
proof
fix $a$ assume $a$ : padr $m$ ‘ $b a s e B$
then obtain $b$ where $a b: a=p a d r m$ and $b: b a s e B$ by auto
hence $b$ : mat-kernel $B$ using $b B$ unfolding KB.Ker.basis-def by auto
hence padr $m b$ : mat-kernel $A$
unfolding Adef using kernel-padr $[O F-B-D]$ by auto
thus $a$ : mat-kernel $A$ using $a b$ by auto
qed
from kernel-basis-exists[OF D]
obtain base $D$ where fin-bD: finite baseD and $b D$ : KD.basis base $D$ by blast hence $b D k D$ : base $D \subseteq$ mat-kernel $D$ unfolding KD.Ker.basis-def by auto
hence $b D c$ : base $D \subseteq$ carrier-vec $m$ using mat-kernel-carrier $[O F D]$ by auto
have bDkA: padl $n$ 'base $D \subseteq$ mat-kernel $A$
proof
fix $a$ assume $a$ : padl $n$ 'baseD
then obtain $d$ where $a d: a=p a d l n d$ and $d:$ base $D$ by auto
hence $d$ : mat-kernel $D$ using $b D$ unfolding KD.Ker.basis-def by auto
hence padl n d: mat-kernel A
unfolding Adef using kernel-padl $[O F-B-D]$ by auto
thus $a$ : mat-kernel $A$ using ad by auto
qed
let ${ }^{2} B D=($ padr $m$ 'base $B \cup$ padl $n$ 'baseD $)$
have finBD: finite ? $B D$ using fin- $b B$ fin- $b D$ by auto
have KA.basis ? $B D$
unfolding KA.Ker.basis-def
proof (intro conjI Set.equalityI)
show $B D k: ? B D \subseteq$ mat-kernel $A$ using $b B k A b D k A$ by auto
also have mat-kernel $A \subseteq$ carrier-vec ( $m+n$ ) using mat-kernel-carrier $A$ by auto
finally have $B D: ? B D \subseteq \operatorname{carrier}(M(n+m))$ by auto
show mat-kernel $A \subseteq$ KA.Ker.span ?BD
unfolding KA.span-same $[O F B D k]$
proof
have $B D: ? B D \subseteq$ carrier-vec $(n+m)($ is $-\subseteq ? R)$
proof (rule)
fix $v$ assume $v$ : ? $B D$
moreover
\{ assume $v$ : padr m'baseB
then obtain $b$ where $b: b a s e B$ and $v b: v=p a d r m b$ by auto
hence $b$ : carrier-vec $n$ using $b B c$ by auto
hence $v: ? R$ unfolding $v b$ apply(subst append-carrier-vec) by auto \} moreover
\{ assume $v$ : padl $n$ 'baseD
then obtain $d$ where $d: b a s e D$ and $v d: v=p a d l n d$ by auto hence $d$ : carrier-vec $m$ using $b D c$ by auto hence $v: ? R$ unfolding $v d$ apply (subst append-carrier-vec) by auto \}
ultimately show $v$ : ?R by auto
qed
fix $a$ assume $a$ : $a$ : mat-kernel $A$
hence $a$ : carrier-vec $(n+m)$ using a mat-kernel-carrier $[O F A]$ by auto
hence $a=$ vec-first a $n @_{v}$ vec-last a $m$ (is - $=$ ?b $@_{v}$ ?d) by simp
also have $\ldots=$ padr $m ? b+$ padl $n ? d$ by auto
finally have $1: a=p a d r m ? b+p a d l n ? d$.
have subkernel: ?b : mat-kernel $B$ ?d : mat-kernel $D$ using mat-kernel-split [OF B D] a Adef by auto
hence ?b : span $n$ base $B$
using $b B$ unfolding KB.Ker.basis-def using KB.span-same by auto
hence padr $m$ ? $b: p a d r ~ m ' ~ s p a n ~ n ~ b a s e B ~ b y ~ a u t o ~$
also have padr $m$ 'span $n$ base $B=\operatorname{span}(n+m)($ padr $m$ 'base $B)$
using span-pad $[O F \quad b B c]$ by auto
also have $\ldots \subseteq \operatorname{span}(n+m)$ ? $B D$ using $M N$.span-is-monotone by auto
finally have 2: padr $m$ ?b: span $(n+m)$ ?BD.
have ?d : span m baseD
using subkernel bD unfolding KD.Ker.basis-def using KD.span-same by
auto
hence padl $n$ ? d : padl $n$ ' span $m$ baseD by auto
also have padl $n$ ' span $m$ baseD $=$ span $(n+m)($ padl $n$ 'baseD $)$
using span-pad $[O F b D c]$ by auto
also have $\ldots \subseteq \operatorname{span}(n+m)$ ? $B D$ using MN.span-is-monotone by auto
finally have 3: padl $n$ ?d : span $(n+m)$ ?BD.
have padr $m ? b+$ padl $n ? d: \operatorname{span}(n+m) ? B D$ using MN.span-add1[OF-2 3] BD by auto
thus $a \in \operatorname{span}(n+m)$ ? $B D$ using 1 by auto
qed
show KA.Ker.span ?BD $\subseteq$ mat-kernel $A$ using KA.Ker.span-closed $[$ OF BDk]
by auto
have $l i: \sim$ lin-dep $n$ base $B \sim$ lin-dep $m$ baseD
using $b B[$ unfolded KB.Ker.basis-def]
unfolding KB.lindep-same [OF bBkB]
using $b D[$ unfolded KD.Ker.basis-def]
unfolding $K D$.lindep-same $[O F b D k D]$ by auto
show ~KA.Ker.lin-dep ?BD
unfolding KA.lindep-same[OF BDk]
apply (rule padr-padl-lindep) using $b B c b D c l i$ by auto

```
    qed
    hence KA.dim = card ?BD using KA.Ker.dim-basis[OF finBD] by auto
    also have card ?BD = card (padr m'baseB) + card (padl n'baseD)
        apply(rule card-Un-disjoint)
        using pad-disjoint[OF bBc bBO bDc] fin-bB fin-bD by auto
    also have ... = card baseB + card baseD
    using card-image[OF subset-inj-on[OF padr-inj]]
    using card-image[OF subset-inj-on[OF padl-inj]] bBc bDc by auto
    also have card baseB = KB.dim using KB.Ker.dim-basis[OF fin-bB] bB by auto
    also have card baseD = KD.dim using KD.Ker.dim-basis[OF fin-bD] bD by
auto
    finally show ?thesis.
qed
lemma similar-mat-wit-kernel-dim: assumes A:A carrier-mat n n
    and wit: similar-mat-wit A B PQ
    shows kernel.dim n A = kernel.dim n B
proof -
    from similar-mat-witD2[OF A wit]
    have QP:Q*P=1m n and AB:A=P*B*Q and
        A:A\incarrier-mat n n and B:B\incarrier-mat n n and P:P\incarrier-mat
nn and Q:Q\incarrier-mat n n by auto
    from P B have PB: P*B\incarrier-mat n n by auto
    show ?thesis unfolding AB mat-kernel-dim-mult-eq-right[OF
mat-kernel-mult-eq[OF B P Q QP]
    by simp
qed
end
```


## 29 Jordan Normal Form - Uniqueness

We prove that the Jordan normal form of a matrix is unique up to permutations of the blocks. We do this via generalized eigenspaces, and an algorithm which computes for each potential jordan block (ev,n), how often it occurs in any Jordan normal form.

```
theory Jordan-Normal-Form-Uniqueness
imports
    Jordan-Normal-Form
    Matrix-Kernel
begin
lemma similar-mat-wit-char-matrix: assumes wit: similar-mat-wit A B PQ
    shows similar-mat-wit (char-matrix A ev) (char-matrix B ev) P Q
proof -
    define n}\mathrm{ where }n=\mathrm{ dim-row }
```

```
    let ?C = carrier-mat n n
    from similar-mat-witD[OF refl wit, folded n-def] have
    A:A\in?C and B:B\in?C and P:P\in?C and Q:Q\in?C
    and PQ:P*Q=1m n and QP:Q*P=1m}
    and AB:A=P*B*Q
    by auto
    have char-matrix A ev = (P*B*Q+(-ev)\cdotm}(P*Q)
    unfolding char-matrix-def n-def[symmetric] unfolding AB PQ
    by (intro eq-matI, insert P B Q, auto)
    also have (-ev) \cdotm}(P*Q)=P*((-ev)\cdotm 1m n)*Q using PQ
    by (metis mult-smult-assoc-mat mult-smult-distrib one-carrier-mat right-mult-one-mat)
    also have P*B*Q+\ldots=(P*B+P*((-ev)\cdotm 1 m}n))*Q\mathrm{ using P B
    by (intro add-mult-distrib-mat[symmetric, OF - - Q, of - n], auto)
    also have P*B+P*((-ev)\cdotm 1m n)=P*(B+(-ev) \cdotm 1m n)
    by (intro mult-add-distrib-mat[symmetric, OF P B], auto)
    also have (B+(-ev) \cdotm 1m n) = char-matrix B ev unfolding char-matrix-def
    by (intro eq-matI, insert B, auto)
    finally have AB: char-matrix A ev = P* char-matrix B ev*Q.
    show similar-mat-wit (char-matrix A ev) (char-matrix B ev) P Q
    by (intro similar-mat-witI[OF PQ QP AB - P Q], insert A B, auto)
qed
context fixes ty :: 'a :: field itself
begin
lemma dim-kernel-non-zero-jordan-block-pow: assumes a: a\not=0
    shows kernel.dim n (jordan-block n ( a :: 'a) ^m}k)=
    by (rule kernel-upper-triangular[OF pow-carrier-mat[OF jordan-block-carrier]],
    unfold jordan-block-pow, insert a, auto simp: diag-mat-def)
lemma dim-kernel-zero-jordan-block-pow:
```



```
?c)
proof -
    have A: ?A \in carrier-mat n n by auto
    hence dim: dim-row ?A = n by simp
    let ?f = \lambdai. min (k+i)n
    have piv: pivot-fun ?A ?f n unfolding jordan-block-zero-pow
        by (intro pivot-funI, auto)
    hence row: row-echelon-form ?A unfolding row-echelon-form-def by auto
    from find-base-vectors(5-6)[OF row A]
    have kernel.dim n ?A = n - length (map fst (pivot-positions ?A)) by auto
    also have length (map fst (pivot-positions ?A)) = card (fst' set (pivot-positions
?A))
    by (subst distinct-card[OF pivot-positions(2)[OF A piv], symmetric], simp)
    also have fst'set (pivot-positions ?A) ={0..< (n-?c)} unfolding pivot-positions(1)[OF
A piv]
            by force
    also have card \ldots=n - ?c by simp
```

```
    finally show ?thesis by simp
qed
definition dim-gen-eigenspace :: 'a mat => ' a => nat }=>\mathrm{ nat where
    dim-gen-eigenspace A ev k=kernel-dim((char-matrix A ev) `m}k
lemma dim-gen-eigenspace-jordan-matrix:
    dim-gen-eigenspace (jordan-matrix n-as) ev k
        =(\sumn\leftarrowmap fst [(n,e)\leftarrown-as.e=ev].min kn)
proof -
    let ?JM = \lambda n-as. jordan-matrix n-as
    let ?CM = \lambda n-as.char-matrix (?JM n-as) ev
    let ?A = \lambdan-as. (?CM n-as) \widehat{m}}
    let ?n = \lambda n-as. sum-list (map fst n-as)
    let ?C = \lambda n-as.carrier-mat (?n n-as) (?n n-as)
    let ?sum = \lambda n-as. \sumn}\leftarrow\operatorname{map fst [(n,e)\leftarrown-as.e=ev]. min kn
    let ?dim = \lambda n-as. sum-list (map fst n-as)
    let ?kdim = \lambda n-as. kernel.dim (?dim n-as) (?A n-as)
    have JM: \bigwedgen-as. ?JM n-as \in?C n-as by auto
    have CM: \bigwedgen-as.?CM n-as \in?C n-as by auto
    have A:\n-as. ?A n-as }\in\mathrm{ ?C n-as by auto
    have dimc: dim-col (?JM n-as) = ?dim n-as by simp
    interpret K: kernel ?dim n-as ?dim n-as ?A n-as
        by (unfold-locales, rule A)
    show ?thesis unfolding dim-gen-eigenspace-def K.kernel-dim
    proof (induct n-as)
        case Nil
        have ?JM Nil = 1m 0 unfolding jordan-matrix-def
            by (intro eq-matI, auto)
    hence id: ?A Nil= 1m O unfolding char-matrix-def by auto
    show ?case unfolding id using kernel-one-mat[of 0] by auto
    next
    case (Cons ne n-as')
    let ? n-as = Cons ne n-as'
    let ?d = ?dim ? n-as
    let ?d' = ?dim n-as'
    obtain n e where ne: ne=(n,e) by force
    have dim:?d = n + ? d' unfolding ne by simp
    let ?jb = jordan-block n e
    let ?cm = char-matrix ?.jb ev
    let ?a=?cm ^}\mp@subsup{}{m}{}
    have a:?a \in carrier-mat n n by simp
    from JM[of n-as'] have dim-rec: dim-row (?JM n-as') = ?d' dim-col (?JM
n-as') = ? d' by auto
    hence JM-id: ?JM ?n-as = four-block-mat ?jb ( }\mp@subsup{0}{m}{}n\mathrm{ n?d') ( }\mp@subsup{0}{m}{}\mathrm{ ? ?d' n) (?JM
n-as')
            unfolding ne jordan-matrix-def using JM[of n-as']
            by (simp add: Let-def)
            have CM-id: ?CM ?n-as = four-block-mat ?cm (0m n ?d') (0m ?d' n) (?CM
```

```
n-as')
    unfolding JM-id
    unfolding char-matrix-def
    by (intro eq-matI, auto)
    have A-id: ?A ?n-as = four-block-mat ?a ( }\mp@subsup{0}{m}{}n\mathrm{ n ?d') ( }\mp@subsup{0}{m}{}\mathrm{ ? ?d' n) (?A n-as')
        unfolding CM-id by (rule pow-four-block-mat[OF - CM], auto)
    have kdim:?kdim ?n-as = kernel.dim n ?a + ?kdim n-as'
        unfolding dim A-id
        by (rule kernel-four-block-0-mat[OF refl a A])
    also have ?kdim n-as' = ?sum n-as' by (rule Cons)
    also have kernel.dim n ?a = (if e=ev then min k n else 0)
        using dim-kernel-zero-jordan-block-pow[of n k]
            dim-kernel-non-zero-jordan-block-pow[of e-ev n k
        unfolding char-matrix-jordan-block
        by (cases e =ev,auto)
    also have ... + ?sum n-as' = ?sum ?n-as unfolding ne by auto
    finally show ?case .
    qed
qed
lemma dim-gen-eigenspace-similar: assumes sim: similar-mat A B
    shows dim-gen-eigenspace A = dim-gen-eigenspace B
proof (intro ext)
    fix ev k
    define }n\mathrm{ where }n=\mathrm{ dim-row }
    from sim[unfolded similar-mat-def] obtain P Q where
        wit: similar-mat-wit A B PQ by auto
    let ?C = carrier-mat n n
    from similar-mat-witD[OF refl wit, folded n-def]
        have A:A\in?C and B:B\in?C and P:P\in?C and Q:Q\in?C
        and}PQ:P*Q=\mp@subsup{1}{m}{}n\mathrm{ and }QP:Q*P=1m 
        by auto
    from similar-mat-wit-pow[OF similar-mat-wit-char-matrix[OF wit, of ev], of k]
    have wit: similar-mat-wit (char-matrix A ev }\mp@subsup{}{m}{*}k\mathrm{ ) (char-matrix B ev _
Q.
    from A B have cA: char-matrix A ev \widehat{m}}k\in\mathrm{ carrier-mat n n
        and cB: char-matrix B ev \widehat{m}}k\in\mathrm{ carrier-mat n n by auto
    hence dim: dim-col (char-matrix A ev \widehat{m}}k\mathrm{ ) = n dim-col (char-matrix Bev }\mp@subsup{\widehat{m}}{m}{
k)=n by auto
    have dim-gen-eigenspace A ev k=kernel-dim(char-matrix A ev `}\mp@subsup{m}{k}{k}\mathrm{ )
    unfolding dim-gen-eigenspace-def using A by simp
    also have ... = kernel-dim (char-matrix B ev \}\mp@subsup{}{m}{}k\mathrm{ ) unfolding kernel-dim-def
dim
    by (rule similar-mat-wit-kernel-dim[OF cA wit])
    also have ... = dim-gen-eigenspace B evk
        unfolding dim-gen-eigenspace-def using B by simp
    finally show dim-gen-eigenspace A ev k=dim-gen-eigenspace B ev k.
qed
```

```
lemma dim-gen-eigenspace: assumes jordan-nf A n-as
    shows dim-gen-eigenspace A ev k
        =(\sumn\leftarrowmap fst [(n,e)\leftarrown-as.e=ev].min kn)
proof -
    from assms[unfolded jordan-nf-def]
    have sim: similar-mat A (jordan-matrix n-as) by auto
    from dim-gen-eigenspace-jordan-matrix[of n-as, folded dim-gen-eigenspace-similar[OF
this]]
    show ?thesis.
qed
```

definition compute-nr-of-jordan-blocks $::$ ' $a$ mat $\Rightarrow{ }^{\prime} a \Rightarrow n a t \Rightarrow n a t$ where
compute-nr-of-jordan-blocks $A$ ev $k=2 *$ dim-gen-eigenspace $A$ ev $k-$
dim-gen-eigenspace $A$ ev $(k-1)$-dim-gen-eigenspace $A$ ev (Suc k)
This lemma finally shows uniqueness of JNFs. Take an arbitrary JNF of
a matrix $A$, (encoded by the list of Jordan-blocks $n$-as), then then number
of occurrences of each Jordan-Block in $n$-as is uniquely determined, namely
by local.compute-nr-of-jordan-blocks. The condition $k \neq 0$ is to ensure that
we do not count blocks of dimension 0 .
lemma compute-nr-of-jordan-blocks: assumes jnf: jordan-nf A n-as
and no-0: $k \neq 0$
shows compute-nr-of-jordan-blocks $A$ ev $k=$ length $($ filter $((=)(k, e v)) n$-as)
proof -
from no-0 obtain $k 1$ where $k$ : $k=$ Suc $k 1$ by (cases $k$, auto)
let $? k=$ Suc $k 1$ let $? k 2=$ Suc $? k$
let ? dim $=$ dim-gen-eigenspace $A$ ev
let ? sizes $=$ map fst $[(n, e) \leftarrow n$-as $\cdot e=e v]$
define sizes where sizes $=$ ? sizes
let ?two $=$ length $($ filter $((=)(k, e v)) n$-as $)$
have compute-nr-of-jordan-blocks A ev $k=$
?dim ? $k+$ ?dim ? $k-$ ? dim k1 - ?dim ?k2 unfolding compute-nr-of-jordan-blocks-def
$k$ by $\operatorname{simp}$
also have $\ldots=$ length (filter $((=) k)$ ?sizes $)$
unfolding dim-gen-eigenspace $[O F$ jnf] $k$ sizes-def $[$ symmetric $]$
proof (rule sym, induct sizes)
case (Cons s sizes)
show ?case
proof (cases $s=? k$ )
case True
let ?sum $=\lambda k$ sizes. sum-list $(\operatorname{map}(\min k)$ sizes $)$
let ?len $=\lambda$ sizes. length $($ filter $((=)$ ? $)$ sizes $)$
have len: ?len ( $s$ \# sizes ) = Suc (?len sizes) unfolding True by simp
have $I H$ : ?len sizes $=$ ?sum ?k sizes + ?sum ?k sizes -
?sum $k 1$ sizes - ?sum ? $k 2$ sizes by (rule Cons)
have ?sum ? $k$ ( $\#$ \# sizes $)+$ ?sum ? $k$ ( $s \#$ sizes $)-$
? sum k1 ( $s$ \# sizes) - ?sum ?k2 ( $s$ \# sizes)
$=$ Suc (?sum ?k sizes + ?sum ?k sizes $)-$

```
            (?sum k1 sizes + ?sum ?k2 sizes)
            using True by simp
            also have \ldots=Suc (?sum ?k sizes + ?sum ?k sizes - (?sum k1 sizes +
?sum ?k2 sizes))
            by (rule Suc-diff-le, induct sizes, auto)
            also have ... = ?len (s # sizes) unfolding len IH by simp
            finally show ?thesis by simp
    qed (insert Cons,auto)
    qed simp
    also have ... = length (filter ((=) (k,ev)) n-as) by (induct n-as, force+)
    finally show ?thesis.
qed
definition compute-set-of-jordan-blocks :: 'a mat 知'a=>(nat \times 'a)list where
    compute-set-of-jordan-blocks A ev \equivlet
        k= Polynomial.order ev (char-poly A);
        as = map (dim-gen-eigenspace A ev) [0 ..< Suc (Suc k)];
        cards = map (\lambdak. (k, 2*as!k-as! (k-1) - as!Suc k)) [1 ..< Suc k]
        in map (\lambda (k,c). (k,ev)) (filter ( }\lambda(k,c).c\not=0) cards
lemma compute-set-of-jordan-blocks: assumes jnf: jordan-nf A n-as
    shows set (compute-set-of-jordan-blocks A ev) = set n-as \capUNIV }\times{ev} (i
?C=? 'N')
proof -
    let ?N = set n-as \capUNIV }\times{ev}-{0}\timesUNI
    have }N:?\mp@subsup{N}{}{\prime}=?N\mathrm{ using jnf[unfolded jordan-nf-def] by force
    note cjb = compute-nr-of-jordan-blocks[OF jnf]
    note d = compute-set-of-jordan-blocks-def Let-def
    define kk}\mathrm{ where kk=Polynomial.order ev (char-poly A)
    define as where as = map (dim-gen-eigenspace A ev) [0 ..< Suc (Suc kk)]
    define cards where cards = map (\lambdak. (k,2*as!k-as! (k-1) - as!Suc
k))[1 ..<Suc kk]
    have C: ?C = set (map ( }\lambda(k,c).(k,ev))(filter ( \lambda (k,c).c\not=0) cards))
        unfolding d as-def kk-def cards-def by (rule refl)
    {
        fix }
        assume i< Suc (Suc kk)
        hence as !i=dim-gen-eigenspace A ev i
            unfolding as-def by (auto simp del: upt-Suc)
    } note as=this
    have cards:cards = map ( }\lambdak.(k,\mathrm{ compute-nr-of-jordan-blocks A ev k)) [1 ..<
Suc kk]
    unfolding cards-def
    by (rule map-cong[OF refl], insert as, unfold compute-nr-of-jordan-blocks-def,
auto)
    have C: ?C ={(k,ev)|k.compute-nr-of-jordan-blocks A ev k\not=0^k\not=0^
k<Suc kk }
    unfolding C cards by force
```

```
{
    fix }
    have }(k,ev)\in?C\longleftrightarrow\longleftrightarrow(k,ev)\in?
    proof (cases k=0)
        case True
        thus ?thesis unfolding C by auto
    next
        case False
        show ?thesis
        proof (cases k < Suc kk)
            case True
            have length (filter ((=) (k,ev)) n-as) \not=0\longleftrightarrow
            set (filter }((=)(k,ev))n-as)\not={} by blas
        have }(k,ev)\in?N\longleftrightarrow set (filter ((=) (k,ev)) n-as)\not={} using False by
auto
            also have ...\longleftrightarrow length (filter ((=) (k,ev)) n-as) \not=0 by blast
            also have ...\longleftrightarrow compute-nr-of-jordan-blocks A ev k\not=0
                    unfolding compute-nr-of-jordan-blocks[OF jnf False] by simp
            also have }\ldots\longleftrightarrow(k,ev)\in\mathrm{ ?C unfolding C using False True by auto
            finally show ?thesis by auto
        next
            case False
            hence ( }k,ev)\not\in?C\mathrm{ unfolding C by auto
            moreover from False kk-def have k: k> Polynomial.order ev (char-poly
A) by auto
            with jordan-nf-block-size-order-bound[OF jnf, of k ev]
            have (k,ev) & ?N by auto
            ultimately show ?thesis by simp
        qed
    qed
    }
    thus ?thesis unfolding C N[symmetric] by auto
qed
lemma jordan-nf-unique: assumes jordan-nf (A :: 'a mat) n-as and jordan-nf A
m-bs
shows set n-as = set m-bs
proof -
    from compute-set-of-jordan-blocks[OF assms(1), unfolded compute-set-of-jordan-blocks[OF
assms(2)]]
    show ?thesis by auto
qed
```

One might get more fine-grained and prove the uniqueness lemma for multisets, so one takes multiplicities into account. For the moment we don't require this for complexity analysis, so it remains as future work.
end
end

## 30 Spectral Radius Theory

The following results show that the spectral radius characterize polynomial growth of matrix powers.
theory Spectral-Radius
imports
Jordan-Normal-Form-Existence
begin
definition spectrum $A=$ Collect (eigenvalue $A$ )
lemma spectrum-root-char-poly: assumes $A:\left(A::{ }^{\prime} a\right.$ :: field mat $) \in$ carrier-mat $n n$
shows spectrum $A=\{k$. poly (char-poly $A) k=0\}$
unfolding spectrum-def eigenvalue-root-char-poly[OF A, symmetric] by auto
lemma card-finite-spectrum: assumes $A$ : ( $A$ :: ' $a$ :: field mat $) \in$ carrier-mat $n n$ shows finite (spectrum A) card (spectrum $A) \leq n$
proof -
define $C P$ where $C P=$ char-poly $A$
from spectrum-root-char-poly[OF $A]$ have $i d$ : spectrum $A=\{k$. poly $C P k=$ 0\}
unfolding $C P$-def by auto
from degree-monic-char-poly[OF $A$ ] have $d$ : degree $C P=n$ and $c$ : coeff $C P n$ $=1$
unfolding $C P$-def by auto
from $c$ have $C P \neq 0$ by auto
from poly-roots-finite[OF this]
show finite (spectrum $A$ ) unfolding $i d$.
from poly-roots-degree $[O F\langle C P \neq 0\rangle]$
show card (spectrum $A$ ) $\leq n$ unfolding $i d$ using $d$ by simp
qed
lemma spectrum-non-empty: assumes $A:(A$ :: complex mat $) \in$ carrier-mat $n n$ and $n: n>0$
shows spectrum $A \neq\{ \}$
proof -
define $C P$ where $C P=$ char-poly $A$
from spectrum-root-char-poly[OF A] have id: spectrum $A=\{k$. poly $C P k=$ $0\}$
unfolding $C P$-def by auto
from degree-monic-char-poly[OF A] have $d$ : degree $C P>0$ using $n$ unfolding $C P$-def by auto
hence $\neg$ constant (poly CP) by (simp add: constant-degree)
from fundamental-theorem-of-algebra[OF this] show ?thesis unfolding id by auto
qed
definition spectral-radius :: complex mat $\Rightarrow$ real where

```
    spectral-radius A = Max (norm'spectrum A)
    lemma spectral-radius-mem-max: assumes A:A\incarrier-mat n n
    and n: n>0
    shows spectral-radius A\in norm'spectrum A (is ?one)
    a\in norm'spectrum A \Longrightarrowa\leqspectral-radius A
proof -
    define SA where SA= norm' spectrum A
    from card-finite-spectrum[OF A]
    have fin: finite SA unfolding SA-def by auto
    from spectrum-non-empty[OF A n] have ne: SA \not={} unfolding SA-def by
auto
    note d = spectral-radius-def SA-def[symmetric] Sup-fin-Max[symmetric]
    show ?one unfolding d
        by (rule Sup-fin.closed[OF fin ne], auto simp: sup-real-def)
    assume a \in norm'spectrum A
    thus a\leqspectral-radius A unfolding d
        by (intro Sup-fin.coboundedI[OF fin])
qed
```

If spectral radius is at most 1, and JNF exists, then we have polynomial growth.
lemma spectral-radius-jnf-norm-bound-le-1: assumes $A: A \in$ carrier-mat $n n$ and sr-1: spectral-radius $A \leq 1$
and jnf-exists: $\exists n$-as. jordan-nf $A$ n-as
shows $\exists c 1 c \mathcal{2} . \forall k$. norm-bound $\left(A \widehat{m}_{m} k\right)(c 1+c \mathcal{Z} *$ of-nat $k \wedge(n-1))$
proof -
let $? p=$ char-poly $A$
from char-poly-factorized $[O F A]$ obtain as where $c A$ : char-poly $A=\left(\prod a \leftarrow a s\right.$. [:- a, 1:])
and len: length as $=n$ by auto
show ?thesis
proof (rule factored-char-poly-norm-bound[OF A cA jnf-exists])
fix $a$
show length $($ filter $((=) a)$ as $) \leq n$ using len by auto
assume $a \in$ set as
from linear-poly-root[OF this]
have poly ? $p a=0$ unfolding $c A$ by simp
with spectrum-root-char-poly[OF A]
have mem: norm $a \in$ norm ' spectrum $A$ by auto
with card-finite-spectrum $[O F A]$ have $n>0$ by (cases $n$, auto)
from spectral-radius-mem-max(2)[OF A this mem] sr-1
show norm $a \leq 1$ by auto
qed
qed
If spectral radius is smaller than 1, and JNF exists, then we have a constant bound.
lemma spectral-radius-jnf-norm-bound-less-1: assumes $A: A \in$ carrier-mat $n n$
and sr-1: spectral-radius $A<1$
and jnf-exists: $\exists$ n-as. jordan-nf $A$ n-as
shows $\exists c . \forall k$. norm-bound $\left(A \widehat{m}_{m} k\right) c$
proof -
let ? $p=$ char-poly $A$
from char-poly-factorized $[O F A]$ obtain as where cA: char-poly $A=\left(\prod a \leftarrow a s\right.$.
[:-a, 1:]) by auto
have $\exists c 1 c 2 . \forall k$. norm-bound $\left(A_{m} k\right)(c 1+c 2 *$ of-nat $k \wedge(0-1))$
proof (rule factored-char-poly-norm-bound[OF A cA jnf-exists])
fix $a$
assume $a \in$ set as
from linear-poly-root[OF this]
have poly ? $p a=0$ unfolding $c A$ by simp
with spectrum-root-char-poly[OF A]
have mem: norm $a \in$ norm 'spectrum $A$ by auto
with card-finite-spectrum [OF A] have $n>0$ by (cases $n$, auto)
from spectral-radius-mem-max(2)[OF A this mem] sr-1
have $l t$ : norm $a<1$ by auto
thus norm $a \leq 1$ by auto
from $l t$ show norm $a=1 \Longrightarrow$ length $($ filter $((=) a)$ as $) \leq 0$ by auto
qed
thus ?thesis by auto
qed
If spectral radius is larger than 1 , than we have exponential growth.
lemma spectral-radius-gt-1: assumes $A: A \in$ carrier-mat $n n$
and $n: n>0$
and sr-1: spectral-radius $A>1$
shows $\exists v c . v \in$ carrier-vec $n \wedge$ norm $c>1 \wedge v \neq 0_{v} n \wedge A \widehat{m}_{m} k *_{v} v=c^{\wedge} k$
$\cdot v v$
proof -
from sr-1 spectral-radius-mem-max[OF An] obtain ev
where $e v: e v \in \operatorname{spectrum} A$ and gt: norm ev>1 by auto
from ev[unfolded spectrum-def eigenvalue-def[abs-def]] obtain $v$ where ev: eigenvector $A v$ ev by auto
from eigenvector-pow[OF A this] this[unfolded eigenvector-def] A gt
show ?thesis
by (intro exI $[o f-v]$, intro exI $[o f-e v]$, auto)
qed
If spectral radius is at most 1 for a complex matrix, then we have polynomial growth.
lemma spectral-radius-jnf-norm-bound-le-1-upper-triangular: assumes $A$ : ( $A::$ complex mat) $\in$ carrier-mat $n n$
and sr-1: spectral-radius $A \leq 1$
shows $\exists c 1 c 2 . \forall k$. norm-bound $\left(A{ }_{m} k\right)(c 1+c \mathcal{2} *$ of-nat $k \wedge(n-1))$
by (rule spectral-radius-jnf-norm-bound-le-1[OF A sr-1],
insert char-poly-factorized $[O F A]$ jordan-nf-exists $[O F A]$, blast)
If spectral radius is less than 1 for a complex matrix, then we have a
constant bound.
lemma spectral-radius-jnf-norm-bound-less-1-upper-triangular: assumes $A$ : $(A::$ complex mat) $\in$ carrier-mat $n n$
and sr-1: spectral-radius $A<1$
shows $\exists c$. $\forall k$. norm-bound $\left(A{ }_{m} k\right) c$
by (rule spectral-radius-jnf-norm-bound-less-1[OF A sr-1], insert char-poly-factorized $[O F A]$ jordan-nf-exists[OF A], blast)

And we can also get a quantative approximation via the multiplicity of the eigenvalues.

```
lemma spectral-radius-poly-bound: fixes A :: complex mat
    assumes A:A\in carrier-mat n n
    and sr-1: spectral-radius A\leq1
    and eq-1: \bigwedgeev k. poly (char-poly A) ev = 0 \Longrightarrow norm ev = 1 \LongrightarrowPolyno-
mial.order ev (char-poly A) \leqd
    shows \exists c1 c2. }\forall\textrm{k}\mathrm{ . norm-bound ( }A\mp@subsup{\widehat{m}}{}{~}k)(c1+c\mathcal{L}*\mathrm{ of-nat k^ }(d-1)
proof -
    {
        fix ev
        assume poly (char-poly A) ev = 0
        with eigenvalue-root-char-poly[OF A] have ev: eigenvalue A ev by simp
        hence norm ev \in norm'spectrum A unfolding spectrum-def by auto
        from spectral-radius-mem-max(2)[OF A eigenvalue-imp-nonzero-dim[OF A ev]
this] sr-1
    have norm ev \leq1 by auto
    } note le-1 = this
    let ?p = char-poly A
    from char-poly-factorized[OF A] obtain as where cA: char-poly A = (\proda\leftarrowas.
[:- a, 1:])
    and lenn: length as = n by auto
    from degree-monic-char-poly[OF A] have deg: degree (char-poly A) =n by auto
    show ?thesis
    proof (rule factored-char-poly-norm-bound[OF A cA jordan-nf-exists[OF A]], rule
cA)
    fix ev
    assume ev \in set as
    hence root: poly (char-poly A) ev = 0 unfolding cA by (rule linear-poly-root)
    from le-1[OF root] show norm ev \leq 1.
    let ?k = length (filter ((=) ev) as)
    have len: length (filter ((=) (-ev)) (map uminus as)) = length (filter ((=) ev)
as)
            by (induct as, auto)
    have prod: (\proda\leftarrowmap uminus as. [:a, 1:]) = (\proda\leftarrowas. [:- a, 1:])
        by (induct as, auto)
    have dvd:[:- ev, 1:] ^ ?k dvd char-poly A unfolding cA using
        poly-linear-exp-linear-factors-rev[of - ev map uminus as]
        unfolding len prod.
    from <ev \in set as` deg lenn
    have degree (char-poly A) \not=0 by (cases as, auto)
```

```
    hence char-poly A\not=0 by auto
    from order-max[OF dvd this] have k:?k\leq Polynomial.order ev (char-poly A)
    assume norm ev = 1
    from eq-1[OF root this] }
    show ?k}\leqd\mathrm{ by simp
    qed
qed
end
```


## 31 Missing Lemmas of List

```
theory DL-Missing-List
imports Main
begin
lemma nth-map-zip:
assumes i< length xs
assumes i< length ys
shows map f(zip xs ys)!i=f(xs!i, ys!i)
    using nth-zip nth-map length-zip by (simp add:assms(1) assms(2))
lemma nth-map-zip2:
assumes i< length (map f (zip xs ys))
shows map f(zip xs ys)!i=f(xs!i, ys!i)
    using nth-zip nth-map length-zip assms by simp
fun find-first where
find-first a [] = undefined 
find-first a (x # xs) = (if x = a then 0 else Suc (find-first a xs))
lemma find-first-le:
assumes a\in set xs
shows find-first a xs < length xs
using assms proof (induction xs)
    case (Cons x xs)
    then show ?case
        using find-first.simps(2) nth-Cons-0 nth-Cons-Suc set-ConsD by auto
qed auto
lemma nth-find-first:
assumes a\in set xs
shows xs!(find-first a xs) =a
using assms proof (induction xs)
    case (Cons x xs)
    then show ?case
    using find-first.simps(2) nth-Cons-0 nth-Cons-Suc set-ConsD by auto
```

```
qed auto
```

lemma find－first－unique：
assumes distinct xs
and $i<$ length $x s$
shows find－first（ $x s!i$ ）xs $=i$
using assms proof（induction xs arbitrary：i）
case（Cons $x$ xs $i$ ）
then show ？case by（cases $i$ ；auto）
qed auto
end

## 32 Matrix Rank

```
theory DL-Rank
imports VS-Connect DL-Missing-List
    Determinant
    Missing-VectorSpace
begin
lemma (in vectorspace) full-dim-span:
assumes S\subseteq carrier V
and finite S
and vectorspace.dim K (span-vs S)= card S
shows lin-indpt S
proof -
    have vectorspace K (span-vs S)
    using field.field-axioms vectorspace-def submodule-is-module[OF span-is-submodule[OF
assms(1)]] by metis
    have S\subseteqcarrier (span-vs S) by (simp add: assms(1) in-own-span)
    have LinearCombinations.module.span K (vs (span S)) S = carrier (vs (span
S))
    using module.span-li-not-depend[OF - span-is-submodule[OF assms(1)]]
    by (simp add: assms(1) in-own-span)
    have vectorspace.basis K (vs (span S)) S
            using vectorspace.dim-gen-is-basis[OF <vectorspace K (span-vs S)〉〈finite S〉
<S\subseteq carrier (span-vs S)>
            <LinearCombinations.module.span K (vs (span S)) S = carrier (vs (span S))>]
<vectorspace.dim K (span-vs S) = card S〉
            by simp
    then have LinearCombinations.module.lin-indpt K (vs (span S)) S
            using vectorspace.basis-def[OF〈vectorspace K (span-vs S)>] by blast
    then show ?thesis using module.span-li-not-depend[OF - span-is-submodule[OF
assms(1)]]
    by (simp add: assms(1) in-own-span)
qed
lemma (in vectorspace) dim-span:
```


## assumes $S \subseteq$ carrier $V$

and finite $S$
and maximal $U(\lambda T . T \subseteq S \wedge$ lin-indpt $T)$
shows vectorspace.dim $K$ (span-vs $S)=$ card $U$
proof -
have lin-indpt $U U \subseteq S$ by (metis assms(3) maximal-def)+
then have $U \subseteq$ span $S$ using in-own-span[OF assms(1)] by blast
then have lin-indpt: LinearCombinations.module.lin-indpt $K$ (span-vs $S$ ) $U$
using module.span-li-not-depend(2)[OF $\langle U \subseteq$ span $S\rangle]$ lin-indpt $U\rangle \operatorname{assms}(1)$
span-is-submodule by blast
have span $U=\operatorname{span} S$
proof (rule ccontr)
assume span $U \neq \operatorname{span} S$
have span $U \subseteq$ span $S$ using span-is-monotone $\langle U \subseteq S\rangle$ by metis
then have $\neg S \subseteq$ span $U$ by (meson $\langle U \subseteq S\rangle\langle$ span $U \neq \operatorname{span} S\rangle \operatorname{assms}(1)$
span-is-submodule
span-is-subset subset-antisym subset-trans)
then obtain $s$ where $s \in S s \notin$ span $U$ by blast
then have lin-indpt $(U \cup\{s\})$ using lindep-span
by (meson $\langle U \subseteq S\rangle\langle l i n-i n d p t ~ U 〉 \operatorname{assms}(1)$ lin-dep-iff-in-span rev-subsetD
span-mem subset-trans)
have $s \notin U$ using $\langle U \subseteq S\rangle\langle s \notin$ span $U\rangle$ assms(1) span-mem by auto
then have $(U \cup\{s\}) \subseteq S \wedge$ lin-indpt $(U \cup\{s\})$ using $\langle U \subseteq S\rangle\langle$ lin-indpt $(U \cup$
$\{s\})\rangle\langle s \in S\rangle$ by auto
then have $\neg$ maximal $U(\lambda T . T \subseteq S \wedge$ lin-indpt $T)$
unfolding maximal-def using Un-subset-iff $\langle s \notin U\rangle$ insert-subset order-refl by auto
then show False using assms by metis
qed
then have span:LinearCombinations.module.span $K($ vs $($ span $S)) U=\operatorname{span} S$ using module.span-li-not-depend $[O F\langle U \subseteq$ span $S\rangle]$
by (simp add: LinearCombinations.module.span-is-submodule assms(1) mod-ule-axioms)
have vectorspace $K(v s($ span $S))$
using field.field-axioms vectorspace-def submodule-is-module[OF span-is-submodule[OF $\operatorname{assms}(1)]$ ] by metis
then have vectorspace.basis $K(v s($ span $S)) U$ using vectorspace.basis-def[OF〈vectorspace $K(v s($ span $S))\rangle]$
by (simp add: span $\langle U \subseteq$ span $S\rangle$ lin-indpt)
then show? thesis
using $\langle U \subseteq S\rangle\langle v e c t o r s p a c e ~ K(v s(\operatorname{span} S))\rangle$ assms(2) infinite-super vec-
torspace.dim-basis by blast
qed
definition (in vec-space) rank ::'a mat $\Rightarrow$ nat
where rank $A=$ vectorspace.dim class-ring (span-vs (set (cols $A))$ )
lemma (in vec-space) rank-card-indpt:
assumes $A \in$ carrier-mat $n n c$

```
assumes maximal S ( }\lambdaT.T\subseteq\operatorname{set}(\mathrm{ cols A)}\wedge lin-indpt T
shows rank A = card S
proof -
    have set (cols A)\subseteqcarrier-vec n using cols-dim assms(1) by blast
    have finite (set (cols A)) by blast
    show ?thesis using dim-span[OF <set (cols A)\subseteq carrier-vec n><finite (set (cols
A))> assms(2)]
    unfolding rank-def by blast
qed
lemma maximal-exists-superset:
    assumes finite S
    assumes maxc: }\bigwedgeA.PA\LongrightarrowA\subseteqS\mathrm{ and }P
    shows }\exists\mathrm{ A. finite }A\wedge\mathrm{ maximal }AP\wedgeB\subseteq
proof -
    have finite (S-B) using assms(1) assms(3) infinite-super maxc by blast
    then show ?thesis using <P B\rangle
    proof (induction S-B arbitrary:B rule: finite-psubset-induct)
        case (psubset B)
        then show ?case
        proof (cases maximal B P)
            case True
            then show ?thesis using order-refl psubset.hyps by (metis assms(1) maxc
psubset.prems rev-finite-subset)
            next
                case False
                    then obtain }\mp@subsup{B}{}{\prime}\mathrm{ where B}\subset\mp@subsup{B}{}{\prime}P\mp@subsup{B}{}{\prime}\mathrm{ using maximal-def psubset.prems by
(metis dual-order.order-iff-strict)
            then have B'\subseteqSB\subseteqS using maxc <P B\rangle by auto
            then have S-\mp@subsup{B}{}{\prime}\subsetS-B using \langleB\subset B'\rangle by blast
            then show ?thesis using psubset(2)[OF <S- B'\subsetS-B\rangle\langleP B'\rangle}]\mathrm{ using
\}\subset\mp@subsup{B}{}{\prime}\rangle\mathrm{ by fast
            qed
    qed
qed
lemma (in vec-space) rank-ge-card-indpt:
assumes A \in carrier-mat n nc
assumes U\subseteq set (cols A)
assumes lin-indpt U
shows rank A \geq card U
proof -
    obtain S where maximal S ( }\lambdaT.T\subseteq\mathrm{ set (cols A) ^ lin-indpt T) UФS finite S
    using maximal-exists-superset[of set (cols A) ( }\lambdaT.T\subseteq\mathrm{ set (cols A) ^ lin-indpt
T) U]
    using List.finite-set assms(2) assms(3) maximal-exists-superset by blast
    then show ?thesis
    unfolding rank-card-indpt[OF<A \in carrier-mat n nc><maximal S ( }\lambdaT.T
set (cols A) ^ lin-indpt T)>]
```

using card-mono by blast
qed
lemma (in vec-space) lin-indpt-full-rank:
assumes $A \in$ carrier-mat $n n c$
assumes distinct (cols A)
assumes lin-indpt (set (cols A))
shows $\operatorname{rank} A=n c$
proof -
have maximal $(\operatorname{set}(\operatorname{cols} A))(\lambda T . T \subseteq \operatorname{set}(\operatorname{cols} A) \wedge$ lin-indpt $T)$
by (simp add: assms(3) maximal-def subset-antisym)
then have $\operatorname{rank} A=\operatorname{card}(\operatorname{set}(\operatorname{cols} A))$ using assms(1) vec-space.rank-card-indpt by blast
then show ?thesis using assms(1) assms(2) distinct-card by fastforce
qed
lemma (in vec-space) rank-le-nc:
assumes $A \in$ carrier-mat $n n c$
shows rank $A \leq n c$
proof -
obtain $S$ where maximal $S(\lambda T . T \subseteq \operatorname{set}(\operatorname{cols} A) \wedge$ lin-indpt $T)$
using maximal-exists $[o f(\lambda T . T \subseteq$ set $(\operatorname{cols} A) \wedge$ lin-indpt $T)$ card (set (cols A)) $\}$ ]
by (meson List.finite-set card-mono empty-iff empty-subsetI finite-lin-indpt2 rev-finite-subset)
then have card $S \leq \operatorname{card}(\operatorname{set}(\operatorname{cols} A))$ by (simp add: card-mono maximal-def)
then have card $S \leq n c$
using assms(1) cols-length card-length carrier-matD(2) by (metis dual-order.trans)
then show?thesis
using rank-card-indpt $[O F\langle A \in$ carrier-mat $n$ nc〉〈maximal $S(\lambda T . T \subseteq$ set $($ cols $A) \wedge$ lin-indpt $T)\rangle]$
by $\operatorname{simp}$
qed
lemma (in vec-space) full-rank-lin-indpt:
assumes $A \in$ carrier-mat $n n c$
assumes rank $A=n c$
assumes distinct (cols A)
shows lin-indpt (set (cols A))
proof -
have 1 :set $($ cols $A) \subseteq$ carrier-vec $n$ using assms(1) cols-dim by blast
have 2:finite ( set (cols A)) by simp
have $\operatorname{card}(\operatorname{set}(\operatorname{cols} A))=n c$
using assms(1) assms(3) distinct-card by fastforce
have 3:vectorspace.dim class-ring (span-vs $(\operatorname{set}(\operatorname{cols} A)))=\operatorname{card}(\operatorname{set}(\operatorname{cols} A))$
using $\langle r a n k ~ A=n c\rangle[u n f o l d e d ~ r a n k-d e f]$
using assms(1) assms(3) distinct-card by fastforce
show ?thesis using full-dim-span[OF 122 3].
qed
lemma（in vec－space）mat－mult－eq－lincomb：
assumes $A \in$ carrier－mat $n n c$
assumes distinct（cols A）
shows $A *_{v}($ vec nc $(\lambda i . a(\operatorname{col} A i)))=\operatorname{lincomb} a(\operatorname{set}(\operatorname{cols} A))$
proof（rule eq－vecI）
have finite（ set（cols A））using assms（1）by simp
then show dim－vec $\left(A *_{v}(\right.$ vec $n c(\lambda i$ ．a $\left.(\operatorname{col} A i)))\right)=$ dim－vec（lincomb a（set （cols A）））
using assms cols－dim vec－space．lincomb－dim by（metis dim－mult－mat－vec car－ rier－matD（1））
fix $i$ assume $i<d i m-v e c(l i n c o m b ~ a(\operatorname{set}(\operatorname{cols} A)))$
then have $i<n$ using＜dim－vec $\left(A *_{v}(\right.$ vec nc $(\lambda i$ ．a（col $\left.\left.A i))\right)\right)=\operatorname{dim-vec}$ （lincomb a（set（cols $A)$ ））＞assms by auto
have set $($ cols $A) \subseteq$ carrier－vec $n$ using cols－dim $\langle A \in$ carrier－mat $n$ nc〉car－ rier－matD（1）by blast
have bij－betw $($ nth $($ cols $A))\{. .<$ length $(\operatorname{cols} A)\}(\operatorname{set}(\operatorname{cols} A))$
unfolding bij－betw－def by（rule conjI，simp add：inj－on－nth 〈distinct（cols A）〉； metis subset－antisym in－set－conv－nth lessThan－iff rev－image－eqI subsetI image－subsetI lessThan－iff nth－mem）
then have $\left(\sum x \in \operatorname{set}(\operatorname{cols} A) . a x * x \$ i\right)=$
$\left(\sum j \in\{. .<\right.$ length $($ cols $A)\} . a($ cols $\left.A!j) *(\operatorname{cols} A!j) \$ i\right)$
using bij－betw－imp－surj－on bij－betw－imp－inj－on by（metis（no－types，lifting） sum．reindex－cong）
also have $\ldots=\left(\sum j \in\{. .<\right.$ length $($ cols $A)\}$ ．a $(\operatorname{col} A j) *($ cols $\left.A!j) \$ i\right)$
using assms（1）assms（2）find－first－unique［OF〈distinct（cols A）〉］$\langle i<n\rangle$ by

## auto

also have $\ldots=\left(\sum j \in\{. .<\right.$ length $($ cols $\left.A)\} .(\operatorname{cols} A!j) \$ i * a(\operatorname{col} A j)\right)$ by （metis mult－commute－abs）
also have $\ldots=\left(\sum j \in\{. .<\right.$ length $($ cols $A)\}$ ．row $\left.A i \$ j * a(\operatorname{col} A j)\right)$ using $\langle i$ $<n\rangle \operatorname{assms}(1) \operatorname{assms(2)}$ by auto
finally show $\left(A *_{v}(v e c n c(\lambda i . a(\operatorname{col} A i)))\right) \$ i=\operatorname{lincomb} a(\operatorname{set}(\operatorname{cols} A)) \$ i$
unfolding lincomb－index $[O F\langle i<n\rangle\langle$ set $(\operatorname{cols} A) \subseteq$ carrier－vec $n\rangle]$
unfolding mult－mat－vec－def scalar－prod－def
using $\langle i<n\rangle \operatorname{assms}(1)$ atLeastOLessThan lessThan－def carrier－matD（1）in－ dex－vec sum．cong by auto
qed
lemma（in vec－space）lincomb－eq－mat－mult：
assumes $A \in$ carrier－mat $n n c$
assumes $v \in$ carrier－vec nc
assumes distinct（cols A）
shows lincomb $(\lambda a . v \$$ find－first a $(\operatorname{cols} A))(\operatorname{set}(\operatorname{cols} A))=\left(A *_{v} v\right)$
proof－
have $\wedge i . i<n c \Longrightarrow$ find－first $(\operatorname{col} A i)(\operatorname{cols} A)=i$
using assms（1）assms（3）find－first－unique by fastforce
then have vec nc $(\lambda i . v \$$ find－first $(\operatorname{col} A i)(\operatorname{cols} A))=v$
using assms（2）by auto
then show ?thesis
using mat-mult-eq-lincomb[where $a=(\lambda a . v \$$ find-first $a(\operatorname{cols} A))$, OF $\operatorname{assms}(1) \operatorname{assms}(3)]$ by auto
qed
lemma (in vec-space) lin-depI:
assumes $A \in$ carrier-mat $n n c$
assumes $v \in$ carrier-vec nc $v \neq 0_{v} n c A *_{v} v=0_{v} n$
assumes distinct (cols A)
shows lin-dep $(\operatorname{set}(\operatorname{cols} A))$
proof -
have 1: finite ( $\operatorname{set}(\operatorname{cols} A))$ by simp
have 2: set $($ cols $A) \subseteq \operatorname{set}(\operatorname{cols} A)$ by auto
have 3: $(\lambda a . v \$$ find-first $a($ cols $A)) \in \operatorname{set}($ cols $A) \rightarrow$ UNIV by simp
obtain $i$ where $v \$ i \neq 0 i<n c$
using $\left\langle v \neq 0_{v} n c\right\rangle$
by (metis assms(2) dim-vec carrier-vecD vec-eq-iff zero-vec-def index-zero-vec(1))
then have $i<\operatorname{dim}$-col $A$ using assms(1) by blast
have $4: \operatorname{col} A i \in \operatorname{set}(\operatorname{cols} A)$
using cols-nth[OF $\langle i<$ dim-col $A\rangle]\langle i<d i m-c o l A\rangle$ in-set-conv-nth by fastforce
have 5:v $\$$ find-first $($ col $A$ i) $($ cols $A) \neq 0$
using find-first-unique $[O F\langle$ distinct (cols A)〉] cols-nth $[O F\langle i<\operatorname{dim}-c o l A\rangle]\langle i$
$<n c\rangle\langle v \$ i \neq 0\rangle$
$\operatorname{assms}(1)$ by auto
have $6: \operatorname{lincomb}(\lambda a . v \$$ find-first $a(\operatorname{cols} A))(\operatorname{set}(\operatorname{cols} A))=0_{v} n$
using assms(1) assms(2) assms(4) assms(5) lincomb-eq-mat-mult by auto
show ?thesis using lin-dep-crit[OF 12-456] by metis
qed
lemma (in vec-space) lin-depE:
assumes $A \in$ carrier-mat $n n c$
assumes lin-dep (set (cols A))
assumes distinct (cols A)
obtains $v$ where $v \in$ carrier-vec $n c v \neq 0_{v} n c A *_{v} v=0_{v} n$
proof -
have finite ( $\operatorname{set}(\operatorname{cols} A))$ by simp
obtain $a w$ where $a \in \operatorname{set}(\operatorname{cols} A) \rightarrow U N I V \operatorname{lincomb} a(\operatorname{set}(\operatorname{cols} A))=O_{v} n w$ $\in \operatorname{set}($ cols A) a $w \neq 0$
using finite-lin-dep $[O F<$ finite $($ set $(\operatorname{cols} A))\rangle\langle l i n-d e p($ set $($ cols $A))\rangle]$
using assms(1) cols-dim carrier-matD(1) by blast
define $v$ where $v=v e c n c(\lambda i . a(\operatorname{col} A i))$
have $1: v \in$ carrier-vec nc by (simp add: v-def)
have 2: $v \neq 0_{v} n c$
proof -
obtain $i$ where $w=\operatorname{col} A$ i $i<$ length $(\operatorname{cols} A)$
by (metis $\langle w \in$ set (cols $A$ ) > cols-length cols-nth in-set-conv-nth)
have $v \$ i \neq 0$
unfolding $v$-def
using $\langle a w \neq 0\rangle[$ unfolded $\langle w=\operatorname{col} A i\rangle]$ index-vec[OF $\langle i<l e n g t h(\operatorname{cols} A)\rangle]$
assms(1) cols-length carrier-matD(2) by (metis (no-types) $\langle A \in$ carrier-mat $n$ nc>
$\langle\backslash f . \operatorname{vec}(\operatorname{length}(\operatorname{cols} A)) f \$ i=f i\rangle\langle a(\operatorname{col} A i) \neq 0\rangle$ cols-length car-rier-matD(2))
then show ?thesis using $\langle i<$ length $(\operatorname{cols} A)\rangle \operatorname{assms}(1)$ by auto
qed
have $3: A *_{v} v=O_{v} n$ unfolding $v$-def
using 〈lincomb $a($ set (cols $\left.A))=0_{v} n\right\rangle$ mat-mult-eq-lincomb $[O F\langle A \in$ car-rier-mat $n n c\rangle\langle d i s t i n c t($ cols $A)\rangle]$ by auto
show thesis using 123 by ( simp add: that)
qed
lemma (in vec-space) non-distinct-low-rank:
assumes $A \in$ carrier-mat $n n$
and $\neg$ distinct (cols $A$ )
shows rank $A<n$
proof -
obtain $S$ where maximal $S(\lambda T . T \subseteq$ set $($ cols $A) \wedge$ lin-indpt $T)$
using maximal-exists $[o f(\lambda T . T \subseteq$ set $(\operatorname{cols} A) \wedge$ lin-indpt $T)$ card (set (cols A)) $\}]$
by (meson List.finite-set card-mono empty-iff empty-subsetI finite-lin-indpt2 rev-finite-subset)
then have card $S \leq$ card (set (cols $A)$ ) by (simp add: card-mono maximal-def)
then have card $S<n$
using assms(1) cols-length card-length $\langle\neg$ distinct (cols A)〉 card-distinct car-rier-matD(2) nat-less-le
by (metis dual-order.antisym dual-order.trans)
then show? thesis
using rank-card-indpt $[O F\langle A \in$ carrier-mat $n n\rangle\langle$ maximal $S(\lambda T . T \subseteq$ set $($ cols $A) \wedge$ lin-indpt $T)\rangle$ ]
by $\operatorname{simp}$
qed
The theorem "det non-zero $\longleftrightarrow$ full rank" is practically proven in det__0_iff__vec_prod_zero_field but without an actual definition of the rank.

```
lemma (in vec-space) det-zero-low-rank:
assumes A\incarrier-mat n n
and }\operatorname{det}A=
shows rank A<n
proof (rule ccontr)
    assume \neg rank A<n
    then have rank A = n using rank-le-nc assms le-neq-implies-less by blast
    obtain v}\mathrm{ where v}\mathrm{ carrier-vec n v}=\mp@subsup{0}{v}{}|\capA*vv=\mp@subsup{0}{v}{}
    using det-0-iff-vec-prod-zero-field[OF assms(1)] assms(2) by blast
    then show False
    proof (cases distinct (cols A))
        case True
        then have lin-indpt (set (cols A)) using full-rank-lin-indpt using<rank A =
n> assms(1) by auto
```

then show False using lin-dep $\left[\right.$ OF assms $(1)\langle v \in$ carrier-vec $n\rangle\left\langle v \neq 0_{v} n\right\rangle$ $\left\langle A *_{v} v=O_{v} n\right\rangle$ True by blast
next
case False
then show False using non-distinct-low-rank $\langle\operatorname{rank} A=n\rangle\langle\neg \operatorname{rank} A<n\rangle$ assms(1) by blast
qed
qed
lemma det-identical-cols:
assumes $A: A \in$ carrier-mat $n n$
and $i j: i \neq j$
and $i: i<n$ and $j: j<n$
and $r: \operatorname{col} A i=\operatorname{col} A j$
shows $\operatorname{det} A=0$
using det-identical-rows det-transpose
by (metis A i ij j carrier-matD(2) transpose-carrier-mat r row-transpose)
lemma (in vec-space) low-rank-det-zero:
assumes $A \in$ carrier-mat $n n$
and $\operatorname{det} A \neq 0$
shows rank $A=n$
proof -
have distinct (cols A)
proof (rule ccontr)
assume $\neg$ distinct (cols A)
then obtain $i j$ where $i \neq j($ cols $A)!i=($ cols $A)!j i<l e n g t h($ cols $A) j<l e n g t h$ (cols A)
using distinct-conv-nth by blast
then have $\operatorname{col} A i=\operatorname{col} A j i<n j<n$ using $\operatorname{assms}(1)$ by auto
then have $\operatorname{det} A=0$ using det-identical-cols using $\langle i \neq j\rangle \operatorname{assms}(1)$ by blast
then show False using $\langle\operatorname{det} A \neq 0\rangle$ by auto
qed
have $\bigwedge v . v \in$ carrier-vec $n \Longrightarrow v \neq 0_{v} n \Longrightarrow A *_{v} v \neq 0_{v} n$
using det-0-iff-vec-prod-zero-field $[$ OF $\operatorname{assms}(1)] \operatorname{assms}(2)$ by auto
then have lin-indpt (set (cols A)) using lin-depE[OF assms(1) - <distinct (cols A) >] by auto
then show ?thesis using lin-indpt-full-rank[OF assms(1) 〈distinct (cols A)〉] by metis

## qed

lemma (in vec-space) det-rank-iff:
assumes $A \in$ carrier-mat $n n$
shows $\operatorname{det} A \neq 0 \longleftrightarrow \operatorname{rank} A=n$
using assms det-zero-low-rank low-rank-det-zero by force

## 33 Subadditivity of rank

Subadditivity is the property of rank, that rank $(\mathrm{A}+\mathrm{B})<=\operatorname{rank} \mathrm{A}+\operatorname{rank}$ B.
lemma (in Module.module) lincomb-add:
assumes finite $(b 1 \cup b 2)$
assumes $b 1 \cup b 2 \subseteq$ carrier $M$
assumes $x 1=$ lincomb a1 b1 a1 $\in(b 1 \rightarrow$ carrier $R)$
assumes $x \mathcal{2}=$ lincomb a2 b2 a2 $\in(b 2 \rightarrow$ carrier $R)$
assumes $x=x 1 \oplus_{M} x 2$
shows lincomb $(\lambda v$. $(\lambda v$. if $v \in$ b1 then a1 $v$ else $\mathbf{0}) v \oplus(\lambda v$. if $v \in b 2$ then a2 $v$
else 0) v) $(b 1 \cup b 2)=x$
proof -
have finite $(b 1 \cup(b 2-b 1))$ finite $(b 2 \cup(b 1-b 2))$ $b 1 \cup(b 2-b 1) \subseteq$ carrier $M b 2 \cup(b 1-b 2) \subseteq$ carrier $M$ $b 1 \cap(b 2-b 1)=\{ \} b 2 \cap(b 1-b 2)=\{ \}$ $\left(\lambda b . \mathbf{0}_{R}\right) \in b 2-b 1 \rightarrow \operatorname{carrier} R\left(\lambda b . \mathbf{0}_{R}\right) \in b 1-b 2 \rightarrow \operatorname{carrier} R$
using $\langle$ finite $(b 1 \cup b 2)\rangle\langle b 1 \cup b 2 \subseteq$ carrier $M\rangle\langle a 2 \in(b 2 \rightarrow$ carrier $R)\rangle$ by auto
have lincomb $\left(\lambda b . \mathbf{0}_{R}\right)(b 2-b 1)=\mathbf{0}_{M}$ lincomb $\left(\lambda b . \mathbf{0}_{R}\right)(b 1-b 2)=\mathbf{0}_{M}$
unfolding lincomb-def using M.finsum-all0 assms(2) lmult-0 subset-iff
by (metis (no-types, lifting) Un-Diff-cancel2 inf-sup-aci(5) le-sup-iff)+
then have $x 1=\operatorname{lincomb}(\lambda v$. if $v \in b 1$ then a1 velse $\mathbf{0})(b 1 \cup b 2)$
$x 2=$ lincomb $(\lambda v$. if $v \in b 2$ then a2 $v$ else 0) $(b 1 \cup b 2)$
using lincomb-union2 $[O F\langle$ finite $(b 1 \cup(b 2-b 1))\rangle\langle b 1 \cup(b 2-b 1) \subseteq$ carrier $M\rangle\langle b 1 \cap(b 2-b 1)=\{ \}\rangle\langle a 1 \in(b 1 \rightarrow$ carrier $R)\rangle\left\langle\left(\lambda b . \mathbf{0}_{R}\right) \in b 2-b 1 \rightarrow\right.$ carrier $R>$ ]
lincomb-union2 $[O F<$ finite $(b 2 \cup(b 1-b 2))$ 〈b2 $\cup(b 1-b 2) \subseteq$ carrier $M>$
$\langle b 2 \cap(b 1-b 2)=\{ \}\rangle\langle a 2 \in(b 2 \rightarrow$ carrier $R)\rangle\left\langle\left(\lambda b . \mathbf{0}_{R}\right) \in b 1-b 2 \rightarrow\right.$ carrier $\left.\left.R\right\rangle\right]$
using $\operatorname{assms}(2) \operatorname{assms}(3) \operatorname{assms}(4) \operatorname{assms}(5) \operatorname{assms}(6)$ by (simp-all add:Un-commute)
have $(\lambda v$. if $v \in b 1$ then a1 $v$ else $\mathbf{0}) \in(b 1 \cup b 2) \rightarrow$ carrier $R$
$(\lambda v$. if $v \in b 2$ then a2 $v$ else $\mathbf{0}) \in(b 1 \cup b 2) \rightarrow$ carrier $R$ using $\operatorname{assms}(4)$
assms(6) by auto
show lincomb $(\lambda v$. $(\lambda v$. if $v \in b 1$ then a1 $v$ else $\mathbf{0}) v \oplus(\lambda v$. if $v \in b 2$ then a2 $v$ else 0) v) $(b 1 \cup b 2)=x$
using lincomb-sum $[O F<$ finite $(b 1 \cup b 2)\rangle\langle b 1 \cup b 2 \subseteq$ carrier $M\rangle$
$\langle(\lambda v$. if $v \in b 1$ then a1 $v$ else $\mathbf{0}) \in(b 1 \cup b 2) \rightarrow$ carrier $R\rangle\langle(\lambda v$. if $v \in b 2$ then a2 $v$ else $\mathbf{0}) \in(b 1 \cup b 2) \rightarrow$ carrier $R\rangle]$
$\langle x 1=$ lincomb $(\lambda v$. if $v \in$ b1 then a1 $v$ else $\mathbf{0})(b 1 \cup b 2)\rangle\langle x 2=\operatorname{lincomb}(\lambda v$. if $v \in b 2$ then a2 $v$ else $\mathbf{0})(b 1 \cup b 2)$ > assms $(7)$ by blast
qed
lemma (in vectorspace) dim-subadditive:
assumes subspace $K W 1$ V
and vectorspace.fin-dim $K$ (vs W1)
assumes subspace K W2 V
and vectorspace.fin-dim $K$ (vs W2)
shows vectorspace.dim $K$ (vs (subspace-sum W1 W2)) $\leq$ vectorspace.dim $K$ (vs W1) + vectorspace.dim $K$ (vs W2)
proof -
have vectorspace $K$（vs W1）vectorspace $K$（vs W2）submodule $K$ W1 V submod－ ule $K$ W2 $V$
by（simp add：〈subspace $K W 1 \quad V\rangle\langle$ subspace $K W 2 V\rangle$ subspace－is－vs）＋
obtain b1 b2 where vectorspace．basis $K$（vs W1）b1 vectorspace．basis $K$（vs W2）
b2 finite b1 finite b2
using vectorspace．finite－basis－exists［OF «vectorspace $K$（vs W1）〉〈vectorspace．fin－dim $K(v s W 1)\rangle]$
using vectorspace．finite－basis－exists $[O F\langle v e c t o r s p a c e ~ K(v s ~ W 2)\rangle\langle v e c t o r s p a c e . f i n-d i m ~$ K（vs W2）＞］
by blast
then have LinearCombinations．module．gen－set $K$（vs W1）b1 LinearCombina－ tions．module．gen－set $K$（vs W2）b2
using 〈vectorspace $K$（vs W1）〉〈vectorspace $K$（vs W2）〉 vectorspace．basis－def by blast＋
then have span $b 1=W 1$ span $b 2=W 2$
using module．span－li－not－depend（1）«submodule K W1 V〉 «submodule K W2 V
$\langle v e c t o r s p a c e ~ K(v s W 1)\rangle\langle v e c t o r s p a c e . b a s i s K(v s W 1) b 1\rangle\langle v e c t o r s p a c e ~ K(v s$ W2）＞

〈vectorspace．basis K（vs W2）b2〉 vectorspace．basis－def by force＋
have $W 1 \subseteq$ carrier $V W 2 \subseteq$ carrier $V$ using 〈subspace $K W 1 V\rangle\langle$ subspace $K$ W2 $V>$ subspace－def submodule－def by metis＋
have $b 1 \subseteq$ carrier $V$
using 〈vectorspace．basis $K$（vs W1）b1〉〈vectorspace $K$（vs W1）〉 vectorspace．basis－def
$\langle W 1 \subseteq$ carrier $V\rangle$ by fastforce
have $b 2 \subseteq$ carrier $V$
using 〈vectorspace．basis $K$（vs W2）b2〉〈vectorspace $K$（vs W2）〉vectorspace．basis－def
$\langle W 2 \subseteq$ carrier $V\rangle$ by fastforce
have finite $(b 1 \cup b 2) b 1 \cup b 2 \subseteq$ carrier $V$
by（simp－all add：＜finite b1〉〈finite b2〉〈b2 $\subseteq$ carrier $V\rangle\langle b 1 \subseteq$ carrier $V\rangle$ ）
have subspace－sum W1 W2 $\subseteq$ span（b1 bb2）
proof（rule subsetI）
fix $x$ assume $x \in$ subspace－sum W1 W2
obtain $x 1 x 2$ where $x 1 \in W 1 x 2 \in W 2 x=x 1 \oplus_{V} x 2$
using imageE $[O F\langle x \in$ subspace－sum W1 W2〉［unfolded submodule－sum－def］］
by（metis（no－types，lifting）BNF－Def．Collect－case－prodD split－def）
obtain a1 where $x 1=$ lincomb a1 b1 a1 $\in(b 1 \rightarrow$ carrier $K)$
using 〈span b1＝W1〉 finite－span $[O F\langle$ finite b1〉〈b1 $\subseteq$ carrier $V\rangle]\langle x 1 \in$ W1＞by auto
obtain $a 2$ where $x 2=$ lincomb a2 b2 a2 $\in(b 2 \rightarrow$ carrier $K)$
using 〈span b2＝W2〉 finite－span $[O F\langle$ finite b2〉〈b2 $\subseteq$ carrier $V\rangle]\langle x 2 \in$ W2）by auto
obtain $a$ where $x=$ lincomb $a(b 1 \cup b 2)$ using lincomb－add $[O F<$ finite（b1 $\cup b 2)\rangle\langle b 1 \cup b 2 \subseteq$ carrier $V\rangle$
$\langle x 1=$ lincomb a1 b1 $\langle\langle a 1 \in($ b1 $\rightarrow$ carrier $K)\rangle\langle x 2=$ lincomb a2 b2 $\rangle\langle a 2 \in$ $(b 2 \rightarrow$ carrier $\left.K)\rangle\left\langle x=x 1 \oplus_{V} x 2\right\rangle\right]$ by blast
then show $x \in \operatorname{span}(b 1 \cup b 2)$ using finite－span［OF＜finite $(b 1 \cup b 2)\rangle\langle(b 1$ $\cup b 2) \subseteq$ carrier $V\rangle]$
using $\langle b 1 \subseteq$ carrier $V\rangle\langle b 2 \subseteq$ carrier $V\rangle\langle$ span $b 1=W 1\rangle\langle$ span $b 2=W 2\rangle$

```
<x\in subspace-sum W1 W2` span-union-is-sum by auto
```

    qed
    have \(b 1 \subseteq W 1 b 2 \subseteq W 2\)
        using «vectorspace \(K\) (vs W1)〉〈vectorspace \(K\) (vs W2)〉〈vectorspace.basis K
    (vs W1) b1>
〈vectorspace.basis K (vs W2) b2〉 vectorspace.basis-def local.carrier-vs-is-self by
blast+
then have $b 1 \cup b 2 \subseteq$ subspace-sum $W 1$ W2 using <submodule $K W 1$ V〉〈sub-
module $K W 2 \quad V>$ in-sum
by (metis assms(1) assms(3) dual-order.trans sup-least vectorspace.vsum-comm
vectorspace-axioms)
have subspace-sum W1 W2 = LinearCombinations.module.span $K$ (vs (subspace-sum
W1 W2)) (b1 (b2)
proof (rule subset-antisym)
have submodule $K$ (subspace-sum W1 W2) V by (simp add: «submodule K W1
$V\rangle\langle$ submodule $K$ W2 $V\rangle$ sum-is-submodule)
show subspace-sum W1 W2 $\subseteq$ LinearCombinations.module.span $K$ (vs (subspace-sum
W1 W2)) (b1 Ub2)
using module.span-li-not-depend $(1)[O F\langle b 1 \cup b 2 \subseteq$ subspace-sum W1 W2〉
«submodule K (subspace-sum W1 W2) V〉]
by (simp add: «subspace-sum W1 W2 $\subseteq \operatorname{span}(b 1 \cup b 2)\rangle)$
show subspace-sum W1 W2 $\supseteq$ LinearCombinations.module.span K (vs (subspace-sum
W1 W2)) (b1 Ub2)
using $\langle b 1 \cup b 2 \subseteq$ subspace-sum W1 W2 $\mathbf{W}$ by (metis (full-types) LinearCombi-
nations.module.span-is-subset2
LinearCombinations.module.submodule-is-module «submodule K (subspace-sum
W1 W2) $V>$ local.carrier-vs-is-self submodule-def)
qed
have vectorspace $K$ (vs (subspace-sum W1 W2)) using assms(1) assms(3) sub-
space-def sum-is-subspace vectorspace.subspace-is-vs by blast
then have vectorspace.dim $K($ vs $($ subspace-sum W1 W2) $) \leq \operatorname{card}(b 1 \cup$ b2 $)$
using vectorspace.gen-ge-dim[OF <vectorspace K (vs (subspace-sum W1 W2))〉
〈finite $(b 1 \cup b 2)$ )]
$\langle b 1 \cup b 2 \subseteq$ subspace-sum W1 W2〉
〈subspace-sum W1 W2 = LinearCombinations.module.span $K$ (vs (subspace-sum
W1 W2)) (b1 $\cup b 2)$ >
local.carrier-vs-is-self by blast
also have $\ldots \leq$ card b1 + card b2 by (simp add: card-Un-le)
also have $\ldots=$ vectorspace. $\operatorname{dim} K(v s W 1)+$ vectorspace. $\operatorname{dim} K$ (vs W2)
by (metis 〈finite b1〉〈finite b2〉〈vectorspace K (vs W1)〉〈vectorspace K (vs
W2) >
〈vectorspace.basis $K$ (vs W1) b1〉〈vectorspace.basis $K$ (vs W2) b2〉vectorspace.dim-basis)
finally show? thesis by auto
qed
lemma (in Module.module) nested-submodules:
assumes submodule $R$ W M
assumes submodule $R X M$
assumes $X \subseteq W$
shows submodule $R X$（md $W$ ）
unfolding submodule－def
using «X $\subseteq W$ 〉 submodule－is－module $[O F$ «submodule $R W M$ ］using «submodule
$R X M\rangle[u n f o l d e d ~ s u b m o d u l e-d e f]$ by auto
lemma（in vectorspace）nested－subspaces：
assumes subspace $K W V$
assumes subspace $K X V$
assumes $X \subseteq W$
shows subspace $K X$（vs $W$ ）
using assms nested－submodules subspace－def subspace－is－vs by blast
lemma（in vectorspace）subspace－dim：
assumes subspace $K X V$ fin－dim vectorspace．fin－dim $K(v s X)$
shows vectorspace．dim $K($ vs $X) \leq \operatorname{dim}$
proof－
have vectorspace $K$（vs $X$ ）using assms（1）subspace－is－vs by auto
then obtain $b$ where vectorspace．basis $K(v s X) b$ using vectorspace．finite－basis－exists using assms（3）by blast
then have $b \subseteq$ carrier $V$ LinearCombinations．module．lin－indpt $K$（vs X）b
using vectorspace．basis－def $[O F\langle v e c t o r s p a c e ~ K(v s X)\rangle]\langle$ subspace $K X V\rangle[$ unfolded
subspace－def submodule－def］by auto
then have lin－indpt b
by（metis LinearCombinations．module．span－li－not－depend（2）«vectorspace $K$（vs $X)\rangle\langle v e c t o r s p a c e . b a s i s ~ K(v s X) b\rangle$
assms（1）is－module local．carrier－vs－is－self submodule－def vectorspace．basis－def） show ？thesis using li－le－dim（2）［OF〈fin－dim〉〈b $\subseteq$ carrier $V\rangle\langle l i n-i n d p t ~ b\rangle]$
using $\langle b \subseteq$ carrier $V\rangle\langle l i n-i n d p t b\rangle\langle v e c t o r s p a c e ~ K(v s X)\rangle\langle v e c t o r s p a c e . b a s i s$ $K(v s X) b\rangle \operatorname{assms}(2)$ fin－dim－li－fin vectorspace．dim－basis by fastforce
qed
lemma（in vectorspace）fin－dim－subspace－sum：
assumes subspace K W1 V
assumes subspace $K$ W2 $V$
assumes vectorspace．fin－dim $K$（vs W1）vectorspace．fin－dim $K$（vs W2）
shows vectorspace．fin－dim $K$（vs（subspace－sum W1 W2））
proof－
obtain b1 where finite b1 b1 $\subseteq$ W1 LinearCombinations．module．gen－set $K$（vs W1）b1
using assms vectorspace．fin－dim－def subspace－is－vs by force
obtain b2 where finite b2 b2 $\subseteq$ W2 LinearCombinations．module．gen－set $K$（vs
W2）b2
using assms vectorspace．fin－dim－def subspace－is－vs by force
have $1:$ finite $(b 1 \cup$ b2）by（simp add：〈finite b1〉〈finite b2〉）
have 2：b1 $\cup$ b2 $\subseteq$ subspace－sum W1 W2
by（metis（no－types，lifting）$\langle b 1 \subseteq W 1\rangle\langle b 2 \subseteq W 2\rangle \operatorname{assms}(1) \operatorname{assms}(2)$
le－sup－iff subset－Un－eq vectorspace．in－sum－vs vectorspace．vsum－comm vectorspace－axioms）
have 3：LinearCombinations．module．gen－set K（vs（subspace－sum W1 W2））（b1

## U b2）

proof（rule subset－antisym）
have 0：LinearCombinations．module．span $K$（vs（subspace－sum W1 W2））（b1 $\cup$ $b 2)=\operatorname{span}(b 1 \cup b 2)$
using span－li－not－depend（1）［OF 〈b1 $\cup$ b2 $\subseteq$ subspace－sum W1 W2〉］sum－is－subspace $[O F$ $\operatorname{assms}(1) \operatorname{assms}(2)]$ by auto
then show LinearCombinations．module．span $K$（vs（subspace－sum W1 W2）） $(b 1 \cup b 2) \subseteq$ carrier $(v s($ subspace－sum W1 W2））
using $\langle b 1 \cup b 2 \subseteq$ subspace－sum W1 W2〉 span－is－subset sum－is－subspace $[O F$ $\operatorname{assms(1)} \operatorname{assms(2)]}$ by auto
show carrier（vs（subspace－sum W1 W2））$\subseteq$ LinearCombinations．module．span $K(v s($ subspace－sum W1 W2）$)(b 1 \cup$ b2 $)$
unfolding 0
proof
fix $x$ assume assumption：$x \in$ carrier（vs（subspace－sum W1 W2））
then have $x \in$ subspace－sum W1 W2 by auto
then obtain $x 1 x 2$ where $x=x 1 \oplus_{V} x 2 x 1 \in W 1 x 2 \in W 2$
using imageE［OF $\langle x \in$ subspace－sum W1 W2〉［unfolded submodule－sum－def］］ by（metis（no－types，lifting）BNF－Def．Collect－case－prodD split－def）
have $x 1 \in$ span b1 $x 2 \in$ span b2
using＜LinearCombinations．module．span $K(v s W 1) b 1=$ carrier（vs W1）〉 $\langle b 1 \subseteq W 1\rangle\langle x 1 \in W 1\rangle$

〈LinearCombinations．module．span $K$（vs W2）b2 $=$ carrier（vs W2）〉
$\langle b 2 \subseteq W 2\rangle\langle x 2 \in W 2\rangle$
$\operatorname{assms}$（1） $\operatorname{assms}(2)$ span－li－not－depend（1）by auto
then have $x 1 \in \operatorname{span}(b 1 \cup b 2) x 2 \in \operatorname{span}(b 1 \cup b 2)$ by（meson le－sup－iff subsetD span－is－monotone subsetI）＋
then show $x \in \operatorname{span}(b 1 \cup b 2)$ unfolding $\left\langle x=x 1 \oplus_{V} x 2\right\rangle$
by（meson 〈b1 $\cup$ b2 $\subseteq$ subspace－sum W1 W2〉assms（1）assms（2）is－module submodule．subset subset－trans sum－is－submodule vectorspace．span－add1 vectorspace－axioms）
qed
qed
show ？thesis using 123 vectorspace．fin－dim－def
by（metis assms（1）assms（2）local．carrier－vs－is－self subspace－def sum－is－subspace vectorspace．subspace－is－vs）
qed
lemma（in vec－space）rank－subadditive：
assumes $A \in$ carrier－mat $n n c$
assumes $B \in$ carrier－mat $n n c$
shows rank $(A+B) \leq \operatorname{rank} A+\operatorname{rank} B$
proof－
define $W 1$ where $W 1=\operatorname{span}(\operatorname{set}(\operatorname{cols} A))$
define $W 2$ where $W 2=\operatorname{span}(\operatorname{set}(\operatorname{cols} B))$
have set $(\operatorname{cols}(A+B)) \subseteq$ subspace－sum W1 W2
proof
fix $x$ assume $x \in \operatorname{set}(\operatorname{cols}(A+B))$
obtain $i$ where $x=\operatorname{col}(A+B) i i<l e n g t h(\operatorname{cols}(A+B))$
using $\langle x \in \operatorname{set}(\operatorname{cols}(A+B))\rangle n t h-f i n d-f i r s t$ cols－nth find－first－le by（metis cols－length）
then have $x=\operatorname{col} A i+\operatorname{col} B i$ using $\langle i<l e n g t h(\operatorname{cols}(A+B))\rangle \operatorname{assms}(1)$ assms（2）by auto
have $\operatorname{col} A i \in \operatorname{span}(\operatorname{set}(\operatorname{cols} A)) \operatorname{col} B i \in \operatorname{span}(\operatorname{set}(\operatorname{cols} B))$
using $\langle i<$ length $($ cols $(A+B))\rangle \operatorname{assms}(1)$ assms（2）in－set－conv－nth
by（metis cols－dim cols－length cols－nth carrier－matD（1）carrier－matD（2）in－ dex－add－mat（3）span－mem）＋
then show $x \in$ subspace－sum W1 W2
unfolding W1－def W2－def $\langle x=\operatorname{col} A i+\operatorname{col} B i\rangle$ submodule－sum－def by blast
qed
have subspace class－ring（subspace－sum W1 W2）V
by（metis W1－def W2－def assms（1）assms（2）cols－dim carrier－matD（1）span－is－submodule subspace－def sum－is－submodule vec－vs）
then have span $($ set $($ cols $(A+B))) \subseteq$ subspace－sum W1 W2
by $($ simp add：$<$ set $($ cols $(A+B)) \subseteq$ subspace－sum W1 W2〉 span－is－subset）
have subspace class－ring（span（set（cols $(A+B))$ ））$V$ by（metis assms（2） cols－dim add－carrier－mat carrier－matD（1）span－is－subspace）
have subspace：subspace class－ring（span（set $($ cols $(A+B)))$ ）（vs（subspace－sum W1 W2））
using nested－subspaces［OF 〈subspace class－ring（subspace－sum W1 W2）V〉〈subspace class－ring（span（set（cols $(A+B)))) V\rangle$

〈span $($ set $($ cols $(A+B))) \subseteq$ subspace－sum W1 W2〉］．
have vectorspace．fin－dim class－ring（vs W1）vectorspace．fin－dim class－ring（vs W2）
subspace class－ring W1 V subspace class－ring W2 V
using span－is－subspace W1－def W2－def assms（1）assms（2）cols－dim carrier－matD fin－dim－span－cols by auto
then have fin－dim：vectorspace．fin－dim class－ring（vs（subspace－sum W1 W2）） using fin－dim－subspace－sum by auto
have vectorspace．fin－dim class－ring（span－vs（set（cols $(A+B))$ ））using assms（2） add－carrier－mat vec－space．fin－dim－span－cols by blast
then have rank $(A+B) \leq$ vectorspace．dim class－ring（vs（subspace－sum W1 W2））unfolding rank－def
using vectorspace．subspace－dim［OF subspace－is－vs［OF «subspace class－ring（subspace－sum W1 W2）$V>]$ subspace fin－dim］by auto
also have vectorspace．dim class－ring（vs（subspace－sum W1 W2））$\leq \operatorname{rank} A+$ rank $B$ unfolding rank－def
using W1－def W2－def 〈subspace class－ring W1 $V\rangle\langle$ subspace class－ring W2 $V\rangle$〈vectorspace．fin－dim class－ring（vs W1）＞

〈vectorspace．fin－dim class－ring（vs W2）〉 subspace－def vectorspace．dim－subadditive by blast
finally show？？thesis by auto
qed
lemma（in vec－space）span－zero：span $\{$ zero $V\}=\{$ zero $V\}$ by（metis（no－types，lifting）empty－subsetI in－own－span span－is－submodule span－is－subset span－is－subset2 subset－antisym vectorspace．span－empty vectorspace－axioms）
lemma（in vec－space）dim－zero－vs：vectorspace．dim class－ring（span－vs $\})=0$ proof－
have vectorspace class－ring（span－vs $\}$ ）using field．field－axioms span－is－submodule submodule－is－module vectorspace－def by auto
have $\} \subseteq$ carrier－vec $n \wedge$ lin－indpt $\}$
by（metis（no－types）empty－subsetI fin－dim finite－basis－exists subset－li－is－li vec－vs vectorspace．basis－def）
then have vectorspace．basis class－ring（span－vs \｛\}) \{\} using vectorspace.basis-def
by（simp add：〈vectorspace class－ring（vs（span \｛\})) > span-is-submodule span-li-not-depend (1) span－li－not－depend（2）vectorspace．basis－def）
then show？？thesis using 〈vectorspace class－ring（vs（span \｛\})) > vectorspace.dim-basis by fastforce
qed
lemma（in vec－space）rank－OI： $\operatorname{rank}\left(O_{m} n n c\right)=0$
proof－
have set $\left(\operatorname{cols}\left(O_{m} n n c\right)\right) \subseteq\left\{0_{v} n\right\}$
by（metis col－zero cols－length cols－nth in－set－conv－nth insertCI index－zero－mat（3） subsetI）
have set $\left(\right.$ cols $\left.\left(0_{m} n n c::^{\prime} a \operatorname{mat}\right)\right)=\{ \} \vee \operatorname{set}\left(\operatorname{cols}\left(0_{m} n n c\right)\right)=\left\{0_{v} n::^{\prime} a\right.$ vec $\}$
by $\left(\right.$ meson 〈set $\left(\operatorname{cols}\left(O_{m} n n c\right)\right) \subseteq\left\{0_{v} \quad n\right\} 〉$ subset－singletonD $)$
then have span $\left(\operatorname{set}\left(\operatorname{cols}\left(0_{m} n n c\right)\right)\right)=\left\{0_{v} n\right\}$
by（metis（no－types）span－empty span－zero vectorspace．span－empty vectorspace－axioms）
then show ？thesis unfolding rank－def 〈span（set $\left.\left(\operatorname{cols}\left(O_{m} n n c\right)\right)\right)=\left\{\begin{array}{ll}0_{v} & n\end{array}\right\}$ 〉
using span－empty dim－zero－vs by simp
qed
lemma（in vec－space）rank－le－1－product－entries：
fixes $f g:: n a t \Rightarrow{ }^{\prime} a$
assumes $A \in$ carrier－mat $n n c$
assumes $\bigwedge r c . r<d i m$－row $A \Longrightarrow c<d i m-\operatorname{col} A \Longrightarrow A \$ \$(r, c)=f r * g c$
shows $\operatorname{rank} A \leq 1$
proof－
have set $($ cols $A) \subseteq \operatorname{span}\{$ vec $n f\}$
proof
fix $v$ assume $v \in \operatorname{set}(\operatorname{cols} A)$
then obtain $c$ where $c<\operatorname{dim}-c o l A v=\operatorname{col} A c$ by（metis cols－length cols－nth
in－set－conv－nth）
have $g c \cdot v$ vec $n f=v$
proof（rule eq－vecI）
show dim－vec $(g c \cdot v$ Matrix．vec $n f)=$ dim－vec $v$ using $\langle v=\operatorname{col} A c\rangle$
assms（1）by auto
fix $r$ assume $r<d i m-v e c v$
then have $r<$ dim－vec（Matrix．vec $n f$ ）using 〈dim－vec（ $g c \cdot{ }_{v}$ Matrix．vec $n f)=$ dim－vec $v>$ by auto
then have $r<n r<$ dim－row Ausing index－smult－vec（2）$\langle A \in$ carrier－mat $n n c>$ by auto

```
    show (g c \cdotv Matrix.vec nf) $ r=v $ r
    unfolding <v = col A c> col-def index-smult-vec(1)[OF<r< dim-vec
(Matrix.vec n f)>]
    index-vec[OF <r<n`] index-vec[OF<r< dim-row A〉] by (simp add: <c <
dim-col A〉\langler < dim-row A> assms(2))
    qed
    then show v\in span {vec nf} using submodule.smult-closed[OF span-is-submodule]
    using UNIV-I empty-subsetI insert-subset span-self dim-vec module-vec-simps(4)
by auto
    qed
    have vectorspace class-ring (vs (span { Matrix.vec n f})) using span-is-subspace[THEN
subspace-is-vs, of {vec n f}] by auto
    have submodule class-ring (span {Matrix.vec nf}) V by (simp add: span-is-submodule)
    have subspace class-ring(span (set (cols A))) (vs (span {Matrix.vec nf}))
    using vectorspace.span-is-subspace[OF <vectorspace class-ring (vs (span {Matrix.vec
nf}))>, of set (cols A), unfolded
    span-li-not-depend(1)[OF <set (cols A)\subseteq span {vec n f}><submodule class-ring
(span {Matrix.vec n f}) V \]]
    <set (cols A)\subseteq span {vec n f}> by auto
    have fin-dim:vectorspace.fin-dim class-ring (vs (span {Matrix.vec n f}))
            vectorspace.fin-dim class-ring (vs (span {Matrix.vec n f})\carrier := span
(set (cols A))D)
    using fin-dim-span fin-dim-span-cols }\langleA\in\mathrm{ carrier-mat n nc> by auto
    have vectorspace.dim class-ring (vs (span {Matrix.vec nf})) \leq 1
    using vectorspace.dim-le1I[OF<vectorspace class-ring (vs (span {Matrix.vec n
f}))>]
    span-mem span-li-not-depend(1)[OF - <submodule class-ring (span {Matrix.vec
nf}) V>] by simp
    then show ?thesis unfolding rank-def using vectorspace.subspace-dim[OF
    vectorspace class-ring (vs (span {Matrix.vec nf}))\rangle\langlesubspace class-ring (span
(set (cols A))) (vs (span {Matrix.vec n f}))>
    fin-dim(1) fin-dim(2)] by simp
qed
end
```


## 34 Missing Lemmas of Sublist

```
theory DL-Missing-Sublist
imports Main
begin
lemma nths-only-one:
assumes {i.i<length xs ^i\inI}={j}
shows nths xs I = [xs!j]
proof -
    have set (nths xs I) ={xs!j}
    unfolding set-nths using subset-antisym assms by fastforce
    moreover have length (nths xs I) = 1
```

unfolding length-nths assms by auto ultimately show ?thesis
by (metis One-nat-def length-0-conv length-Suc-conv the-elem-eq the-elem-set) qed
lemma nths-replicate:
nths (replicate $n x) A=($ replicate $(\operatorname{card}\{i . i<n \wedge i \in A\}) x)$
proof (induction $n$ )
case 0
then show? case by simp
next
case (Suc n)
then show ?case
proof (cases $n \in A$ )
case True
then have $0:($ if $0 \in\{j . j+$ length (replicate $n x) \in A\}$ then $[x]$ else []$)=[x]$
by $\operatorname{simp}$
have $\{i . i<$ Suc $n \wedge i \in A\}=$ insert $n\{i . i<n \wedge i \in A\}$ using True by auto
have Suc $(\operatorname{card}\{i . i<n \wedge i \in A\})=\operatorname{card}\{i . i<\operatorname{Suc} n \wedge i \in A\}$
unfolding «\{i. $i<\operatorname{Suc} n \wedge i \in A\}=$ insert $n\{i . i<n \wedge i \in A\}$ 〉
using finite-Collect-conjI[THEN card-insert-if] finite-Collect-less-nat less-irrefl-nat mem-Collect-eq by simp
then show ?thesis unfolding replicate-Suc replicate-append-same[symmetric] nths-append Suc nths-singleton 0
unfolding replicate-append-same replicate-Suc[symmetric] by simp
next
case False
then have $0:($ if $0 \in\{j . j+$ length $($ replicate $n x) \in A\}$ then $[x]$ else []$)=[]$ by $\operatorname{simp}$
have $\{i . i<$ Suc $n \wedge i \in A\}=\{i . i<n \wedge i \in A\}$ using False using le-less less-Suc-eq-le by auto
then show ?thesis unfolding replicate-Suc replicate-append-same[symmetric] nths-append Suc nths-singleton 0
by simp
qed
qed
lemma length-nths-even:
assumes even (length xs)
shows length ( $n$ ths xs (Collect even)) $=$ length ( $n$ ths xs (Collect odd))
using assms proof (induction length xs div 2 arbitrary:xs)
case 0
then have length $x s=0$
by (auto elim: evenE)
then show? case by simp
next
case (Suc l xs)
then have length-drop2: length (nths (drop 2 xs) (Collect even)) $=$ length ( $n$ ths
(drop 2 xs $)$ \{a. odd a\}) by simp
have length (take $2 x s$ ) $=2$ using Suc.hyps(2) by auto
then have plus-odd: $\{j . j+$ length $($ take $2 x s) \in$ Collect odd $\}=$ Collect odd and plus-even: $\{j . j+$ length (take 2 xs $) \in$ Collect even $\}=$ Collect even by
simp-all
have nths-take2: nths (take 2 xs) (Collect even) $=[$ take 2 xs! 0] nths (take 2 xs) $($ Collect odd $)=[$ take 2 xs ! 1]
using <length (take $2 x s$ ) $=2$ 2 less-2-cases nths-only-one[of take 2 xs Collect even 0]
nths-only-one[of take 2 xs Collect odd 1]
by fastforce+
then have length (nths (take 2 xs @ drop 2 xs) (Collect even))
= length (nths (take 2 xs @ drop 2 xs) \{a. odd a\})
unfolding nths-append length-append plus-odd plus-even nths-take2 length-drop2
by auto
then show ?case using append-take-drop-id[of 2 xs] by simp
qed
lemma nths-map:
nths (map $f$ xs) $A=\operatorname{map} f$ ( $n$ ths xs $A$ )
proof (induction xs arbitrary:A)
case Nil
then show? case by simp
next
case (Cons $x \times s$ )
then show ?case
by (simp add: nths-Cons)
qed

## 35 Pick

fun pick :: nat set $\Rightarrow$ nat $\Rightarrow$ nat where
pick S $0=(L E A S T$ a. $a \in S) \mid$
pick $S(S u c n)=($ LEAST $a . a \in S \wedge a>$ pick $S n)$
lemma pick-in-set-inf:
assumes infinite $S$
shows pick $S n \in S$
proof (cases $n$ )
show $n=0 \Longrightarrow$ pick $S n \in S$
unfolding pick.simps using 〈infinite $S$ 〉LeastI pick.simps(1) by (metis Col-
lect-mem-eq not-finite-existsD)
next
fix $n^{\prime}$ assume $n=S u c n^{\prime}$
obtain $a$ where $a \in S \wedge a>$ pick $S n^{\prime}$ using assms by (metis bounded-nat-set-is-finite
less-Suc-eq nat-neq-iff)
show pick $S n \in S$ unfolding $\left\langle n=S u c n^{\prime}\right\rangle$ pick.simps(2)
using LeastI[of $\lambda a . a \in S \wedge$ pick $\left.S n^{\prime}<a a, O F\left\langle a \in S \wedge a>p i c k S n^{\prime}\right\rangle\right]$ by

```
blast
qed
lemma pick-mono-inf:
assumes infinite S
shows m<n\Longrightarrow pick S m < pick S n
using assms proof (induction n)
    case 0
    then show ?case by auto
next
    case (Suc n)
    then obtain a where a GS ^ pick S n<a by (metis bounded-nat-set-is-finite
less-Suc-eq nat-neq-iff)
    then have pick S n < pick S (Suc n) unfolding pick.simps
        using LeastI[of \lambdaa.a 
simp
    then show ?case using Suc.IH Suc.prems(1) assms dual-order.strict-trans less-Suc-eq
by auto
qed
lemma pick-eq-iff-inf:
assumes infinite S
shows }x=y\longleftrightarrow pickS x= pick S y
    by (metis assms nat-neq-iff pick-mono-inf)
lemma card-le-pick-inf:
assumes infinite S
and pick S n\geqi
shows card {a\inS.a<i}\leqn
using assms proof (induction n arbitrary:i)
    case 0
    then show ?case unfolding pick.simps using not-less-Least
    by (metis (no-types, lifting) Collect-empty-eq card-0-eq card-ge-0-finite dual-order.strict-trans1
leI le-0-eq)
next
    case (Suc n)
    then show ?case
    proof -
    have card {a\inS.a<pickS n}\leqn using Suc by blast
    have {a\inS.a<i}\subseteq{a\inS.a<pickS (Suc n)} using Suc.prems(2) by
auto
    have {a\inS.a<pickS (Suc n)}={a\inS.a<pickS n} \cup{pickS n}
        apply (rule subset-antisym; rule subsetI)
        using not-less-Least UnCI mem-Collect-eq nat-neq-iff singleton-conv
        pick-mono-inf[OF Suc.prems(1), of n Suc n] pick-in-set-inf[OF Suc.prems(1),
of n] by fastforce+
    then have card {a\inS.a<i}\leqcard {a\inS.a<pickSn}+card {pick S
n}
    using card-Un-disjoint card-mono[OF-<{a\inS.a<i}\subseteq{a\inS.a<pick
```

```
S (Suc n)}>] by simp
        then show ?thesis using <card {a\inS.a<pickS n}\leqn` by auto
    qed
qed
lemma card-pick-inf:
assumes infinite S
shows card {a\inS.a< pickS n}=n
using assms proof (induction n)
    case 0
    then show ?case unfolding pick.simps using not-less-Least by auto
next
    case (Suc n)
    then show card {a\inS.a< pick S (Suc n)}=Suc n
    proof -
        have {a\inS.a<pickS (Suc n)}={a\inS.a<pickS n}\cup{pickS n}
            apply (rule subset-antisym; rule subsetI)
            using not-less-Least UnCI mem-Collect-eq nat-neq-iff singleton-conv
                pick-mono-inf[OF Suc.prems, of n Suc n] pick-in-set-inf[OF Suc.prems, of
n] by fastforce+
    then have card {a\inS.a<pickS (Suc n)}=card {a\inS.a<pickSn}+
card {pickS n} using card-Un-disjoint by auto
    then show ?thesis by (metis One-nat-def Suc-eq-plus1 Suc card.empty card-insert-if
empty-iff finite.emptyI)
    qed
qed
lemma
assumes n<card S
shows
    pick-in-set-le:pick S n \inS and
    card-pick-le: card {a\inS.a< pick S n} = n and
    pick-mono-le: m < n\Longrightarrow pick Sm< pickS n
using assms proof (induction n)
    assume 0< card S
    then obtain x where x\inS by fastforce
    then show pick S 0 GS unfolding pick.simps by (meson LeastI)
    then show card {a\inS.a< pickS 0} = 0 using not-less-Least by auto
    show m<0\Longrightarrow pickS m< pickS 0 by auto
next
    fix n
    assume n<card S\Longrightarrow pick S n \inS
        and n<card S\Longrightarrowcard {a\inS.a< pick S n}=n
        and Suc n< card S
        and m<n\Longrightarrown<card S\Longrightarrow pickSm< pickS n
    then have card {a\inS.a< pickS n}=n pick S n\inS by linarith+
    have card {a\inS.a> pickSn}>0
    proof -
        have}S={a\inS.a< pickSn}\cup{a\inS.a\geq pickS n} by fastforc
```

then have $\operatorname{card}\{a \in S . a \geq$ pick $S n\}>1$
using 〈Suc $n<$ card $S\rangle\langle$ card $\{a \in S . a<$ pick $S n\}=n$ 〉
card－Un－le $[o f\{a \in S . a<$ pick $S n\}\{a \in S$ ．pick $S n \leq a\}]$ by force
then have $0:\{a \in S . a \geq$ pick $S n\} \subseteq\{$ pick $S n\} \cup\{a \in S . a>$ pick $S n\}$ by auto
have 1：finite（ $\{$ pick $S n\} \cup\{a \in S$ ．pick $S n<a\}$ ）
unfolding finite－Un using Collect－mem－eq assms card．infinite conjI by force
have $1<$ card $\{$ pick $S n\}+$ card $\{a \in S$ ．pick $S n<a\}$
using card－mono［OF 1 0］card－Un－le［of \｛pick $S n\}\{a \in S . a>$ pick $S n\}]$
$\langle\operatorname{card}\{a \in S . a \geq$ pick $S n\}>1\rangle$
by linarith
then show？？thesis by simp
qed
then show pick $S($ Suc $n) \in S$ unfolding pick．simps
by（metis（no－types，lifting）Collect－empty－eq LeastI card－0－eq card．infinite less－numeral－extra（3））
have pick $S(S u c n)>$ pick $S n$
by（metis（no－types，lifting）pick．simps（2）〈card $\{a \in S . a>$ pick $S n\}>0\rangle$ Collect－empty－eq LeastI card－0－eq card．infinite less－numeral－extra（3））
then show $m<S u c n \Longrightarrow$ pick $S m<$ pick $S$（Suc $n$ ）
using $\langle m<n \Longrightarrow n<$ card $S \Longrightarrow$ pick $S m<$ pick $S n\rangle$
using $\langle S u c n<$ card $S\rangle$ dual－order．strict－trans less－Suc－eq by auto
then show card $\{a \in S . a<$ pick $S($ Suc $n)\}=$ Suc $n$
proof－
have $\{a \in S . a<$ pick $S(S u c n)\}=\{a \in S . a<$ pick $S n\} \cup\{$ pick $S n\}$
apply（rule subset－antisym；rule subsetI）
using pick．simps not－less－Least 〈pick $S(S u c n)>$ pick $S n\rangle\langle p i c k S n \in S\rangle$
by fastforce＋
then have card $\{a \in S . a<$ pick $S(S u c n)\}=\operatorname{card}\{a \in S . a<$ pick $S n\}+$ card $\{$ pick $S n\}$ using card－Un－disjoint by auto
then show ？thesis by（metis One－nat－def Suc－eq－plus1 «card $\{a \in S . a<p i c k$ $S n\}=n>$ card．empty card－insert－if empty－iff finite．emptyI）
qed
qed
lemma card－le－pick－le：
assumes $n<$ card $S$
and pick $S n \geq i$
shows card $\{a \in S . a<i\} \leq n$
using assms proof（induction $n$ arbitrary：$i$ ）
case 0
then show ？case unfolding pick．simps using not－less－Least
by（metis（no－types，lifting）Collect－empty－eq card－0－eq card－ge－0－finite dual－order．strict－trans1
leI le－0－eq）
next
case（Suc n）
have card $\{a \in S . a<$ pick $S n\} \leq n$ using Suc by（simp add：less－eq－Suc－le nat－less－le）
have $\{a \in S . a<i\} \subseteq\{a \in S . a<\operatorname{pick} S$（Suc n）$\}$ using Suc．prems（2）by
auto
have $\{a \in S . a<$ pick $S($ Suc $n)\}=\{a \in S . a<$ pick $S n\} \cup\{$ pick $S n\}$
apply (rule subset-antisym; rule subsetI)
using pick.simps not-less-Least pick-mono-le[OF Suc.prems(1), of n, OF lessI] pick-in-set-le[of n S] Suc by fastforce+
then have card $\{a \in S . a<i\} \leq$ card $\{a \in S . a<$ pick $S n\}+$ card $\{$ pick $S n\}$
using card-Un-disjoint card-mono $[O F-\langle\{a \in S . a<i\} \subseteq\{a \in S . a<$ pick $S$ (Suc n) \}>] by simp
then show ?case using 〈card $\{a \in S . a<$ pick $S n\} \leq n\rangle$ by auto
qed

## lemma

assumes $n<$ card $S \vee$ infinite $S$
shows
pick-in-set:pick $S n \in S$ and
card-le-pick: $i \leq$ pick $S n==>$ card $\{a \in S . a<i\} \leq n$ and
card-pick: card $\{a \in S . a<$ pick $S n\}=n$ and
pick-mono: $m<n \Longrightarrow$ pick $S m<$ pick $S n$
using assms pick-in-set-inf pick-in-set-le card-pick-inf card-pick-le card-le-pick-le card-le-pick-inf
pick-mono-inf pick-mono-le by auto

```
lemma pick-card:
pick \(I(\) card \(\{a \in I . a<i\})=(\) LEAST \(a . a \in I \wedge a \geq i)\)
proof (induction i)
    case 0
    then show ?case by (simp add: pick-in-set-le)
next
    case (Suc i)
    then show ?case
    proof (cases \(i \in I\) )
        case True
        then have 1:pick \(I\) (card \(\{a \in I . a<i\})=i\) by (metis (mono-tags, lifting)
Least-equality Suc.IH order-refl)
    have \(\{a \in I . a<\) Suc \(i\}=\{a \in I . a<i\} \cup\{i\}\) using True by auto
    then have 2:card \(\{a \in I . a<S u c i\}=\) Suc (card \(\{a \in I . a<i\}\) ) by auto
    then show ?thesis unfolding 2 pick.simps 1 using Suc-le-eq by auto
    next
        case False
    then have \(1:\{a \in I . a<\) Suc \(i\}=\{a \in I . a<i\}\) using Collect-cong less-Suc-eq
by auto
    have 2: \(\wedge a .(a \in I \wedge\) Suc \(i \leq a)=(a \in I \wedge i \leq a)\) using False Suc-leD
le-less-Suc-eq not-le by blast
    then show ?thesis unfolding 12 using Suc.IH by blast
    qed
qed
lemma pick-card-in-set: \(i \in I \Longrightarrow\) pick \(I(\) card \(\{a \in I . a<i\})=i\)
    unfolding pick-card using Least-equality order-refl by (metis (no-types, lifting))
```


## 36 Sublist

```
lemma nth-nths-card:
assumes j<length xs
and j\inJ
shows nths xs J!card {j0.j0<j^j0\inJ} = xs!j
using assms proof (induction xs rule:rev-induct)
    case Nil
    then show ?case using gr-implies-not0 list.size(3) by auto
next
    case (snoc x xs)
    then show ?case
    proof (cases j < length xs)
        case True
        have {j0.j0<j^j0\inJ}\subset{i.i<length xs \wedge i\inJ}
            using True snoc.prems(2) by auto
            then have card {j0.j0<j^j0\inJ}<length (nths xs J) unfolding
length-nths
            using psubset-card-mono[of {i. i< length xs ^ i\inJ}] by simp
            then show ?thesis unfolding nths-append nth-append by (simp add: True
snoc.IH snoc.prems(2))
    next
        case False
        then have length xs = j
            using length-append-singleton less-antisym snoc.prems(1) by auto
    then show ?thesis unfolding nths-append nth-append length-nths <length xs =
j>
            by (simp add: snoc.prems(2))
    qed
qed
lemma pick-reduce-set:
assumes i<card {a. a<m\wedgea\inI}
shows pick I i = pick {a.a<m^a\inI} i
using assms proof (induction i)
    let ?L = LEAST a. a\in{a.a<m^a\inI}
    case 0
    then have {a.a<m\wedgea\inI}\not={} using card.empty less-numeral-extra(3)
by fastforce
    then have ?L }\inI?L<m\mathrm{ by (metis (mono-tags, lifting) Collect-empty-eq LeastI
mem-Collect-eq)+
    have }\x.x\in{a.a<m\wedgea\inI}\Longrightarrow?L\leqx by (simp add: Least-le
    have \x. x }\=I\Longrightarrow?L\leq
        by (metis (mono-tags)<?L<m><\x. x < {a. a<m^a\inI} \Longrightarrow?L\leqx\rangle
dual-order.strict-trans2 le-cases mem-Collect-eq)
    then show ?case unfolding pick.simps using Least-equality[of \lambdax. x\inI, OF
< LL E I`] by blast
next
    case (Suc i)
```

let $? L=L E A S T x . x \in\{a . a<m \wedge a \in I\} \wedge$ pick $I i<x$
have 0:pick $\{a . a<m \wedge a \in I\} i=$ pick $I i$ using Suc-lessD Suc by linarith
then have $? L \in\{a . a<m \wedge a \in I\}$ pick $I i<$ ? $L$
using LeastI[of $\lambda a . a \in\{a . a<m \wedge a \in I\} \wedge$ pick $I i<a]$ using Suc.prems pick-in-set-le pick-mono-le by fastforce+
then have ? $L \in I$ by blast
show ?case unfolding pick.simps 0 using Least-equality[of $\lambda a . a \in I \wedge$ pick $I$ $i<a ? L]$
by (metis (no-types, lifting) Least-le $\langle ? L \in\{a . a<m \wedge a \in I\}\rangle\langle p i c k I i<$ ?L> mem-Collect-eq not-le not-less-iff-gr-or-eq order.trans)
qed
lemma nth-nths:
assumes $i<$ card $\{i . i<$ length $x s \wedge i \in I\}$
shows nths xs $I!i=x s!$ pick $I i$
proof -
have $\{a \in\{i . i<$ length $x s \wedge i \in I\} . a<$ pick $\{i . i<$ length $x s \wedge i \in I\} i\}$ $=\{a . a<$ pick $\{i . i<$ length $x s \wedge i \in I\} i \wedge a \in I\}$
using assms pick-in-set by fastforce
then have card $\{a . a<$ pick $\{i . i<l e n g t h ~ x s ~ \wedge i \in I\} i \wedge a \in I\}=i$ using card-pick-le[OF assms] by simp
then have nths xs $I!i=x s!$ pick $\{i . i<$ length $x s \wedge i \in I\} i$
using nth-nths-card[where $j=$ pick $\{i . i<l e n g t h ~ x s \wedge i \in I\} i$, of xs $I]$
assms pick-in-set pick-in-set by auto
then show ?thesis using pick-reduce-set using assms by auto qed
lemma pick-UNIV: pick UNIV $j=j$
by (induction $j$, simp, metis (no-types, lifting) LeastI pick.simps(2) Suc-mono UNIV-I less-Suc-eq not-less-Least)
lemma pick-le:
assumes $n<\operatorname{card}\{a . a<i \wedge a \in S\}$
shows pick $S n<i$
proof -
have $0:\{a \in\{a . a<i \wedge a \in S\} . a<i\}=\{a . a<i \wedge a \in S\}$ by blast
show ?thesis apply (rule ccontr)
using card-le-pick-le[OF assms, unfolded pick-reduce-set[OF assms, symmetric], of $i$, unfolded 0]
assms not-less not-le by blast
qed
lemma prod-list-complementary-nthss:
fixes $f::^{\prime} a \Rightarrow$ ' $b::$ comm-monoid-mult
shows prod-list $(\operatorname{map} f x s)=\operatorname{prod}-l i s t(\operatorname{map} f(n t h s x s A)) * \operatorname{prod}-l i s t(\operatorname{map} f(n t h s$
xs ( $-A$ )) )
proof (induction xs rule:rev-induct)
case Nil
then show? case by simp

```
next
    case (snoc x xs)
    show ?case unfolding map-append prod-list.append nths-append nths-singleton
snoc
    by (cases (length xs)\inA; simp;metis mult.assoc mult.commute)
qed
lemma nths-zip: nths (zip xs ys) I = zip (nths xs I) (nths ys I)
proof (rule nth-equalityI)
    show length (nths (zip xs ys)I) = length (zip (nths xs I) (nths ys I))
    proof (cases length xs \leq length ys)
        case True
        then have {i.i< length xs ^i\inI}\subseteq{i.i<length ys }\wedgei\inI} by (simp
add: Collect-mono less-le-trans)
    then have card {i. i< length xs ^i\inI}\leqcard {i.i<length ys }\wedgei\inI
    by (metis (mono-tags, lifting) card-mono finite-nat-set-iff-bounded mem-Collect-eq)
    then show ?thesis unfolding length-nths length-zip using True using min-def
by linarith
    next
        case False
        then have {i.i< length ys }\wedgei\inI}\subseteq{i.i<length xs \wedgei\inI} by (simp
add: Collect-mono less-le-trans)
    then have card {i. i< length ys ^i\inI}\leqcard {i.i<length xs ^i\inI}
    by (metis (mono-tags, lifting) card-mono finite-nat-set-iff-bounded mem-Collect-eq)
    then show ?thesis unfolding length-nths length-zip using False using min-def
by linarith
    qed
    show nths (zip xs ys) I!i=zip (nths xs I) (nths ys I)!i if i<length (nths
(zip xs ys)I) for i
    proof -
    have i< length (nths xs I) i< length (nths ys I)
        using that by (simp-all add:<length (nths (zip xs ys) I) = length (zip (nths
xs I) (nths ys I))>)
    show nths (zip xs ys) I!i=zip (nths xs I) (nths ys I)!i
            unfolding nth-nths[OF<i< length (nths (zip xs ys) I)>[unfolded length-nths]]
            unfolding nth-zip[OF <i< length (nths xs I)\rangle\langlei < length (nths ys I)>]
            unfolding nth-zip[OF pick-le[OF<i < length (nths xs I)>[unfolded length-nths]]
                pick-le[OF<i< length (nths ys I)>[unfolded length-nths]]]
    by (metis (full-types) <i< length (nths xs I)\rangle\langlei< length (nths ys I)> length-nths
nth-nths)
    qed
qed
```


## 37 weave

definition weave :: nat set $\Rightarrow{ }^{\prime}$ 'a list $\Rightarrow{ }^{\prime}$ a list $\Rightarrow$ 'a list where
weave $A$ xs ys $=\operatorname{map}(\lambda i$. if $i \in A$ then xs! (card $\{a \in A . a<i\})$ else ys! $(\operatorname{card}\{a \in-A$. $a<i\}))[0 . .<$ length $x s+$ length $y s]$
lemma length－weave：
shows length（weave $A$ xs ys）＝length $x s+$ length ys
unfolding weave－def length－map by simp
lemma nth－weave：
assumes $i<$ length（weave $A$ xs ys）
shows weave $A$ xs ys $!i=($ if $i \in A$ then xs！（card $\{a \in A . a<i\})$ else ys！（card $\{a \in-A$ ．
$a<i\})$ ）
proof－
have $i<$ length $x s+$ length ys using length－weave using assms by metis
then have $i<$ length $[0 . .<$ length $x s+l$ length $y s]$ by auto
then have $[0 . .<$ length $x s+$ length $y s]!i=i$
by（metis $\langle i<$ length $x s+$ length ys〉 add．left－neutral nth－upt）
then show ？thesis
unfolding weave－def nth－map $[O F<i<l e n g t h[0 . .<$ length $x s+l e n g t h y s]\rangle]$ by presburger
qed
lemma weave－append1：
assumes length $x s+$ length $y s \in A$
assumes length $x s=$ card $\{a \in A . a<$ length $x s+$ length $y s\}$
shows weave $A(x s$＠$[x]) y s=w e a v e ~ A x s y s ~ @ ~[x]$
proof（rule nth－equalityI）
show length（weave $A$（xs＠$[x]$ ）ys）＝length（weave A xs ys＠$[x]$ ）
unfolding weave－def length－map by simp
show weave $A(x s @[x]) y s!i=($ weave $A$ xs ys＠$[x])!i$
if $i<$ length（weave $A(x s @[x]) y s)$ for $i$
proof－
show weave $A(x s @[x]) y s!i=($ weave $A$ xs $y s$＠$[x])!i$
proof（cases $i=$ length $x s+$ length $y s)$
case True
then have（weave $A$ xs ys＠［x］）！$i=x$ using length－weave by（metis nth－append－length）
have card $\{a \in A . a<i\}=$ length $x s$ using assms（2）True by auto
then show ？thesis unfolding nth－weave $[O F<i<$ length（weave $A(x s @[x]$ ）
$y s)\rangle$ ］
〈（weave $A$ xs ys＠$[x])!i=x\rangle$ using True assms（1）by simp
next
case False
have $i<$ length（weave $A$ xs ys）using $\langle i<$ length（weave $A(x s @[x]) y s)\rangle$
＜length（weave $A(x s$＠$[x]) y s)=$ length（weave $A$ xs ys＠$[x]$ ）〉length－append－singleton
length－weave less－antisym False by fastforce
then have（weave $A$ xs ys＠$[x]$ ）！$i=($ weave $A$ xs ys）！i by（simp add：
nth－append）
\｛
assume $i \in A$
have $i<$ length $x s+$ length $y s$ by（metis $\langle i<l e n g t h ~(w e a v e ~ A x s ~ y s)\rangle$
length－weave）
then have $\{a \in A . a<i\} \subset\{a \in A . a<$ length $x s+$ length $y s\}$
using assms（1）〈i＜length $x s+$ length $y s\rangle\langle i \in A\rangle$ by auto
then have card $\{a \in A . a<i\}<$ card $\{a \in A . a<$ length $x s+$ length ys $\}$
using psubset－card－mono［of $\{a \in A . a<$ length $x s+l e n g t h ~ y s\}\{a \in A . a$
$<i\}$ ］by simp
then have（xs＠$[x]$ ）！card $\{a \in A . a<i\}=x s!$ card $\{a \in A . a<i\}$
by（metis（no－types，lifting）assms（2）nth－append）
\}
then show ？thesis unfolding nth－weave $[O F<i<l e n g t h(w e a v e ~ A(x s @[x])$ $y s)>]$
$\langle($ weave $A$ xs ys＠$[x])!i=($ weave $A$ xs $y s)!i\rangle n t h-w e a v e[O F<i<$ length （weave $A$ xs $y s)\rangle$ ］
by $\operatorname{simp}$ qed
qed
qed
lemma weave－append2：
assumes length $x s+$ length $y s \notin A$
assumes length ys $=$ card $\{a \in-A . a<$ length $x s+$ length $y s\}$
shows weave $A$ xs（ys＠［y］）＝weave A xs ys＠［y］
proof（rule nth－equalityI）
show length（weave Axs（ys＠［y］））＝length（weave A xs ys＠［y］）
unfolding weave－def length－map by simp
show weave $A$ xs（ys＠［y］）！$i=($ weave $A$ xs ys＠$[y])!i$ if $i<l e n g t h(w e a v e$ $A$ xs（ $y s$＠$[y])$ ）for $i$
proof－
show weave $A$ xs $(y s$＠$[y])!i=($ weave $A$ xs ys＠$[y])!i$
proof（cases $i=$ length $x s+$ length $y s$ ）
case True
then have（weave $A$ xs ys＠［y］）！$i=y$ using length－weave by（metis nth－append－length）
have card $\{a \in-A . a<i\}=$ length ys using assms（2）True by auto then show ？thesis unfolding nth－weave $[O F<i<l e n g t h$（weave $A$ xs（ys＠ ［y］））＞］
«（weave $A$ xs ys＠$[y])!i=y\rangle$ using True assms（1）by simp
next
case False
have $i<$ length（weave $A$ xs ys）using $\langle i<$ length（weave $A$ xs（ys＠［y］））＞
〈length（weave A xs（ys＠$[y])$ ）＝length（weave A xs ys＠$[y]$ ）〉 length－append－singleton length－weave less－antisym False by fastforce
then have（weave $A$ xs ys＠$[y]$ ）！$i=($ weave $A$ xs ys）！$i$ by（simp add：
nth－append）
\｛
assume $i \notin A$
have $i<l e n g t h ~ x s+l e n g t h ~ y s ~ b y ~(m e t i s ~ 〈 i<l e n g t h ~(w e a v e ~ A ~ x s ~ y s)\rangle$ length－weave）
then have $\{a \in-A . a<i\} \subset\{a \in-A . a<$ length $x s+$ length $y s\}$
using assms（1）〈i＜length xs + length $y s\rangle\langle i \notin A\rangle$ by auto
then have card $\{a \in-A . a<i\}<\operatorname{card}\{a \in-A . a<$ length $x s+$ length
$y s\}$
using psubset-card-mono[of $\{a \in-A . a<$ length $x s+l e n g t h ~ y s\}\{a \in-A$. $a<i\}$ ] by simp
then have (ys @ [y])!card $\{a \in-A . a<i\}=y s!$ card $\{a \in-A . a<i\}$ by (metis (no-types, lifting) assms(2) nth-append)
\}
then show ?thesis unfolding nth-weave $[O F<i<l e n g t h$ (weave $A$ xs (ys @ $[y]))\rangle]$
$\langle($ weave $A$ xs ys @ $[y])!i=($ weave $A$ xs ys $)!i\rangle$ nth-weave $[$ OF $\langle i<$ length (weave $A$ xs $y s$ )〉]
by $\operatorname{simp}$
qed
qed
qed
lemma nths-nth:
assumes $n \in A \quad n<$ length $x s$
shows nths xs $A!($ card $\{i . i<n \wedge i \in A\})=x s!n$
using assms proof (induction xs rule:rev-induct)
case (snoc $x x s$ )
then show ?case
proof (cases $n=$ length $x s$ )
case True
then show ?thesis unfolding nths-append $[$ of $x s[x] A]$ nth-append
using length-nths[of xs A] nths-singleton snoc.prems(1) by auto
next
case False
then have $n<$ length xs using snoc by auto
then have 0 :nths xs $A!$ card $\{i . i<n \wedge i \in A\}=x s!n$ using snoc by auto
have $\{i . i<n \wedge i \in A\} \subset\{i . i<$ length $x s \wedge i \in A\}$ using $\langle n<$ length $x s\rangle$ snoc by force
then have card $\{i . i<n \wedge i \in A\}<$ length (nths xs $A$ ) unfolding length-nths by (simp add: psubset-card-mono)
then show ?thesis unfolding nths-append $[$ of $x s[x] A]$ nth-append using 0 by (simp add: $\langle n<$ length $x s\rangle$ )
qed
qed $\operatorname{simp}$
lemma list-all2-nths:
assumes list-all2 $P$ (nths xs A) (nths ys A)
and list-all2 $P$ (nths $x s(-A)$ ) (nths ys $(-A))$
shows list-all2 P xs ys
proof -
have length $x s=$ length ys
proof (rule ccontr; cases length $x s<l e n g t h ~ y s)$
case True
then show False
proof (cases length $x s \in A$ )

```
    case False
    have {i. i< length xs ^i\in-A}\subset{i.i<length ys }\wedgei\in-A
        using False «length xs < length ys> by force
    then have length (nths ys (-A)) > length (nths xs ( }-A\mathrm{ ))
        unfolding length-nths by (simp add: psubset-card-mono)
        then show False using assms(2) list-all2-lengthD not-less-iff-gr-or-eq by
blast
    next
        case True
        have {i. i< length xs ^i\inA}\subset{i.i<length ys }\wedgei\inA
            using True <length xs < length ys> by force
        then have length (nths ys A) > length (nths xs A)
            unfolding length-nths by (simp add: psubset-card-mono)
            then show False using assms(1) list-all2-lengthD not-less-iff-gr-or-eq by
blast
    qed
    next
    assume length xs \not= length ys
    case False
    then have length xs > length ys using <length xs \not= length ys> by auto
    then show False
    proof (cases length ys }\inA\mathrm{ )
        case False
        have {i. i< length ys }\wedgei\in-A}\subset{i.i<length xs \wedgei\in-A
            using False〈length xs > length ys` by force
        then have length (nths xs ( }-A)\mathrm{ ) > length (nths ys ( }-A\mathrm{ ))
            unfolding length-nths by (simp add: psubset-card-mono)
    then show False using assms(2) list-all2-lengthD dual-order.strict-implies-not-eq
by blast
    next
        case True
        have {i. i< length ys }\wedgei\inA}\subset{i.i<length xs \wedgei\inA
            using True 〈length xs > length ys> by force
        then have length (nths xs A) > length (nths ys A)
        unfolding length-nths by (simp add: psubset-card-mono)
    then show False using assms(1) list-all2-lengthD dual-order.strict-implies-not-eq
by blast
    qed
qed
have }\n.n< length xs \LongrightarrowP(xs!n)(ys!n
proof -
    fix n assume n< length xs
    then have }n<length ys using <length xs = length ys> by aut
    then show P (xs!n) (ys!n)
    proof (cases n\inA)
            case True
            have {i. i<n^i\inA}\subset{i.i<length xs ^i\inA} using <n<length xs>
<n\inA> by force
```

```
    then have card {i.i<n\wedgei\inA}<length (nths xs A) unfolding length-nths
        by (simp add: psubset-card-mono)
    show ?thesis using nths-nth[OF <n\inA〉\langlen < length xs〉] nths-nth[OF <n\inA〉
<n < length ys`]
        list-all2-nthD[OF assms(1), of card {i.i<n\wedgei\inA}] length-nths
        by (simp add: <card {i.i<n\wedge i\inA}< length (nths xs A)〉)
    next
            case False then have }n\in-A\mathrm{ by auto
    have {i.i<n\wedgei\in-A}\subset{i.i<length xs \wedge i\in-A} using <n<length
xs\rangle\langlen\in-A\rangle by force
            then have card {i.i<n\wedgei\in-A}<length (nths xs (-A)) unfolding
length-nths
            by (simp add: psubset-card-mono)
            show ?thesis using nths-nth[OF <n\in-A〉\langlen < length xs〉] nths-nth[OF
<n\in-A〉\langlen< length ys`]
            list-all2-nthD[OF assms(2), of card {i. i<n\wedge i\in - A}] length-nths
            using <card {i.i<n\wedgei\in-A}< length (nths xs (-A))> by auto next
        qed
    qed
    then show ?thesis using <length xs = length ys> list-all2-all-nthI by blast
qed
lemma nths-weave:
assumes length xs = card {a\inA. a< length xs + length ys }
assumes length ys = card {a\in(-A).a<length xs + length ys }
shows nths (weave A xs ys) A = xs ^ nths (weave A xs ys) (-A) = ys
using assms proof (induction length xs + length ys arbitrary: xs ys)
    case 0
    then show ?case
    unfolding weave-def nths-map by simp
next
    case (Suc l)
    then show ?case
    proof (cases l l\inA)
        case True
    then have l\in{a\inA.a<length xs + length ys} using Suc.hyps mem-Collect-eq
zero-less-Suc by auto
    then have length xs > 0 using Suc by fastforce
            then obtain x\mp@subsup{s}{}{\prime}}x\mathrm{ where xs =xs' @ [x] by (metis append-butlast-last-id
length-greater-0-conv)
    then have l= length xs' + length ys using Suc.hyps by simp
    have length-xs':length xs' = card {a\inA.a<length xs' }+\mathrm{ length ys }
    proof -
            have {a\inA.a<length xs + length ys} = {a\inA.a<length xs' + length
ys}\cup{l}
            using <xs = xs'@ @x]><l\in{a\inA.a<length xs + length ys }><l = length
xs}\mp@subsup{}{}{\prime}+ length ys
            by force
            then have card {a\inA.a<length xs + length ys } = card {a\inA.a<
```

length $x s^{\prime}+$ length $\left.y s\right\}+1$
using $\left\langle l=\right.$ length $x s^{\prime}+$ length $\left.y s\right\rangle$ by fastforce
then show ?thesis by (metis One-nat-def Suc.prems(1) 〈xs $\left.=x s^{\prime} @[x]\right\rangle$
add-right-imp-eq
length-Cons length-append list.size(3))
qed
have length-ys:length ys $=$ card $\left\{a \in-A . a<\right.$ length $x s^{\prime}+$ length $\left.y s\right\}$
proof -
have $l \notin\{a \in-A . a<$ length $x s+$ length $y s\}$ using $\langle l \in A\rangle\left\langle l=\right.$ length $x s^{\prime}+$
length ys> by blast
have $\{a \in-A . a<$ length $x s+$ length $y s\}=\left\{a \in-A . a<\right.$ length $x s^{\prime}+$
length $y s\}$
apply (rule subset-antisym)
using $\left\langle l=\right.$ length $x s^{\prime}+$ length $\left.y s\right\rangle\langle S u c l=$ length $x s+l e n g t h ~ y s\rangle\langle l \notin\{a \in$
$-A . a<$ length $x s+$ length $y s\}>$
apply (metis (no-types, lifting) Collect-mono less-Suc-eq mem-Collect-eq)
using Collect-mono Suc.hyps(2) $\left\langle l=\right.$ length $x s^{\prime}+$ length ys by auto
then show ?thesis using Suc.prems(2) by auto
qed
have length $x s^{\prime}+$ length $y s \in A$ using $\langle l \in A\rangle\left\langle l=\right.$ length $x s^{\prime}+$ length $\left.y s\right\rangle$ by
blast
then have nths（weave $A$ xs ys）$A=n$ ths（weave $A x s^{\prime} y s$＠$[x]$ ）$A$ unfolding $\left\langle x s=x s^{\prime} @[x]\right\rangle$ using weave－append1 $\left[O F 〈\right.$ length $x s^{\prime}+$ length $\left.y s \in A\right\rangle$ length－xs $\}$ by metis
also have $\ldots=$ nths $\left(\right.$ weave $\left.A x s^{\prime} y s\right) A @ n t h s[x]\left\{a . a+\left(\right.\right.$ length $x s^{\prime}+$ length $y s) \in A\}$
using nths－append length－weave by metis
also have $\ldots=$ nths（weave $A x s^{\prime} y s$ ）$A$＠$[x]$
using nths－singleton 〈length $x s^{\prime}+$ length $y s \in A$ 〉 by auto
also have $\ldots=x s$ using Suc．hyps（1）［OF $\left\langle l=\right.$ length $x s^{\prime}+$ length $\left.y s\right\rangle$ length－xs ${ }^{\prime}$ length－ys］
$\left\langle x s=x s^{\prime} @[x]\right\rangle$ by presburger
finally have nths（weave $A$ xs ys）$A=x s$ by metis
have nths（weave A xs ys）$(-A)=$ nths（weave $\left.A x s^{\prime} y s @[x]\right)(-A)$ unfolding $\left\langle x s=x s^{\prime} @[x]\right\rangle$ using weave－append1 $\left[O F 〈 l e n g t h ~ x s^{\prime}+\right.$ length $\left.y s \in A\right\rangle$ length－xs $\dagger$ by metis
also have $\ldots=$ nths（weave $\left.A x s^{\prime} y s\right)(-A) @ n t h s[x]\left\{a . a+\left(\right.\right.$ length $x s^{\prime}+$ length $y s) \in(-A)\}$
using nths－append length－weave by metis
also have $\ldots=$ nths（weave $A x s^{\prime}$ ys）$(-A)$
using nths－singleton 〈length $x s^{\prime}+$ length $y s \in A$ 〉 by auto
also have $\ldots=y s$
using Suc．hyps（1）［OF $\left\langle l=\right.$ length $x s^{\prime}+$ length $\left.y s\right\rangle$ length－xs ${ }^{\prime}$ length－ys $]$ by presburger
finally show ？thesis using «nths（weave $A x s y s) A=x s\rangle$ by auto next
case False
then have $l \notin\{a \in A . a<$ length $x s+$ length ys $\}$ using Suc．hyps mem－Collect－eq zero－less－Suc by auto
then have length ys $>0$ using Suc by fastforce
then obtain $y s^{\prime} y$ where $y s=y s^{\prime} @[y]$ by（metis append－butlast－last－id length－greater－0－conv）
then have $l=$ length $x s+$ length ys＇using Suc．hyps by simp
have length－ys＇：length $y s^{\prime}=$ card $\{a \in-A . a<$ length $x s+l e n g t h ~ y s '\}$
proof－
have $\{a \in-A . a<$ length $x s+$ length $y s\}=\{a \in-A . a<$ length $x s+$ length $\left.y s^{\prime}\right\} \cup\{l\}$
using $\left\langle y s=y s^{\prime} @[y]\right\rangle\langle l \notin\{a \in A . a<$ length $x s+$ length $y s\}\rangle\langle l=$ length $x s+$ length $y s^{\prime}$＞
by force
then have card $\{a \in-$ A．$a<$ length $x s+$ length $y s\}=\operatorname{card}\{a \in-A . a<$ length $x s+$ length $\left.y s^{\prime}\right\}+1$ using $\langle l=$ length $x s+$ length $y s$＇$>$ by fastforce
then show ？thesis by（metis One－nat－def Suc．prems（2）〈ys＝ys＇＠［y］〉 add－right－imp－eq
length－Cons length－append list．size（3））
qed
have length－xs：length $x s=$ card $\{a \in A . a<$ length $x s+$ length $y s '\}$
proof－
have $l \notin\{a \in A . a<$ length $x s+$ length $y s\}$ using $\langle l \notin A\rangle\langle l=$ length $x s+$ length $y s^{\prime}>$ by blast
have $\{a \in A . a<$ length $x s+$ length $y s\}=\{a \in A . a<$ length $x s+$ length $\left.y s^{\prime}\right\}$
apply（rule subset－antisym）
using $\left\langle l=\right.$ length $x s+$ length $\left.y s^{\prime}\right\rangle\langle S u c l=$ length $x s+l e n g t h ~ y s\rangle\langle l \notin\{a \in$ A．$a<$ length $x s+$ length $y s\}>$
apply（metis（no－types，lifting）Collect－mono less－Suc－eq mem－Collect－eq）
using Collect－mono Suc．hyps（2）$<l=$ length $x s+l e n g t h ~ y s '>$ by auto
then show ？thesis using Suc．prems（1）by auto
qed
have length $x s+l e n g t h ~ y s^{\prime} \notin A$ using $\langle l \notin A\rangle\left\langle l=\right.$ length $x s+$ length $\left.y s s^{\prime}\right\rangle$ by blast
then have nths（weave $A$ xs ys）$A=n$ ths（weave $A$ xs ys ${ }^{\prime} @[y]$ ）$A$ unfolding
$\left\langle y s=y s^{\prime} @[y]\right\rangle$ using weave－append2［OF〈length $x s+$ length $\left.y s^{\prime} \notin A\right\rangle$ length－ys $\}$ by metis
also have $\ldots=$ nths（weave $A$ xs ys $\left.{ }^{\prime}\right) A$＠nths $[y]\{a . a+($ length $x s+$ length $\left.\left.y s^{\prime}\right) \in A\right\}$
using nths－append length－weave by metis
also have $\ldots=$ nths（weave $A$ xs ys＇）$A$
using nths－singleton 〈length $x s+$ length $y s^{\prime} \notin A$ 〉 by auto
also have ．．．$=x s$
using Suc．hyps（1）［OF $\left\langle l=\right.$ length $x s+$ length $\left.y s^{\prime}\right\rangle$ length－xs length－ys $]$ by auto
finally have nths（weave $A$ xs ys）$A=x s$ by auto
have nths（weave A xs ys）$(-A)=$ nths（weave $A$ xs ys $\left.{ }^{\prime} @[y]\right)(-A)$ unfolding $\left\langle y s=y s^{\prime} @[y]\right\rangle$ using weave－append2［OF〈length xs + length $\left.y s^{\prime} \notin A\right\rangle$ length－ys $\dagger$ by metis
also have $\ldots=$ nths（weave $A$ xs $\left.y s^{\prime}\right)(-A) @ n t h s[y]\{a . a+($ length $x s+$ length $\left.\left.y s^{\prime}\right) \in(-A)\right\}$
using nths－append length－weave by metis
also have $\ldots=$ nths（weave $A$ xs $\left.y s^{\prime}\right)(-A) @[y]$
using $n$ ths－singleton 〈length $x s+$ length $y s^{\prime} \notin A$ 〉by auto
also have $\ldots=y s$
using Suc．hyps（1）［OF $\langle l=$ length $x s+$ length $y s$＇$>$ length－xs length－ys $]\langle y s=$ $y s^{\prime} @[y]>$ by $\operatorname{simp}$
finally show ？thesis using «nths（weave $A$ xs ys）$A=x s\rangle$ by auto qed
qed
lemma set－weave：
assumes length $x s=$ card $\{a \in A . a<$ length $x s+$ length $y s\}$
assumes length ys $=$ card $\{a \in-A . a<$ length $x s+l e n g t h ~ y s ~\}$
shows set（weave $A$ xs ys）$=$ set $x s \cup$ set ys
proof
show set（weave $A$ xs ys）$\subseteq$ set $x s \cup$ set ys
proof
fix $x$ assume $x \in$ set（weave $A$ xs ys）
then obtain $i$ where weave $A$ xs ys ！$i=x i<$ length（weave $A$ xs ys）by （meson in－set－conv－nth）
show $x \in$ set $x s \cup$ set ys
proof（cases $i \in A$ ）
case True
then have $i \in\{a \in A . a<$ length $x s+$ length $y s\}$ unfolding length－weave by（metis $\langle i<$ length（weave $A$ xs ys）＞length－weave mem－Collect－eq）
then have $\{a \in A . a<i\} \subset\{a \in A . a<$ length $x s+$ length $y s\}$
using Collect－mono $\langle i<$ length（weave $A$ xs ys）$\rangle[u n f o l d e d ~ l e n g t h-w e a v e] ~$ le－Suc－ex less－imp－le－nat trans－less－add1
le－neq－trans less－irrefl mem－Collect－eq by auto
then have card $\{a \in A . a<i\}<$ card $\{a \in A . a<$ length $x s+l e n g t h y s\}$ by （simp add：psubset－card－mono）
then show $x \in$ set $x s \cup$ set ys
unfolding nth－weave $[O F<i<l e n g t h(w e a v e ~ A x s y s)\rangle$ ，unfolded «weave $A$ xs $y s!i=x\rangle$ ］using True
using UnI1 assms（1）nth－mem by auto
next
case False
have $i \notin A \Longrightarrow i \in\{a \in-A . a<$ length $x s+$ length $y s\}$ unfolding length－weave by（metis ComplI 〈i＜length（weave $A$ xs ys）＞length－weave mem－Collect－eq） then have $\{a \in-A . a<i\} \subset\{a \in-A . a<$ length $x s+$ length $y s\}$
using Collect－mono $\langle i<$ length（weave $A$ xs ys）$\rangle[u n f o l d e d ~ l e n g t h-w e a v e] ~$
le－Suc－ex less－imp－le－nat trans－less－add1 le－neq－trans less－irrefl mem－Collect－eq using False by auto
then have card $\{a \in-A . a<i\}<\operatorname{card}\{a \in-A . a<$ length $x s+$ length $y s\}$

```
by (simp add: psubset-card-mono)
```

    then show \(x \in\) set \(x s \cup\) set \(y s\)
            unfolding nth-weave \([O F<i<\) length (weave \(A\) xs ys) 〉, unfolded 〈weave \(A\) xs
    ys $!i=x\rangle$ ] using False
using UnI1 assms(2) nth-mem by auto
qed
qed
show set $x s \cup$ set $y s \subseteq$ set (weave $A$ xs ys)
using nths-weave[OF assms] by (metis Un-subset-iff set-nths-subset)
qed
lemma weave-complementary-nthss [simp]:
weave $A$ (nths xs $A$ ) (nths xs $(-A))=x s$
proof (induction xs rule:rev-induct)
case Nil
then show?case by (metis gen-length-def length-0-conv length-code length-weave nths-nil)
next
case (snoc $x x s$ )
have length-xs:length $x s=$ length ( $n$ ths $x s A$ ) + length $(n t h s x s(-A)$ ) by (metis length-weave snoc.IH)
show? case
proof (cases (length $x s) \in A$ )
case True
have 0:length (nths xs $A$ ) + length (nths xs $(-A)) \in A$ using length-xs True by metis
have 1:length (nths xs $A)=$ card $\{a \in A . a<$ length (nths xs $A)+$ length (nths xs ( $-A$ )) \}
using length-nths $[$ of xs $A]$ by (metis (no-types, lifting) Collect-cong length-xs)
have 2:nths (xs @ $[x]$ ) $A=n$ ths xs $A$ @ $[x]$
unfolding nths-append [of xs [x] A] using nths-singleton True by auto
have 3:nths (xs @ $[x])(-A)=$ nths xs $(-A)$
unfolding nths-append $[$ of $x s[x]-A]$ using True by auto
show ?thesis unfolding 23 weave-append1[OF 0 1] snoc.IH by metis
next
case False
have 0:length (nths xs A) + length (nths xs $(-A)) \notin A$ using length-xs False by metis
have 1:length (nths xs $(-A))=$ card $\{a \in-A . a<$ length ( $n$ ths xs $A$ ) + length ( $n$ ths xs ( $-A$ )) \}
using length-nths $[$ of $x s-A]$ by (metis (no-types, lifting) Collect-cong length-xs)
have 2:nths (xs @ $[x]$ ) $A=n$ ths xs $A$
unfolding nths-append [of xs [x] A] using nths-singleton False by auto
have 3:nths $(x s$ @ $[x])(-A)=n$ ths $x s(-A) @[x]$
unfolding nths-append $[$ of $x s[x]-A]$ using False by auto
show ?thesis unfolding 23 weave-append2[OF 0 1] snoc.IH by metis qed
qed
lemma length-nths': length (nths xs $I$ ) $=$ card $\{i \in I . i<l e n g t h ~ x s\}$
unfolding length-nths by meson
end

## 38 Submatrices

theory DL-Submatrix
imports Matrix DL-Missing-Sublist
begin

## 39 Submatrix

definition submatrix :: 'a mat $\Rightarrow$ nat set $\Rightarrow$ nat set $\Rightarrow$ 'a mat where submatrix $A I J=\operatorname{mat}($ card $\{i$. $i<$ dim-row $A \wedge i \in I\})($ card $\{j . j<\operatorname{dim}$-col $A \wedge$ $j \in J\})(\lambda(i, j) . A \$ \$($ pick $I$ i, pick $J j))$
lemma dim-submatrix: dim-row (submatrix $A I J)=\operatorname{card}\{i . i<\operatorname{dim}$-row $A \wedge i \in I\}$ dim-col (submatrix $A I J)=\operatorname{card}\{j . j<\operatorname{dim}-\operatorname{col} A \wedge j \in J\}$
unfolding submatrix-def by simp-all
lemma submatrix-index:
assumes $i<$ card $\{i . i<$ dim-row $A \wedge i \in I\}$
assumes $j<$ card $\{j . j<\operatorname{dim}$-col $A \wedge j \in J\}$
shows submatrix A I J \$\$ $(i, j)=A \$ \$$ (pick I i, pick $J j)$
unfolding submatrix-def by (simp add: assms(1) assms(2))
lemma set-le-in:\{a. $a<n \wedge a \in I\}=\{a \in I . a<n\}$ by meson
lemma submatrix-index-card:
assumes $i<$ dim-row $A j<d i m$-col $A \quad i \in I j \in J$
shows submatrix A I J $\$ \$($ card $\{a \in I . a<i\}$, card $\{a \in J . a<j\})=A \$ \$(i, j)$
proof -
have $i=$ pick $I(\operatorname{card}\{a \in I . a<i\})$
$j=$ pick $J($ card $\{a \in J . a<j\})$ using pick-card-in-set assms by auto
have $\{a \in I . a<i\} \subset\{i . i<$ dim-row $A \wedge i \in I\}$
$\{a \in J . a<j\} \subset\{j . j<\operatorname{dim}-\operatorname{col} A \wedge j \in J\}$
unfolding set-le-in using $\langle i<$ dim-row $A\rangle\langle j<$ dim-col $A\rangle$ Collect-mono less-imp-le less-le-trans $\langle i \in I\rangle\langle j \in J\rangle$ by auto
then have card $\{a \in I . a<i\}<\operatorname{card}\{i . i<\operatorname{dim}$-row $A \wedge i \in I\}$
card $\{a \in J . a<j\}<$ card $\{j . j<$ dim-col $A \wedge j \in J\}$ by (simp-all add:
psubset-card-mono)
then show ?thesis
using $\langle i=$ pick $I(\operatorname{card}\{a \in I . a<i\})\rangle\langle j=\operatorname{pick} J(\operatorname{card}\{a \in J . a<j\})\rangle$
submatrix-index by fastforce
qed
lemma submatrix-split: submatrix A $I J=$ submatrix (submatrix A UNIV J) I UNIV
proof (rule eq-matI)
show dim-row (submatrix A I J) $=$ dim-row (submatrix (submatrix A UNIV J) I UNIV) by (simp add: dim-submatrix(1))
show dim-col (submatrix A I J) = dim-col (submatrix (submatrix A UNIV J) I UNIV) by (simp add: dim-submatrix(2))
fix $i j$ assume $i j$-le:i < dim-row (submatrix (submatrix A UNIV J) I UNIV) $j$ $<$ dim-col (submatrix (submatrix A UNIV J) I UNIV)
then have $i j$-le1: $i<$ card $\{i . i<\operatorname{dim}$-row $A \wedge i \in I\} j<\operatorname{card}\{i . i<\operatorname{dim}$-col $A$ $\wedge i \in J\}$ by (simp-all add: dim-submatrix)
then have $i j$-le2: $i<$ card $\{i . i<$ dim-row (submatrix $A$ UNIV $J) \wedge i \in I\} j<$ card $\{i . i<$ dim-col (submatrix A UNIV J) $\wedge i \in U N I V\}$
by (simp-all add: dim-submatrix)
then have $i$-le3:pick $I i<$ card $\{i . i<\operatorname{dim}$-row $A \wedge i \in U N I V\}$
using ij-le1(1) pick-le by auto
have $j$-le3: pick UNIV $j<\operatorname{card}\{i . i<\operatorname{dim}-c o l ~ A \wedge i \in J\}$ unfolding pick-UNIV by (simp add: ij-le1 (2))
then show submatrix A I J \$\$ $(i, j)=$ submatrix (submatrix A UNIV J) I UNIV $\$ \$(i, j)$
unfolding submatrix-index[OF ij-le1] submatrix-index[OF ij-le2] submatrix-index[OF i-le3 j-le3]
unfolding pick-UNIV by metis
qed
end

## 40 Rank and Submatrices

theory DL-Rank-Submatrix
imports DL-Rank DL-Submatrix Matrix
begin
lemma row-submatrix-UNIV:
assumes $i<\operatorname{card}\{i . i<$ dim-row $A \wedge i \in I\}$
shows row (submatrix A I UNIV) $i=$ row $A$ (pick I $i$ )
proof (rule eq-vecI)
show dim-eq:dim-vec (row (submatrix A I UNIV) i) $=$ dim-vec (row A (pick I i))
unfolding carrier-vecD[OF row-carrier $]$ dim-submatrix by auto
fix $j$ assume $j<$ dim-vec (row $A($ pick $I i)$ )
then have $j<$ dim-col (submatrix A I UNIV) $j<\operatorname{dim}$-col $A j<\operatorname{card}\{j . j<$ $\operatorname{dim}-c o l ~ A \wedge j \in U N I V\}$ using dim-eq by auto
show row (submatrix A I UNIV) $i \$ j=$ row $A($ pick $I i) \$ j$
unfolding row-def index-vec[OF $\langle j<$ dim-col (submatrix A I UNIV) $>]$ in-dex-vec $[O F<j<$ dim-col $A\rangle]$
using submatrix－index［OF assms $\langle j<\operatorname{card}\{j . j<\operatorname{dim}-c o l ~ A \wedge j \in U N I V\}>]$ using pick－UNIV by auto
qed
lemma distinct－cols－submatrix－UNIV：
assumes distinct（cols（submatrix A I UNIV））
shows distinct（cols A）
using assms proof（rule contrapos－pp）
assume $\neg$ distinct（cols $A$ ）
then obtain $i j$ where $i<\operatorname{dim}-\operatorname{col} A j<\operatorname{dim}-\operatorname{col} A(\operatorname{cols} A)!i=(\operatorname{cols} A)!j i \neq j$ using distinct－conv－nth cols－length by metis
have $i<$ dim－col（submatrix A I UNIV）$j<$ dim－col（submatrix A I UNIV）
unfolding dim－submatrix using $\langle i<$ dim－col $A\rangle\langle j<$ dim－col $A\rangle$ by simp－all
then have $i<$ length（cols（submatrix A I UNIV））$j<$ length（cols（submatrix A I UNIV））
unfolding cols－length by simp－all
have（cols（submatrix A I UNIV））！$i=($ cols $($ submatrix A I UNIV $))!j$
proof（rule eq－vecI）
show dim－vec（cols（submatrix A I UNIV）！i）＝dim－vec（cols（submatrix A I UNIV）！j）
by（simp add：$\langle i<\operatorname{dim}$－col（submatrix A I UNIV）$\langle j<$ dim－col（submatrix A I UNIV）＞）
fix $k$ assume $k<$ dim－vec（cols（submatrix A I UNIV）！j）
then have $k<$ dim－row（submatrix A I UNIV）
using $\langle j<$ length（cols（submatrix A I UNIV））〉 by auto
then have $k<\operatorname{card}\{j . j<$ dim－row $A \wedge j \in I\}$ using dim－submatrix（1）by metis
have $i$－transfer：cols（submatrix A I UNIV）！$i \$ k=($ cols A）！$i \$($ pick $I k)$
unfolding cols－nth［OF $\langle i<$ dim－col（submatrix A I UNIV）$\rangle$ ］col－def in－ dex－vec $[O F\langle k<$ dim－row（submatrix A I UNIV）$\rangle]$
unfolding submatrix－index $[O F\langle k<\operatorname{card}\{j . j<\operatorname{dim}$－row $A \wedge j \in I\}\rangle\langle i<$ dim－col（submatrix A I UNIV）＞［unfolded dim－submatrix］］
unfolding pick－UNIV cols－nth［OF $\langle i<d i m-c o l ~ A\rangle]$ col－def index－vec $[O F$ pick－le［OF $\langle k<\operatorname{card}\{j . j<$ dim－row $A \wedge j \in I\}\rangle]]$
by metis
have $j$－transfer：cols（submatrix A I UNIV）！$j \$ k=($ cols A）！$j \$($ pick I k）
unfolding cols－nth［OF $\langle j<$ dim－col（submatrix A I UNIV）$\rangle$ ］col－def in－ dex－vec $[O F\langle k<$ dim－row（submatrix A I UNIV）$\rangle]$
unfolding submatrix－index $[O F\langle k<$ card $\{j . j<$ dim－row $A \wedge j \in I\}\rangle\langle j<$ dim－col（submatrix A I UNIV）$>[$ unfolded dim－submatrix］］
unfolding pick－UNIV cols－nth $[O F\langle j<$ dim－col $A\rangle]$ col－def index－vec $[O F$ pick－le［OF $\langle k<\operatorname{card}\{j . j<$ dim－row $A \wedge j \in I\}\rangle]]$
by metis
show cols（submatrix A I UNIV）！$i \$ k=$ cols（submatrix A I UNIV）$!j \$ k$
using $\langle\operatorname{cols} A!i=$ cols $A!j\rangle i$－transfer $j$－transfer by auto
qed
then show $\neg$ distinct（cols（submatrix A I UNIV））unfolding distinct－conv－nth
using $\langle i<$ length（cols（submatrix A I UNIV））〉〈j＜length（cols（submatrix A I UNIV））$\langle\langle i \neq j\rangle$ by blast

## qed

lemma cols-submatrix-subset: set $($ cols $($ submatrix A UNIV J) $) \subseteq \operatorname{set}(\operatorname{cols} A)$ proof
fix $c$ assume $c \in \operatorname{set}($ cols (submatrix A UNIV J))
then obtain $j$ where $j<$ length (cols (submatrix A UNIV J)) cols (submatrix A UNIV J) ! $j=c$
by (meson in-set-conv-nth)
then have $j<$ dim-col (submatrix A UNIV J) by simp
then have $j<\operatorname{card}\{j . j<\operatorname{dim}-c o l ~ A \wedge j \in J\}$ by (simp add: dim-submatrix(2))
have cols (submatrix A UNIV J)! $j=$ cols $A!($ pick $J j)$
unfolding cols-nth[OF $\langle j<$ dim-col (submatrix A UNIV J)〉] cols-nth[OF pick-le[OF $\langle j<\operatorname{card}\{j . j<\operatorname{dim-col} A \wedge j \in J\}\rangle]]$
proof (rule eq-vecI)
show dim-vec $(\operatorname{col}($ submatrix A UNIV J) $j)=\operatorname{dim-vec}(\operatorname{col} A(p i c k J j))$
unfolding dim-col dim-submatrix by auto
fix $i$ assume $i<\operatorname{dim}$-vec $(\operatorname{col} A($ pick $J j))$
then have $i<d i m$-row $A$ by simp
then have $i<$ dim-row (submatrix A UNIV J) using 〈dim-vec (col (submatrix
A UNIV $J) j$ ) $=$ dim-vec $(\operatorname{col} A($ pick $J j))>$ by auto
show col (submatrix A UNIV J) $j \$ i=\operatorname{col} A($ pick $J j) \$ i$
unfolding col-def index-vec $[O F\langle i<d i m-r o w(s u b m a t r i x ~ A ~ U N I V ~ J)\rangle] ~ i n-~$ dex-vec[OF $\langle i<\operatorname{dim}$-row $A\rangle]$
using submatrix-index by (metis (no-types, lifting) «dim-vec (col (submatrix A UNIV J) j) $=$ dim-vec $($ col $A($ pick $J j))$ )
$\langle i<\operatorname{dim}-v e c($ col $A($ pick $J j))\rangle\langle j<d i m-c o l($ submatrix A UNIV J) $)$ dim-col dim-submatrix (1) dim-submatrix(2) pick-UNIV)
qed
then show $c \in \operatorname{set}($ cols $A)$
using <cols (submatrix A UNIV J)! $j=c\rangle$
using pick-le $[O F<j<\operatorname{card}\{j . j<\operatorname{dim}-c o l A \wedge j \in J\}\rangle]$ by (metis cols-length nth-mem)
qed
lemma (in vec-space) lin-dep-submatrix-UNIV:
assumes $A \in$ carrier-mat $n n c$
assumes lin-dep (set (cols A))
assumes distinct (cols (submatrix A I UNIV))
shows LinearCombinations.module.lin-dep class-ring (module-vec TYPE('a) (card $\{i . i<n \wedge i \in I\}))($ set $($ cols (submatrix A I UNIV) $))$
(is LinearCombinations.module.lin-dep class-ring ?M (set ? $S^{\prime}$ ))
proof -
obtain $v$ where 2:v $\in$ carrier-vec $n c$ and $3: v \neq 0_{v} n c$ and $A *_{v} v=0_{v} n$
using vec-space.lin-depE[OF assms(1) assms(2) distinct-cols-submatrix-UNIV[OF assms(3)]] by auto
have 1: submatrix A I UNIV $\in$ carrier-mat (card $\{i . i<n \wedge i \in I\}) n c$
apply (rule carrier-matI) unfolding dim-submatrix using $\langle A \in$ carrier-mat $n$ $n c>$ by auto
have 4:submatrix A I UNIV $*_{v} v=0_{v}(\operatorname{card}\{i . i<n \wedge i \in I\})$
proof (rule eq-vecI)
show dim-eq:dim-vec (submatrix A I UNIV $*_{v} v$ ) $=$ dim-vec ( $0_{v}$ (card $\{i . i<$ $n \wedge i \in I\})$ ) using 1 by auto
fix $i$ assume $i<\operatorname{dim-vec}\left(O_{v}(\operatorname{card}\{i . i<n \wedge i \in I\})\right)$
then have $i$-le: $i<$ card $\{i . i<n \wedge i \in I\}$ by auto
have (submatrix A I UNIV $\left.*_{v} v\right) \$ i=$ row (submatrix A I UNIV) $i \cdot v$ using dim-eq i-le by auto
also have $\ldots=$ row $A($ pick $I i) \cdot v$ using row-submatrix-UNIV
by (metis (no-types, lifting) dim-eq dim-mult-mat-vec dim-submatrix(1) $1 i<$ dim-vec $\left.\left.\left(0_{v}(\operatorname{card}\{i . i<n \wedge i \in I\})\right)\right\rangle\right)$
also have $\ldots=0$
using $\left\langle A *_{v} v=O_{v} n\right\rangle i-l e[$ THEN pick-le] by (metis assms(1) index-mult-mat-vec carrier-matD (1) index-zero-vec (1))
also have $\ldots=O_{v}(\operatorname{card}\{i . i<n \wedge i \in I\}) \$ i$ by (simp add: $i$-le $)$
finally show (submatrix A IUNIV $\left.*_{v} v\right) \$ i=O_{v}(\operatorname{card}\{i . i<n \wedge i \in I\})$
$\$ i$ by metis
qed
show ?thesis using vec-space.lin-depI[OF $\left.1 \begin{array}{llll}2 & 3 & 4\end{array}\right]$ using assms(3) by auto qed
lemma (in vec-space) rank-gt-minor:
assumes $A \in$ carrier-mat $n$ nc
assumes $\operatorname{det}($ submatrix $A I J) \neq 0$
shows card $\{j . j<n c \wedge j \in J\} \leq \operatorname{rank} A$
proof -
have square:dim-row (submatrix A I J) $=$ dim-col (submatrix A I J)
using det-def $\langle\operatorname{det}$ (submatrix A I J) $\neq 0\rangle$ by metis
then have full-rank:vec-space.rank (dim-row (submatrix A I J)) (submatrix A I $J)=$ dim-row (submatrix A I J)
using vec-space.low-rank-det-zero assms(2) carrier-matI by auto
then have distinct:distinct (cols (submatrix A I J))
using vec-space.non-distinct-low-rank square less-irrefl carrier-matI by metis
then have indpt:LinearCombinations.module.lin-indpt class-ring (module-vec TYPE ('a)
(dim-row (submatrix A I J)) ) (set (cols (submatrix A I J)) )
using vec-space.full-rank-lin-indpt[OF - full-rank distinct $]$ square by fastforce
have distinct2: distinct (cols (submatrix (submatrix A UNIV J) I UNIV)) using submatrix-split distinct by metis
have indpt2:LinearCombinations.module.lin-indpt class-ring (module-vec TYPE ('a)
(card $\{i . i<n \wedge i \in I\}))($ set (cols (submatrix (submatrix A UNIV J) I UNIV)) )
using submatrix-split dim-submatrix(1) indpt by (metis (full-types) assms(1)
carrier-matD(1))
have submatrix A UNIV $J \in$ carrier-mat $n$ (dim-col (submatrix A UNIV J))
apply (rule carrier-matI) unfolding dim-submatrix(1) using $\langle A \in$ carrier-mat $n$ nc> carrier-matD by simp-all
have lin-indpt (set (cols (submatrix A UNIV J)))
using indpt2 vec-space.lin-dep-submatrix-UNIV[OF «submatrix A UNIV J $\in$ carrier-mat $n$ (dim-col (submatrix A UNIV J)) >-distinct2] by blast

```
    have distinct3:distinct (cols (submatrix A UNIV J)) by (metis distinct dis-
tinct-cols-submatrix-UNIV submatrix-split)
    show ?thesis using
    rank-ge-card-indpt[OF <A \in carrier-mat n nc> cols-submatrix-subset <lin-indpt
(set (cols (submatrix A UNIV J)))>,
    unfolded distinct-card[OF distinct3, unfolded cols-length dim-submatrix], un-
folded carrier-matD(2)[OF<A \in carrier-mat n nc`]]
    by blast
qed
end
```


## References

[1] M. Avanzini, C. Sternagel, and R. Thiemann. Certification of complexity proofs using CeTA. In Proc. RTA 2015, LIPIcs 36, pages 23-39, 2015.
[2] J. Divasón and J. Aransay. Gauss-jordan algorithm and its applications. Archive of Formal Proofs, Sept. 2014. http://isa-afp.org/entries/Gauss_ Jordan.shtml, Formal proof development.
[3] J. Endrullis, J. Waldmann, and H. Zantema. Matrix Interpretations for Proving Termination of Term Rewriting. Journal of Automated Reasoning, 40(2-3):195-220, 2008.
[4] A. Lochbihler. Light-weight containers. Archive of Formal Proofs, Apr. 2013. http://isa-afp.org/entries/Containers.shtml, Formal proof development.
[5] R. Piziak and P. L. Odell. Matrix theory: from generalized inverses to Jordan form. CRC Press, 2007.
[6] C. Sternagel and R. Thiemann. Executable matrix operations on matrices of arbitrary dimensions. Archive of Formal Proofs, June 2010. http://isa-afp.org/entries/Matrix.shtml, Formal proof development.
[7] R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In Proc. TPHOLs'09, LNCS 5674, pages 452-468, 2009.


[^0]:    *Supported by FWF (Austrian Science Fund) project Y757.

