

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

René Thiemann and Akihisa Yamada

December 14, 2021

## 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 be 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 HMA-representation 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

|          |                                             |          |
|----------|---------------------------------------------|----------|
| <b>1</b> | <b>Introduction</b>                         | <b>4</b> |
| <b>2</b> | <b>Material missing in the distribution</b> | <b>5</b> |
| <b>3</b> | <b>Missing Ring</b>                         | <b>8</b> |
| 3.1      | Instantiations . . . . .                    | 13       |

---

\*Supported by FWF (Austrian Science Fund) project Y757.

|           |                                                     |            |
|-----------|-----------------------------------------------------|------------|
| <b>4</b>  | <b>Vectors and Matrices</b>                         | <b>14</b>  |
| 4.1       | Vectors . . . . .                                   | 14         |
| 4.2       | Matrices . . . . .                                  | 22         |
| 4.3       | Update Operators . . . . .                          | 39         |
| 4.4       | Block Vectors and Matrices . . . . .                | 40         |
| 4.5       | Homomorphism properties . . . . .                   | 49         |
| <b>5</b>  | <b>Code Generation for Basic Matrix Operations</b>  | <b>58</b>  |
| <b>6</b>  | <b>Gauss-Jordan Algorithm</b>                       | <b>62</b>  |
| 6.1       | Row Operations . . . . .                            | 62         |
| 6.2       | Gauss-Jordan Elimination . . . . .                  | 65         |
| <b>7</b>  | <b>Code Generation for Basic Matrix Operations</b>  | <b>73</b>  |
| <b>8</b>  | <b>Elementary Column Operations</b>                 | <b>74</b>  |
| <b>9</b>  | <b>Determinants</b>                                 | <b>78</b>  |
| <b>10</b> | <b>Code Equations for Determinants</b>              | <b>91</b>  |
| 10.1      | Properties of triangular matrices . . . . .         | 92         |
| 10.2      | Algorithms for Triangulization . . . . .            | 93         |
| 10.3      | Finding Non-Zero Elements . . . . .                 | 95         |
| 10.4      | Determinant Preserving Growth of Triangle . . . . . | 96         |
| 10.5      | Recursive Triangulization of Columns . . . . .      | 96         |
| 10.6      | Triangulization . . . . .                           | 97         |
| 10.7      | Divisor will not be 0 . . . . .                     | 97         |
| 10.8      | Determinant Preservation Results . . . . .          | 98         |
| 10.9      | Determinant Computation . . . . .                   | 99         |
| <b>11</b> | <b>Converting Matrices to Strings</b>               | <b>101</b> |
| <b>12</b> | <b>Characteristic Polynomial</b>                    | <b>102</b> |
| <b>13</b> | <b>Jordan Normal Form</b>                           | <b>106</b> |
| 13.1      | Application for Complexity . . . . .                | 111        |
| <b>14</b> | <b>Missing Vector Spaces</b>                        | <b>111</b> |
| <b>15</b> | <b>Matrices as Vector Spaces</b>                    | <b>121</b> |
| <b>16</b> | <b>Gram-Schmidt Orthogonalization</b>               | <b>132</b> |
| 16.1      | Orthogonality with Conjugates . . . . .             | 132        |
| 16.2      | The Algorithm . . . . .                             | 133        |
| 16.3      | Properties of the Algorithms . . . . .              | 133        |

|                                                                                                               |            |
|---------------------------------------------------------------------------------------------------------------|------------|
| <b>17 Schur Decomposition</b>                                                                                 | <b>137</b> |
| <b>18 Computing Jordan Normal Forms</b>                                                                       | <b>140</b> |
| 18.1 Pseudo Code Algorithm . . . . .                                                                          | 141        |
| 18.2 Real Algorithm . . . . .                                                                                 | 142        |
| 18.3 Preservation of Dimensions . . . . .                                                                     | 145        |
| 18.4 Properties of Auxiliary Algorithms . . . . .                                                             | 146        |
| 18.5 Proving Similarity . . . . .                                                                             | 148        |
| 18.6 Invariants for Proving that Result is in JNF . . . . .                                                   | 148        |
| 18.7 Alternative Characterization of <i>identify-blocks</i> in Presence of<br><i>local.ev-block</i> . . . . . | 153        |
| 18.8 Proving the Invariants . . . . .                                                                         | 153        |
| 18.9 Combination with Schur-decomposition . . . . .                                                           | 157        |
| 18.10 Application for Complexity . . . . .                                                                    | 157        |
| <b>19 Code Equations for All Algorithms</b>                                                                   | <b>158</b> |
| <b>20 Strassen’s algorithm for matrix multiplication.</b>                                                     | <b>159</b> |
| <b>21 Strassen’s Algorithm as Code Equation</b>                                                               | <b>162</b> |
| <b>22 Comparison of Matrices</b>                                                                              | <b>162</b> |
| <b>23 Matrix Conversions</b>                                                                                  | <b>173</b> |
| <b>24 Derivation Bounds</b>                                                                                   | <b>174</b> |
| <b>25 Complexity Carrier</b>                                                                                  | <b>175</b> |
| <b>26 Converting Arctic Numbers to Strings</b>                                                                | <b>176</b> |
| <b>27 Application: Complexity of Matrix Orderings</b>                                                         | <b>178</b> |
| 27.1 Locales for Carriers of Matrix Interpretations and Polynomial<br>Orders . . . . .                        | 178        |
| 27.2 The Integers as Carrier . . . . .                                                                        | 179        |
| 27.3 The Rational and Real Numbers as Carrier . . . . .                                                       | 179        |
| 27.4 The Arctic Numbers as Carrier . . . . .                                                                  | 179        |
| <b>28 Matrix Kernel</b>                                                                                       | <b>179</b> |
| <b>29 Jordan Normal Form – Uniqueness</b>                                                                     | <b>186</b> |
| <b>30 Spectral Radius Theory</b>                                                                              | <b>187</b> |
| <b>31 Missing Lemmas of List</b>                                                                              | <b>189</b> |

|                                     |            |
|-------------------------------------|------------|
| <b>32 Matrix Rank</b>               | <b>190</b> |
| <b>33 Subadditivity of rank</b>     | <b>193</b> |
| <b>34 Missing Lemmas of Sublist</b> | <b>194</b> |
| <b>35 Pick</b>                      | <b>195</b> |
| <b>36 Sublist</b>                   | <b>196</b> |
| <b>37 weave</b>                     | <b>197</b> |
| <b>38 Submatrices</b>               | <b>198</b> |
| <b>39 Submatrix</b>                 | <b>198</b> |
| <b>40 Rank and Submatrices</b>      | <b>199</b> |

## 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  $\mathcal{O}(|c|^n \cdot n^{k-1})$  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, characteristic-function). 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*:

$\langle \text{finite } (f \text{ ' } A) \longleftrightarrow \text{finite } A \rangle$  **if**  $\langle \text{inj-on } f \ A \rangle$   
 $\langle \text{proof} \rangle$

The following lemma is slightly generalized from Determinants.thy in HMA.

**lemma** *finite-bounded-functions*:

**assumes**  $fS$ : *finite*  $S$   
**shows**  $\text{finite } T \implies \text{finite } \{f. (\forall i \in T. f \ i \in S) \wedge (\forall i. i \notin T \longrightarrow f \ i = i)\}$   
 $\langle \text{proof} \rangle$

**lemma** *finite-bounded-functions'*:

**assumes**  $fS$ : *finite*  $S$   
**shows**  $\text{finite } T \implies \text{finite } \{f. (\forall i \in T. f \ i \in S) \wedge (\forall i. i \notin T \longrightarrow f \ i = j)\}$   
 $\langle \text{proof} \rangle$

**lemma** *permutes-less* [*simp*]:

**assumes**  $p$ :  $p$  *permutes*  $\{0..<(n :: \text{nat})\}$   
**shows**  
 $i < n \implies p \ i < n$   
 $i < n \implies \text{inv } p \ i < n$   
 $p \ (\text{inv } p \ i) = i$   
 $\text{inv } p \ (p \ i) = i$   
 $\langle \text{proof} \rangle$

**lemma** *permutes-prod*:

**assumes**  $p$ :  $p$  *permutes*  $S$   
**shows**  $(\prod s \in S. f \ (p \ s) \ s) = (\prod s \in S. f \ s \ (\text{inv } p \ s))$   
 $(\text{is } ?l = ?r)$   
 $\langle \text{proof} \rangle$

**lemma** *permutes-sum*:

**assumes**  $p$ :  $p$  *permutes*  $S$   
**shows**  $(\sum s \in S. f \ (p \ s) \ s) = (\sum s \in S. f \ s \ (\text{inv } p \ s))$   
 $(\text{is } ?l = ?r)$   
 $\langle \text{proof} \rangle$

**context**

**fixes**  $A :: 'a \ \text{set}$   
**and**  $B :: 'b \ \text{set}$   
**and**  $a\text{-to-}b :: 'a \implies 'b$   
**and**  $b\text{-to-}a :: 'b \implies 'a$   
**assumes**  $ab$ :  $\bigwedge a. a \in A \implies a\text{-to-}b \ a \in B$   
**and**  $ba$ :  $\bigwedge b. b \in B \implies b\text{-to-}a \ b \in A$   
**and**  $ab\text{-}ba$ :  $\bigwedge a. a \in A \implies b\text{-to-}a \ (a\text{-to-}b \ a) = a$

**and**  $ba-ab: \bigwedge b. b \in B \implies a-to-b (b-to-a b) = b$   
**begin**

**qualified lemma** *permutes-memb*: **fixes**  $p :: 'b \Rightarrow 'b$   
**assumes**  $p: p \text{ permutes } B$   
**and**  $a: a \in A$   
**defines**  $ip \equiv \text{Hilbert-Choice.inv } p$   
**shows**  $a \in A \ a-to-b \ a \in B \ ip \ (a-to-b \ a) \in B \ p \ (a-to-b \ a) \in B$   
 $b-to-a \ (p \ (a-to-b \ a)) \in A \ b-to-a \ (ip \ (a-to-b \ a)) \in A$   
 $\langle \text{proof} \rangle$

**lemma** *permutes-bij-main*:  
 $\{p . p \text{ permutes } A\} \supseteq (\lambda p \ a. \text{if } a \in A \text{ then } b-to-a \ (p \ (a-to-b \ a)) \text{ else } a) \text{ ' } \{p . p \text{ permutes } B\}$   
**(is**  $?A \supseteq ?f \text{ ' } ?B$ )  
 $\langle \text{proof} \rangle$

**end**

**lemma** *permutes-bij'*: **assumes**  $ab: \bigwedge a. a \in A \implies a-to-b \ a \in B$   
**and**  $ba: \bigwedge b. b \in B \implies b-to-a \ b \in A$   
**and**  $ab-ba: \bigwedge a. a \in A \implies b-to-a \ (a-to-b \ a) = a$   
**and**  $ba-ab: \bigwedge b. b \in B \implies a-to-b \ (b-to-a \ b) = b$   
**shows**  $\{p . p \text{ permutes } A\} = (\lambda p \ a. \text{if } a \in A \text{ then } b-to-a \ (p \ (a-to-b \ a)) \text{ else } a) \text{ ' } \{p . p \text{ permutes } B\}$   
**(is**  $?A = ?f \text{ ' } ?B$ )  
 $\langle \text{proof} \rangle$

**lemma** *permutes-others*:  
**assumes**  $p: p \text{ permutes } S$  **and**  $x: x \notin S$  **shows**  $p \ x = x$   
 $\langle \text{proof} \rangle$

**lemma** *inj-on-nat-permutes*: **assumes**  $i: \text{inj-on } f \ (S :: \text{nat set})$   
**and**  $fS: f \in S \rightarrow S$   
**and**  $fin: \text{finite } S$   
**and**  $f: \bigwedge i. i \notin S \implies f \ i = i$   
**shows**  $f \text{ permutes } S$   
 $\langle \text{proof} \rangle$

**abbreviation** (*input*)  $\text{signof} :: \langle \text{nat} \Rightarrow \text{nat} \rangle \Rightarrow 'a :: \text{ring-1}$   
**where**  $\langle \text{signof } p \equiv \text{of-int } (\text{sign } p) \rangle$

**lemma** *signof-id*:  
 $\text{signof } id = 1$   
 $\text{signof } (\lambda x. x) = 1$   
 $\langle \text{proof} \rangle$

**lemma** *signof-inv*:  $\text{finite } S \implies p \text{ permutes } S \implies \text{signof } (\text{inv } p) = \text{signof } p$   
 $\langle \text{proof} \rangle$

**lemma** *signof-pm-one*:  $\text{signof } p \in \{1, -1\}$   
 ⟨*proof*⟩

**lemma** *signof-compose*:  
**assumes**  $p$  permutes  $\{0..<(n :: \text{nat})\}$   
**and**  $q$  permutes  $\{0..<(m :: \text{nat})\}$   
**shows**  $\text{signof } (p \circ q) = \text{signof } p * \text{signof } q$   
 ⟨*proof*⟩

**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* ⟨*proof*⟩

**end**

partially ordered variant

**class** *ordered-semiring-strict* = *semiring* + *comm-monoid-add* + *ordered-cancel-ab-semigroup-add*  
 +

**assumes** *mult-strict-left-mono*:  $a < b \implies 0 < c \implies c * a < c * b$

**assumes** *mult-strict-right-mono*:  $a < b \implies 0 < c \implies a * c < b * c$

**begin**

**subclass** *semiring-0-cancel* ⟨*proof*⟩

**subclass** *ordered-semiring*

⟨*proof*⟩

**lemma** *mult-pos-pos[simp]*:  $0 < a \implies 0 < b \implies 0 < a * b$

⟨*proof*⟩

**lemma** *mult-pos-neg*:  $0 < a \implies b < 0 \implies a * b < 0$

⟨*proof*⟩

**lemma** *mult-neg-pos*:  $a < 0 \implies 0 < b \implies a * b < 0$



*<proof>*

Legacy - use *mult-neg-pos*

**lemma** *mult-pos-neg2*:  $0 < a \implies b < 0 \implies b * a < 0$

*<proof>*

Strict monotonicity in both arguments

**lemma** *mult-strict-mono*:

**assumes**  $a < b$  **and**  $c < d$  **and**  $0 < b$  **and**  $0 \leq c$

**shows**  $a * c < b * d$

*<proof>*

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**  $a * c < b * d$

*<proof>*

**lemma** *mult-less-le-imp-less*:

**assumes**  $a < b$  **and**  $c \leq d$  **and**  $0 \leq a$  **and**  $0 < c$

**shows**  $a * c < b * d$

*<proof>*

**lemma** *mult-le-less-imp-less*:

**assumes**  $a \leq b$  **and**  $c < d$  **and**  $0 < a$  **and**  $0 \leq c$

**shows**  $a * c < b * d$

*<proof>*

**end**

**class** *ordered-idom* = *idom* + *ordered-semiring-strict* +  
**assumes** *zero-less-one* [*simp*]:  $0 < 1$  **begin**

**subclass** *semiring-1* *<proof>*

**subclass** *comm-ring-1* *<proof>*

**subclass** *ordered-ring* *<proof>*

**subclass** *ordered-comm-semiring* *<proof>*

**subclass** *ordered-ab-semigroup-add* *<proof>*

**lemma** *of-nat-ge-0*[*simp*]:  $of\text{-}nat\ x \geq 0$

*<proof>*

**lemma** *of-nat-eq-0*[*simp*]:  $of\text{-}nat\ x = 0 \iff x = 0$

*<proof>*

**lemma** *inj-of-nat*:  $inj\ (of\text{-}nat\ ::\ nat \Rightarrow 'a)$

*<proof>*

**subclass** *ring-char-0* *<proof>*

**end**

**context** *comm-monoid*  
**begin**

**lemma** *finprod-reindex-bij-betw*: *bij-betw h S T*  
 $\implies g \in h \text{ ' } S \rightarrow \text{carrier } G$   
 $\implies \text{finprod } G (\lambda x. g (h x)) S = \text{finprod } G g T$   
{*proof*}

**lemma** *finprod-reindex-bij-witness*:

**assumes** *witness*:

$\bigwedge a. a \in S \implies i (j a) = a$

$\bigwedge a. a \in S \implies j a \in T$

$\bigwedge b. b \in T \implies j (i b) = b$

$\bigwedge b. b \in T \implies i b \in S$

**assumes** *eq*:

$\bigwedge a. a \in S \implies h (j a) = g a$

**assumes** *g*:  $g \in S \rightarrow \text{carrier } G$

**and** *h*:  $h \in j \text{ ' } S \rightarrow \text{carrier } G$

**shows**  $\text{finprod } G g S = \text{finprod } G h T$

{*proof*}

**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* {*proof*}

**end**

**lemma** (**in** *comm-monoid*) *finprod-one'*:

$(\bigwedge a. a \in A \implies f a = \mathbf{1}) \implies \text{finprod } G f A = \mathbf{1}$

{*proof*}

**lemma** (**in** *comm-monoid*) *finprod-split*:

$\text{finite } A \implies f \text{ ' } A \subseteq \text{carrier } G \implies a \in A \implies \text{finprod } G f A = f a \otimes \text{finprod } G f$   
 $(A - \{a\})$

{*proof*}

**lemma** (**in** *comm-monoid*) *finprod-finprod*:

$\text{finite } A \implies \text{finite } B \implies (\bigwedge a b. a \in A \implies b \in B \implies g a b \in \text{carrier } G) \implies$

$\text{finprod } G (\lambda a. \text{finprod } G (g a) B) A = \text{finprod } G (\lambda (a,b). g a b) (A \times B)$

{*proof*}

**lemma** (in *comm-monoid*) *finprod-swap*:  
**assumes** *finite A finite B*  $\wedge$  *a b. a ∈ A ⇒ b ∈ B ⇒ g a b ∈ carrier G*  
**shows** *finprod G (λ (b,a). g a b) (B × A) = finprod G (λ (a,b). g a b) (A × B)*  
⟨*proof*⟩

**lemma** (in *comm-monoid*) *finprod-finprod-swap*:  
*finite A ⇒ finite B ⇒ (λ a b. a ∈ A ⇒ b ∈ B ⇒ g a b ∈ carrier G) ⇒*  
*finprod G (λ a. finprod G (g a) B) A = finprod G (λ b. finprod G (λ a. g a b)*  
*A) B*  
⟨*proof*⟩

**lemmas** (in *semiring*) *finsum-zero' = add.finprod-one'*  
**lemmas** (in *semiring*) *finsum-split = add.finprod-split*  
**lemmas** (in *semiring*) *finsum-finsum-swap = add.finprod-finprod-swap*

**lemma** (in *csemiring*) *finprod-zero*:  
*finite A ⇒ f ∈ A → carrier R ⇒ ∃ a ∈ A. f a = 0*  
 $\Rightarrow$  *finprod R f A = 0*  
⟨*proof*⟩

**lemma** (in *semiring*) *finsum-product*:  
**assumes** *A: finite A and B: finite B*  
**and** *f: f ∈ A → carrier R and g: g ∈ B → carrier R*  
**shows** *finsum R f A ⊗ finsum R g B = (⊕ i ∈ A. ⊕ j ∈ B. f i ⊗ g j)*  
⟨*proof*⟩

**lemma** (in *semiring*) *Units-one-side-I*:  
*a ∈ carrier R ⇒ p ∈ Units R ⇒ p ⊗ a = 1 ⇒ a ∈ Units R*  
*a ∈ carrier R ⇒ p ∈ Units R ⇒ a ⊗ p = 1 ⇒ a ∈ Units R*  
⟨*proof*⟩

**lemma** *permutes-funcset*: *p permutes A ⇒ (p ` A → B) = (A → B)*  
⟨*proof*⟩

**context** *comm-monoid*

**begin**

**lemma** *finprod-permute*:  
**assumes** *p: p permutes S*  
**and** *f: f ∈ S → carrier G*  
**shows** *finprod G f S = finprod G (f ∘ p) S*  
⟨*proof*⟩

**lemma** *finprod-singleton-set[simp]*: **assumes** *f a ∈ carrier G*  
**shows** *finprod G f {a} = f a*

```

⟨proof⟩
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 ∈ {p. p permutes S} → carrier R
  shows finsum R f {p. p permutes S} = finsum R (λp. f(Hilbert-Choice.inv p))
  {p. p permutes S}
  (is ?lhs = ?rhs)
⟨proof⟩

lemma finsum-permutations-compose-right: assumes q: q permutes S
  and *: f ∈ {p. p permutes S} → carrier R
  shows finsum R f {p. p permutes S} = finsum R (λp. f(p ∘ q)) {p. p permutes
  S}
  (is ?lhs = ?rhs)
⟨proof⟩

end

end

theory Conjugate
  imports HOL.Complex HOL-Library.Complex-Order
begin

class conjugate =
  fixes conjugate :: 'a ⇒ 'a
  assumes conjugate-id[simp]: conjugate (conjugate a) = a
  and conjugate-cancel-iff[simp]: conjugate a = conjugate b ⟷ a = b

class conjugatable-ring = ring + conjugate +
  assumes conjugate-dist-mul: conjugate (a * b) = conjugate a * conjugate b
  and conjugate-dist-add: conjugate (a + b) = conjugate a + 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 ⟷ a = 0
  ⟨proof⟩
end

class conjugatable-field = conjugatable-ring + field

lemma sum-conjugate:

```

```

fixes f :: 'b ⇒ 'a :: conjugatable-ring
assumes finX: finite X
shows conjugate (sum f X) = sum (λx. conjugate (f x)) X
⟨proof⟩

class conjugatable-ordered-ring = conjugatable-ring + ordered-comm-monoid-add
+
assumes conjugate-square-positive: a * conjugate a ≥ 0

class conjugatable-ordered-field = conjugatable-ordered-ring + field
begin
  subclass conjugatable-field⟨proof⟩
end

lemma conjugate-square-0:
  fixes a :: 'a :: {conjugatable-ordered-ring, semiring-no-zero-divisors}
  shows a * conjugate a = 0 ⇒ a = 0 ⟨proof⟩

```

### 3.1 Instantiations

```

instantiation complex :: conjugatable-ordered-field
begin
  definition [simp]: conjugate ≡ cnj

instance
  ⟨proof⟩

end

instantiation real :: conjugatable-ordered-field
begin
  definition [simp]: conjugate (x::real) ≡ x
  instance ⟨proof⟩
end

instantiation rat :: conjugatable-ordered-field
begin
  definition [simp]: conjugate (x::rat) ≡ x
  instance ⟨proof⟩
end

instantiation int :: conjugatable-ordered-ring
begin
  definition [simp]: conjugate (x::int) ≡ x
  instance ⟨proof⟩
end

lemma conjugate-square-eq-0 [simp]:
  fixes x :: 'a :: {conjugatable-ring, semiring-no-zero-divisors}

```

**shows**  $x * \text{conjugate } x = 0 \longleftrightarrow x = 0 \text{ conjugate } x * x = 0 \longleftrightarrow x = 0$   
 ⟨proof⟩

**lemma** *conjugate-square-greater-0* [simp]:  
**fixes**  $x :: 'a :: \{\text{conjugatable-ordered-ring, ring-no-zero-divisors}\}$   
**shows**  $x * \text{conjugate } x > 0 \longleftrightarrow x \neq 0$   
 ⟨proof⟩

**lemma** *conjugate-square-smaller-0* [simp]:  
**fixes**  $x :: 'a :: \{\text{conjugatable-ordered-ring, ring-no-zero-divisors}\}$   
**shows**  $\neg x * \text{conjugate } x < 0$   
 ⟨proof⟩

**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* ::  $\text{nat} \Rightarrow 'a$  **where**  
*undef-vec*  $i \equiv [] ! i$

**definition** *mk-vec* ::  $\text{nat} \Rightarrow (\text{nat} \Rightarrow 'a) \Rightarrow (\text{nat} \Rightarrow 'a)$  **where**  
*mk-vec*  $n f \equiv \lambda i. \text{if } i < n \text{ then } f i \text{ else } \text{undef-vec } (i - n)$

**typedef**  $'a \text{ vec} = \{(n, \text{mk-vec } n f) \mid n f :: \text{nat} \Rightarrow 'a. \text{True}\}$

*<proof>*

**setup-lifting** *type-definition-vec*

**lift-definition** *dim-vec* :: 'a vec  $\Rightarrow$  nat **is** *fst* *<proof>*

**lift-definition** *vec-index* :: 'a vec  $\Rightarrow$  (nat  $\Rightarrow$  'a) (**infixl** \$ 100) **is** *snd* *<proof>*

**lift-definition** *vec* :: nat  $\Rightarrow$  (nat  $\Rightarrow$  'a)  $\Rightarrow$  'a vec

**is**  $\lambda$  n f. (n, *mk-vec* n f) *<proof>*

**lift-definition** *vec-of-list* :: 'a list  $\Rightarrow$  'a vec **is**

$\lambda$  v. (*length* v, *mk-vec* (*length* v) (*nth* v)) *<proof>*

**lift-definition** *list-of-vec* :: 'a vec  $\Rightarrow$  'a list **is**

$\lambda$  (n,v). *map* v [0 ..< n] *<proof>*

**definition** *carrier-vec* :: nat  $\Rightarrow$  '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) *<proof>*

**lemma** *dim-vec[simp]*: *dim-vec* (*vec* n f) = n *<proof>*

**lemma** *vec-carrier[simp]*: *vec* n f  $\in$  *carrier-vec* n *<proof>*

**lemma** *index-vec[simp]*: i < n  $\Longrightarrow$  *vec* n f \$ i = f i *<proof>*

**lemma** *eq-vecI[intro]*: ( $\bigwedge$  i. i < *dim-vec* w  $\Longrightarrow$  v \$ i = w \$ i)  $\Longrightarrow$  *dim-vec* v = *dim-vec* w

$\Longrightarrow$  v = w

*<proof>*

**lemma** *carrier-dim-vec*: v  $\in$  *carrier-vec* n  $\longleftrightarrow$  *dim-vec* v = n

*<proof>*

**lemma** *carrier-vecD[simp]*: v  $\in$  *carrier-vec* n  $\Longrightarrow$  *dim-vec* v = n *<proof>*

**lemma** *carrier-vecI*: *dim-vec* v = n  $\Longrightarrow$  v  $\in$  *carrier-vec* n *<proof>*

**instantiation** *vec* :: (*plus*) *plus*

**begin**

**definition** *plus-vec* :: 'a vec  $\Rightarrow$  'a vec  $\Rightarrow$  'a :: *plus* *vec* **where**

v<sub>1</sub> + v<sub>2</sub>  $\equiv$  *vec* (*dim-vec* v<sub>2</sub>) ( $\lambda$  i. v<sub>1</sub> \$ i + v<sub>2</sub> \$ i)

**instance** *<proof>*

**end**

**instantiation** *vec* :: (*minus*) *minus*

**begin**

**definition** *minus-vec* :: 'a vec  $\Rightarrow$  'a vec  $\Rightarrow$  'a :: *minus* *vec* **where**

v<sub>1</sub> - v<sub>2</sub>  $\equiv$  *vec* (*dim-vec* v<sub>2</sub>) ( $\lambda$  i. v<sub>1</sub> \$ i - v<sub>2</sub> \$ i)

**instance** *<proof>*

**end**

**definition**

$zero\text{-}vec :: nat \Rightarrow 'a :: zero\text{-}vec (0_v)$   
**where**  $0_v\ n \equiv vec\ n (\lambda\ i.\ 0)$

**lemma**  $zero\text{-}carrier\text{-}vec[simp]$ :  $0_v\ n \in carrier\text{-}vec\ n$   
 $\langle proof \rangle$

**lemma**  $index\text{-}zero\text{-}vec[simp]$ :  $i < n \implies 0_v\ n\ \$\ i = 0$   $dim\text{-}vec\ (0_v\ n) = n$   
 $\langle proof \rangle$

**lemma**  $vec\text{-}of\text{-}dim\text{-}0[simp]$ :  $dim\text{-}vec\ v = 0 \longleftrightarrow v = 0_v\ 0$   $\langle proof \rangle$

**definition**

$unit\text{-}vec :: nat \Rightarrow nat \Rightarrow ('a :: zero\text{-}neq\text{-}one)\ vec$   
**where**  $unit\text{-}vec\ n\ i = vec\ n (\lambda\ j.\ if\ j = i\ then\ 1\ else\ 0)$

**lemma**  $index\text{-}unit\text{-}vec[simp]$ :  
 $i < n \implies j < n \implies unit\text{-}vec\ n\ i\ \$\ j = (if\ j = i\ then\ 1\ else\ 0)$   
 $i < n \implies unit\text{-}vec\ n\ i\ \$\ i = 1$   
 $dim\text{-}vec\ (unit\text{-}vec\ n\ i) = n$   
 $\langle proof \rangle$

**lemma**  $unit\text{-}vec\text{-}eq[simp]$ :  
**assumes**  $i: i < n$   
**shows**  $(unit\text{-}vec\ n\ i = unit\text{-}vec\ n\ j) = (i = j)$   
 $\langle proof \rangle$

**lemma**  $unit\text{-}vec\text{-}nonzero[simp]$ :  
**assumes**  $i\text{-}n: i < n$  **shows**  $unit\text{-}vec\ n\ i \neq zero\text{-}vec\ n$  (**is**  $?l \neq ?r$ )  
 $\langle proof \rangle$

**lemma**  $unit\text{-}vec\text{-}carrier[simp]$ :  $unit\text{-}vec\ n\ i \in carrier\text{-}vec\ n$   
 $\langle proof \rangle$

**definition**  $unit\text{-}vecs :: nat \Rightarrow 'a :: zero\text{-}neq\text{-}one\ vec\ list$   
**where**  $unit\text{-}vecs\ n = map\ (unit\text{-}vec\ n) [0..<n]$

List of first i units

**fun**  $unit\text{-}vecs\text{-}first :: nat \Rightarrow nat \Rightarrow 'a :: zero\text{-}neq\text{-}one\ vec\ list$   
**where**  $unit\text{-}vecs\text{-}first\ n\ 0 = []$   
 $| unit\text{-}vecs\text{-}first\ n\ (Suc\ i) = unit\text{-}vecs\text{-}first\ n\ i @ [unit\text{-}vec\ n\ i]$

**lemma**  $unit\text{-}vecs\text{-}first$ :  $unit\text{-}vecs\ n = unit\text{-}vecs\text{-}first\ n\ n$   
 $\langle proof \rangle$

list of last i units

**fun**  $unit\text{-}vecs\text{-}last :: nat \Rightarrow nat \Rightarrow 'a :: zero\text{-}neq\text{-}one\ vec\ list$   
**where**  $unit\text{-}vecs\text{-}last\ n\ 0 = []$   
 $| unit\text{-}vecs\text{-}last\ n\ (Suc\ i) = unit\text{-}vec\ n\ (n - Suc\ i) \# unit\text{-}vecs\text{-}last\ n\ i$



**lemma** *unit-vecs-last-carrier*:  $set (unit-vecs-last\ n\ i) \subseteq carrier-vec\ n$   
 ⟨proof⟩

**lemma** *unit-vecs-last[code]*:  $unit-vecs\ n = unit-vecs-last\ n\ n$   
 ⟨proof⟩

**lemma** *unit-vecs-carrier*:  $set (unit-vecs\ n) \subseteq carrier-vec\ n$   
 ⟨proof⟩

**lemma** *unit-vecs-last-distinct*:  
 $j \leq n \implies i < n - j \implies unit-vec\ n\ i \notin set (unit-vecs-last\ n\ j)$   
 ⟨proof⟩

**lemma** *unit-vecs-first-distinct*:  
 $i \leq j \implies j < n \implies unit-vec\ n\ j \notin set (unit-vecs-first\ n\ i)$   
 ⟨proof⟩

**definition** *map-vec* **where**  $map-vec\ f\ v \equiv vec (dim-vec\ v) (\lambda i. f (v\ \$\ i))$

**instantiation** *vec* :: (*uminus*) *uminus*

**begin**

**definition** *uminus-vec* :: 'a :: *uminus* *vec*  $\Rightarrow$  'a *vec* **where**

–  $v \equiv vec (dim-vec\ v) (\lambda i. - (v\ \$\ i))$

**instance** ⟨proof⟩

**end**

**definition** *smult-vec* :: 'a :: *times*  $\Rightarrow$  'a *vec*  $\Rightarrow$  'a *vec* (**infixl**  $\cdot_v$  70)  
**where**  $a \cdot_v v \equiv vec (dim-vec\ v) (\lambda i. a * v\ \$\ i)$

**definition** *scalar-prod* :: 'a *vec*  $\Rightarrow$  'a *vec*  $\Rightarrow$  'a :: *semiring-0* (**infix**  $\cdot$  70)  
**where**  $v \cdot w \equiv \sum i \in \{0 ..< dim-vec\ w\}. v\ \$\ i * w\ \$\ i$

**definition** *monoid-vec* :: 'a *itself*  $\Rightarrow$  *nat*  $\Rightarrow$  ('a :: *monoid-add* *vec*) *monoid* **where**  
*monoid-vec* *ty* *n*  $\equiv$  (  
*carrier* = *carrier-vec* *n*,  
*mult* = (+),  
*one* =  $0_v\ n$ )

**definition** *module-vec* ::  
 'a :: *semiring-1* *itself*  $\Rightarrow$  *nat*  $\Rightarrow$  ('a, 'a *vec*) *module* **where**  
*module-vec* *ty* *n*  $\equiv$  (  
*carrier* = *carrier-vec* *n*,  
*mult* = *undefined*,  
*one* = *undefined*,  
*zero* =  $0_v\ n$ ,  
*add* = (+),  
*smult* = ( $\cdot_v$ )

**lemma** *monoid-vec-simps*:

$mult (monoid-vec\ ty\ n) = (+)$   
 $carrier (monoid-vec\ ty\ n) = carrier-vec\ n$   
 $one (monoid-vec\ ty\ n) = 0_v\ n$   
(proof)

**lemma** *module-vec-simps*:

$add (module-vec\ ty\ n) = (+)$   
 $zero (module-vec\ ty\ n) = 0_v\ n$   
 $carrier (module-vec\ ty\ n) = carrier-vec\ n$   
 $smult (module-vec\ ty\ n) = (\cdot_v)$   
(proof)

**definition** *finsum-vec* :: 'a :: monoid-add itself  $\Rightarrow$  nat  $\Rightarrow$  ('c  $\Rightarrow$  'a vec)  $\Rightarrow$  'c set  
 $\Rightarrow$  'a vec **where**

$finsum-vec\ ty\ n = finprod (monoid-vec\ ty\ n)$

**lemma** *index-add-vec[simp]*:

$i < dim-vec\ v_2 \Longrightarrow (v_1 + v_2) \$ i = v_1 \$ i + v_2 \$ i$   
 $dim-vec (v_1 + v_2) = dim-vec\ v_2$   
(proof)

**lemma** *index-minus-vec[simp]*:

$i < dim-vec\ v_2 \Longrightarrow (v_1 - v_2) \$ i = v_1 \$ i - v_2 \$ i$   
 $dim-vec (v_1 - v_2) = dim-vec\ v_2$   
(proof)

**lemma** *index-map-vec[simp]*:

$i < dim-vec\ v \Longrightarrow map-vec\ f\ v \$ i = f (v \$ i)$   
 $dim-vec (map-vec\ f\ v) = dim-vec\ v$   
(proof)

**lemma** *map-carrier-vec[simp]*:  $map-vec\ h\ v \in carrier-vec\ n = (v \in carrier-vec\ n)$

(proof)

**lemma** *index-uminus-vec[simp]*:

$i < dim-vec\ v \Longrightarrow (- v) \$ i = - (v \$ i)$   
 $dim-vec (- v) = dim-vec\ v$   
(proof)

**lemma** *index-smult-vec[simp]*:

$i < dim-vec\ v \Longrightarrow (a \cdot_v v) \$ i = a * v \$ i$   
 $dim-vec (a \cdot_v v) = dim-vec\ v$   
(proof)

**lemma** *add-carrier-vec[simp]*:

$v_1 \in carrier-vec\ n \Longrightarrow v_2 \in carrier-vec\ n \Longrightarrow v_1 + v_2 \in carrier-vec\ n$   
(proof)

**lemma** *minus-carrier-vec[simp]*:

$v_1 \in \text{carrier-vec } n \implies v_2 \in \text{carrier-vec } n \implies v_1 - v_2 \in \text{carrier-vec } n$   
*<proof>*

**lemma** *comm-add-vec*[*ac-simps*]:

$(v_1 :: 'a :: \text{ab-semigroup-add vec}) \in \text{carrier-vec } n \implies v_2 \in \text{carrier-vec } n \implies v_1 + v_2 = v_2 + v_1$   
*<proof>*

**lemma** *assoc-add-vec*[*simp*]:

$(v_1 :: 'a :: \text{semigroup-add vec}) \in \text{carrier-vec } n \implies v_2 \in \text{carrier-vec } n \implies v_3 \in \text{carrier-vec } n$   
 $\implies (v_1 + v_2) + v_3 = v_1 + (v_2 + v_3)$   
*<proof>*

**lemma** *zero-minus-vec*[*simp*]:  $(v :: 'a :: \text{group-add vec}) \in \text{carrier-vec } n \implies 0_v n - v = - v$   
*<proof>*

**lemma** *minus-zero-vec*[*simp*]:  $(v :: 'a :: \text{group-add vec}) \in \text{carrier-vec } n \implies v - 0_v n = v$   
*<proof>*

**lemma** *minus-cancel-vec*[*simp*]:  $(v :: 'a :: \text{group-add vec}) \in \text{carrier-vec } n \implies v - v = 0_v n$   
*<proof>*

**lemma** *minus-add-uminus-vec*:  $(v :: 'a :: \text{group-add vec}) \in \text{carrier-vec } n \implies w \in \text{carrier-vec } n \implies v - w = v + (- w)$   
*<proof>*

**lemma** *comm-monoid-vec*: *comm-monoid* (*monoid-vec TYPE ('a :: comm-monoid-add) n*)  
*<proof>*

**lemma** *left-zero-vec*[*simp*]:  $(v :: 'a :: \text{monoid-add vec}) \in \text{carrier-vec } n \implies 0_v n + v = v$  *<proof>*

**lemma** *right-zero-vec*[*simp*]:  $(v :: 'a :: \text{monoid-add vec}) \in \text{carrier-vec } n \implies v + 0_v n = v$  *<proof>*

**lemma** *uminus-carrier-vec*[*simp*]:

$(- v \in \text{carrier-vec } n) = (v \in \text{carrier-vec } n)$   
*<proof>*

**lemma** *uminus-r-inv-vec*[*simp*]:

$(v :: 'a :: \text{group-add vec}) \in \text{carrier-vec } n \implies (v + - v) = 0_v n$   
*<proof>*

**lemma** *uminus-l-inv-vec*[simp]:

$(v :: 'a :: \text{group-add vec}) \in \text{carrier-vec } n \implies (-v + v) = 0_v n$   
(proof)

**lemma** *add-inv-exists-vec*:

$(v :: 'a :: \text{group-add vec}) \in \text{carrier-vec } n \implies \exists w \in \text{carrier-vec } n. w + v = 0_v n$   
 $n \wedge v + w = 0_v n$   
(proof)

**lemma** *comm-group-vec*: *comm-group* (*monoid-vec TYPE ('a :: ab-group-add) n*)

(proof)

**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*, *unfolded monoid-vec-simps*]

**lemma** *smult-carrier-vec*[simp]:  $(a \cdot_v v \in \text{carrier-vec } n) = (v \in \text{carrier-vec } n)$

(proof)

**lemma** *scalar-prod-left-zero*[simp]:  $v \in \text{carrier-vec } n \implies 0_v n \cdot v = 0$

(proof)

**lemma** *scalar-prod-right-zero*[simp]:  $v \in \text{carrier-vec } n \implies v \cdot 0_v n = 0$

(proof)

**lemma** *scalar-prod-left-unit*[simp]: **assumes**  $v: (v :: 'a :: \text{semiring-1 vec}) \in \text{carrier-vec } n$  **and**  $i: i < n$

**shows**  $\text{unit-vec } n \ i \cdot v = v \ \$ \ i$

(proof)

**lemma** *scalar-prod-right-unit*[simp]: **assumes**  $i: i < n$

**shows**  $(v :: 'a :: \text{semiring-1 vec}) \cdot \text{unit-vec } n \ i = v \ \$ \ i$

(proof)

**lemma** *add-scalar-prod-distrib*: **assumes**  $v: v_1 \in \text{carrier-vec } n \ v_2 \in \text{carrier-vec } n \ v_3 \in \text{carrier-vec } n$

**shows**  $(v_1 + v_2) \cdot v_3 = v_1 \cdot v_3 + v_2 \cdot v_3$

(proof)

**lemma** *scalar-prod-add-distrib*: **assumes**  $v: v_1 \in \text{carrier-vec } n \ v_2 \in \text{carrier-vec } n \ v_3 \in \text{carrier-vec } n$

**shows**  $v_1 \cdot (v_2 + v_3) = v_1 \cdot v_2 + v_1 \cdot v_3$   
 ⟨proof⟩

**lemma** *smult-scalar-prod-distrib*[simp]: **assumes**  $v: v_1 \in \text{carrier-vec } n \ v_2 \in \text{carrier-vec } n$   
**shows**  $(a \cdot_v v_1) \cdot v_2 = a * (v_1 \cdot v_2)$   
 ⟨proof⟩

**lemma** *scalar-prod-smult-distrib*[simp]: **assumes**  $v: v_1 \in \text{carrier-vec } n \ v_2 \in \text{carrier-vec } n$   
**shows**  $v_1 \cdot (a \cdot_v v_2) = (a :: 'a :: \text{comm-ring}) * (v_1 \cdot v_2)$   
 ⟨proof⟩

**lemma** *comm-scalar-prod*: **assumes**  $(v_1 :: 'a :: \text{comm-semiring-0 } \text{vec}) \in \text{carrier-vec } n \ v_2 \in \text{carrier-vec } n$   
**shows**  $v_1 \cdot v_2 = v_2 \cdot v_1$   
 ⟨proof⟩

**lemma** *add-smult-distrib-vec*:  
 $((a :: 'a :: \text{ring}) + b) \cdot_v v = a \cdot_v v + b \cdot_v v$   
 ⟨proof⟩

**lemma** *smult-add-distrib-vec*:  
**assumes**  $v \in \text{carrier-vec } n \ w \in \text{carrier-vec } n$   
**shows**  $(a :: 'a :: \text{ring}) \cdot_v (v + w) = a \cdot_v v + a \cdot_v w$   
 ⟨proof⟩

**lemma** *smult-smult-assoc*:  
 $a \cdot_v (b \cdot_v v) = (a * b :: 'a :: \text{ring}) \cdot_v v$   
 ⟨proof⟩

**lemma** *one-smult-vec* [simp]:  
 $(1 :: 'a :: \text{ring-1}) \cdot_v v = v$  ⟨proof⟩

**lemma** *uminus-zero-vec*[simp]:  $-(0_v \ n) = (0_v \ n :: 'a :: \text{group-add } \text{vec})$   
 ⟨proof⟩

**lemma** *index-finsum-vec*: **assumes** *finite*  $F$  **and**  $i: i < n$   
**and**  $vs: vs \in F \rightarrow \text{carrier-vec } n$   
**shows**  $\text{finsum-vec } \text{TYPE}('a :: \text{comm-monoid-add}) \ n \ vs \ F \ \$ \ i = \text{sum } (\lambda f. vs \ f \ \$ \ i) \ F$   
 ⟨proof⟩

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**  $\text{vec} :: (\text{ord}) \ \text{ord}$   
**begin**

**definition**  $\text{less-eq-vec} :: 'a \ \text{vec} \Rightarrow 'a \ \text{vec} \Rightarrow \text{bool}$  **where**

$less-eq-vec\ v\ w = (dim-vec\ v = dim-vec\ w \wedge (\forall\ i < dim-vec\ w. v\ \$\ i \leq w\ \$\ i))$

**definition**  $less-vec :: 'a\ vec \Rightarrow 'a\ vec \Rightarrow bool$  **where**

$less-vec\ v\ w = (v \leq w \wedge \neg (w \leq v))$

**instance**  $\langle proof \rangle$

**end**

**instantiation**  $vec :: (preorder)\ preorder$

**begin**

**instance**

$\langle proof \rangle$

**end**

**instantiation**  $vec :: (order)\ order$

**begin**

**instance**

$\langle proof \rangle$

**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**  $undef-mat :: nat \Rightarrow nat \Rightarrow (nat \times nat \Rightarrow 'a) \Rightarrow nat \times nat \Rightarrow 'a$  **where**

$undef-mat\ nr\ nc\ f \equiv \lambda\ (i,j). [[f\ (i,j). j <- [0 ..< nc]] . i <- [0 ..< nr]] ! i\ !\ j$

**lemma**  $undef-cong-mat$ : **assumes**  $\bigwedge\ i\ j. i < nr \Longrightarrow j < nc \Longrightarrow f\ (i,j) = f'\ (i,j)$

**shows**  $undef-mat\ nr\ nc\ f\ x = undef-mat\ nr\ nc\ f'\ x$

$\langle proof \rangle$

**definition**  $mk-mat :: nat \Rightarrow nat \Rightarrow (nat \times nat \Rightarrow 'a) \Rightarrow (nat \times nat \Rightarrow 'a)$  **where**

$mk-mat\ nr\ nc\ f \equiv \lambda\ (i,j). if\ i < nr \wedge j < nc\ then\ f\ (i,j)\ else\ undef-mat\ nr\ nc\ f\ (i,j)$

**lemma**  $cong-mk-mat$ : **assumes**  $\bigwedge\ i\ j. i < nr \Longrightarrow j < nc \Longrightarrow f\ (i,j) = f'\ (i,j)$

**shows**  $mk-mat\ nr\ nc\ f = mk-mat\ nr\ nc\ f'$

$\langle proof \rangle$

**typedef**  $'a\ mat = \{(nr, nc, mk-mat\ nr\ nc\ f) \mid nr\ nc\ f :: nat \times nat \Rightarrow 'a.\ True\}$

$\langle proof \rangle$

**setup-lifting**  $type-definition-mat$

**lift-definition**  $dim-row :: 'a\ mat \Rightarrow nat$  **is**  $fst$   $\langle proof \rangle$

**lift-definition**  $dim-col :: 'a\ mat \Rightarrow nat$  **is**  $fst\ o\ snd$   $\langle proof \rangle$

**lift-definition**  $index-mat :: 'a\ mat \Rightarrow (nat \times nat \Rightarrow 'a)$  (**infixl**  $\$ \$ 100$ ) **is**  $snd\ o\ snd$   $\langle proof \rangle$

**lift-definition**  $mat :: nat \Rightarrow nat \Rightarrow (nat \times nat \Rightarrow 'a) \Rightarrow 'a\ mat$

**is**  $\lambda nr\ nc\ f. (nr, nc, mk\text{-}mat\ nr\ nc\ f)$   $\langle proof \rangle$   
**lift-definition**  $mat\text{-}of\text{-}row\text{-}fun :: nat \Rightarrow nat \Rightarrow (nat \Rightarrow 'a\ vec) \Rightarrow 'a\ mat\ (mat_r)$   
**is**  $\lambda nr\ nc\ f. (nr, nc, mk\text{-}mat\ nr\ nc\ (\lambda (i,j). f\ i\ \$\ j))$   $\langle proof \rangle$

**definition**  $mat\text{-}to\text{-}list :: 'a\ mat \Rightarrow 'a\ list\ list$  **where**  
 $mat\text{-}to\text{-}list\ A = [ [A\ \$\$ (i,j) . j <- [0 ..< dim\text{-}col\ A]] . i <- [0 ..< dim\text{-}row\ A]]$

**fun**  $square\text{-}mat :: 'a\ mat \Rightarrow bool$  **where**  $square\text{-}mat\ A = (dim\text{-}col\ A = dim\text{-}row\ A)$

**definition**  $upper\text{-}triangular :: 'a::zero\ mat \Rightarrow bool$   
**where**  $upper\text{-}triangular\ A \equiv$   
 $\forall i < dim\text{-}row\ A. \forall j < i. A\ \$\$ (i,j) = 0$

**lemma**  $upper\text{-}triangularD[elim] :$   
 $upper\text{-}triangular\ A \Longrightarrow j < i \Longrightarrow i < dim\text{-}row\ A \Longrightarrow A\ \$\$ (i,j) = 0$   
 $\langle proof \rangle$

**lemma**  $upper\text{-}triangularI[intro] :$   
 $(\bigwedge i\ j. j < i \Longrightarrow i < dim\text{-}row\ A \Longrightarrow A\ \$\$ (i,j) = 0) \Longrightarrow upper\text{-}triangular\ A$   
 $\langle proof \rangle$

**lemma**  $dim\text{-}row\text{-}mat[simp] : dim\text{-}row\ (mat\ nr\ nc\ f) = nr\ dim\text{-}row\ (mat_r\ nr\ nc\ g)$   
 $= nr$   
 $\langle proof \rangle$

**lemma**  $dim\text{-}col\text{-}mat[simp] : dim\text{-}col\ (mat\ nr\ nc\ f) = nc\ dim\text{-}col\ (mat_r\ nr\ nc\ g) =$   
 $nc$   
 $\langle proof \rangle$

**definition**  $carrier\text{-}mat :: nat \Rightarrow nat \Rightarrow 'a\ mat\ set$   
**where**  $carrier\text{-}mat\ nr\ nc = \{ m . dim\text{-}row\ m = nr \wedge dim\text{-}col\ m = nc \}$

**lemma**  $carrier\text{-}mat\text{-}triv[simp] : m \in carrier\text{-}mat\ (dim\text{-}row\ m)\ (dim\text{-}col\ m)$   
 $\langle proof \rangle$

**lemma**  $mat\text{-}carrier[simp] : mat\ nr\ nc\ f \in carrier\text{-}mat\ nr\ nc$   
 $\langle proof \rangle$

**definition**  $elements\text{-}mat :: 'a\ mat \Rightarrow 'a\ set$   
**where**  $elements\text{-}mat\ A = set\ [A\ \$\$ (i,j). i <- [0 ..< dim\text{-}row\ A], j <- [0 ..< dim\text{-}col\ A]]$

**lemma**  $elements\text{-}matD [dest] :$   
 $a \in elements\text{-}mat\ A \Longrightarrow \exists i\ j. i < dim\text{-}row\ A \wedge j < dim\text{-}col\ A \wedge a = A\ \$\$ (i,j)$   
 $\langle proof \rangle$

**lemma**  $elements\text{-}matI [intro] :$   
 $A \in carrier\text{-}mat\ nr\ nc \Longrightarrow i < nr \Longrightarrow j < nc \Longrightarrow a = A\ \$\$ (i,j) \Longrightarrow a \in$   
 $elements\text{-}mat\ A$

*<proof>*

**lemma** *index-mat[simp]*:  $i < nr \implies j < nc \implies \text{mat } nr \ nc \ f \ \$\$ (i,j) = f (i,j)$   
 $i < nr \implies j < nc \implies \text{mat}_r \ nr \ nc \ g \ \$\$ (i,j) = g \ i \ \$ \ j$   
*<proof>*

**lemma** *eq-matI[intro]*:  $(\bigwedge i \ j . i < \text{dim-row } B \implies j < \text{dim-col } B \implies A \ \$\$ (i,j) = B \ \$\$ (i,j))$   
 $\implies \text{dim-row } A = \text{dim-row } B$   
 $\implies \text{dim-col } A = \text{dim-col } B$   
 $\implies A = B$   
*<proof>*

**lemma** *carrier-matI[intro]*:  
**assumes**  $\text{dim-row } A = nr \ \text{dim-col } A = nc$  **shows**  $A \in \text{carrier-mat } nr \ nc$   
*<proof>*

**lemma** *carrier-matD[dest,simp]*: **assumes**  $A \in \text{carrier-mat } nr \ nc$   
**shows**  $\text{dim-row } A = nr \ \text{dim-col } A = nc$  *<proof>*

**lemma** *cong-mat*: **assumes**  $nr = nr' \ nc = nc' \ \bigwedge i \ j . i < nr \implies j < nc \implies f (i,j) = f' (i,j)$  **shows**  $\text{mat } nr \ nc \ f = \text{mat } nr' \ nc' \ f'$   
*<proof>*

**definition** *row* :: 'a mat  $\Rightarrow$  nat  $\Rightarrow$  'a vec **where**  
 $\text{row } A \ i = \text{vec } (\text{dim-col } A) (\lambda j . A \ \$\$ (i,j))$

**definition** *rows* :: 'a mat  $\Rightarrow$  'a vec list **where**  
 $\text{rows } A = \text{map } (\text{row } A) [0..<\text{dim-row } A]$

**lemma** *row-carrier[simp]*:  $\text{row } A \ i \in \text{carrier-vec } (\text{dim-col } A)$  *<proof>*

**lemma** *rows-carrier[simp]*:  $\text{set } (\text{rows } A) \subseteq \text{carrier-vec } (\text{dim-col } A)$  *<proof>*

**lemma** *length-rows[simp]*:  $\text{length } (\text{rows } A) = \text{dim-row } A$  *<proof>*

**lemma** *nth-rows[simp]*:  $i < \text{dim-row } A \implies \text{rows } A \ ! \ i = \text{row } A \ i$   
*<proof>*

**lemma** *row-mat-of-row-fun[simp]*:  $i < nr \implies \text{dim-vec } (f \ i) = nc \implies \text{row } (\text{mat}_r \ nr \ nc \ f) \ i = f \ i$   
*<proof>*

**lemma** *set-rows-carrier*:  
**assumes**  $A \in \text{carrier-mat } m \ n$  **and**  $v \in \text{set } (\text{rows } A)$  **shows**  $v \in \text{carrier-vec } n$   
*<proof>*

**definition** *mat-of-rows* :: nat  $\Rightarrow$  'a vec list  $\Rightarrow$  'a mat  
**where**  $\text{mat-of-rows } n \ rs = \text{mat } (\text{length } rs) \ n (\lambda(i,j) . rs \ ! \ i \ \$ \ j)$



**definition** *mat-of-rows-list* ::  $\text{nat} \Rightarrow 'a \text{ list list} \Rightarrow 'a \text{ mat}$  **where**  
 $\text{mat-of-rows-list } nc \text{ } rs = \text{mat } (\text{length } rs) \text{ } nc \text{ } (\lambda (i,j). rs ! i ! j)$

**lemma** *mat-of-rows-carrier[simp]*:  
 $\text{mat-of-rows } n \text{ } vs \in \text{carrier-mat } (\text{length } vs) \text{ } n$   
 $\text{dim-row } (\text{mat-of-rows } n \text{ } vs) = \text{length } vs$   
 $\text{dim-col } (\text{mat-of-rows } n \text{ } vs) = n$   
 $\langle \text{proof} \rangle$

**lemma** *mat-of-rows-row[simp]*:  
**assumes**  $i < \text{length } vs$  **and**  $n: vs ! i \in \text{carrier-vec } n$   
**shows**  $\text{row } (\text{mat-of-rows } n \text{ } vs) \text{ } i = vs ! i$   
 $\langle \text{proof} \rangle$

**lemma** *rows-mat-of-rows[simp]*:  
**assumes**  $set \text{ } vs \subseteq \text{carrier-vec } n$  **shows**  $\text{rows } (\text{mat-of-rows } n \text{ } vs) = vs$   
 $\langle \text{proof} \rangle$

**lemma** *mat-of-rows-rows[simp]*:  
 $\text{mat-of-rows } (\text{dim-col } A) \text{ } (\text{rows } A) = A$   
 $\langle \text{proof} \rangle$

**definition** *col* ::  $'a \text{ mat} \Rightarrow \text{nat} \Rightarrow 'a \text{ vec}$  **where**  
 $\text{col } A \text{ } j = \text{vec } (\text{dim-row } A) \text{ } (\lambda i. A \text{ } \$\$ (i,j))$

**definition** *cols* ::  $'a \text{ mat} \Rightarrow 'a \text{ vec list}$  **where**  
 $\text{cols } A = \text{map } (\text{col } A) \text{ } [0..<\text{dim-col } A]$

**definition** *mat-of-cols* ::  $\text{nat} \Rightarrow 'a \text{ vec list} \Rightarrow 'a \text{ mat}$   
**where**  $\text{mat-of-cols } n \text{ } cs = \text{mat } n \text{ } (\text{length } cs) \text{ } (\lambda(i,j). cs ! j \$ i)$

**definition** *mat-of-cols-list* ::  $\text{nat} \Rightarrow 'a \text{ list list} \Rightarrow 'a \text{ mat}$  **where**  
 $\text{mat-of-cols-list } nr \text{ } cs = \text{mat } nr \text{ } (\text{length } cs) \text{ } (\lambda (i,j). cs ! j ! i)$

**lemma** *col-dim[simp]*:  $\text{col } A \text{ } i \in \text{carrier-vec } (\text{dim-row } A) \langle \text{proof} \rangle$

**lemma** *dim-col[simp]*:  $\text{dim-vec } (\text{col } A \text{ } i) = \text{dim-row } A \langle \text{proof} \rangle$

**lemma** *cols-dim[simp]*:  $set \text{ } (\text{cols } A) \subseteq \text{carrier-vec } (\text{dim-row } A) \langle \text{proof} \rangle$

**lemma** *cols-length[simp]*:  $\text{length } (\text{cols } A) = \text{dim-col } A \langle \text{proof} \rangle$

**lemma** *cols-nth[simp]*:  $i < \text{dim-col } A \implies \text{cols } A ! i = \text{col } A \text{ } i$   
 $\langle \text{proof} \rangle$

**lemma** *mat-of-cols-carrier[simp]*:  
 $\text{mat-of-cols } n \text{ } vs \in \text{carrier-mat } n \text{ } (\text{length } vs)$

$\text{dim-row } (\text{mat-of-cols } n \text{ vs}) = n$   
 $\text{dim-col } (\text{mat-of-cols } n \text{ vs}) = \text{length vs}$   
 <proof>

**lemma** *col-mat-of-cols*[simp]:  
**assumes**  $j < \text{length vs}$  **and**  $n: \text{vs} ! j \in \text{carrier-vec } n$   
**shows**  $\text{col } (\text{mat-of-cols } n \text{ vs}) j = \text{vs} ! j$   
 <proof>

**lemma** *cols-mat-of-cols*[simp]:  
**assumes**  $\text{set vs} \subseteq \text{carrier-vec } n$  **shows**  $\text{cols } (\text{mat-of-cols } n \text{ vs}) = \text{vs}$   
 <proof>

**lemma** *mat-of-cols-cols*[simp]:  
 $\text{mat-of-cols } (\text{dim-row } A) (\text{cols } A) = A$   
 <proof>

**instantiation** *mat* :: (ord) ord  
**begin**

**definition** *less-eq-mat* :: 'a mat  $\Rightarrow$  'a mat  $\Rightarrow$  bool **where**  
 $\text{less-eq-mat } A B = (\text{dim-row } A = \text{dim-row } B \wedge \text{dim-col } A = \text{dim-col } B \wedge$   
 $(\forall i < \text{dim-row } B. \forall j < \text{dim-col } B. A \text{ $$ } (i,j) \leq B \text{ $$ } (i,j)))$

**definition** *less-mat* :: 'a mat  $\Rightarrow$  'a mat  $\Rightarrow$  bool **where**  
 $\text{less-mat } A B = (A \leq B \wedge \neg (B \leq A))$

**instance** <proof>  
**end**

**instantiation** *mat* :: (preorder) preorder  
**begin**  
**instance**  
 <proof>  
**end**

**instantiation** *mat* :: (order) order  
**begin**  
**instance**  
 <proof>  
**end**

**instantiation** *mat* :: (plus) plus  
**begin**  
**definition** *plus-mat* :: ('a :: plus) mat  $\Rightarrow$  'a mat  $\Rightarrow$  'a mat **where**  
 $A + B \equiv \text{mat } (\text{dim-row } B) (\text{dim-col } B) (\lambda ij. A \text{ $$ } ij + B \text{ $$ } ij)$   
**instance** <proof>  
**end**

**definition** *map-mat* :: ('a ⇒ 'b) ⇒ 'a mat ⇒ 'b mat **where**  
*map-mat* f A ≡ mat (dim-row A) (dim-col A) (λ ij. f (A \$\$ ij))

**definition** *smult-mat* :: 'a :: times ⇒ 'a mat ⇒ 'a mat (**infixl** ·<sub>m</sub> 70)  
**where** a ·<sub>m</sub> A ≡ map-mat (λ b. a \* b) A

**definition** *zero-mat* :: nat ⇒ nat ⇒ 'a :: zero mat (0<sub>m</sub>) **where**  
0<sub>m</sub> nr nc ≡ mat nr nc (λ ij. 0)

**lemma** *elements-0-mat* [simp]: elements-mat (0<sub>m</sub> nr nc) ⊆ {0}  
⟨proof⟩

**definition** *transpose-mat* :: 'a mat ⇒ 'a mat **where**  
*transpose-mat* A ≡ mat (dim-col A) (dim-row A) (λ (i,j). A \$\$ (j,i))

**definition** *one-mat* :: nat ⇒ 'a :: {zero,one} mat (1<sub>m</sub>) **where**  
1<sub>m</sub> n ≡ mat n n (λ (i,j). if i = j then 1 else 0)

**instantiation** *mat* :: (uminus) uminus  
**begin**

**definition** *uminus-mat* :: 'a :: uminus mat ⇒ 'a mat **where**  
– A ≡ mat (dim-row A) (dim-col A) (λ ij. – (A \$\$ ij))

**instance** ⟨proof⟩  
**end**

**instantiation** *mat* :: (minus) minus  
**begin**

**definition** *minus-mat* :: ('a :: minus) mat ⇒ 'a mat ⇒ 'a mat **where**  
A – B ≡ mat (dim-row B) (dim-col B) (λ ij. A \$\$ ij – B \$\$ ij)

**instance** ⟨proof⟩  
**end**

**instantiation** *mat* :: (semiring-0) times  
**begin**

**definition** *times-mat* :: 'a :: semiring-0 mat ⇒ 'a mat ⇒ 'a mat  
**where** A \* B ≡ mat (dim-row A) (dim-col B) (λ (i,j). row A i · col B j)

**instance** ⟨proof⟩  
**end**

**definition** *mult-mat-vec* :: 'a :: semiring-0 mat ⇒ 'a vec ⇒ 'a vec (**infixl** \*\_<sub>v</sub> 70)  
**where** A \*\_<sub>v</sub> v ≡ vec (dim-row A) (λ i. row A i · v)

**definition** *inverts-mat* :: 'a :: semiring-1 mat ⇒ 'a mat ⇒ bool **where**  
*inverts-mat* A B ≡ A \* B = 1<sub>m</sub> (dim-row A)

**definition** *invertible-mat* :: 'a :: semiring-1 mat ⇒ bool  
**where** *invertible-mat* A ≡ square-mat A ∧ (∃ B. *inverts-mat* A B ∧ *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*  $\equiv$   $\langle$   
*carrier* = *carrier-mat nr nc*,  
*mult* = (+),  
*one* =  $0_m$  nr nc  $\rangle$

**definition** *ring-mat* :: 'a :: semiring-1 itself  $\Rightarrow$  nat  $\Rightarrow$  'b  $\Rightarrow$  ('a mat, 'b) ring-scheme  
**where**

*ring-mat ty n b*  $\equiv$   $\langle$   
*carrier* = *carrier-mat n n*,  
*mult* = (\*),  
*one* =  $1_m$  n,  
*zero* =  $0_m$  n n,  
*add* = (+),  
... = b  $\rangle$

**definition** *module-mat* :: 'a :: semiring-1 itself  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  ('a, 'a mat) module  
**where**

*module-mat ty nr nc*  $\equiv$   $\langle$   
*carrier* = *carrier-mat nr nc*,  
*mult* = (\*),  
*one* =  $1_m$  nr,  
*zero* =  $0_m$  nr nc,  
*add* = (+),  
*smult* = ( $\cdot_m$ )  $\rangle$

**lemma** *ring-mat-simps*:

*mult* (*ring-mat ty n b*) = (\*)  
*add* (*ring-mat ty n b*) = (+)  
*one* (*ring-mat ty n b*) =  $1_m$  n  
*zero* (*ring-mat ty n b*) =  $0_m$  n n  
*carrier* (*ring-mat ty n b*) = *carrier-mat n n*  
 $\langle$ proof  $\rangle$

**lemma** *module-mat-simps*:

*mult* (*module-mat ty nr nc*) = (\*)  
*add* (*module-mat ty nr nc*) = (+)  
*one* (*module-mat ty nr nc*) =  $1_m$  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*) = ( $\cdot_m$ )  
 $\langle$ proof  $\rangle$

**lemma** *index-zero-mat[simp]*:  $i < nr \Longrightarrow j < nc \Longrightarrow 0_m$  nr nc  $\$$ (i,j) = 0  
*dim-row* ( $0_m$  nr nc) = nr *dim-col* ( $0_m$  nr nc) = nc  
 $\langle$ proof  $\rangle$$

**lemma** *index-one-mat[simp]*:  $i < n \Longrightarrow j < n \Longrightarrow 1_m$  n  $\$$ (i,j) = (if i = j then$

1 else 0)

$$\dim\text{-row } (1_m \ n) = n \quad \dim\text{-col } (1_m \ n) = n$$

*<proof>*

**lemma** *index-add-mat*[simp]:

$$i < \dim\text{-row } B \implies j < \dim\text{-col } B \implies (A + B) \ \$\$ (i,j) = A \ \$\$ (i,j) + B \ \$\$ (i,j)$$

$$\dim\text{-row } (A + B) = \dim\text{-row } B \quad \dim\text{-col } (A + B) = \dim\text{-col } B$$

*<proof>*

**lemma** *index-minus-mat*[simp]:

$$i < \dim\text{-row } B \implies j < \dim\text{-col } B \implies (A - B) \ \$\$ (i,j) = A \ \$\$ (i,j) - B \ \$\$ (i,j)$$

$$\dim\text{-row } (A - B) = \dim\text{-row } B \quad \dim\text{-col } (A - B) = \dim\text{-col } B$$

*<proof>*

**lemma** *index-map-mat*[simp]:

$$i < \dim\text{-row } A \implies j < \dim\text{-col } A \implies \text{map-mat } f \ A \ \$\$ (i,j) = f \ (A \ \$\$ (i,j))$$

$$\dim\text{-row } (\text{map-mat } f \ A) = \dim\text{-row } A \quad \dim\text{-col } (\text{map-mat } f \ A) = \dim\text{-col } A$$

*<proof>*

**lemma** *index-smult-mat*[simp]:

$$i < \dim\text{-row } A \implies j < \dim\text{-col } A \implies (a \cdot_m A) \ \$\$ (i,j) = a * A \ \$\$ (i,j)$$

$$\dim\text{-row } (a \cdot_m A) = \dim\text{-row } A \quad \dim\text{-col } (a \cdot_m A) = \dim\text{-col } A$$

*<proof>*

**lemma** *index-uminus-mat*[simp]:

$$i < \dim\text{-row } A \implies j < \dim\text{-col } A \implies (- A) \ \$\$ (i,j) = - (A \ \$\$ (i,j))$$

$$\dim\text{-row } (- A) = \dim\text{-row } A \quad \dim\text{-col } (- A) = \dim\text{-col } A$$

*<proof>*

**lemma** *index-transpose-mat*[simp]:

$$i < \dim\text{-col } A \implies j < \dim\text{-row } A \implies \text{transpose-mat } A \ \$\$ (i,j) = A \ \$\$ (j,i)$$

$$\dim\text{-row } (\text{transpose-mat } A) = \dim\text{-col } A \quad \dim\text{-col } (\text{transpose-mat } A) = \dim\text{-row } A$$

*<proof>*

**lemma** *index-mult-mat*[simp]:

$$i < \dim\text{-row } A \implies j < \dim\text{-col } B \implies (A * B) \ \$\$ (i,j) = \text{row } A \ i \cdot \text{col } B \ j$$

$$\dim\text{-row } (A * B) = \dim\text{-row } A \quad \dim\text{-col } (A * B) = \dim\text{-col } B$$

*<proof>*

**lemma** *dim-mult-mat-vec*[simp]:  $\dim\text{-vec } (A *_v v) = \dim\text{-row } A$

*<proof>*

**lemma** *index-mult-mat-vec*[simp]:  $i < \dim\text{-row } A \implies (A *_v v) \ \$ i = \text{row } A \ i \cdot v$

*<proof>*

**lemma** *index-row*[simp]:

$$i < \dim\text{-row } A \implies j < \dim\text{-col } A \implies \text{row } A \ i \ \$ j = A \ \$\$ (i,j)$$

$$\dim\text{-vec } (\text{row } A \ i) = \dim\text{-col } A$$

*<proof>*

**lemma** *index-col[simp]*:  $i < \dim\text{-row } A \implies j < \dim\text{-col } A \implies \text{col } A \ j \ \$ \ i = A \ \$ \ i$   
*(i,j)*  
*<proof>*

**lemma** *upper-triangular-one[simp]*: *upper-triangular*  $(1_m \ n)$   
*<proof>*

**lemma** *upper-triangular-zero[simp]*: *upper-triangular*  $(0_m \ n \ n)$   
*<proof>*

**lemma** *mat-row-carrierI[intro,simp]*:  $\text{mat}_r \ nr \ nc \ r \in \text{carrier-mat } nr \ nc$   
*<proof>*

**lemma** *eq-rowI*: **assumes** *rows*:  $\bigwedge i. i < \dim\text{-row } B \implies \text{row } A \ i = \text{row } B \ i$   
**and** *dims*:  $\dim\text{-row } A = \dim\text{-row } B \ \dim\text{-col } A = \dim\text{-col } B$   
**shows**  $A = B$   
*<proof>*

**lemma** *row-mat[simp]*:  $i < nr \implies \text{row } (\text{mat } nr \ nc \ f) \ i = \text{vec } nc \ (\lambda j. f \ (i,j))$   
*<proof>*

**lemma** *col-mat[simp]*:  $j < nc \implies \text{col } (\text{mat } nr \ nc \ f) \ j = \text{vec } nr \ (\lambda i. f \ (i,j))$   
*<proof>*

**lemma** *zero-carrier-mat[simp]*:  $0_m \ nr \ nc \in \text{carrier-mat } nr \ nc$   
*<proof>*

**lemma** *smult-carrier-mat[simp]*:  
 $A \in \text{carrier-mat } nr \ nc \implies k \cdot_m A \in \text{carrier-mat } nr \ nc$   
*<proof>*

**lemma** *add-carrier-mat[simp]*:  
 $B \in \text{carrier-mat } nr \ nc \implies A + B \in \text{carrier-mat } nr \ nc$   
*<proof>*

**lemma** *one-carrier-mat[simp]*:  $1_m \ n \in \text{carrier-mat } n \ n$   
*<proof>*

**lemma** *uminus-carrier-mat*:  
 $A \in \text{carrier-mat } nr \ nc \implies (- A) \in \text{carrier-mat } nr \ nc$   
*<proof>*

**lemma** *uminus-carrier-iff-mat[simp]*:  
 $(- A \in \text{carrier-mat } nr \ nc) = (A \in \text{carrier-mat } nr \ nc)$   
*<proof>*

**lemma** *minus-carrier-mat*:  
 $B \in \text{carrier-mat } nr \ nc \implies (A - B) \in \text{carrier-mat } nr \ nc$

*<proof>*

**lemma** *transpose-carrier-mat[simp]*:  $(\text{transpose-mat } A \in \text{carrier-mat } nr \ nc) = (A \in \text{carrier-mat } nr \ nc)$

*<proof>*

**lemma** *row-carrier-vec[simp]*:  $i < nr \implies A \in \text{carrier-mat } nr \ nc \implies \text{row } A \ i \in \text{carrier-vec } nc$

*<proof>*

**lemma** *col-carrier-vec[simp]*:  $j < nc \implies A \in \text{carrier-mat } nr \ nc \implies \text{col } A \ j \in \text{carrier-vec } nr$

*<proof>*

**lemma** *mult-carrier-mat[simp]*:

$A \in \text{carrier-mat } nr \ n \implies B \in \text{carrier-mat } n \ nc \implies A * B \in \text{carrier-mat } nr \ nc$

*<proof>*

**lemma** *mult-mat-vec-carrier[simp]*:

$A \in \text{carrier-mat } nr \ n \implies v \in \text{carrier-vec } n \implies A *_v v \in \text{carrier-vec } nr$

*<proof>*

**lemma** *comm-add-mat[ac-simps]*:

$(A :: 'a :: \text{comm-monoid-add mat}) \in \text{carrier-mat } nr \ nc \implies B \in \text{carrier-mat } nr \ nc \implies A + B = B + A$

*<proof>*

**lemma** *minus-r-inv-mat[simp]*:

$(A :: 'a :: \text{group-add mat}) \in \text{carrier-mat } nr \ nc \implies (A - A) = 0_m \ nr \ nc$

*<proof>*

**lemma** *uminus-l-inv-mat[simp]*:

$(A :: 'a :: \text{group-add mat}) \in \text{carrier-mat } nr \ nc \implies (- A + A) = 0_m \ nr \ nc$

*<proof>*

**lemma** *add-inv-exists-mat*:

$(A :: 'a :: \text{group-add mat}) \in \text{carrier-mat } nr \ nc \implies \exists B \in \text{carrier-mat } nr \ nc. B + A = 0_m \ nr \ nc \wedge A + B = 0_m \ nr \ nc$

*<proof>*

**lemma** *assoc-add-mat[simp]*:

$(A :: 'a :: \text{monoid-add mat}) \in \text{carrier-mat } nr \ nc \implies B \in \text{carrier-mat } nr \ nc \implies C \in \text{carrier-mat } nr \ nc$

$\implies (A + B) + C = A + (B + C)$

*<proof>*

**lemma** *uminus-add-mat*: **fixes**  $A :: 'a :: \text{group-add mat}$

**assumes**  $A \in \text{carrier-mat } nr \ nc$   
**and**  $B \in \text{carrier-mat } nr \ nc$   
**shows**  $-(A + B) = -B + -A$   
 $\langle \text{proof} \rangle$

**lemma** *transpose-transpose[simp]*:  
 $\text{transpose-mat } (\text{transpose-mat } A) = A$   
 $\langle \text{proof} \rangle$

**lemma** *transpose-one[simp]*:  $\text{transpose-mat } (1_m \ n) = (1_m \ n)$   
 $\langle \text{proof} \rangle$

**lemma** *row-transpose[simp]*:  
 $j < \text{dim-col } A \implies \text{row } (\text{transpose-mat } A) \ j = \text{col } A \ j$   
 $\langle \text{proof} \rangle$

**lemma** *col-transpose[simp]*:  
 $i < \text{dim-row } A \implies \text{col } (\text{transpose-mat } A) \ i = \text{row } A \ i$   
 $\langle \text{proof} \rangle$

**lemma** *row-zero[simp]*:  
 $i < nr \implies \text{row } (0_m \ nr \ nc) \ i = 0_v \ nc$   
 $\langle \text{proof} \rangle$

**lemma** *col-zero[simp]*:  
 $j < nc \implies \text{col } (0_m \ nr \ nc) \ j = 0_v \ nr$   
 $\langle \text{proof} \rangle$

**lemma** *row-one[simp]*:  
 $i < n \implies \text{row } (1_m \ n) \ i = \text{unit-vec } n \ i$   
 $\langle \text{proof} \rangle$

**lemma** *col-one[simp]*:  
 $j < n \implies \text{col } (1_m \ n) \ j = \text{unit-vec } n \ j$   
 $\langle \text{proof} \rangle$

**lemma** *transpose-add*:  $A \in \text{carrier-mat } nr \ nc \implies B \in \text{carrier-mat } nr \ nc$   
 $\implies \text{transpose-mat } (A + B) = \text{transpose-mat } A + \text{transpose-mat } B$   
 $\langle \text{proof} \rangle$

**lemma** *transpose-minus*:  $A \in \text{carrier-mat } nr \ nc \implies B \in \text{carrier-mat } nr \ nc$   
 $\implies \text{transpose-mat } (A - B) = \text{transpose-mat } A - \text{transpose-mat } B$   
 $\langle \text{proof} \rangle$

**lemma** *transpose-uminus*:  $A \in \text{carrier-mat } nr \ nc \implies \text{transpose-mat } (-A) = -$   
 $(\text{transpose-mat } A)$   
 $\langle \text{proof} \rangle$

**lemma** *row-add[simp]*:



$A \in \text{carrier-mat } nr \ nc \implies B \in \text{carrier-mat } nr \ nc \implies i < nr$   
 $\implies \text{row } (A + B) \ i = \text{row } A \ i + \text{row } B \ i$   
 $i < \text{dim-row } A \implies \text{dim-row } B = \text{dim-row } A \implies \text{dim-col } B = \text{dim-col } A \implies \text{row}$   
 $(A + B) \ i = \text{row } A \ i + \text{row } B \ i$   
 ⟨proof⟩

**lemma** *col-add[simp]*:

$A \in \text{carrier-mat } nr \ nc \implies B \in \text{carrier-mat } nr \ nc \implies j < nc$   
 $\implies \text{col } (A + B) \ j = \text{col } A \ j + \text{col } B \ j$   
 ⟨proof⟩

**lemma** *row-mult[simp]*: **assumes**  $m$ :  $A \in \text{carrier-mat } nr \ n \ B \in \text{carrier-mat } n \ nc$

**and**  $i$ :  $i < nr$

**shows**  $\text{row } (A * B) \ i = \text{vec } nc \ (\lambda j. \text{row } A \ i \cdot \text{col } B \ j)$

⟨proof⟩

**lemma** *col-mult[simp]*: **assumes**  $m$ :  $A \in \text{carrier-mat } nr \ n \ B \in \text{carrier-mat } n \ nc$

**and**  $j$ :  $j < nc$

**shows**  $\text{col } (A * B) \ j = \text{vec } nr \ (\lambda i. \text{row } A \ i \cdot \text{col } B \ j)$

⟨proof⟩

**lemma** *transpose-mult*:

$(A :: 'a :: \text{comm-semiring-0 mat}) \in \text{carrier-mat } nr \ n \implies B \in \text{carrier-mat } n \ nc$

$\implies \text{transpose-mat } (A * B) = \text{transpose-mat } B * \text{transpose-mat } A$

⟨proof⟩

**lemma** *left-add-zero-mat[simp]*:

$(A :: 'a :: \text{monoid-add mat}) \in \text{carrier-mat } nr \ nc \implies 0_m \ nr \ nc + A = A$

⟨proof⟩

**lemma** *add-uminus-minus-mat*:  $A \in \text{carrier-mat } nr \ nc \implies B \in \text{carrier-mat } nr \ nc$

$\implies$

$A + (- B) = A - (B :: 'a :: \text{group-add mat})$

⟨proof⟩

**lemma** *right-add-zero-mat[simp]*:  $A \in \text{carrier-mat } nr \ nc \implies$

$A + 0_m \ nr \ nc = (A :: 'a :: \text{monoid-add mat})$

⟨proof⟩

**lemma** *left-mult-zero-mat*:

$A \in \text{carrier-mat } n \ nc \implies 0_m \ nr \ n * A = 0_m \ nr \ nc$

⟨proof⟩

**lemma** *left-mult-zero-mat'[simp]*:  $\text{dim-row } A = n \implies 0_m \ nr \ n * A = 0_m \ nr$   
 $(\text{dim-col } A)$

⟨proof⟩

**lemma** *right-mult-zero-mat*:

$A \in \text{carrier-mat } nr \ n \implies A * 0_m \ n \ nc = 0_m \ nr \ nc$

*<proof>*

**lemma** *right-mult-zero-mat'*[simp]:  $\dim\text{-col } A = n \implies A * 0_m \text{ nr } nc = 0_m (\dim\text{-row } A) \text{ nc}$   
*<proof>*

**lemma** *left-mult-one-mat*:  
 $(A :: 'a :: \text{semiring-1 mat}) \in \text{carrier-mat nr nc} \implies 1_m \text{ nr} * A = A$   
*<proof>*

**lemma** *left-mult-one-mat'*[simp]:  $\dim\text{-row } (A :: 'a :: \text{semiring-1 mat}) = n \implies 1_m \text{ nr} * A = A$   
*<proof>*

**lemma** *right-mult-one-mat*:  
 $(A :: 'a :: \text{semiring-1 mat}) \in \text{carrier-mat nr nc} \implies A * 1_m \text{ nc} = A$   
*<proof>*

**lemma** *right-mult-one-mat'*[simp]:  $\dim\text{-col } (A :: 'a :: \text{semiring-1 mat}) = n \implies A * 1_m \text{ nr} = A$   
*<proof>*

**lemma** *one-mult-mat-vec*[simp]:  
 $(v :: 'a :: \text{semiring-1 vec}) \in \text{carrier-vec n} \implies 1_m \text{ nr} *_v v = v$   
*<proof>*

**lemma** *minus-add-uminus-mat*: **fixes**  $A :: 'a :: \text{group-add mat}$   
**shows**  $A \in \text{carrier-mat nr nc} \implies B \in \text{carrier-mat nr nc} \implies A - B = A + (- B)$   
*<proof>*

**lemma** *add-mult-distrib-mat*[algebra-simps]: **assumes**  $m: A \in \text{carrier-mat nr n}$   
 $B \in \text{carrier-mat nr n}$   $C \in \text{carrier-mat n nc}$   
**shows**  $(A + B) * C = A * C + B * C$   
*<proof>*

**lemma** *mult-add-distrib-mat*[algebra-simps]: **assumes**  $m: A \in \text{carrier-mat nr n}$   
 $B \in \text{carrier-mat n nc}$   $C \in \text{carrier-mat n nc}$   
**shows**  $A * (B + C) = A * B + A * C$   
*<proof>*

**lemma** *add-mult-distrib-mat-vec*[algebra-simps]: **assumes**  $m: A \in \text{carrier-mat nr nc}$   
 $B \in \text{carrier-mat nr nc}$   $v \in \text{carrier-vec nc}$   
**shows**  $(A + B) *_v v = A *_v v + B *_v v$   
*<proof>*

**lemma** *mult-add-distrib-mat-vec*[algebra-simps]: **assumes**  $m: A \in \text{carrier-mat nr nc}$   
 $B \in \text{carrier-mat nr nc}$   $v \in \text{carrier-vec nc}$   
**shows**  $A *_v (B + C) = A *_v B + A *_v C$   
*<proof>*

$v_1 \in \text{carrier-vec } nc \ v_2 \in \text{carrier-vec } nc$   
**shows**  $A *_v (v_1 + v_2) = A *_v v_1 + A *_v v_2$   
 $\langle \text{proof} \rangle$

**lemma** *mult-mat-vec*:

**assumes**  $m: (A::'a::\text{field mat}) \in \text{carrier-mat } nr \ nc$  **and**  $v: v \in \text{carrier-vec } nc$   
**shows**  $A *_v (k \cdot_v v) = k \cdot_v (A *_v v)$  (**is** ?l = ?r)  
 $\langle \text{proof} \rangle$

**lemma** *assoc-scalar-prod*: **assumes**  $*$ :  $v_1 \in \text{carrier-vec } nr \ A \in \text{carrier-mat } nr \ nc$   
 $v_2 \in \text{carrier-vec } nc$

**shows**  $\text{vec } nc \ (\lambda j. v_1 \cdot \text{col } A \ j) \cdot v_2 = v_1 \cdot \text{vec } nr \ (\lambda i. \text{row } A \ i \cdot v_2)$   
 $\langle \text{proof} \rangle$

**lemma** *assoc-mult-mat[simp]*:

$A \in \text{carrier-mat } n_1 \ n_2 \implies B \in \text{carrier-mat } n_2 \ n_3 \implies C \in \text{carrier-mat } n_3 \ n_4$   
 $\implies (A * B) * C = A * (B * C)$   
 $\langle \text{proof} \rangle$

**lemma** *assoc-mult-mat-vec[simp]*:

$A \in \text{carrier-mat } n_1 \ n_2 \implies B \in \text{carrier-mat } n_2 \ n_3 \implies v \in \text{carrier-vec } n_3$   
 $\implies (A * B) *_v v = A *_v (B *_v v)$   
 $\langle \text{proof} \rangle$

**lemma** *comm-monoid-mat*: *comm-monoid* (*monoid-mat*  $\text{TYPE}('a :: \text{comm-monoid-add})$   
 $nr \ nc$ )

$\langle \text{proof} \rangle$

**lemma** *comm-group-mat*: *comm-group* (*monoid-mat*  $\text{TYPE}('a :: \text{ab-group-add})$   $nr$   
 $nc$ )

$\langle \text{proof} \rangle$

**lemma** *semiring-mat*: *semiring* (*ring-mat*  $\text{TYPE}('a :: \text{semiring-1})$   $n \ b$ )

$\langle \text{proof} \rangle$

**lemma** *ring-mat*: *ring* (*ring-mat*  $\text{TYPE}('a :: \text{comm-ring-1})$   $n \ b$ )

$\langle \text{proof} \rangle$

**lemma** *abelian-group-mat*: *abelian-group* (*module-mat*  $\text{TYPE}('a :: \text{comm-ring-1})$   
 $nr \ nc$ )

$\langle \text{proof} \rangle$

**lemma** *row-smult[simp]*: **assumes**  $i: i < \text{dim-row } A$

**shows**  $\text{row } (k \cdot_m A) \ i = k \cdot_v (\text{row } A \ i)$   
 $\langle \text{proof} \rangle$

**lemma** *col-smult[simp]*: **assumes**  $i: i < \text{dim-col } A$

**shows**  $\text{col } (k \cdot_m A) \ i = k \cdot_v (\text{col } A \ i)$   
 $\langle \text{proof} \rangle$

**lemma** *row-uminus[simp]*: **assumes**  $i: i < \dim\text{-row } A$   
**shows**  $\text{row } (- A) i = - (\text{row } A i)$   
 $\langle \text{proof} \rangle$

**lemma** *scalar-prod-uminus-left[simp]*: **assumes**  $\dim: \dim\text{-vec } v = \dim\text{-vec } (w :: 'a$   
 $:: \text{ring vec})$   
**shows**  $- v \cdot w = - (v \cdot w)$   
 $\langle \text{proof} \rangle$

**lemma** *col-uminus[simp]*: **assumes**  $i: i < \dim\text{-col } A$   
**shows**  $\text{col } (- A) i = - (\text{col } A i)$   
 $\langle \text{proof} \rangle$

**lemma** *scalar-prod-uminus-right[simp]*: **assumes**  $\dim: \dim\text{-vec } v = \dim\text{-vec } (w ::$   
 $'a :: \text{ring vec})$   
**shows**  $v \cdot - w = - (v \cdot w)$   
 $\langle \text{proof} \rangle$

**context** **fixes**  $A B :: 'a :: \text{ring mat}$   
**assumes**  $\dim: \dim\text{-col } A = \dim\text{-row } B$   
**begin**

**lemma** *uminus-mult-left-mat[simp]*:  $(- A * B) = - (A * B)$   
 $\langle \text{proof} \rangle$

**lemma** *uminus-mult-right-mat[simp]*:  $(A * - B) = - (A * B)$   
 $\langle \text{proof} \rangle$

**end**

**lemma** *minus-mult-distrib-mat[algebra-simps]*: **fixes**  $A :: 'a :: \text{ring mat}$   
**assumes**  $m: A \in \text{carrier-mat } nr \ n \ B \in \text{carrier-mat } nr \ n \ C \in \text{carrier-mat } n \ nc$   
**shows**  $(A - B) * C = A * C - B * C$   
 $\langle \text{proof} \rangle$

**lemma** *minus-mult-distrib-mat-vec[algebra-simps]*: **assumes**  $A: (A :: 'a :: \text{ring$   
 $\text{mat}) \in \text{carrier-mat } nr \ nc$   
**and**  $B: B \in \text{carrier-mat } nr \ nc$   
**and**  $v: v \in \text{carrier-vec } nc$   
**shows**  $(A - B) *_v v = A *_v v - B *_v v$   
 $\langle \text{proof} \rangle$

**lemma** *mult-minus-distrib-mat-vec[algebra-simps]*: **assumes**  $A: (A :: 'a :: \text{ring$   
 $\text{mat}) \in \text{carrier-mat } nr \ nc$   
**and**  $v: v \in \text{carrier-vec } nc$   
**and**  $w: w \in \text{carrier-vec } nc$   
**shows**  $A *_v (v - w) = A *_v v - A *_v w$   
 $\langle \text{proof} \rangle$

**lemma** *mult-minus-distrib-mat[algebra-simps]*: **fixes**  $A :: 'a :: \text{ring mat}$

**assumes**  $m: A \in \text{carrier-mat } nr \ n \ B \in \text{carrier-mat } n \ nc \ C \in \text{carrier-mat } n \ nc$   
**shows**  $A * (B - C) = A * B - A * C$   
 $\langle \text{proof} \rangle$

**lemma** *uminus-mult-mat-vec*[simp]: **assumes**  $v: \text{dim-vec } v = \text{dim-col } (A :: 'a :: \text{ring mat})$   
**shows**  $- A *_v v = - (A *_v v)$   
 $\langle \text{proof} \rangle$

**lemma** *uminus-zero-vec-eq*: **assumes**  $v: (v :: 'a :: \text{group-add vec}) \in \text{carrier-vec } n$   
**shows**  $(- v = 0_v \ n) = (v = 0_v \ n)$   
 $\langle \text{proof} \rangle$

**lemma** *map-carrier-mat*[simp]:  
 $(\text{map-mat } f \ A \in \text{carrier-mat } nr \ nc) = (A \in \text{carrier-mat } nr \ nc)$   
 $\langle \text{proof} \rangle$

**lemma** *col-map-mat*[simp]:  
**assumes**  $j < \text{dim-col } A$  **shows**  $\text{col } (\text{map-mat } f \ A) \ j = \text{map-vec } f \ (\text{col } A \ j)$   
 $\langle \text{proof} \rangle$

**lemma** *scalar-vec-one*[simp]:  $1 \cdot_v (v :: 'a :: \text{semiring-1 vec}) = v$   
 $\langle \text{proof} \rangle$

**lemma** *scalar-prod-smult-right*[simp]:  
 $\text{dim-vec } w = \text{dim-vec } v \implies w \cdot (k \cdot_v v) = (k :: 'a :: \text{comm-semiring-0}) * (w \cdot v)$   
 $\langle \text{proof} \rangle$

**lemma** *scalar-prod-smult-left*[simp]:  
 $\text{dim-vec } w = \text{dim-vec } v \implies (k \cdot_v w) \cdot v = (k :: 'a :: \text{comm-semiring-0}) * (w \cdot v)$   
 $\langle \text{proof} \rangle$

**lemma** *mult-smult-distrib*: **assumes**  $A: A \in \text{carrier-mat } nr \ n$  **and**  $B: B \in \text{carrier-mat } n \ nc$   
**shows**  $A * (k \cdot_m B) = (k :: 'a :: \text{comm-semiring-0}) \cdot_m (A * B)$   
 $\langle \text{proof} \rangle$

**lemma** *add-smult-distrib-left-mat*: **assumes**  $A \in \text{carrier-mat } nr \ nc \ B \in \text{carrier-mat } nr \ nc$   
**shows**  $k \cdot_m (A + B) = (k :: 'a :: \text{semiring}) \cdot_m A + k \cdot_m B$   
 $\langle \text{proof} \rangle$

**lemma** *add-smult-distrib-right-mat*: **assumes**  $A \in \text{carrier-mat } nr \ nc$   
**shows**  $(k + l) \cdot_m A = (k :: 'a :: \text{semiring}) \cdot_m A + l \cdot_m A$   
 $\langle \text{proof} \rangle$

**lemma** *mult-smult-assoc-mat*: **assumes**  $A: A \in \text{carrier-mat } nr \ n$  **and**  $B: B \in \text{carrier-mat } n \ nc$   
**shows**  $(k \cdot_m A) * B = (k :: 'a :: \text{comm-semiring-0}) \cdot_m (A * B)$

*<proof>*

**definition** *similar-mat-wit* :: 'a :: semiring-1 mat  $\Rightarrow$  'a mat  $\Rightarrow$  'a mat  $\Rightarrow$  '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<sub>m</sub> n  $\wedge$  Q \* P = 1<sub>m</sub> n  $\wedge$   
A = P \* B \* Q)

**definition** *similar-mat* :: 'a :: semiring-1 mat  $\Rightarrow$  '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<sub>m</sub> n  $\wedge$  Q \* P = 1<sub>m</sub> n  $\wedge$  A = P \* B \* Q

*<proof>*

**lemma** *similar-matI*: **assumes** {A,B,P,Q}  $\subseteq$  carrier-mat n n P \* Q = 1<sub>m</sub> n Q \* P = 1<sub>m</sub> n A = P \* B \* Q

**shows** *similar-mat* A B *<proof>*

**fun** *pow-mat* :: 'a :: semiring-1 mat  $\Rightarrow$  nat  $\Rightarrow$  'a mat (**infixr**  $\hat{\ }_m$  75) **where**

A  $\hat{\ }_m$  0 = 1<sub>m</sub> (dim-row A)  
| A  $\hat{\ }_m$  (Suc k) = A  $\hat{\ }_m$  k \* A

**lemma** *pow-mat-dim[simp]*:

dim-row (A  $\hat{\ }_m$  k) = dim-row A  
dim-col (A  $\hat{\ }_m$  k) = (if k = 0 then dim-row A else dim-col A)

*<proof>*

**lemma** *pow-mat-dim-square[simp]*:

A  $\in$  carrier-mat n n  $\Longrightarrow$  dim-row (A  $\hat{\ }_m$  k) = n  
A  $\in$  carrier-mat n n  $\Longrightarrow$  dim-col (A  $\hat{\ }_m$  k) = n

*<proof>*

**lemma** *pow-carrier-mat[simp]*: A  $\in$  carrier-mat n n  $\Longrightarrow$  A  $\hat{\ }_m$  k  $\in$  carrier-mat n n

*<proof>*

**definition** *diag-mat* :: 'a mat  $\Rightarrow$  'a list **where**

*diag-mat* A = map ( $\lambda$  i. A \$\$\$ (i,i)) [0 ..< dim-row A]

**lemma** *prod-list-diag-prod*: prod-list (diag-mat A) = ( $\prod$  i = 0 ..< dim-row A. A \$\$\$ (i,i))

*<proof>*

**lemma** *diag-mat-transpose[simp]*: dim-row A = dim-col A  $\Longrightarrow$

*diag-mat* (transpose-mat A) = *diag-mat* A *<proof>*

**lemma** *diag-mat-zero[simp]*: *diag-mat* (0<sub>m</sub> n n) = replicate n 0

*<proof>*

**lemma** *diag-mat-one*[simp]:  $\text{diag-mat } (1_m \ n) = \text{replicate } n \ 1$   
*<proof>*

**lemma** *pow-mat-ring-pow*: **assumes**  $A: (A :: ('a :: \text{semiring-1})\text{mat}) \in \text{carrier-mat } n \ n$

**shows**  $A \hat{\ }_m \ k = A \ [\ ]_{\text{ring-mat TYPE('a) } n \ b \ k}$   
**(is**  $- = A \ [\ ]_{?C} \ k$ )

*<proof>*

**definition** *diagonal-mat* ::  $'a::\text{zero mat} \Rightarrow \text{bool}$  **where**

$\text{diagonal-mat } A \equiv \forall i < \text{dim-row } A. \forall j < \text{dim-col } A. i \neq j \longrightarrow A \ \$\$ (i,j) = 0$

**definition** (**in** *comm-monoid-add*) *sum-mat* ::  $'a \text{ mat} \Rightarrow 'a$  **where**

$\text{sum-mat } A = \text{sum } (\lambda ij. A \ \$\$ ij) (\{0 ..< \text{dim-row } A\} \times \{0 ..< \text{dim-col } A\})$

**lemma** *sum-mat-0*[simp]:  $\text{sum-mat } (0_m \ nr \ nc) = (0 :: 'a :: \text{comm-monoid-add})$   
*<proof>*

**lemma** *sum-mat-add*: **assumes**  $A: (A :: 'a :: \text{comm-monoid-add mat}) \in \text{carrier-mat } nr \ nc$  **and**  $B: B \in \text{carrier-mat } nr \ nc$

**shows**  $\text{sum-mat } (A + B) = \text{sum-mat } A + \text{sum-mat } B$

*<proof>*

### 4.3 Update Operators

**definition** *update-vec* ::  $'a \ \text{vec} \Rightarrow \text{nat} \Rightarrow 'a \Rightarrow 'a \ \text{vec} \ (- \ |_v \ - \mapsto - \ [60,61,62] \ 60)$   
**where**  $v \ |_v \ i \mapsto a = \text{vec } (\text{dim-vec } v) \ (\lambda i'. \text{if } i' = i \text{ then } a \ \text{else } v \ \$ i')$

**definition** *update-mat* ::  $'a \ \text{mat} \Rightarrow \text{nat} \times \text{nat} \Rightarrow 'a \Rightarrow 'a \ \text{mat} \ (- \ |_m \ - \mapsto - \ [60,61,62] \ 60)$

**where**  $A \ |_m \ ij \mapsto a = \text{mat } (\text{dim-row } A) \ (\text{dim-col } A) \ (\lambda ij'. \text{if } ij' = ij \text{ then } a \ \text{else } A \ \$\$ ij')$

**lemma** *dim-update-vec*[simp]:

$\text{dim-vec } (v \ |_v \ i \mapsto a) = \text{dim-vec } v$  *<proof>*

**lemma** *index-update-vec1*[simp]:

**assumes**  $i < \text{dim-vec } v$  **shows**  $(v \ |_v \ i \mapsto a) \ \$ i = a$   
*<proof>*

**lemma** *index-update-vec2*[simp]:

**assumes**  $i' \neq i$  **shows**  $(v \ |_v \ i \mapsto a) \ \$ i' = v \ \$ i'$   
*<proof>*

**lemma** *dim-update-mat*[simp]:

$\text{dim-row } (A \ |_m \ ij \mapsto a) = \text{dim-row } A$   
 $\text{dim-col } (A \ |_m \ ij \mapsto a) = \text{dim-col } A$  *<proof>*

**lemma** *index-update-mat1*[simp]:

**assumes**  $i < \dim\text{-row } A$   $j < \dim\text{-col } A$  **shows**  $(A \mid_m (i,j) \mapsto a) \text{ $$ } (i,j) = a$   
 ⟨proof⟩

**lemma** *index-update-mat2*[simp]:

**assumes**  $i': i' < \dim\text{-row } A$  **and**  $j': j' < \dim\text{-col } A$  **and**  $\text{neq}: (i',j') \neq ij$   
**shows**  $(A \mid_m ij \mapsto a) \text{ $$ } (i',j') = A \text{ $$ } (i',j')$   
 ⟨proof⟩

#### 4.4 Block Vectors and Matrices

**definition** *append-vec* ::  $'a \text{ vec} \Rightarrow 'a \text{ vec} \Rightarrow 'a \text{ vec}$  (**infixr**  $@_v$  65) **where**

$v @_v w \equiv \text{let } n = \dim\text{-vec } v; m = \dim\text{-vec } w \text{ in}$   
 $\text{vec } (n + m) (\lambda i. \text{if } i < n \text{ then } v \$ i \text{ else } w \$ (i - n))$

**lemma** *index-append-vec*[simp]:  $i < \dim\text{-vec } v + \dim\text{-vec } w$

$\implies (v @_v w) \$ i = (\text{if } i < \dim\text{-vec } v \text{ then } v \$ i \text{ else } w \$ (i - \dim\text{-vec } v))$   
 $\dim\text{-vec } (v @_v w) = \dim\text{-vec } v + \dim\text{-vec } w$   
 ⟨proof⟩

**lemma** *append-carrier-vec*[simp,intro]:

$v \in \text{carrier-vec } n1 \implies w \in \text{carrier-vec } n2 \implies v @_v w \in \text{carrier-vec } (n1 + n2)$   
 ⟨proof⟩

**lemma** *scalar-prod-append*: **assumes**  $v1 \in \text{carrier-vec } n1$   $v2 \in \text{carrier-vec } n2$

$w1 \in \text{carrier-vec } n1$   $w2 \in \text{carrier-vec } n2$   
**shows**  $(v1 @_v v2) \cdot (w1 @_v w2) = v1 \cdot w1 + v2 \cdot w2$   
 ⟨proof⟩

**definition** *vec-first*  $v \ n \equiv \text{vec } n (\lambda i. v \$ i)$

**definition** *vec-last*  $v \ n \equiv \text{vec } n (\lambda i. v \$ (\dim\text{-vec } v - n + i))$

**lemma** *dim-vec-first*[simp]:  $\dim\text{-vec } (\text{vec-first } v \ n) = n$  ⟨proof⟩

**lemma** *dim-vec-last*[simp]:  $\dim\text{-vec } (\text{vec-last } v \ n) = n$  ⟨proof⟩

**lemma** *vec-first-carrier*[simp]:  $\text{vec-first } v \ n \in \text{carrier-vec } n$  ⟨proof⟩

**lemma** *vec-last-carrier*[simp]:  $\text{vec-last } v \ n \in \text{carrier-vec } n$  ⟨proof⟩

**lemma** *vec-first-last-append*[simp]:

**assumes**  $v \in \text{carrier-vec } (n+m)$  **shows**  $\text{vec-first } v \ n @_v \text{vec-last } v \ m = v$   
 ⟨proof⟩

**lemma** *append-vec-le*: **assumes**  $v \in \text{carrier-vec } n$  **and**  $w: w \in \text{carrier-vec } n$

**shows**  $v @_v v' \leq w @_v w' \longleftrightarrow v \leq w \wedge v' \leq w'$   
 ⟨proof⟩

**lemma** *all-vec-append*:  $(\forall x \in \text{carrier-vec } (n + m). P \ x) \longleftrightarrow (\forall x1 \in \text{carrier-vec } n. \forall x2 \in \text{carrier-vec } m. P \ (x1 @_v x2))$



*<proof>*

**definition** *four-block-mat* :: 'a mat  $\Rightarrow$  'a mat  $\Rightarrow$  'a mat  $\Rightarrow$  'a mat  $\Rightarrow$  'a mat **where**

*four-block-mat* A B C D =

(let nra = dim-row A; nrd = dim-row D;

nca = dim-col A; ncd = dim-col D

in

mat (nra + nrd) (nca + ncd) ( $\lambda$  (i,j). if i < nra then

if j < nca then A \$\$ (i,j) else B \$\$ (i,j - nca)

else if j < nca then C \$\$ (i - nra, j) else D \$\$ (i - nra, j - nca)))

**lemma** *index-mat-four-block[simp]*:

$i < \text{dim-row } A + \text{dim-row } D \implies j < \text{dim-col } A + \text{dim-col } D \implies \text{four-block-mat}$

$A \ B \ C \ D \ \$\$ (i,j)$

= (if i < dim-row A then

if j < dim-col A then A \$\$ (i,j) else B \$\$ (i,j - dim-col A)

else if j < dim-col A then C \$\$ (i - dim-row A, j) else D \$\$ (i - dim-row A, j - dim-col A))

dim-row (four-block-mat A B C D) = dim-row A + dim-row D

dim-col (four-block-mat A B C D) = dim-col A + dim-col D

*<proof>*

**lemma** *four-block-carrier-mat[simp]*:

$A \in \text{carrier-mat } nr1 \ nc1 \implies D \in \text{carrier-mat } nr2 \ nc2 \implies$

$\text{four-block-mat } A \ B \ C \ D \in \text{carrier-mat } (nr1 + nr2) \ (nc1 + nc2)$

*<proof>*

**lemma** *cong-four-block-mat*:  $A1 = B1 \implies A2 = B2 \implies A3 = B3 \implies A4 = B4 \implies$

$\text{four-block-mat } A1 \ A2 \ A3 \ A4 = \text{four-block-mat } B1 \ B2 \ B3 \ B4$  *<proof>*

**lemma** *four-block-one-mat[simp]*:

$\text{four-block-mat } (1_m \ n1) \ (0_m \ n1 \ n2) \ (0_m \ n2 \ n1) \ (1_m \ n2) = 1_m \ (n1 + n2)$

*<proof>*

**lemma** *four-block-zero-mat[simp]*:

$\text{four-block-mat } (0_m \ nr1 \ nc1) \ (0_m \ nr1 \ nc2) \ (0_m \ nr2 \ nc1) \ (0_m \ nr2 \ nc2) = 0_m$

$(nr1 + nr2) \ (nc1 + nc2)$

*<proof>*

**lemma** *row-four-block-mat*:

**assumes** c:  $A \in \text{carrier-mat } nr1 \ nc1 \ B \in \text{carrier-mat } nr1 \ nc2$

$C \in \text{carrier-mat } nr2 \ nc1 \ D \in \text{carrier-mat } nr2 \ nc2$

**shows**

$i < nr1 \implies \text{row } (\text{four-block-mat } A \ B \ C \ D) \ i = \text{row } A \ i \ @_v \ \text{row } B \ i$  (**is** -  $\implies$  ?AB)

$\neg i < nr1 \implies i < nr1 + nr2 \implies \text{row } (\text{four-block-mat } A \ B \ C \ D) \ i = \text{row } C \ (i$

–  $nr1$ )  $@_v$  row  $D$  ( $i - nr1$ )  
 (is  $- \implies - \implies ?CD$ )  
 $\langle proof \rangle$

**lemma** *col-four-block-mat*:

**assumes**  $c$ :  $A \in \text{carrier-mat } nr1 \ nc1$   $B \in \text{carrier-mat } nr1 \ nc2$   
 $C \in \text{carrier-mat } nr2 \ nc1$   $D \in \text{carrier-mat } nr2 \ nc2$   
**shows**  
 $j < nc1 \implies \text{col } (\text{four-block-mat } A \ B \ C \ D) \ j = \text{col } A \ j \ @_v \ \text{col } C \ j$  (is  $- \implies ?AC$ )  
 $\neg j < nc1 \implies j < nc1 + nc2 \implies \text{col } (\text{four-block-mat } A \ B \ C \ D) \ j = \text{col } B \ (j - nc1) \ @_v \ \text{col } D \ (j - nc1)$   
 (is  $- \implies - \implies ?BD$ )  
 $\langle proof \rangle$

**lemma** *mult-four-block-mat*: **assumes**

$c1$ :  $A1 \in \text{carrier-mat } nr1 \ n1$   $B1 \in \text{carrier-mat } nr1 \ n2$   $C1 \in \text{carrier-mat } nr2 \ n1$   
 $D1 \in \text{carrier-mat } nr2 \ n2$  **and**  
 $c2$ :  $A2 \in \text{carrier-mat } n1 \ nc1$   $B2 \in \text{carrier-mat } n1 \ nc2$   $C2 \in \text{carrier-mat } n2 \ nc1$   
 $D2 \in \text{carrier-mat } n2 \ nc2$   
**shows**  $\text{four-block-mat } A1 \ B1 \ C1 \ D1 \ * \ \text{four-block-mat } A2 \ B2 \ C2 \ D2$   
 $= \text{four-block-mat } (A1 \ * \ A2 + B1 \ * \ C2) \ (A1 \ * \ B2 + B1 \ * \ D2)$   
 $(C1 \ * \ A2 + D1 \ * \ C2) \ (C1 \ * \ B2 + D1 \ * \ D2)$  (is  $?M1 \ * \ ?M2 = -$ )  
 $\langle proof \rangle$

**definition** *append-rows* ::  $'a :: \text{zero mat} \implies 'a \ \text{mat} \implies 'a \ \text{mat}$  (**infixr**  $@_r$  65) **where**  
 $A \ @_r \ B = \text{four-block-mat } A \ (0_m \ (\text{dim-row } A) \ 0) \ B \ (0_m \ (\text{dim-row } B) \ 0)$

**lemma** *carrier-append-rows[simp,intro]*:  $A \in \text{carrier-mat } nr1 \ nc \implies B \in \text{carrier-mat } nr2 \ nc \implies$

$A \ @_r \ B \in \text{carrier-mat } (nr1 + nr2) \ nc$   
 $\langle proof \rangle$

**lemma** *col-mult2[simp]*:

**assumes**  $A$ :  $A : \text{carrier-mat } nr \ n$   
**and**  $B$ :  $B : \text{carrier-mat } n \ nc$   
**and**  $j$ :  $j < nc$   
**shows**  $\text{col } (A \ * \ B) \ j = A \ *_v \ \text{col } B \ j$   
 $\langle proof \rangle$

**lemma** *mat-vec-as-mat-mat-mult*: **assumes**  $A$ :  $A \in \text{carrier-mat } nr \ nc$

**and**  $v$ :  $v \in \text{carrier-vec } nc$   
**shows**  $A \ *_v \ v = \text{col } (A \ * \ \text{mat-of-cols } nc \ [v]) \ 0$   
 $\langle proof \rangle$

**lemma** *mat-mult-append*: **assumes**  $A$ :  $A \in \text{carrier-mat } nr1 \ nc$

**and**  $B$ :  $B \in \text{carrier-mat } nr2 \ nc$   
**and**  $v$ :  $v \in \text{carrier-vec } nc$   
**shows**  $(A \ @_r \ B) \ *_v \ v = (A \ *_v \ v) \ @_v \ (B \ *_v \ v)$   
 $\langle proof \rangle$

**lemma** *append-rows-le*: **assumes**  $A: A \in \text{carrier-mat } nr1 \ nc$   
**and**  $B: B \in \text{carrier-mat } nr2 \ nc$   
**and**  $a: a \in \text{carrier-vec } nr1$   
**and**  $v: v \in \text{carrier-vec } nc$   
**shows**  $(A @_r B) *_v v \leq (a @_v b) \iff A *_v v \leq a \wedge B *_v v \leq b$   
 $\langle \text{proof} \rangle$

**lemma** *elements-four-block-mat*:  
**assumes**  $c: A \in \text{carrier-mat } nr1 \ nc1 \ B \in \text{carrier-mat } nr1 \ nc2$   
 $C \in \text{carrier-mat } nr2 \ nc1 \ D \in \text{carrier-mat } nr2 \ nc2$   
**shows**  
 $\text{elements-mat } (\text{four-block-mat } A \ B \ C \ D) \subseteq$   
 $\text{elements-mat } A \cup \text{elements-mat } B \cup \text{elements-mat } C \cup \text{elements-mat } D$   
**(is**  $\text{elements-mat } ?\text{four} \subseteq -)$   
 $\langle \text{proof} \rangle$

**lemma** *assoc-four-block-mat*: **fixes**  $FB :: 'a \ \text{mat} \Rightarrow 'a \ \text{mat} \Rightarrow 'a :: \text{zero mat}$   
**defines**  $FB: FB \equiv \lambda \ Bb \ Cc. \ \text{four-block-mat } Bb \ (0_m \ (\text{dim-row } Bb) \ (\text{dim-col } Cc))$   
 $(0_m \ (\text{dim-row } Cc) \ (\text{dim-col } Bb)) \ Cc$   
**shows**  $FB \ A \ (FB \ B \ C) = FB \ (FB \ A \ B) \ C$  **(is**  $?L = ?R)$   
 $\langle \text{proof} \rangle$

**definition** *split-block*  $:: 'a \ \text{mat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow ('a \ \text{mat} \times 'a \ \text{mat} \times 'a \ \text{mat} \times 'a \ \text{mat})$   
**where**  $\text{split-block } A \ sr \ sc = (\text{let}$   
 $nr = \text{dim-row } A; \ nc = \text{dim-col } A;$   
 $nr2 = nr - sr; \ nc2 = nc - sc;$   
 $A1 = \text{mat } sr \ sc \ (\lambda \ ij. \ A \ \$$ \ ij);$   
 $A2 = \text{mat } sr \ nc2 \ (\lambda \ (i,j). \ A \ \$$ \ (i,j+sc));$   
 $A3 = \text{mat } nr2 \ sc \ (\lambda \ (i,j). \ A \ \$$ \ (i+sr,j));$   
 $A4 = \text{mat } nr2 \ nc2 \ (\lambda \ (i,j). \ A \ \$$ \ (i+sr,j+sc))$   
 $\text{in } (A1, A2, A3, A4))$

**lemma** *split-block*: **assumes**  $\text{res: split-block } A \ sr1 \ sc1 = (A1, A2, A3, A4)$   
**and**  $\text{dims: dim-row } A = sr1 + sr2 \ \text{dim-col } A = sc1 + sc2$   
**shows**  $A1 \in \text{carrier-mat } sr1 \ sc1 \ A2 \in \text{carrier-mat } sr1 \ sc2$   
 $A3 \in \text{carrier-mat } sr2 \ sc1 \ A4 \in \text{carrier-mat } sr2 \ sc2$   
 $A = \text{four-block-mat } A1 \ A2 \ A3 \ A4$   
 $\langle \text{proof} \rangle$

Using *four-block-mat* we define block-diagonal matrices.

**fun** *diag-block-mat*  $:: 'a :: \text{zero mat list} \Rightarrow 'a \ \text{mat}$  **where**  
 $\text{diag-block-mat } [] = 0_m \ 0 \ 0$   
 $| \text{diag-block-mat } (A \ \# \ As) = (\text{let}$   
 $B = \text{diag-block-mat } As$   
 $\text{in } \text{four-block-mat } A \ (0_m \ (\text{dim-row } A) \ (\text{dim-col } B)) \ (0_m \ (\text{dim-row } B) \ (\text{dim-col } A)) \ B)$

**lemma** *dim-diag-block-mat*:

$dim\text{-row } (diag\text{-block-mat } As) = sum\text{-list } (map\ dim\text{-row } As) \text{ (is ?row)}$

$dim\text{-col } (diag\text{-block-mat } As) = sum\text{-list } (map\ dim\text{-col } As) \text{ (is ?col)}$

$\langle proof \rangle$

**lemma** *diag-block-mat-singleton[simp]*:  $diag\text{-block-mat } [A] = A$

$\langle proof \rangle$

**lemma** *diag-block-mat-append*:  $diag\text{-block-mat } (As @ Bs) =$

$(let\ A = diag\text{-block-mat } As; B = diag\text{-block-mat } Bs$

$in\ four\text{-block-mat } A\ (0_m\ (dim\text{-row } A)\ (dim\text{-col } B))\ (0_m\ (dim\text{-row } B)\ (dim\text{-col } A))\ B)$

$\langle proof \rangle$

**lemma** *diag-block-mat-last*:  $diag\text{-block-mat } (As @ [B]) =$

$(let\ A = diag\text{-block-mat } As$

$in\ four\text{-block-mat } A\ (0_m\ (dim\text{-row } A)\ (dim\text{-col } B))\ (0_m\ (dim\text{-row } B)\ (dim\text{-col } A))\ B)$

$\langle proof \rangle$

**lemma** *diag-block-mat-square*:

$Ball\ (set\ As)\ square\text{-mat} \implies square\text{-mat } (diag\text{-block-mat } As)$

$\langle proof \rangle$

**lemma** *diag-block-one-mat[simp]*:

$diag\text{-block-mat } (map\ (\lambda A.\ 1_m\ (dim\text{-row } A))\ As) = (1_m\ (sum\text{-list } (map\ dim\text{-row } As)))$

$\langle proof \rangle$

**lemma** *elements-diag-block-mat*:

$elements\text{-mat } (diag\text{-block-mat } As) \subseteq \{0\} \cup \bigcup (set\ (map\ elements\text{-mat } As))$

$\langle proof \rangle$

**lemma** *diag-block-pow-mat*: **assumes**  $sq: Ball\ (set\ As)\ square\text{-mat}$

**shows**  $diag\text{-block-mat } As \hat{=}^m n = diag\text{-block-mat } (map\ (\lambda A.\ A \hat{=}^m n)\ As) \text{ (is ?As } \hat{=}^m - = -)$

$\langle proof \rangle$

**lemma** *diag-block-upper-triangular*: **assumes**

$\bigwedge A\ i\ j.\ A \in set\ As \implies j < i \implies i < dim\text{-row } A \implies A \$\$ (i,j) = 0$

**and**  $Ball\ (set\ As)\ square\text{-mat}$

**and**  $j < i < dim\text{-row } (diag\text{-block-mat } As)$

**shows**  $diag\text{-block-mat } As \$\$ (i,j) = 0$

$\langle proof \rangle$

**lemma** *smult-four-block-mat*: **assumes**  $c: A \in carrier\text{-mat } nr1\ nc1\ B \in carrier\text{-mat } nr1\ nc2$

$C \in \text{carrier-mat } nr2 \ nc1 \ D \in \text{carrier-mat } nr2 \ nc2$   
**shows**  $a \cdot_m \text{four-block-mat } A \ B \ C \ D = \text{four-block-mat } (a \cdot_m A) \ (a \cdot_m B) \ (a \cdot_m C) \ (a \cdot_m D)$   
 ⟨proof⟩

**lemma map-four-block-mat: assumes**  $c: A \in \text{carrier-mat } nr1 \ nc1 \ B \in \text{carrier-mat } nr1 \ nc2$   
 $C \in \text{carrier-mat } nr2 \ nc1 \ D \in \text{carrier-mat } nr2 \ nc2$   
**shows**  $\text{map-mat } f \ (\text{four-block-mat } A \ B \ C \ D) = \text{four-block-mat } (\text{map-mat } f \ A) \ (\text{map-mat } f \ B) \ (\text{map-mat } f \ C) \ (\text{map-mat } f \ D)$   
 ⟨proof⟩

**lemma add-four-block-mat: assumes**  
 $c1: A1 \in \text{carrier-mat } nr1 \ nc1 \ B1 \in \text{carrier-mat } nr1 \ nc2 \ C1 \in \text{carrier-mat } nr2 \ nc1 \ D1 \in \text{carrier-mat } nr2 \ nc2$  **and**  
 $c2: A2 \in \text{carrier-mat } nr1 \ nc1 \ B2 \in \text{carrier-mat } nr1 \ nc2 \ C2 \in \text{carrier-mat } nr2 \ nc1 \ D2 \in \text{carrier-mat } nr2 \ nc2$   
**shows**  $\text{four-block-mat } A1 \ B1 \ C1 \ D1 + \text{four-block-mat } A2 \ B2 \ C2 \ D2 = \text{four-block-mat } (A1 + A2) \ (B1 + B2) \ (C1 + C2) \ (D1 + D2)$   
 ⟨proof⟩

**lemma diag-four-block-mat: assumes**  $c: A \in \text{carrier-mat } n1 \ n1$   
 $D \in \text{carrier-mat } n2 \ n2$   
**shows**  $\text{diag-mat } (\text{four-block-mat } A \ B \ C \ D) = \text{diag-mat } A \ @ \ \text{diag-mat } D$   
 ⟨proof⟩

**definition mk-diagonal**  $:: 'a::\text{zero list} \Rightarrow 'a \ \text{mat}$   
**where**  $\text{mk-diagonal } as = \text{diag-block-mat } (\text{map } (\lambda a. \text{mat } (\text{Suc } 0) \ (\text{Suc } 0) \ (\lambda -. a)) \ as)$

**lemma mk-diagonal-dim:**  
 $\text{dim-row } (\text{mk-diagonal } as) = \text{length } as \ \text{dim-col } (\text{mk-diagonal } as) = \text{length } as$   
 ⟨proof⟩

**lemma mk-diagonal-diagonal:**  $\text{diagonal-mat } (\text{mk-diagonal } as)$   
 ⟨proof⟩

**definition orthogonal-mat**  $:: 'a::\text{semiring-0} \ \text{mat} \Rightarrow \text{bool}$   
**where**  $\text{orthogonal-mat } A \equiv$   
 $\text{let } B = \text{transpose-mat } A * A \ \text{in}$   
 $\text{diagonal-mat } B \wedge (\forall i < \text{dim-col } A. B \ \$\$ \ (i, i) \neq 0)$

**lemma orthogonal-matD[elim]:**  
 $\text{orthogonal-mat } A \Longrightarrow$   
 $i < \text{dim-col } A \Longrightarrow j < \text{dim-col } A \Longrightarrow (\text{col } A \ i \cdot \text{col } A \ j = 0) = (i \neq j)$   
 ⟨proof⟩

**lemma orthogonal-matI[intro]:**

$(\bigwedge i j. i < \dim\text{-col } A \implies j < \dim\text{-col } A \implies (\text{col } A \ i \cdot \text{col } A \ j = 0) = (i \neq j))$   
 $\implies$   
*orthogonal-mat* *A*  
 ⟨*proof*⟩

**definition** *orthogonal* :: 'a::semiring-0 *vec list*  $\Rightarrow$  *bool*  
**where** *orthogonal vs*  $\equiv$   
 $\forall i j. i < \text{length } vs \longrightarrow j < \text{length } vs \longrightarrow$   
 $(vs \ ! \ i \cdot vs \ ! \ j = 0) = (i \neq j)$

**lemma** *orthogonalD[elim]*:  
*orthogonal vs*  $\implies i < \text{length } vs \implies j < \text{length } vs \implies$   
 $(\text{nth } vs \ i \cdot \text{nth } vs \ j = 0) = (i \neq j)$   
 ⟨*proof*⟩

**lemma** *orthogonalI[intro]*:  
 $(\bigwedge i j. i < \text{length } vs \implies j < \text{length } vs \implies (\text{nth } vs \ i \cdot \text{nth } vs \ j = 0) = (i \neq j))$   
 $\implies$   
*orthogonal vs*  
 ⟨*proof*⟩

**lemma** *transpose-four-block-mat*: **assumes** \*: *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** *transpose-mat (four-block-mat A B C D) =*  
*four-block-mat (transpose-mat A) (transpose-mat C) (transpose-mat B) (transpose-mat D)*  
 ⟨*proof*⟩

**lemma** *zero-transpose-mat[simp]*: *transpose-mat (0<sub>m</sub> n m) = (0<sub>m</sub> m n)*  
 ⟨*proof*⟩

**lemma** *upper-triangular-four-block*: **assumes** *AD*: *A*  $\in$  *carrier-mat n n* *D*  $\in$  *carrier-mat m m*  
**and** *ut*: *upper-triangular A upper-triangular D*  
**shows** *upper-triangular (four-block-mat A B (0<sub>m</sub> m n) D)*  
 ⟨*proof*⟩

**lemma** *pow-four-block-mat*: **assumes** *A*: *A*  $\in$  *carrier-mat n n*  
**and** *B*: *B*  $\in$  *carrier-mat m m*  
**shows** *(four-block-mat A (0<sub>m</sub> n m) (0<sub>m</sub> m n) B)  $\hat{\ }_m$  k =*  
*four-block-mat (A  $\hat{\ }_m$  k) (0<sub>m</sub> n m) (0<sub>m</sub> m n) (B  $\hat{\ }_m$  k)*  
 ⟨*proof*⟩

**lemma** *uminus-scalar-prod*:  
**assumes** [*simp*]: *v* : *carrier-vec n* *w* : *carrier-vec n*  
**shows**  $- ((v::'a::field \text{vec}) \cdot w) = (- v) \cdot w$   
 ⟨*proof*⟩

**lemma** *append-vec-eq*:

**assumes** [*simp*]:  $v : \text{carrier-vec } n \ v' : \text{carrier-vec } n$   
**shows** [*simp*]:  $v @_v w = v' @_v w' \longleftrightarrow v = v' \wedge w = w'$  (**is** ?*L*  $\longleftrightarrow$  ?*R*)  
 <proof>

**lemma** *append-vec-add*:

**assumes** [*simp*]:  $v : \text{carrier-vec } n \ v' : \text{carrier-vec } n$   
**and** [*simp*]:  $w : \text{carrier-vec } m \ w' : \text{carrier-vec } m$   
**shows**  $(v @_v w) + (v' @_v w') = (v + v') @_v (w + w')$  (**is** ?*L* = ?*R*)  
 <proof>

**lemma** *mult-mat-vec-split*:

**assumes** *A*:  $A : \text{carrier-mat } n \ n$   
**and** *D*:  $D : \text{carrier-mat } m \ m$   
**and** *a*:  $a : \text{carrier-vec } n$   
**and** *d*:  $d : \text{carrier-vec } m$   
**shows** *four-block-mat*  $A (0_m \ n \ m) (0_m \ m \ n) D *_v (a @_v d) = A *_v a @_v D *_v d$   
 (**is** ?*A00D* \*\_v - = ?*r*)  
 <proof>

**lemma** *similar-mat-witI*: **assumes**  $P * Q = 1_m \ n \ Q * P = 1_m \ n \ A = P * B * Q$   
 $A \in \text{carrier-mat } n \ n \ B \in \text{carrier-mat } n \ n \ P \in \text{carrier-mat } n \ n \ Q \in \text{carrier-mat } n \ n$   
**shows** *similar-mat-wit*  $A \ B \ P \ Q$  <proof>

**lemma** *similar-mat-witD*: **assumes**  $n = \text{dim-row } A$  *similar-mat-wit*  $A \ B \ P \ Q$   
**shows**  $P * Q = 1_m \ n \ Q * P = 1_m \ n \ A = P * B * Q$   
 $A \in \text{carrier-mat } n \ n \ B \in \text{carrier-mat } n \ n \ P \in \text{carrier-mat } n \ n \ Q \in \text{carrier-mat } n \ n$   
 <proof>

**lemma** *similar-mat-witD2*: **assumes**  $A \in \text{carrier-mat } n \ m$  *similar-mat-wit*  $A \ B \ P \ Q$   
**shows**  $P * Q = 1_m \ n \ Q * P = 1_m \ n \ A = P * B * Q$   
 $A \in \text{carrier-mat } n \ n \ B \in \text{carrier-mat } n \ n \ P \in \text{carrier-mat } n \ n \ Q \in \text{carrier-mat } n \ n$   
 <proof>

**lemma** *similar-mat-wit-sym*: **assumes** *sim*: *similar-mat-wit*  $A \ B \ P \ Q$   
**shows** *similar-mat-wit*  $B \ A \ Q \ P$   
 <proof>

**lemma** *similar-mat-wit-refl*: **assumes** *A*:  $A \in \text{carrier-mat } n \ n$   
**shows** *similar-mat-wit*  $A \ A (1_m \ n) (1_m \ n)$

*<proof>*

**lemma** *similar-mat-wit-trans*: **assumes** *AB*: *similar-mat-wit A B P Q*  
**and** *BC*: *similar-mat-wit B C P' Q'*  
**shows** *similar-mat-wit A C (P \* P') (Q' \* Q)*  
*<proof>*

**lemma** *similar-mat-refl*:  $A \in \text{carrier-mat } n \ n \implies \text{similar-mat } A \ A$   
*<proof>*

**lemma** *similar-mat-trans*:  $\text{similar-mat } A \ B \implies \text{similar-mat } B \ C \implies \text{similar-mat } A \ C$   
*<proof>*

**lemma** *similar-mat-sym*:  $\text{similar-mat } A \ B \implies \text{similar-mat } B \ A$   
*<proof>*

**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*:  $URA = (P1 * UR * Q2)$   
  **and** *LLA*:  $LLA = (P2 * LL * Q1)$   
  **and** *A1*:  $A1 \in \text{carrier-mat } n \ n$   
  **and** *A2*:  $A2 \in \text{carrier-mat } m \ m$   
  **and** *LL*:  $LL \in \text{carrier-mat } m \ n$   
  **and** *UR*:  $UR \in \text{carrier-mat } n \ m$   
**shows** *similar-mat-wit (four-block-mat A1 URA LLA A2) (four-block-mat B1 UR LL B2)*  
  (*four-block-mat P1 (0<sub>m</sub> n m) (0<sub>m</sub> m n) P2) (four-block-mat Q1 (0<sub>m</sub> n m) (0<sub>m</sub> m n) Q2)*  
  (**is similar-mat-wit ?A ?B ?P ?Q**)  
*<proof>*

**lemma** *similar-mat-four-block-0-ex*: **assumes**  
  1: *similar-mat A1 B1*  
  **and** 2: *similar-mat A2 B2*  
  **and** *A0*:  $A0 \in \text{carrier-mat } n \ m$   
  **and** *A1*:  $A1 \in \text{carrier-mat } n \ n$   
  **and** *A2*:  $A2 \in \text{carrier-mat } m \ m$   
**shows**  $\exists B0. B0 \in \text{carrier-mat } n \ m \wedge \text{similar-mat (four-block-mat A1 A0 (0<sub>m</sub> m n) A2)$   
  (*four-block-mat B1 B0 (0<sub>m</sub> m n) B2*)  
*<proof>*

**lemma** *similar-mat-four-block-0-0*: **assumes**  
  1: *similar-mat A1 B1*  
  **and** 2: *similar-mat A2 B2*  
  **and** *A1*:  $A1 \in \text{carrier-mat } n \ n$



**and**  $A2$ :  $A2 \in \text{carrier-mat } m \ m$   
**shows**  $\text{similar-mat } (\text{four-block-mat } A1 \ (0_m \ n \ m) \ (0_m \ m \ n) \ A2)$   
 $(\text{four-block-mat } B1 \ (0_m \ n \ m) \ (0_m \ m \ n) \ B2)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{similar-diag-mat-block-mat}$ : **assumes**  $\bigwedge A \ B. (A,B) \in \text{set } Ms \implies \text{similar-mat } A \ B$   
**shows**  $\text{similar-mat } (\text{diag-block-mat } (\text{map fst } Ms)) \ (\text{diag-block-mat } (\text{map snd } Ms))$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{similar-mat-wit-pow}$ : **assumes**  $\text{wit: similar-mat-wit } A \ B \ P \ Q$   
**shows**  $\text{similar-mat-wit } (A \ \widehat{\ }_m \ k) \ (B \ \widehat{\ }_m \ k) \ P \ Q$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{similar-mat-wit-pow-id}$ :  $\text{similar-mat-wit } A \ B \ P \ Q \implies A \ \widehat{\ }_m \ k = P * B \ \widehat{\ }_m \ k * Q$   
 $\langle \text{proof} \rangle$

## 4.5 Homomorphism properties

**context**  $\text{semiring-hom}$

**begin**

**abbreviation**  $\text{mat-hom} :: 'a \ \text{mat} \Rightarrow 'b \ \text{mat} \ (\text{mat}_h)$   
**where**  $\text{mat}_h \equiv \text{map-mat } \text{hom}$

**abbreviation**  $\text{vec-hom} :: 'a \ \text{vec} \Rightarrow 'b \ \text{vec} \ (\text{vec}_h)$   
**where**  $\text{vec}_h \equiv \text{map-vec } \text{hom}$

**lemma**  $\text{vec-hom-zero}$ :  $\text{vec}_h \ (0_v \ n) = 0_v \ n$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{mat-hom-one}$ :  $\text{mat}_h \ (1_m \ n) = 1_m \ n$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{mat-hom-mult}$ : **assumes**  $A$ :  $A \in \text{carrier-mat } nr \ n$  **and**  $B$ :  $B \in \text{carrier-mat } n \ nc$   
**shows**  $\text{mat}_h \ (A * B) = \text{mat}_h \ A * \text{mat}_h \ B$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{mult-mat-vec-hom}$ : **assumes**  $A$ :  $A \in \text{carrier-mat } nr \ n$  **and**  $v$ :  $v \in \text{carrier-vec } n$   
**shows**  $\text{vec}_h \ (A *_v \ v) = \text{mat}_h \ A *_v \ \text{vec}_h \ v$   
 $\langle \text{proof} \rangle$   
**end**

**lemma**  $\text{vec-eq-iff}$ :  $(x = y) = (\text{dim-vec } x = \text{dim-vec } y \wedge (\forall i < \text{dim-vec } y. x \$ i = y \$ i))$  (**is**  $?l = ?r$ )  
 $\langle \text{proof} \rangle$

**lemma** *mat-eq-iff*:  $(x = y) = (dim\text{-row } x = dim\text{-row } y \wedge dim\text{-col } x = dim\text{-col } y \wedge (\forall i j. i < dim\text{-row } y \longrightarrow j < dim\text{-col } y \longrightarrow x \text{ \$(i,j) } = y \text{ \$(i,j)}))$  (**is** ?l = ?r)  
 ⟨proof⟩

**lemma** (**in** *inj-semiring-hom*) *vec-hom-zero-iff*[simp]:  $(vec_h x = 0_v n) = (x = 0_v n)$   
 ⟨proof⟩

**lemma** (**in** *inj-semiring-hom*) *mat-hom-inj*:  $mat_h A = mat_h B \implies A = B$   
 ⟨proof⟩

**lemma** (**in** *inj-semiring-hom*) *vec-hom-inj*:  $vec_h v = vec_h w \implies v = w$   
 ⟨proof⟩

**lemma** (**in** *semiring-hom*) *mat-hom-pow*: **assumes**  $A \in carrier\text{-mat } n \ n$   
**shows**  $mat_h (A \hat{\ }_m k) = (mat_h A) \hat{\ }_m k$   
 ⟨proof⟩

**lemma** (**in** *semiring-hom*) *hom-sum-mat*:  $hom (sum\text{-mat } A) = sum\text{-mat } (mat_h A)$   
 ⟨proof⟩

**lemma** (**in** *semiring-hom*) *vec-hom-smult*:  $vec_h (ev \cdot_v v) = hom \ ev \cdot_v \ vec_h v$   
 ⟨proof⟩

**lemma** *minus-scalar-prod-distrib*: **fixes**  $v_1 :: 'a :: ring \ vec$   
**assumes**  $v: v_1 \in carrier\text{-vec } n \ v_2 \in carrier\text{-vec } n \ v_3 \in carrier\text{-vec } n$   
**shows**  $(v_1 - v_2) \cdot v_3 = v_1 \cdot v_3 - v_2 \cdot v_3$   
 ⟨proof⟩

**lemma** *scalar-prod-minus-distrib*: **fixes**  $v_1 :: 'a :: ring \ vec$   
**assumes**  $v: v_1 \in carrier\text{-vec } n \ v_2 \in carrier\text{-vec } n \ v_3 \in carrier\text{-vec } n$   
**shows**  $v_1 \cdot (v_2 - v_3) = v_1 \cdot v_2 - v_1 \cdot v_3$   
 ⟨proof⟩

**lemma** *uminus-add-minus-vec*:  
**assumes**  $l \in carrier\text{-vec } n \ r \in carrier\text{-vec } n$   
**shows**  $- ((l :: 'a :: ab\text{-group-add } vec) + r) = (- l - r)$   
 ⟨proof⟩

**lemma** *minus-add-minus-vec*: **fixes**  $u :: 'a :: ab\text{-group-add } vec$   
**assumes**  $u \in carrier\text{-vec } n \ v \in carrier\text{-vec } n \ w \in carrier\text{-vec } n$   
**shows**  $u - (v + w) = u - v - w$   
 ⟨proof⟩

**lemma** *uminus-add-minus-mat*:  
**assumes**  $l \in carrier\text{-mat } nr \ nc \ r \in carrier\text{-mat } nr \ nc$   
**shows**  $- ((l :: 'a :: ab\text{-group-add } mat) + r) = (- l - r)$   
 ⟨proof⟩

**lemma** *minus-add-minus-mat*: **fixes**  $u :: 'a :: \text{ab-group-add mat}$   
**assumes**  $u \in \text{carrier-mat nr nc } v \in \text{carrier-mat nr nc } w \in \text{carrier-mat nr nc}$   
**shows**  $u - (v + w) = u - v - w$   
 $\langle \text{proof} \rangle$

**lemma** *uminus-uminus-vec[simp]*:  $- (- (v :: 'a :: \text{group-add vec})) = v$   
 $\langle \text{proof} \rangle$

**lemma** *uminus-eq-vec[simp]*:  $- (v :: 'a :: \text{group-add vec}) = - w \longleftrightarrow v = w$   
 $\langle \text{proof} \rangle$

**lemma** *uminus-uminus-mat[simp]*:  $- (- (A :: 'a :: \text{group-add mat})) = A$   
 $\langle \text{proof} \rangle$

**lemma** *uminus-eq-mat[simp]*:  $- (A :: 'a :: \text{group-add mat}) = - B \longleftrightarrow A = B$   
 $\langle \text{proof} \rangle$

**lemma** *smult-zero-mat[simp]*:  $(k :: 'a :: \text{mult-zero}) \cdot_m 0_m \text{ nr nc} = 0_m \text{ nr nc}$   
 $\langle \text{proof} \rangle$

**lemma** *similar-mat-wit-smult*: **fixes**  $A :: 'a :: \text{comm-ring-1 mat}$   
**assumes** *similar-mat-wit*  $A B P Q$   
**shows** *similar-mat-wit*  $(k \cdot_m A) (k \cdot_m B) P Q$   
 $\langle \text{proof} \rangle$

**lemma** *similar-mat-smult*: **fixes**  $A :: 'a :: \text{comm-ring-1 mat}$   
**assumes** *similar-mat*  $A B$   
**shows** *similar-mat*  $(k \cdot_m A) (k \cdot_m B)$   
 $\langle \text{proof} \rangle$

**definition** *mat-diag* ::  $\text{nat} \Rightarrow (\text{nat} \Rightarrow 'a :: \text{zero}) \Rightarrow 'a \text{ mat}$  **where**  
 $\text{mat-diag } n f = \text{Matrix.mat } n n (\lambda (i,j). \text{if } i = j \text{ then } f j \text{ else } 0)$

**lemma** *mat-diag-dim[simp]*:  $\text{mat-diag } n f \in \text{carrier-mat } n n$   
 $\langle \text{proof} \rangle$

**lemma** *mat-diag-mult-left*: **assumes**  $A: A \in \text{carrier-mat } n nr$   
**shows**  $\text{mat-diag } n f * A = \text{Matrix.mat } n nr (\lambda (i,j). f i * A \$\$ (i,j))$   
 $\langle \text{proof} \rangle$

**lemma** *mat-diag-mult-right*: **assumes**  $A: A \in \text{carrier-mat } nr n$   
**shows**  $A * \text{mat-diag } n f = \text{Matrix.mat } nr n (\lambda (i,j). A \$\$ (i,j) * f j)$   
 $\langle \text{proof} \rangle$

**lemma** *mat-diag-diag[simp]*:  $\text{mat-diag } n f * \text{mat-diag } n g = \text{mat-diag } n (\lambda i. f i * g i)$   
 $\langle \text{proof} \rangle$

**lemma** *mat-diag-one[simp]*:  $\text{mat-diag } n (\lambda x. 1) = 1_m n$   $\langle \text{proof} \rangle$

Interpret vector as row-matrix

**definition** *mat-of-row*  $y = \text{mat } 1 \ (\text{dim-vec } y) \ (\lambda ij. y \$ (\text{snd } ij))$

**lemma** *mat-of-row-carrier*[*simp,intro*]:

$y \in \text{carrier-vec } n \implies \text{mat-of-row } y \in \text{carrier-mat } 1 \ n$

$y \in \text{carrier-vec } n \implies \text{mat-of-row } y \in \text{carrier-mat } (\text{Suc } 0) \ n$

*<proof>*

**lemma** *mat-of-row-dim*[*simp*]:  $\text{dim-row } (\text{mat-of-row } y) = 1$

$\text{dim-col } (\text{mat-of-row } y) = \text{dim-vec } y$

*<proof>*

**lemma** *mat-of-row-index*[*simp*]:  $x < \text{dim-vec } y \implies \text{mat-of-row } y \$ \$ (0, x) = y \$ x$

*<proof>*

**lemma** *row-mat-of-row*[*simp*]:  $\text{row } (\text{mat-of-row } y) \ 0 = y$

*<proof>*

**lemma** *mat-of-row-mult-append-rows*: **assumes**  $y1: y1 \in \text{carrier-vec } nr1$

**and**  $y2: y2 \in \text{carrier-vec } nr2$

**and**  $A1: A1 \in \text{carrier-mat } nr1 \ nc$

**and**  $A2: A2 \in \text{carrier-mat } nr2 \ nc$

**shows**  $\text{mat-of-row } (y1 \ @_v \ y2) * (A1 \ @_r \ A2) =$

$\text{mat-of-row } y1 * A1 + \text{mat-of-row } y2 * A2$

*<proof>*

Allowing to construct and deconstruct vectors like lists

**abbreviation** *vNil* **where**  $vNil \equiv \text{vec } 0 \ ((!) \ [])$

**definition** *vCons* **where**  $vCons \ a \ v \equiv \text{vec } (\text{Suc } (\text{dim-vec } v)) \ (\lambda i. \text{case } i \ \text{of } 0 \Rightarrow a \mid \text{Suc } i \Rightarrow v \$ i)$

**lemma** *vec-index-vCons-0* [*simp*]:  $vCons \ a \ v \$ 0 = a$

*<proof>*

**lemma** *vec-index-vCons-Suc* [*simp*]:

**fixes**  $v :: 'a \ \text{vec}$

**shows**  $vCons \ a \ v \$ \text{Suc } n = v \$ n$

*<proof>*

**lemma** *vec-index-vCons*:  $vCons \ a \ v \$ n = (\text{if } n = 0 \ \text{then } a \ \text{else } v \$ (n - 1))$

*<proof>*

**lemma** *dim-vec-vCons* [*simp*]:  $\text{dim-vec } (vCons \ a \ v) = \text{Suc } (\text{dim-vec } v)$

*<proof>*

**lemma** *vCons-carrier-vec*[*simp*]:  $vCons \ a \ v \in \text{carrier-vec } (\text{Suc } n) \longleftrightarrow v \in \text{carrier-vec } n$

*<proof>*

**lemma** *vec-Suc*:  $vec (Suc\ n)\ f = vCons\ (f\ 0)\ (vec\ n\ (f\ \circ\ Suc))$  (**is**  $?l = ?r$ )  
 ⟨*proof*⟩

**declare** *Abs-vec-cases*[*cases del*]

**lemma** *vec-cases* [*case-names vNil vCons, cases type: vec*]:  
**assumes**  $v = vNil \implies thesis$  **and**  $\bigwedge a\ w.\ v = vCons\ a\ w \implies thesis$   
**shows** *thesis*  
 ⟨*proof*⟩

**lemma** *vec-induct* [*case-names vNil vCons, induct type: vec*]:  
**assumes**  $P\ vNil$  **and**  $\bigwedge a\ v.\ P\ v \implies P\ (vCons\ a\ v)$   
**shows**  $P\ v$   
 ⟨*proof*⟩

**lemma** *carrier-vec-induct* [*consumes 1, case-names 0 Suc, induct set: carrier-vec*]:  
**assumes**  $v: v \in carrier-vec\ n$   
**and**  $1: P\ 0\ vNil$  **and**  $2: \bigwedge n\ a\ v.\ v \in carrier-vec\ n \implies P\ n\ v \implies P\ (Suc\ n)$   
 ( $vCons\ a\ v$ )  
**shows**  $P\ n\ v$   
 ⟨*proof*⟩

**lemma** *vec-of-list-Cons*[*simp*]:  $vec-of-list\ (a\ \#\ as) = vCons\ a\ (vec-of-list\ as)$   
 ⟨*proof*⟩

**lemma** *vec-of-list-Nil*[*simp*]:  $vec-of-list\ [] = vNil$   
 ⟨*proof*⟩

**lemma** *scalar-prod-vCons*[*simp*]:  
 $vCons\ a\ v \cdot vCons\ b\ w = a * b + v \cdot w$   
 ⟨*proof*⟩

**lemma** *zero-vec-Suc*:  $0_v\ (Suc\ n) = vCons\ 0\ (0_v\ n)$   
 ⟨*proof*⟩

**lemma** *zero-vec-zero*[*simp*]:  $0_v\ 0 = vNil$  ⟨*proof*⟩

**lemma** *vCons-eq-vCons*[*simp*]:  $vCons\ a\ v = vCons\ b\ w \iff a = b \wedge v = w$  (**is**  $?l \iff ?r$ )  
 ⟨*proof*⟩

**lemma** *vec-carrier-vec*[*simp*]:  $vec\ n\ f \in carrier-vec\ m \iff n = m$   
 ⟨*proof*⟩

**notation** *transpose-mat* ( $(-^T)$  [1000])

**lemma** *map-mat-transpose*:  $(map-mat\ f\ A)^T = map-mat\ f\ A^T$  ⟨*proof*⟩

**lemma** *cols-transpose*[simp]:  $\text{cols } A^T = \text{rows } A$   $\langle \text{proof} \rangle$   
**lemma** *rows-transpose*[simp]:  $\text{rows } A^T = \text{cols } A$   $\langle \text{proof} \rangle$   
**lemma** *list-of-vec-vec* [simp]:  $\text{list-of-vec } (\text{vec } n \ f) = \text{map } f \ [0..<n]$   
 $\langle \text{proof} \rangle$

**lemma** *list-of-vec-0* [simp]:  $\text{list-of-vec } (0_v \ n) = \text{replicate } n \ 0$   
 $\langle \text{proof} \rangle$

**lemma** *diag-mat-map*:  
**assumes** *M-carrier*:  $M \in \text{carrier-mat } n \ n$   
**shows**  $\text{diag-mat } (\text{map-mat } f \ M) = \text{map } f \ (\text{diag-mat } M)$   
 $\langle \text{proof} \rangle$

**lemma** *mat-of-rows-map* [simp]:  
**assumes** *x*:  $\text{set } vs \subseteq \text{carrier-vec } n$   
**shows**  $\text{mat-of-rows } n \ (\text{map } (\text{map-vec } f) \ vs) = \text{map-mat } f \ (\text{mat-of-rows } n \ vs)$   
 $\langle \text{proof} \rangle$

**lemma** *mat-of-cols-map* [simp]:  
**assumes** *x*:  $\text{set } vs \subseteq \text{carrier-vec } n$   
**shows**  $\text{mat-of-cols } n \ (\text{map } (\text{map-vec } f) \ vs) = \text{map-mat } f \ (\text{mat-of-cols } n \ vs)$   
 $\langle \text{proof} \rangle$

**lemma** *vec-of-list-map* [simp]:  $\text{vec-of-list } (\text{map } f \ xs) = \text{map-vec } f \ (\text{vec-of-list } xs)$   
 $\langle \text{proof} \rangle$

**lemma** *map-vec*:  $\text{map-vec } f \ (\text{vec } n \ g) = \text{vec } n \ (f \ o \ g)$   $\langle \text{proof} \rangle$

**lemma** *mat-of-cols-Cons-index-0*:  $i < n \implies \text{mat-of-cols } n \ (w \ \# \ ws) \ \$\$ \ (i, 0) = w \ \$ \ i$   
 $\langle \text{proof} \rangle$

**lemma** *nth-map-out-of-bound*:  $i \geq \text{length } xs \implies \text{map } f \ xs \ ! \ i = [] \ ! \ (i - \text{length } xs)$   
 $\langle \text{proof} \rangle$

**lemma** *mat-of-cols-Cons-index-Suc*:  
 $i < n \implies \text{mat-of-cols } n \ (w \ \# \ ws) \ \$\$ \ (i, \text{Suc } j) = \text{mat-of-cols } n \ ws \ \$\$ \ (i, j)$   
 $\langle \text{proof} \rangle$

**lemma** *mat-of-cols-index*:  $i < n \implies j < \text{length } ws \implies \text{mat-of-cols } n \ ws \ \$\$ \ (i, j) = ws \ ! \ j \ \$ \ i$   
 $\langle \text{proof} \rangle$

**lemma** *mat-of-rows-index*:  $i < \text{length } rs \implies j < n \implies \text{mat-of-rows } n \ rs \ \$\$ \ (i, j) = rs \ ! \ i \ \$ \ j$   
 $\langle \text{proof} \rangle$

**lemma** *transpose-mat-of-rows*:  $(\text{mat-of-rows } n \ vs)^T = \text{mat-of-cols } n \ vs$   
 $\langle \text{proof} \rangle$

**lemma** *transpose-mat-of-cols*:  $(\text{mat-of-cols } n \text{ vs})^T = \text{mat-of-rows } n \text{ vs}$   
 ⟨proof⟩

**lemma** *nth-list-of-vec* [simp]:  
 assumes  $i < \text{dim-vec } v$  shows  $\text{list-of-vec } v ! i = v \$ i$   
 ⟨proof⟩

**lemma** *length-list-of-vec* [simp]:  
 $\text{length } (\text{list-of-vec } v) = \text{dim-vec } v$  ⟨proof⟩

**lemma** *vec-eq-0-iff*:  
 $v = 0_v \ n \longleftrightarrow n = \text{dim-vec } v \wedge (n = 0 \vee \text{set } (\text{list-of-vec } v) = \{0\})$  (is ?l  $\longleftrightarrow$   
 ?r)  
 ⟨proof⟩

**lemma** *list-of-vec-vCons*[simp]:  $\text{list-of-vec } (v\text{Cons } a \ v) = a \# \text{list-of-vec } v$  (is ?l =  
 ?r)  
 ⟨proof⟩

**lemma** *append-vec-vCons*[simp]:  $v\text{Cons } a \ v \ @_v \ w = v\text{Cons } a \ (v \ @_v \ w)$  (is ?l =  
 ?r)  
 ⟨proof⟩

**lemma** *append-vec-vNil*[simp]:  $v\text{Nil} \ @_v \ v = v$   
 ⟨proof⟩

**lemma** *list-of-vec-append*[simp]:  $\text{list-of-vec } (v \ @_v \ w) = \text{list-of-vec } v \ @ \ \text{list-of-vec } w$   
 ⟨proof⟩

**lemma** *transpose-mat-eq*[simp]:  $A^T = B^T \longleftrightarrow A = B$   
 ⟨proof⟩

**lemma** *mat-col-eqI*: assumes  $\text{cols}: \bigwedge i. i < \text{dim-col } B \implies \text{col } A \ i = \text{col } B \ i$   
 and  $\text{dims}: \text{dim-row } A = \text{dim-row } B \ \text{dim-col } A = \text{dim-col } B$   
 shows  $A = B$   
 ⟨proof⟩

**lemma** *upper-triangular-imp-distinct*:  
 assumes  $A: A \in \text{carrier-mat } n \ n$   
 and  $\text{tri}: \text{upper-triangular } A$   
 and  $\text{diag}: 0 \notin \text{set } (\text{diag-mat } A)$   
 shows  $\text{distinct } (\text{rows } A)$   
 ⟨proof⟩

**lemma** *dim-vec-of-list*[simp]:  $\text{dim-vec } (\text{vec-of-list } as) = \text{length } as$  ⟨proof⟩

**lemma** *list-vec*:  $\text{list-of-vec } (\text{vec-of-list } xs) = xs$   
 ⟨proof⟩

**lemma** *vec-list*:  $vec\text{-of-list } (list\text{-of-vec } v) = v$   
*<proof>*

**lemma** *index-vec-of-list*:  $i < length\ xs \implies (vec\text{-of-list } xs) \$ i = xs ! i$   
*<proof>*

**lemma** *vec-of-list-index*:  $vec\text{-of-list } xs \$ j = xs ! j$   
*<proof>*

**lemma** *list-of-vec-index*:  $list\text{-of-vec } v ! j = v \$ j$   
*<proof>*

**lemma** *list-of-vec-map*:  $list\text{-of-vec } xs = map ((\$) xs) [0..<dim-vec xs]$  *<proof>*

**definition** *component-mult*  $v\ w = vec\ (min\ (dim\text{-vec } v)\ (dim\text{-vec } w))\ (\lambda i. v \$ i * w \$ i)$

**definition** *vec-set*:  $'a\ vec \implies 'a\ set\ (set_v)$   
**where**  $vec\text{-set } v = vec\text{-index } v\ ' \{..<dim\text{-vec } v\}$

**lemma** *index-component-mult*:  
**assumes**  $i < dim\text{-vec } v\ i < dim\text{-vec } w$   
**shows**  $component\text{-mult } v\ w \$ i = v \$ i * w \$ i$   
*<proof>*

**lemma** *dim-component-mult*:  
 $dim\text{-vec } (component\text{-mult } v\ w) = min\ (dim\text{-vec } v)\ (dim\text{-vec } w)$   
*<proof>*

**lemma** *vec-setE*:  
**assumes**  $a \in set_v\ v$   
**obtains**  $i$  **where**  $v \$ i = a\ i < dim\text{-vec } v$  *<proof>*

**lemma** *vec-setI*:  
**assumes**  $v \$ i = a\ i < dim\text{-vec } v$   
**shows**  $a \in set_v\ v$  *<proof>*

**lemma** *set-list-of-vec*:  $set\ (list\text{-of-vec } v) = set_v\ v$  *<proof>*

**instantiation** *vec* :: (*conjugate*) *conjugate*  
**begin**

**definition** *conjugate-vec* ::  $'a :: conjugate\ vec \implies 'a\ vec$   
**where**  $conjugate\ v = vec\ (dim\text{-vec } v)\ (\lambda i. conjugate\ (v \$ i))$

**lemma** *conjugate-vCons* [*simp*]:  
 $conjugate\ (vCons\ a\ v) = vCons\ (conjugate\ a)\ (conjugate\ v)$   
*<proof>*



**lemma** *dim-vec-conjugate[simp]*:  $\dim\text{-vec} (\text{conjugate } v) = \dim\text{-vec } v$   
*<proof>*

**lemma** *carrier-vec-conjugate[simp]*:  $v \in \text{carrier-vec } n \implies \text{conjugate } v \in \text{carrier-vec } n$   
*<proof>*

**lemma** *vec-index-conjugate[simp]*:  
**shows**  $i < \dim\text{-vec } v \implies \text{conjugate } v \$ i = \text{conjugate } (v \$ i)$   
*<proof>*

**instance**  
*<proof>*

**end**

**lemma** *conjugate-add-vec*:  
**fixes**  $v w :: 'a :: \text{conjugatable-ring } \text{vec}$   
**assumes**  $\dim: v : \text{carrier-vec } n \ w : \text{carrier-vec } n$   
**shows**  $\text{conjugate } (v + w) = \text{conjugate } v + \text{conjugate } w$   
*<proof>*

**lemma** *uminus-conjugate-vec*:  
**fixes**  $v w :: 'a :: \text{conjugatable-ring } \text{vec}$   
**shows**  $-(\text{conjugate } v) = \text{conjugate } (-v)$   
*<proof>*

**lemma** *conjugate-zero-vec[simp]*:  
 $\text{conjugate } (0_v \ n :: 'a :: \text{conjugatable-ring } \text{vec}) = 0_v \ n$  *<proof>*

**lemma** *conjugate-vec-0[simp]*:  
 $\text{conjugate } (\text{vec } 0 \ f) = \text{vec } 0 \ f$  *<proof>*

**lemma** *sprod-vec-0[simp]*:  $v \cdot \text{vec } 0 \ f = 0$   
*<proof>*

**lemma** *conjugate-zero-iff-vec[simp]*:  
**fixes**  $v :: 'a :: \text{conjugatable-ring } \text{vec}$   
**shows**  $\text{conjugate } v = 0_v \ n \longleftrightarrow v = 0_v \ n$   
*<proof>*

**lemma** *conjugate-smult-vec*:  
**fixes**  $k :: 'a :: \text{conjugatable-ring}$   
**shows**  $\text{conjugate } (k \cdot_v v) = \text{conjugate } k \cdot_v \text{conjugate } v$   
*<proof>*

**lemma** *conjugate-sprod-vec*:  
**fixes**  $v w :: 'a :: \text{conjugatable-ring } \text{vec}$

**assumes**  $v : \text{carrier-vec } n$  **and**  $w : \text{carrier-vec } n$   
**shows**  $\text{conjugate } (v \cdot w) = \text{conjugate } v \cdot \text{conjugate } w$   
 $\langle \text{proof} \rangle$

**abbreviation**  $\text{cscalar-prod} :: 'a \text{ vec} \Rightarrow 'a \text{ vec} \Rightarrow 'a :: \text{conjugatable-ring}$  (**infix**  $\cdot c$   
 $70$ )  
**where**  $(\cdot c) \equiv \lambda v w. v \cdot \text{conjugate } w$

**lemma**  $\text{conjugate-conjugate-sprod}[simp]$ :  
**assumes**  $v[simp] : \text{carrier-vec } n$  **and**  $w[simp] : \text{carrier-vec } n$   
**shows**  $\text{conjugate } (\text{conjugate } v \cdot w) = v \cdot c w$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{conjugate-vec-sprod-comm}$ :  
**fixes**  $v w :: 'a :: \{\text{conjugatable-ring, comm-ring}\} \text{vec}$   
**assumes**  $v : \text{carrier-vec } n$  **and**  $w : \text{carrier-vec } n$   
**shows**  $v \cdot c w = (\text{conjugate } w \cdot v)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{conjugate-square-ge-0-vec}[intro!]$ :  
**fixes**  $v :: 'a :: \text{conjugatable-ordered-ring } \text{vec}$   
**shows**  $v \cdot c v \geq 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{conjugate-square-eq-0-vec}[simp]$ :  
**fixes**  $v :: 'a :: \{\text{conjugatable-ordered-ring, semiring-no-zero-divisors}\} \text{vec}$   
**assumes**  $v \in \text{carrier-vec } n$   
**shows**  $v \cdot c v = 0 \longleftrightarrow v = 0_v n$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{conjugate-square-greater-0-vec}[simp]$ :  
**fixes**  $v :: 'a :: \{\text{conjugatable-ordered-ring, semiring-no-zero-divisors}\} \text{vec}$   
**assumes**  $v \in \text{carrier-vec } n$   
**shows**  $v \cdot c v > 0 \longleftrightarrow v \neq 0_v n$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{vec-conjugate-rat}[simp]$ :  $(\text{conjugate} :: \text{rat } \text{vec} \Rightarrow \text{rat } \text{vec}) = (\lambda x. x)$   $\langle \text{proof} \rangle$   
**lemma**  $\text{vec-conjugate-real}[simp]$ :  $(\text{conjugate} :: \text{real } \text{vec} \Rightarrow \text{real } \text{vec}) = (\lambda x. x)$   $\langle \text{proof} \rangle$

**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} <proof>
typedef 'a mat-impl = {(nr,nc,m :: 'a iarray iarray).
  IArray.length m = nr ∧ IArray.all (λ r. IArray.length r = nc) m}
  <proof>

setup-lifting type-definition-vec-impl
setup-lifting type-definition-mat-impl

lift-definition vec-impl :: 'a vec-impl ⇒ 'a vec is
  λ (n,v). (n,mk-vec n (IArray.sub v)) <proof>

lift-definition vec-add-impl :: 'a::plus vec-impl ⇒ 'a vec-impl ⇒ 'a vec-impl is
  λ (n,v) (m,w).
  (n, IArray.of-fun (λ i. IArray.sub v i + IArray.sub w i) n)
  <proof>

lift-definition mat-impl :: 'a mat-impl ⇒ 'a mat is
  λ (nr,nc,m). (nr,nc,mk-mat nr nc (λ (i,j). IArray.sub (IArray.sub m i) j)) <proof>

lift-definition vec-of-list-impl :: 'a list ⇒ 'a vec-impl is
  λ v. (length v, IArray v) <proof>

lift-definition list-of-vec-impl :: 'a vec-impl ⇒ 'a list is
  λ (n,v). IArray.list-of v <proof>

lift-definition vec-of-fun :: nat ⇒ (nat ⇒ 'a) ⇒ 'a vec-impl is
  λ n f. (n, IArray.of-fun f n) <proof>

lift-definition mat-of-fun :: nat ⇒ nat ⇒ (nat × nat ⇒ 'a) ⇒ 'a mat-impl is
  λ nr nc f. (nr, nc, IArray.of-fun (λ i. IArray.of-fun (λ j. f (i,j)) nc) nr) <proof>

lift-definition vec-index-impl :: 'a vec-impl ⇒ nat ⇒ 'a
  is λ (n,v). IArray.sub v <proof>

lift-definition index-mat-impl :: 'a mat-impl ⇒ nat × nat ⇒ 'a
  is λ (nr,nc,m) (i,j). if i < nr then IArray.sub (IArray.sub m i) j
  else IArray.sub (IArray ([] ! (i - nr))) j <proof>

lift-definition vec-equal-impl :: 'a vec-impl ⇒ 'a vec-impl ⇒ bool

```

**is**  $\lambda (n1, v1) (n2, v2). n1 = n2 \wedge v1 = v2$  *<proof>*

**lift-definition** *mat-equal-impl* :: 'a mat-impl  $\Rightarrow$  'a mat-impl  $\Rightarrow$  bool  
**is**  $\lambda (nr1, nc1, m1) (nr2, nc2, m2). nr1 = nr2 \wedge nc1 = nc2 \wedge m1 = m2$  *<proof>*

**lift-definition** *dim-vec-impl* :: 'a vec-impl  $\Rightarrow$  nat **is** *fst* *<proof>*

**lift-definition** *dim-row-impl* :: 'a mat-impl  $\Rightarrow$  nat **is** *fst* *<proof>*  
**lift-definition** *dim-col-impl* :: 'a mat-impl  $\Rightarrow$  nat **is** *fst o snd* *<proof>*

**code-datatype** *vec-impl*  
**code-datatype** *mat-impl*

**lemma** *vec-code*[code]: *vec* n f = *vec-impl* (*vec-of-fun* n f)  
*<proof>*

**lemma** *mat-code*[code]: *mat* nr nc f = *mat-impl* (*mat-of-fun* nr nc f)  
*<proof>*

**lemma** *vec-of-list*[code]: *vec-of-list* v = *vec-impl* (*vec-of-list-impl* v)  
*<proof>*

**lemma** *list-of-vec-code*[code]: *list-of-vec* (*vec-impl* v) = *list-of-vec-impl* v  
*<proof>*

**lemma** *empty-nth*:  $\neg i < \text{length } x \implies x ! i = [] ! (i - \text{length } x)$   
*<proof>*

**lemma** *undef-vec*:  $\neg i < \text{length } x \implies \text{undef-vec } (i - \text{length } x) = x ! i$   
*<proof>*

**lemma** *vec-index-code*[code]: (*vec-impl* v) \$ i = *vec-index-impl* v i  
*<proof>*

**lemma** *index-mat-code*[code]: (*mat-impl* m) \$\$ ij = (*index-mat-impl* m ij :: 'a)  
*<proof>*

**lift-definition** (*code-dt*) *mat-of-rows-list-impl* :: nat  $\Rightarrow$  'a list list  $\Rightarrow$  'a mat-impl  
*option is*  
 $\lambda n \text{ rows. if list-all } (\lambda r. \text{length } r = n) \text{ rows then Some } (\text{length rows}, n, \text{IArray } (\text{map } \text{IArray rows}))$   
*else None*  
*<proof>*

**lemma** *mat-of-rows-list-impl*: *mat-of-rows-list-impl* n rs = Some A  $\implies$  *mat-impl* A = *mat-of-rows-list* n rs  
*<proof>*

**lemma** *mat-of-rows-list-code*[code]: *mat-of-rows-list* nc vs =

(*case mat-of-rows-list-impl nc vs of Some A ⇒ mat-impl A*  
| *None ⇒ mat-of-rows nc (map (λ v. vec nc (nth v)) vs)*)  
⟨*proof*⟩

**lemma** *dim-vec-code*[code]: *dim-vec (vec-impl v) = dim-vec-impl v*  
⟨*proof*⟩

**lemma** *dim-row-code*[code]: *dim-row (mat-impl m) = dim-row-impl m*  
⟨*proof*⟩

**lemma** *dim-col-code*[code]: *dim-col (mat-impl m) = dim-col-impl m*  
⟨*proof*⟩

**instantiation** *vec* :: (*type*)*equal*

**begin**

**definition** (*equal-vec* :: ('*a* *vec* ⇒ '*a* *vec* ⇒ *bool*)) = (=)

**instance**

⟨*proof*⟩

**end**

**instantiation** *mat* :: (*type*)*equal*

**begin**

**definition** (*equal-mat* :: ('*a* *mat* ⇒ '*a* *mat* ⇒ *bool*)) = (=)

**instance**

⟨*proof*⟩

**end**

**lemma** *vec-equal-code*[code]: *HOL.equal (vec-impl (v1 :: '*a* vec-impl)) (vec-impl v2) = vec-equal-impl v1 v2*  
⟨*proof*⟩

**lemma** *mat-equal-code*[code]: *HOL.equal (mat-impl (m1 :: '*a* mat-impl)) (mat-impl m2) = mat-equal-impl m1 m2*  
⟨*proof*⟩

**declare** *prod.set-conv-list*[code *del*, *code-unfold*]

**derive** (*eq*) *ceq mat vec*

**derive** (*no*) *compare mat vec*

**derive** (*dlist*) *set-impl mat vec*

**derive** (*no*) *cenum mat vec*

**lemma** *carrier-mat-code*[code]: *carrier-mat nr nc = Collect-set (λ A. dim-row A = nr ∧ dim-col A = nc)* ⟨*proof*⟩

**lemma** *carrier-vec-code*[code]: *carrier-vec n = Collect-set (λ v. dim-vec v = n)*  
⟨*proof*⟩

**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* :: ('a ⇒ 'a ⇒ 'a) ⇒ nat ⇒ 'a ⇒ 'a mat ⇒ 'a mat  
**where**

*mat-multrow-gen mul k a A = mat (dim-row A) (dim-col A)*  
*(λ (i,j). if k = i then mul a (A \$\$ (i,j)) else A \$\$ (i,j))*

**abbreviation** *mat-multrow* :: nat ⇒ 'a :: semiring-1 ⇒ 'a mat ⇒ 'a mat (*multrow*)  
**where**

*multrow ≡ mat-multrow-gen ((\*))*

**lemmas** *mat-multrow-def = mat-multrow-gen-def*

**definition** *multrow-mat* :: nat ⇒ nat ⇒ 'a :: semiring-1 ⇒ 'a mat **where**

*multrow-mat n k a = mat n n*  
*(λ (i,j). if k = i ∧ k = j then a else if i = j then 1 else 0)*

**definition** *mat-swaprows* :: nat ⇒ nat ⇒ 'a mat ⇒ 'a mat (*swaprows*)**where**

*swaprows k l A = mat (dim-row A) (dim-col A)*  
*(λ (i,j). if k = i then A \$\$ (l,j) else if l = i then A \$\$ (k,j) else A \$\$ (i,j))*

**definition** *swaprows-mat* :: nat ⇒ nat ⇒ nat ⇒ 'a :: semiring-1 mat **where**

*swaprows-mat n k l = mat n n*  
*(λ (i,j). if k = i ∧ l = j ∨ k = j ∧ l = i ∨ i = j ∧ i ≠ k ∧ i ≠ l then 1 else 0)*

**definition** *mat-addrow-gen* :: ('a ⇒ 'a ⇒ 'a) ⇒ ('a ⇒ 'a ⇒ 'a) ⇒ 'a ⇒ nat ⇒ nat ⇒ 'a mat ⇒ 'a mat **where**

*mat-addrow-gen ad mul a k l A = mat (dim-row A) (dim-col A)*  
*(λ (i,j). if k = i then ad (mul a (A \$\$ (l,j))) (A \$\$ (i,j)) else A \$\$ (i,j))*

**abbreviation** *mat-addrow* :: 'a :: semiring-1 ⇒ nat ⇒ nat ⇒ 'a mat ⇒ 'a mat (*addrow*) **where**

*addrow ≡ mat-addrow-gen (+) ((\*))*

**lemmas** *mat-addrow-def = mat-addrow-gen-def*

**definition** *addrow-mat* :: nat ⇒ 'a :: semiring-1 ⇒ nat ⇒ nat ⇒ 'a mat **where**

*addrow-mat n a k l = mat n n (λ (i,j).*  
*(if k = i ∧ l = j then (+) a else id) (if i = j then 1 else 0))*

**lemma** *index-mat-multrow[simp]*:

$i < \dim\text{-row } A \implies j < \dim\text{-col } A \implies \text{mat-multrow-gen mul } k \ a \ A \ \$\$ (i,j) = (\text{if } k = i \text{ then mul } a \ (A \ \$\$ (i,j)) \text{ else } A \ \$\$ (i,j))$

$i < \dim\text{-row } A \implies j < \dim\text{-col } A \implies \text{mat-multrow-gen mul } i \ a \ A \ \$\$ (i,j) = \text{mul } a \ (A \ \$\$ (i,j))$

$i < \dim\text{-row } A \implies j < \dim\text{-col } A \implies k \neq i \implies \text{mat-multrow-gen mul } k \ a \ A \ \$\$ (i,j) = A \ \$\$ (i,j)$

$\dim\text{-row } (\text{mat-multrow-gen mul } k \ a \ A) = \dim\text{-row } A \ \dim\text{-col } (\text{mat-multrow-gen mul } k \ a \ A) = \dim\text{-col } A$

*<proof>*

**lemma** *index-mat-multrow-mat[simp]*:

$i < n \implies j < n \implies \text{multrow-mat } n \ k \ a \ \$\$ (i,j) = (\text{if } k = i \wedge k = j \text{ then } a \text{ else if } i = j$

$\text{then } 1 \text{ else } 0)$

$\dim\text{-row } (\text{multrow-mat } n \ k \ a) = n \ \dim\text{-col } (\text{multrow-mat } n \ k \ a) = n$

*<proof>*

**lemma** *index-mat-swaprows[simp]*:

$i < \dim\text{-row } A \implies j < \dim\text{-col } A \implies \text{swaprows } k \ l \ A \ \$\$ (i,j) = (\text{if } k = i \text{ then } A \ \$\$ (l,j) \text{ else}$

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

$\dim\text{-row } (\text{swaprows } k \ l \ A) = \dim\text{-row } A \ \dim\text{-col } (\text{swaprows } k \ l \ A) = \dim\text{-col } A$

*<proof>*

**lemma** *index-mat-swaprows-mat[simp]*:

$i < n \implies j < n \implies \text{swaprows-mat } n \ k \ l \ \$\$ (i,j) =$

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

$\dim\text{-row } (\text{swaprows-mat } n \ k \ l) = n \ \dim\text{-col } (\text{swaprows-mat } n \ k \ l) = n$

*<proof>*

**lemma** *index-mat-addrow[simp]*:

$i < \dim\text{-row } A \implies j < \dim\text{-col } A \implies \text{mat-addrow-gen ad mul } a \ k \ l \ A \ \$\$ (i,j) = (\text{if } k = i \text{ then}$

$\text{ad } (\text{mul } a \ (A \ \$\$ (l,j))) \ (A \ \$\$ (i,j)) \text{ else } A \ \$\$ (i,j))$

$i < \dim\text{-row } A \implies j < \dim\text{-col } A \implies \text{mat-addrow-gen ad mul } a \ i \ l \ A \ \$\$ (i,j) = \text{ad } (\text{mul } a \ (A \ \$\$ (l,j))) \ (A \ \$\$ (i,j))$

$i < \dim\text{-row } A \implies j < \dim\text{-col } A \implies k \neq i \implies \text{mat-addrow-gen ad mul } a \ k \ l \ A \ \$\$ (i,j) = A \ \$\$ (i,j)$

$\dim\text{-row } (\text{mat-addrow-gen ad mul } a \ k \ l \ A) = \dim\text{-row } A \ \dim\text{-col } (\text{mat-addrow-gen ad mul } a \ k \ l \ A) = \dim\text{-col } A$

*<proof>*

**lemma** *index-mat-addrow-mat[simp]*:

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

$(\text{if } k = i \wedge l = j \text{ then } (+) \ a \ \text{else } \text{id}) \ (\text{if } i = j \text{ then } 1 \text{ else } 0)$

$\dim\text{-row } (\text{addrow-mat } n \ a \ k \ l) = n \ \dim\text{-col } (\text{addrow-mat } n \ a \ k \ l) = n$

*<proof>*

**lemma** *multrow-carrier[simp]*:  $(\text{mat-multrow-gen mul } k \ a \ A \in \text{carrier-mat } n \ nc)$   
 $= (A \in \text{carrier-mat } n \ nc)$   
 $\langle \text{proof} \rangle$

**lemma** *multrow-mat-carrier[simp]*:  $\text{multrow-mat } n \ k \ a \in \text{carrier-mat } n \ n$   
 $\langle \text{proof} \rangle$

**lemma** *addrow-mat-carrier[simp]*:  $\text{addrow-mat } n \ a \ k \ l \in \text{carrier-mat } n \ n$   
 $\langle \text{proof} \rangle$

**lemma** *swaprows-mat-carrier[simp]*:  $\text{swaprows-mat } n \ k \ l \in \text{carrier-mat } n \ n$   
 $\langle \text{proof} \rangle$

**lemma** *swaprows-carrier[simp]*:  $(\text{swaprows } k \ l \ A \in \text{carrier-mat } n \ nc) = (A \in \text{carrier-mat } n \ nc)$   
 $\langle \text{proof} \rangle$

**lemma** *addrow-carrier[simp]*:  $(\text{mat-addrow-gen ad mul } a \ k \ l \ A \in \text{carrier-mat } n \ nc)$   
 $= (A \in \text{carrier-mat } n \ nc)$   
 $\langle \text{proof} \rangle$

**lemma** *row-multrow*:  $k \neq i \implies i < n \implies \text{row } (\text{multrow-mat } n \ k \ a) \ i = \text{unit-vec } n \ i$   
 $k < n \implies \text{row } (\text{multrow-mat } n \ k \ a) \ k = a \cdot_v \text{unit-vec } n \ k$   
 $\langle \text{proof} \rangle$

**lemma** *multrow-mat*: **assumes**  $A: A \in \text{carrier-mat } n \ nc$   
**shows**  $\text{multrow } k \ a \ A = \text{multrow-mat } n \ k \ a \ * \ A$   
 $\langle \text{proof} \rangle$

**lemma** *row-addrow*:  
 $k \neq i \implies i < n \implies \text{row } (\text{addrow-mat } n \ a \ k \ l) \ i = \text{unit-vec } n \ i$   
 $k < n \implies l < n \implies \text{row } (\text{addrow-mat } n \ a \ k \ l) \ k = a \cdot_v \text{unit-vec } n \ l + \text{unit-vec } n \ k$   
 $\langle \text{proof} \rangle$

**lemma** *addrow-mat*: **assumes**  $A: A \in \text{carrier-mat } n \ nc$   
**and**  $l: l < n$   
**shows**  $\text{addrow } a \ k \ l \ A = \text{addrow-mat } n \ a \ k \ l \ * \ A$   
 $\langle \text{proof} \rangle$

**lemma** *row-swaprows*:  
 $l < n \implies \text{row } (\text{swaprows-mat } n \ l \ l) \ l = \text{unit-vec } n \ l$   
 $i \neq k \implies i \neq l \implies i < n \implies \text{row } (\text{swaprows-mat } n \ k \ l) \ i = \text{unit-vec } n \ i$   
 $k < n \implies l < n \implies \text{row } (\text{swaprows-mat } n \ k \ l) \ l = \text{unit-vec } n \ k$   
 $k < n \implies l < n \implies \text{row } (\text{swaprows-mat } n \ k \ l) \ k = \text{unit-vec } n \ l$   
 $\langle \text{proof} \rangle$



**lemma** *swaprows-mat*: **assumes**  $A: A \in \text{carrier-mat } n \text{ nc}$  **and**  $k: k < n$   $l: l < n$   
**shows**  $\text{swaprows } k \ l \ A = \text{swaprows-mat } n \ k \ l \ * \ A$   
 $\langle \text{proof} \rangle$

**lemma** *swaprows-mat-inv*: **assumes**  $k: k < n$  **and**  $l: l < n$   
**shows**  $\text{swaprows-mat } n \ k \ l \ * \ \text{swaprows-mat } n \ k \ l = 1_m \ n$   
 $\langle \text{proof} \rangle$

**lemma** *swaprows-mat-Unit*: **assumes**  $k: k < n$  **and**  $l: l < n$   
**shows**  $\text{swaprows-mat } n \ k \ l \ \in \text{Units } (\text{ring-mat } \text{TYPE}('a :: \text{semiring-1}) \ n \ b)$   
 $\langle \text{proof} \rangle$

**lemma** *addrow-mat-inv*: **assumes**  $k: k < n$  **and**  $l: l < n$  **and**  $\text{neg}: k \neq l$   
**shows**  $\text{addrow-mat } n \ a \ k \ l \ * \ \text{addrow-mat } n \ (- \ (a :: 'a :: \text{comm-ring-1})) \ k \ l =$   
 $1_m \ n$   
 $\langle \text{proof} \rangle$

**lemma** *addrow-mat-Unit*: **assumes**  $k: k < n$  **and**  $l: l < n$  **and**  $\text{neg}: k \neq l$   
**shows**  $\text{addrow-mat } n \ a \ k \ l \ \in \text{Units } (\text{ring-mat } \text{TYPE}('a :: \text{comm-ring-1}) \ n \ b)$   
 $\langle \text{proof} \rangle$

**lemma** *multrow-mat-inv*: **assumes**  $k: k < n$  **and**  $a: (a :: 'a :: \text{division-ring}) \neq 0$   
**shows**  $\text{multrow-mat } n \ k \ a \ * \ \text{multrow-mat } n \ k \ (\text{inverse } a) = 1_m \ n$   
 $\langle \text{proof} \rangle$

**lemma** *multrow-mat-Unit*: **assumes**  $k: k < n$  **and**  $a: (a :: 'a :: \text{division-ring}) \neq 0$   
**shows**  $\text{multrow-mat } n \ k \ a \ \in \text{Units } (\text{ring-mat } \text{TYPE}('a) \ n \ b)$   
 $\langle \text{proof} \rangle$

## 6.2 Gauss-Jordan Elimination

**fun** *eliminate-entries-rec* **where**

$\text{eliminate-entries-rec } B \ i \ [] = B$   
 $| \text{eliminate-entries-rec } B \ i \ ((ai'j, i') \# \ is) = ($   
 $\text{eliminate-entries-rec } (\text{mat-addrow-gen } ((+) :: 'b :: \text{ring-1} \Rightarrow 'b \Rightarrow 'b) \ (*) \ ai'j \ i'$   
 $i \ B) \ i \ is)$

**context**

**fixes**  $\text{minus} :: 'a \Rightarrow 'a \Rightarrow 'a$

**and**  $\text{times} :: 'a \Rightarrow 'a \Rightarrow 'a$

**begin**

**definition** *eliminate-entries-gen*  $:: (\text{nat} \Rightarrow 'a) \Rightarrow 'a \ \text{mat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \ \text{mat}$   
**where**

$\text{eliminate-entries-gen } v \ A \ I \ J = \text{mat } (\text{dim-row } A) \ (\text{dim-col } A) \ (\lambda \ (i, j).$   
 $\text{if } i \neq I \ \text{then } \text{minus } (A \ \$\$ \ (i, j)) \ (\text{times } (v \ i) \ (A \ \$\$ \ (I, j))) \ \text{else } A \ \$\$ \ (i, j))$

**lemma** *dim-eliminate-entries-gen[simp]*:  $\text{dim-row } (\text{eliminate-entries-gen } v \ B \ i \ as)$   
 $= \text{dim-row } B$

$\text{dim-col } (\text{eliminate-entries-gen } v \ B \ i \ as) = \text{dim-col } B$   
 ⟨proof⟩

**lemma** *dimc-eliminate-entries-rec[simp]*:  $\text{dim-col } (\text{eliminate-entries-rec } B \ i \ as) = \text{dim-col } B$   
 ⟨proof⟩

**lemma** *dimr-eliminate-entries-rec[simp]*:  $\text{dim-row } (\text{eliminate-entries-rec } B \ i \ as) = \text{dim-row } B$   
 ⟨proof⟩

**lemma** *carrier-eliminate-entries*:  $A \in \text{carrier-mat } nr \ nc \implies \text{eliminate-entries-gen } v \ A \ i \ bs \in \text{carrier-mat } nr \ nc$   
 $B \in \text{carrier-mat } nr \ nc \implies \text{eliminate-entries-rec } B \ i \ as \in \text{carrier-mat } nr \ nc$   
 ⟨proof⟩  
**end**

**abbreviation** *eliminate-entries*  $\equiv \text{eliminate-entries-gen } (-) \ ((*) :: 'a :: \text{ring-1} \Rightarrow 'a \Rightarrow 'a)$

**lemma** *eliminate-entries-convert*:

**assumes**  $jA: J < \text{dim-col } A$  **and**  $*: I < \text{dim-row } A$   $\text{dim-row } B = \text{dim-row } A$

**shows**  $\text{eliminate-entries } (\lambda \ i. A \ \$\$ \ (i, J)) \ B \ I \ J =$

$\text{eliminate-entries-rec } B \ I \ (\text{map } (\lambda \ i. (- \ A \ \$\$ \ (i, J), i)) \ (\text{filter } (\lambda \ i. i \neq I) \ [0 \ .. < \ \text{dim-row } A]))$   
 ⟨proof⟩

**lemma** *Unit-prod-eliminate-entries*:  $i < nr \implies (\bigwedge \ a \ i'. (a, i') \in \text{set } is \implies i' < nr \wedge i' \neq i)$   
 $\implies \exists \ P \in \text{Units } (\text{ring-mat } \text{TYPE}('a :: \text{comm-ring-1}) \ nr \ b) . \forall \ B \ nc. B \in \text{carrier-mat } nr \ nc \longrightarrow \text{eliminate-entries-rec } B \ i \ is = P * B$   
 ⟨proof⟩

**function** *gauss-jordan-main*  $:: 'a :: \text{field mat} \Rightarrow 'a \text{ mat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ mat} \times 'a \text{ mat}$  **where**

*gauss-jordan-main*  $A \ B \ i \ j = (\text{let } nr = \text{dim-row } A; \ nc = \text{dim-col } A \ \text{in}$

*if*  $i < nr \wedge j < nc$  *then* *let*  $a_{ij} = A \ \$\$ \ (i, j)$  *in* *if*  $a_{ij} = 0$  *then*

$(\text{case } [ \ i' . i' < - \ [ \ \text{Suc } i \ .. < \ nr], \ A \ \$\$ \ (i', j) \neq 0 ]$

*of*  $[] \Rightarrow \text{gauss-jordan-main } A \ B \ i \ (\text{Suc } j)$

$| \ (i' \# \ -) \Rightarrow \text{gauss-jordan-main } (\text{swaprows } i \ i' \ A) \ (\text{swaprows } i \ i' \ B) \ i \ j$

*else if*  $a_{ij} = 1$  *then* *let*

$v = (\lambda \ i. A \ \$\$ \ (i, j))$  *in*

*gauss-jordan-main*

$(\text{eliminate-entries } v \ A \ i \ j) \ (\text{eliminate-entries } v \ B \ i \ j) \ (\text{Suc } i) \ (\text{Suc } j)$

*else let*  $ia_{ij} = \text{inverse } a_{ij}$  *in* *gauss-jordan-main*  $(\text{multrow } i \ ia_{ij} \ A) \ (\text{multrow } i \ ia_{ij} \ B) \ i \ j$

*else*  $(A, B)$

⟨proof⟩

**termination**

*<proof>*

**declare** *gauss-jordan-main.simps*[*simp del*]

**definition** *gauss-jordan*  $A B \equiv \text{gauss-jordan-main } A B 0 0$

**lemma** *gauss-jordan-transform*: **assumes**  $A: A \in \text{carrier-mat } nr \ nc$  **and**  $B: B \in \text{carrier-mat } nr \ nc'$

**and** *res*: *gauss-jordan*  $(A :: 'a :: \text{field mat}) B = (A', B')$

**shows**  $\exists P \in \text{Units } (\text{ring-mat } \text{TYPE}('a) \ nr \ b). A' = P * A \wedge B' = P * B$

*<proof>*

**lemma** *gauss-jordan-carrier*: **assumes**  $A: (A :: 'a :: \text{field mat}) \in \text{carrier-mat } nr \ nc$

**and**  $B: B \in \text{carrier-mat } nr \ nc'$

**and** *res*: *gauss-jordan*  $A B = (A', B')$

**shows**  $A' \in \text{carrier-mat } nr \ nc \ B' \in \text{carrier-mat } nr \ nc'$

*<proof>*

**definition** *pivot-fun*  $:: 'a :: \{\text{zero, one}\} \text{ mat} \Rightarrow (\text{nat} \Rightarrow \text{nat}) \Rightarrow \text{nat} \Rightarrow \text{bool}$  **where**

*pivot-fun*  $A f \ nc \equiv \text{let } nr = \text{dim-row } A \text{ in}$

$(\forall i < nr. f i \leq nc \wedge$

$(f i < nc \longrightarrow A \ \S\ \$ (i, f i) = 1 \wedge (\forall i' < nr. i' \neq i \longrightarrow A \ \S\ \$ (i', f i) = 0)) \wedge$

$(\forall j < f i. A \ \S\ \$ (i, j) = 0) \wedge$

$(\text{Suc } i < nr \longrightarrow f (\text{Suc } i) > f i \vee f (\text{Suc } i) = nc))$

**lemma** *pivot-funI*: **assumes**  $d: \text{dim-row } A = nr$

**and**  $*$ :  $\bigwedge i. i < nr \Longrightarrow f i \leq nc$

$\bigwedge i \ j. i < nr \Longrightarrow j < f i \Longrightarrow A \ \S\ \$ (i, j) = 0$

$\bigwedge i. i < nr \Longrightarrow \text{Suc } i < nr \Longrightarrow f (\text{Suc } i) > f i \vee f (\text{Suc } i) = nc$

$\bigwedge i. i < nr \Longrightarrow f i < nc \Longrightarrow A \ \S\ \$ (i, f i) = 1$

$\bigwedge i \ i'. i < nr \Longrightarrow f i < nc \Longrightarrow i' < nr \Longrightarrow i' \neq i \Longrightarrow A \ \S\ \$ (i', f i) = 0$

**shows** *pivot-fun*  $A f \ nc$

*<proof>*

**lemma** *pivot-funD*: **assumes**  $d: \text{dim-row } A = nr$

**and**  $p: \text{pivot-fun } A f \ nc$

**shows**  $\bigwedge i. i < nr \Longrightarrow f i \leq nc$

$\bigwedge i \ j. i < nr \Longrightarrow j < f i \Longrightarrow A \ \S\ \$ (i, j) = 0$

$\bigwedge i. i < nr \Longrightarrow \text{Suc } i < nr \Longrightarrow f (\text{Suc } i) > f i \vee f (\text{Suc } i) = nc$

$\bigwedge i. i < nr \Longrightarrow f i < nc \Longrightarrow A \ \S\ \$ (i, f i) = 1$

$\bigwedge i \ i'. i < nr \Longrightarrow f i < nc \Longrightarrow i' < nr \Longrightarrow i' \neq i \Longrightarrow A \ \S\ \$ (i', f i) = 0$

*<proof>*

**lemma** *pivot-fun-multrow*: **assumes**  $p: \text{pivot-fun } A f \ jj$

**and**  $d: \text{dim-row } A = nr \ \text{dim-col } A = nc$

**and**  $fi: f i 0 = jj$

**and**  $jj: jj \leq nc$   
**shows**  $\text{pivot-fun } (\text{multrow } i0 \ a \ A) \ f \ jj$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{pivot-fun-swaprows}$ : **assumes**  $p: \text{pivot-fun } A \ f \ jj$   
**and**  $d: \text{dim-row } A = nr \ \text{dim-col } A = nc$   
**and**  $flk: f \ l = jj \ f \ k = jj$   
**and**  $nr: l < nr \ k < nr$   
**and**  $jj: jj \leq nc$   
**shows**  $\text{pivot-fun } (\text{swaprows } l \ k \ A) \ f \ jj$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{pivot-fun-eliminate-entries}$ : **assumes**  $p: \text{pivot-fun } A \ f \ jj$   
**and**  $d: \text{dim-row } A = nr \ \text{dim-col } A = nc$   
**and**  $fl: f \ l = jj$   
**and**  $nr: l < nr$   
**and**  $jj: jj \leq nc$   
**shows**  $\text{pivot-fun } (\text{eliminate-entries } vs \ A \ l \ j) \ f \ jj$   
 $\langle \text{proof} \rangle$

**definition**  $\text{row-echelon-form} :: 'a :: \{\text{zero,one}\} \ \text{mat} \Rightarrow \text{bool}$  **where**  
 $\text{row-echelon-form } A \equiv \exists \ f. \ \text{pivot-fun } A \ f \ (\text{dim-col } A)$

**lemma**  $\text{pivot-fun-init}$ :  $\text{pivot-fun } A \ (\lambda \ -. \ 0) \ 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{gauss-jordan-main-row-echelon}$ :  
**assumes**  
 $A \in \text{carrier-mat } nr \ nc$   
 $\text{gauss-jordan-main } A \ B \ i \ j = (A', B')$   
 $\text{pivot-fun } A \ f \ j$   
 $\bigwedge \ i'. \ i' < i \Longrightarrow f \ i' < j \ \bigwedge \ i'. \ i' \geq i \Longrightarrow f \ i' = j$   
 $i \leq nr \ j \leq nc$   
**shows**  $\text{row-echelon-form } A'$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{gauss-jordan-row-echelon}$ :  
**assumes**  $A: A \in \text{carrier-mat } nr \ nc$   
**and**  $\text{res}: \text{gauss-jordan } A \ B = (A', B')$   
**shows**  $\text{row-echelon-form } A'$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{pivot-bound}$ : **assumes**  $\text{dim}: \text{dim-row } A = nr$   
**and**  $\text{pivot}: \text{pivot-fun } A \ f \ n$   
**shows**  $i + j < nr \Longrightarrow f \ (i + j) = n \vee f \ (i + j) \geq j + f \ i$   
 $\langle \text{proof} \rangle$

**context**  
**fixes**  $\text{zero} :: 'a$

```

and A :: 'a mat
and nr nc :: nat
begin
function pivot-positions-main-gen :: nat ⇒ nat ⇒ (nat × nat) list where
  pivot-positions-main-gen i j = (
    if i < nr then
      if j < nc 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 [] ⟨proof⟩

termination ⟨proof⟩

declare pivot-positions-main-gen.simps[simp del]
end

context
  fixes A :: 'a :: semiring-1 mat
  and nr nc :: nat
begin

abbreviation pivot-positions-main ≡ pivot-positions-main-gen (0 :: 'a) A nr nc

lemma pivot-positions-main: assumes A: A ∈ carrier-mat nr nc
  and pivot: pivot-fun A f nc
  shows j ≤ f i ∨ i ≥ nr ⇒
    set (pivot-positions-main i j) = {(i', f i') | i'. i ≤ i' ∧ i' < nr} - UNIV ×
    {nc}
    ∧ distinct (map snd (pivot-positions-main i j))
    ∧ distinct (map fst (pivot-positions-main i j))
  ⟨proof⟩
end

lemma pivot-fun-zero-row-iff: assumes pivot: pivot-fun (A :: 'a :: semiring-1 mat)
  f nc
  and A: A ∈ carrier-mat nr nc
  and i: i < nr
  shows f i = nc ⟷ row A i = 0v nc
  ⟨proof⟩

definition pivot-positions-gen :: 'a ⇒ 'a mat ⇒ (nat × nat) list where
  pivot-positions-gen zer A ≡ pivot-positions-main-gen zer A (dim-row A) (dim-col
  A) 0 0

abbreviation pivot-positions :: 'a :: semiring-1 mat ⇒ (nat × nat) list where
  pivot-positions ≡ pivot-positions-gen 0

```

**lemmas** *pivot-positions-def* = *pivot-positions-gen-def*

**lemma** *pivot-positions*: **assumes**  $A \in \text{carrier-mat } nr \ nc$

**and** *pivot*: *pivot-fun*  $A \ f \ nc$

**shows**

$\text{set } (\text{pivot-positions } A) = \{(i, f \ i) \mid i. \ i < nr \wedge f \ i \neq nc\}$

$\text{distinct } (\text{map } \text{fst } (\text{pivot-positions } A))$

$\text{distinct } (\text{map } \text{snd } (\text{pivot-positions } A))$

$\text{length } (\text{pivot-positions } A) = \text{card } \{i. \ i < nr \wedge \text{row } A \ i \neq 0_v \ nc\}$

*<proof>*

**context**

**fixes** *uminus* :: 'a  $\Rightarrow$  'a

**and** *zero* :: 'a

**and** *one* :: 'a

**begin**

**definition** *non-pivot-base-gen* :: 'a *mat*  $\Rightarrow$  (nat  $\times$  nat)*list*  $\Rightarrow$  nat  $\Rightarrow$  'a *vec* **where**

*non-pivot-base-gen*  $A \ \text{pivots} \equiv \text{let } nr = \text{dim-row } A; \ nc = \text{dim-col } A;$

$\text{invers} = \text{map-of } (\text{map } \text{prod.swap } \text{pivots})$

$\text{in } (\lambda \ \text{qj}. \ \text{vec } nc \ (\lambda \ i.$

$\text{if } i = \text{qj} \ \text{then } \text{one} \ \text{else } (\text{case } \text{invers } i \ \text{of } \text{Some } j \Rightarrow \text{uminus } (A \ \$\$ \ (j, \text{qj})) \mid \text{None} \Rightarrow \text{zero}))$ )

**definition** *find-base-vectors-gen* :: 'a *mat*  $\Rightarrow$  'a *vec list* **where**

*find-base-vectors-gen*  $A \equiv$

*let*

$pp = \text{pivot-positions-gen } \text{zero } A;$

$\text{cands} = \text{filter } (\lambda \ j. \ j \notin \text{set } (\text{map } \text{snd } pp)) \ [0 \ ..< \ \text{dim-col } A]$

$\text{in } \text{map } (\text{non-pivot-base-gen } A \ pp) \ \text{cands}$

**end**

**abbreviation** *non-pivot-base*  $\equiv \text{non-pivot-base-gen } \text{uminus } 0 \ (1 :: 'a :: \text{comm-ring-1})$

**abbreviation** *find-base-vectors*  $\equiv \text{find-base-vectors-gen } \text{uminus } 0 \ (1 :: 'a :: \text{comm-ring-1})$

**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**

*find-base-vector*  $A \equiv$

*let*

$pp = \text{pivot-positions } A;$

$\text{cands} = \text{filter } (\lambda \ j. \ j \notin \text{set } (\text{map } \text{snd } pp)) \ [0 \ ..< \ \text{dim-col } A]$

$\text{in } \text{non-pivot-base } A \ pp \ (\text{hd } \text{cands})$

**context**

**fixes**  $A :: 'a :: \text{field } \text{mat}$  **and**  $nr \ nc :: \text{nat}$  **and**  $p :: \text{nat} \Rightarrow \text{nat}$

**assumes** *ref*: *row-echelon-form*  $A$

**and**  $A: A \in \text{carrier-mat } nr \ nc$   
**begin**

**lemma** *non-pivot-base*:

**defines**  $pp: pp \equiv \text{pivot-positions } A$   
**assumes**  $qj: qj < nc \ qj \notin \text{snd } ' \text{ set } pp$   
**shows**  $\text{non-pivot-base } A \ pp \ qj \in \text{carrier-vec } nc$   
 $\text{non-pivot-base } A \ pp \ qj \ \$ \ qj = 1$   
 $A *_v \ \text{non-pivot-base } A \ pp \ qj = 0_v \ nr$   
 $\bigwedge \ qj'. \ qj' < nc \implies \ qj' \notin \text{snd } ' \ \text{set } pp \implies \ qj \neq \ qj' \implies \ \text{non-pivot-base } A \ pp \ qj$   
 $\$ \ qj' = 0$   
 $\langle \text{proof} \rangle$

**lemma** *find-base-vector*: **assumes**  $\text{snd } ' \ \text{set } (\text{pivot-positions } A) \neq \{0 \ .. < nc\}$

**shows**  
 $\text{find-base-vector } A \in \text{carrier-vec } nc$   
 $\text{find-base-vector } A \neq 0_v \ nc$   
 $A *_v \ \text{find-base-vector } A = 0_v \ nr$   
 $\langle \text{proof} \rangle$   
**end**

**lemma** *row-echelon-form-imp-1-or-0-row*: **assumes**  $A: A \in \text{carrier-mat } n \ n$

**and**  $\text{row: row-echelon-form } A$   
**shows**  $A = 1_m \ n \ \vee \ (n > 0 \ \wedge \ \text{row } A \ (n - 1) = 0_v \ n)$   
 $\langle \text{proof} \rangle$

**context**

**fixes**  $A :: 'a :: \text{field mat}$  **and**  $n :: \text{nat}$  **and**  $p :: \text{nat} \implies \text{nat}$   
**assumes**  $\text{ref: row-echelon-form } A$   
**and**  $A: A \in \text{carrier-mat } n \ n$   
**and**  $1: A \neq 1_m \ n$   
**begin**

**lemma** *find-base-vector-not-1-pivot-positions*: **assumes**  $\text{snd } ' \ \text{set } (\text{pivot-positions } A) \neq \{0 \ .. < n\}$   
 $\langle \text{proof} \rangle$

**lemma** *find-base-vector-not-1*:

$\text{find-base-vector } A \in \text{carrier-vec } n$   
 $\text{find-base-vector } A \neq 0_v \ n$   
 $A *_v \ \text{find-base-vector } A = 0_v \ n$   
 $\langle \text{proof} \rangle$   
**end**

**lemma** *gauss-jordan*: **assumes**  $A: A \in \text{carrier-mat } nr \ nc$

**and**  $B: B \in \text{carrier-mat } nr \ nc2$   
**and**  $\text{gauss: gauss-jordan } A \ B = (C, D)$   
**shows**  $x \in \text{carrier-vec } nc \implies (A *_v \ x = 0_v \ nr) = (C *_v \ x = 0_v \ nr)$  (**is** -  $\implies$   
 $?l = ?r$ )

$X \in \text{carrier-mat } nc \ nc2 \implies (A * X = B) = (C * X = D)$  (**is** -  $\implies$  ?l2 = ?r2)  
 $C \in \text{carrier-mat } nr \ nc$   
 $D \in \text{carrier-mat } nr \ nc2$   
 <proof>

**definition** *gauss-jordan-single* :: 'a :: field mat  $\Rightarrow$  'a mat **where**  
*gauss-jordan-single* A = fst (gauss-jordan A (0<sub>m</sub> (dim-row A) 0))

**lemma** *gauss-jordan-single*: **assumes** A: A  $\in$  carrier-mat nr nc  
**and** gauss: *gauss-jordan-single* A = C  
**shows** x  $\in$  carrier-vec nc  $\implies$  (A \*<sub>v</sub> x = 0<sub>v</sub> nr) = (C \*<sub>v</sub> x = 0<sub>v</sub> nr)  
 $C \in \text{carrier-mat } nr \ nc$   
*row-echelon-form* C  
 $\exists P Q. C = P * A \wedge P \in \text{carrier-mat } nr \ nr \wedge Q \in \text{carrier-mat } nr \ nr \wedge P * Q = 1_m \ nr \wedge Q * P = 1_m \ nr$  (**is** ?ex)  
 <proof>

**lemma** *gauss-jordan-inverse-one-direction*:  
**assumes** A: A  $\in$  carrier-mat n n **and** B: B  $\in$  carrier-mat n nc  
**and** res: *gauss-jordan* A B = (1<sub>m</sub> n, B')  
**shows** A  $\in$  Units (ring-mat TYPE('a :: field) n b)  
 $B = 1_m \ n \implies A * B' = 1_m \ n \wedge B' * A = 1_m \ n$   
 <proof>

**lemma** *gauss-jordan-inverse-other-direction*:  
**assumes** AU: A  $\in$  Units (ring-mat TYPE('a :: field) n b) **and** B: B  $\in$  carrier-mat n nc  
**shows** fst (gauss-jordan A B) = 1<sub>m</sub> n  
 <proof>

**lemma** *gauss-jordan-compute-inverse*:  
**assumes** A: A  $\in$  carrier-mat n n  
**and** res: *gauss-jordan* A (1<sub>m</sub> n) = (1<sub>m</sub> n, B')  
**shows** A \* B' = 1<sub>m</sub> n B' \* A = 1<sub>m</sub> n B'  $\in$  carrier-mat n n  
 <proof>

**lemma** *gauss-jordan-check-invertable*: **assumes** A: A  $\in$  carrier-mat n n **and** B:  
 $B \in \text{carrier-mat } n \ nc$   
**shows** (A  $\in$  Units (ring-mat TYPE('a :: field) n b))  $\longleftrightarrow$  fst (gauss-jordan A B) = 1<sub>m</sub> n  
 (**is** ?l = ?r)  
 <proof>

**definition** *mat-inverse* :: 'a :: field mat  $\Rightarrow$  'a mat option **where**  
*mat-inverse* A = (if dim-row A = dim-col A then  
 let one = 1<sub>m</sub> (dim-row A) in



(case gauss-jordan  $A$  one of  
 $(B, C) \Rightarrow$  if  $B = \text{one}$  then  $\text{Some } C$  else  $\text{None}$ ) else  $\text{None}$ )

**lemma** *mat-inverse*: **assumes**  $A: A \in \text{carrier-mat } n \ n$   
**shows**  $\text{mat-inverse } A = \text{None} \Longrightarrow A \notin \text{Units } (\text{ring-mat } \text{TYPE}('a :: \text{field}) \ n \ b)$   
 $\text{mat-inverse } A = \text{Some } B \Longrightarrow A * B = 1_m \ n \wedge B * A = 1_m \ n \wedge B \in \text{carrier-mat}$   
 $n \ n$   
 $\langle \text{proof} \rangle$   
**end**

## 7 Code Generation for Basic Matrix Operations

In this theory we provide efficient implementations for the elementary row-transformations. 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* ::  $\text{nat} \Rightarrow \text{nat} \Rightarrow 'a \ \text{mat-impl} \Rightarrow 'a \ \text{mat-impl}$  **is**  
 $\lambda \ i \ j \ (nr, nc, A).$  if  $i < nr \wedge j < nr$  then  
 $\text{let } Ai = \text{IArray.sub } A \ i;$   
 $Aj = \text{IArray.sub } A \ j;$   
 $Arows = \text{IArray.list-of } A;$   
 $A' = \text{IArray.IArray } (Arows [i := Aj, j := Ai])$   
 $\text{in } (nr, nc, A')$   
 else  $(nr, nc, A)$   
 $\langle \text{proof} \rangle$

**lemma** [code]: *mat-swaprows*  $k \ l \ (\text{mat-impl } A) = (\text{let } nr = \text{dim-row-impl } A \ \text{in}$   
 if  $l < nr \wedge k < nr$  then  
 $\text{mat-impl } (\text{mat-swaprows-impl } k \ l \ A)$  else  $\text{Code.abort } (\text{STR } \text{"index out of bounds in mat-swaprows"})$   
 $(\lambda \ -. \ \text{mat-swaprows } k \ l \ (\text{mat-impl } A))$ ) (**is**  $?l = ?r$ )  
 $\langle \text{proof} \rangle$

**lift-definition** *mat-multrow-gen-impl* ::  $('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow \text{nat} \Rightarrow 'a \Rightarrow 'a \ \text{mat-impl}$   
 $\Rightarrow 'a \ \text{mat-impl}$  **is**  
 $\lambda \ \text{mul } k \ a \ (nr, nc, A).$  let  $Ak = \text{IArray.sub } A \ k;$   $Arows = \text{IArray.list-of } A;$   
 $Ak' = \text{IArray.IArray } (\text{map } (\text{mul } a) \ (\text{IArray.list-of } Ak));$   
 $A' = \text{IArray.IArray } (Arows [k := Ak'])$   
 $\text{in } (nr, nc, A')$   
 $\langle \text{proof} \rangle$

**lemma** [code]: *mat-multrow-gen mul k a (mat-impl A) = mat-impl (mat-multrow-gen-impl mul k a A)*  
 ⟨proof⟩

**lift-definition** *mat-addrow-gen-impl*

∴ ('a ⇒ 'a ⇒ 'a) ⇒ ('a ⇒ 'a ⇒ 'a) ⇒ 'a ⇒ nat ⇒ nat ⇒ 'a mat-impl ⇒ 'a mat-impl **is**

λ ad mul a k l (nr,nc,A). if l < nr then let Ak = IArray.sub A k; Al = IArray.sub A l;

Ak' = IArray.of-fun (λ i. ad (mul a (Al !! i)) (Ak !! i)) (min (IArray.length Ak) (IArray.length Al));

A' = IArray.of-fun (λ i. if i = k then Ak' else A !! i) (IArray.length A)

in (nr,nc,A') else (nr,nc,A)

⟨proof⟩

**lemma** *mat-addrow-gen-impl*[code]: *mat-addrow-gen ad mul a k l (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")*

*(λ -. mat-addrow-gen ad mul a k l (mat-impl A))) (is ?l = ?r)*

⟨proof⟩

**lemma** *gauss-jordan-main-code*[code]:

*gauss-jordan-main A B i j = (let nr = dim-row A; nc = dim-col A in*

*if i < nr ∧ j < nc then let aij = A \$\$ (i,j) in if aij = 0 then*

*(case [ i' . i' <- [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 = (λ i. 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 ia ij = inverse aij; A' = multrow i ia ij A; B' = multrow i ia ij B;*

*v = (λ i. A' \$\$ (i,j)) in gauss-jordan-main*

*(eliminate-entries v A' i j) (eliminate-entries v B' i j) (Suc i) (Suc j)*

*else (A,B)) (is ?l = ?r)*

⟨proof⟩

**end**

## 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*  $\Rightarrow$  '*a* :: *semiring-1*  $\Rightarrow$  '*a* *mat*  $\Rightarrow$  '*a* *mat* (*multcol*)  
**where**

*multcol* *k a A* = *mat* (*dim-row A*) (*dim-col A*)  
 $(\lambda (i,j). \text{if } k = j \text{ then } a * A \text{ \$\$ } (i,j) \text{ else } A \text{ \$\$ } (i,j))$

**definition** *mat-swapcols* :: *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  '*a* *mat*  $\Rightarrow$  '*a* *mat* (*swapcols*)**where**

*swapcols* *k l A* = *mat* (*dim-row A*) (*dim-col A*)  
 $(\lambda (i,j). \text{if } k = j \text{ then } A \text{ \$\$ } (i,l) \text{ else if } l = j \text{ then } A \text{ \$\$ } (i,k) \text{ else } A \text{ \$\$ } (i,j))$

**definition** *mat-addcol-vec* :: *nat*  $\Rightarrow$  '*a* :: *plus vec*  $\Rightarrow$  '*a* *mat*  $\Rightarrow$  '*a* *mat* **where**

*mat-addcol-vec* *k v A* = *mat* (*dim-row A*) (*dim-col A*)  
 $(\lambda (i,j). \text{if } k = j \text{ then } v \$ i + A \text{ \$\$ } (i,j) \text{ else } A \text{ \$\$ } (i,j))$

**definition** *mat-addcol* :: '*a* :: *semiring-1*  $\Rightarrow$  *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  '*a* *mat*  $\Rightarrow$  '*a* *mat* (*addcol*)  
**where**

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

**lemma** *index-mat-multcol[simp]*:

$i < \text{dim-row } A \Longrightarrow j < \text{dim-col } A \Longrightarrow \text{multcol } k \ a \ A \ \text{\$\$ } (i,j) = (\text{if } k = j \text{ then } a * A \ \text{\$\$ } (i,j) \text{ else } A \ \text{\$\$ } (i,j))$   
 $i < \text{dim-row } A \Longrightarrow j < \text{dim-col } A \Longrightarrow \text{multcol } j \ a \ A \ \text{\$\$ } (i,j) = a * A \ \text{\$\$ } (i,j)$   
 $i < \text{dim-row } A \Longrightarrow j < \text{dim-col } A \Longrightarrow k \neq j \Longrightarrow \text{multcol } k \ a \ A \ \text{\$\$ } (i,j) = A \ \text{\$\$ } (i,j)$   
 $\text{dim-row } (\text{multcol } k \ a \ A) = \text{dim-row } A \ \text{dim-col } (\text{multcol } k \ a \ A) = \text{dim-col } A$   
*<proof>*

**lemma** *index-mat-swapcols[simp]*:

$i < \text{dim-row } A \Longrightarrow j < \text{dim-col } A \Longrightarrow \text{swapcols } k \ l \ A \ \text{\$\$ } (i,j) = (\text{if } k = j \text{ then } A \ \text{\$\$ } (i,l) \text{ else if } l = j \text{ then } A \ \text{\$\$ } (i,k) \text{ else } A \ \text{\$\$ } (i,j))$   
 $\text{dim-row } (\text{swapcols } k \ l \ A) = \text{dim-row } A \ \text{dim-col } (\text{swapcols } k \ l \ A) = \text{dim-col } A$   
*<proof>*

**lemma** *index-mat-addcol[simp]*:

$i < \text{dim-row } A \Longrightarrow j < \text{dim-col } A \Longrightarrow \text{addcol } a \ k \ l \ A \ \text{\$\$ } (i,j) = (\text{if } k = j \text{ then } a * A \ \text{\$\$ } (i,l) + A \ \text{\$\$ } (i,j) \text{ else } A \ \text{\$\$ } (i,j))$   
 $i < \text{dim-row } A \Longrightarrow j < \text{dim-col } A \Longrightarrow \text{addcol } a \ j \ l \ A \ \text{\$\$ } (i,j) = a * A \ \text{\$\$ } (i,l) + A \ \text{\$\$ } (i,j)$   
 $i < \text{dim-row } A \Longrightarrow j < \text{dim-col } A \Longrightarrow k \neq j \Longrightarrow \text{addcol } a \ k \ l \ A \ \text{\$\$ } (i,j) = A \ \text{\$\$ } (i,j)$   
 $\text{dim-row } (\text{addcol } a \ k \ l \ A) = \text{dim-row } A \ \text{dim-col } (\text{addcol } a \ k \ l \ A) = \text{dim-col } A$   
*<proof>*

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

**lemma** *col-addrow*:

$l \neq i \implies i < n \implies \text{col}(\text{addrow-mat } n \ a \ k \ l) \ i = \text{unit-vec } n \ i$   
 $k < n \implies l < n \implies \text{col}(\text{addrow-mat } n \ a \ k \ l) \ l = a \cdot_v \text{unit-vec } n \ k + \text{unit-vec } n \ l$   
 ⟨proof⟩

**lemma** *col-addcol[simp]*:

$k < \text{dim-col } A \implies l < \text{dim-col } A \implies \text{col}(\text{addcol } a \ k \ l \ A) \ k = a \cdot_v \text{col } A \ l + \text{col } A \ k$   
 ⟨proof⟩

**lemma** *addcol-mat*: **assumes**  $A: A \in \text{carrier-mat } nr \ n$

**and**  $k: k < n$

**shows**  $\text{addcol}(a :: 'a :: \text{comm-semiring-1}) \ l \ k \ A = A * \text{addrow-mat } n \ a \ k \ l$

⟨proof⟩

**lemma** *col-multrow*:  $k \neq i \implies i < n \implies \text{col}(\text{multrow-mat } n \ k \ a) \ i = \text{unit-vec } n \ i$

$k < n \implies \text{col}(\text{multrow-mat } n \ k \ a) \ k = a \cdot_v \text{unit-vec } n \ k$

⟨proof⟩

**lemma** *multcol-mat*: **assumes**  $A: (A :: 'a :: \text{comm-ring-1 mat}) \in \text{carrier-mat } nr \ n$

**shows**  $\text{multcol } k \ a \ A = A * \text{multrow-mat } n \ k \ a$

⟨proof⟩

**lemma** *col-swaprows*:

$l < n \implies \text{col}(\text{swaprows-mat } n \ l \ l) \ l = \text{unit-vec } n \ l$

$i \neq k \implies i \neq l \implies i < n \implies \text{col}(\text{swaprows-mat } n \ k \ l) \ i = \text{unit-vec } n \ i$

$k < n \implies l < n \implies \text{col}(\text{swaprows-mat } n \ k \ l) \ l = \text{unit-vec } n \ k$

$k < n \implies l < n \implies \text{col}(\text{swaprows-mat } n \ k \ l) \ k = \text{unit-vec } n \ l$

⟨proof⟩

**lemma** *swapcols-mat*: **assumes**  $A: A \in \text{carrier-mat } nr \ n$  **and**  $k: k < n \ l < n$

**shows**  $\text{swapcols } k \ l \ A = A * \text{swaprows-mat } n \ k \ l$

⟨proof⟩

Combining row and column-operations yields similarity transformations.

**definition** *add-col-sub-row* ::  $'a :: \text{ring-1} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \ \text{mat} \Rightarrow 'a \ \text{mat}$  **where**

$\text{add-col-sub-row } a \ k \ l \ A = \text{addrow } (- \ a) \ k \ l \ (\text{addcol } a \ l \ k \ A)$

**definition** *mult-col-div-row* ::  $'a :: \text{field} \Rightarrow \text{nat} \Rightarrow 'a \ \text{mat} \Rightarrow 'a \ \text{mat}$  **where**

$\text{mult-col-div-row } a \ k \ A = \text{multrow } k \ (\text{inverse } a) \ (\text{multcol } k \ a \ A)$

**definition** *swap-cols-rows* ::  $\text{nat} \Rightarrow \text{nat} \Rightarrow 'a \ \text{mat} \Rightarrow 'a \ \text{mat}$  **where**

$\text{swap-cols-rows } k \ l \ A = \text{swaprows } k \ l \ (\text{swapcols } k \ l \ A)$

**lemma** *add-col-sub-row-carrier[simp]*:

$\text{dim-row } (\text{add-col-sub-row } a \ k \ l \ A) = \text{dim-row } A$

$\dim\text{-col } (\text{add-col-sub-row } a \ k \ l \ A) = \dim\text{-col } A$   
 $A \in \text{carrier-mat } n \ n \implies \text{add-col-sub-row } a \ k \ l \ A \in \text{carrier-mat } n \ n$   
 ⟨proof⟩

**lemma** *add-col-sub-index-row*[simp]:

$i < \dim\text{-row } A \implies i < \dim\text{-col } A \implies j < \dim\text{-row } A \implies j < \dim\text{-col } A \implies l < \dim\text{-row } A$   
 $\implies \text{add-col-sub-row } a \ k \ l \ A \ \$\$ (i,j) = (\text{if}$   
 $\quad i = k \wedge j = l \text{ then } A \ \$\$ (i, j) + a * A \ \$\$ (i, i) - a * a * A \ \$\$ (j, i) - a * A \ \$\$ (j, j) \text{ else if}$   
 $\quad i = k \wedge j \neq l \text{ then } A \ \$\$ (i, j) - a * A \ \$\$ (l, j) \text{ else if}$   
 $\quad i \neq k \wedge j = l \text{ then } A \ \$\$ (i, j) + a * A \ \$\$ (i, k) \text{ else } A \ \$\$ (i,j))$   
 ⟨proof⟩

**lemma** *mult-col-div-index-row*[simp]:

$i < \dim\text{-row } A \implies i < \dim\text{-col } A \implies j < \dim\text{-row } A \implies j < \dim\text{-col } A \implies a \neq 0$   
 $\implies \text{mult-col-div-row } a \ k \ A \ \$\$ (i,j) = (\text{if}$   
 $\quad i = k \wedge j \neq i \text{ then inverse } a * A \ \$\$ (i, j) \text{ else if}$   
 $\quad j = k \wedge j \neq i \text{ then } a * A \ \$\$ (i, j) \text{ else } A \ \$\$ (i,j))$   
 ⟨proof⟩

**lemma** *mult-col-div-row-carrier*[simp]:

$\dim\text{-row } (\text{mult-col-div-row } a \ k \ A) = \dim\text{-row } A$   
 $\dim\text{-col } (\text{mult-col-div-row } a \ k \ A) = \dim\text{-col } A$   
 $A \in \text{carrier-mat } n \ n \implies \text{mult-col-div-row } a \ k \ A \in \text{carrier-mat } n \ n$   
 ⟨proof⟩

**lemma** *swap-cols-rows-carrier*[simp]:

$\dim\text{-row } (\text{swap-cols-rows } k \ l \ A) = \dim\text{-row } A$   
 $\dim\text{-col } (\text{swap-cols-rows } k \ l \ A) = \dim\text{-col } A$   
 $A \in \text{carrier-mat } n \ n \implies \text{swap-cols-rows } k \ l \ A \in \text{carrier-mat } n \ n$   
 ⟨proof⟩

**lemma** *swap-cols-rows-index*[simp]:

$i < \dim\text{-row } A \implies i < \dim\text{-col } A \implies j < \dim\text{-row } A \implies j < \dim\text{-col } A \implies a < \dim\text{-row } A \implies b < \dim\text{-row } A$   
 $\implies \text{swap-cols-rows } a \ b \ A \ \$\$ (i,j) = A \ \$\$ (\text{if } i = a \text{ then } b \text{ else if } i = b \text{ then } a \text{ else } i,$   
 $\quad \text{if } j = a \text{ then } b \text{ else if } j = b \text{ then } a \text{ else } j)$   
 ⟨proof⟩

**lemma** *add-col-sub-row-similar*: **assumes**  $A: A \in \text{carrier-mat } n \ n$  **and**  $kl: k < n \ l < n \ k \neq l$

**shows** *similar-mat*  $(\text{add-col-sub-row } a \ k \ l \ A) (A :: 'a :: \text{comm-ring-1 mat})$   
 ⟨proof⟩

**lemma** *mult-col-div-row-similar*: **assumes**  $A: A \in \text{carrier-mat } n \ n$  **and**  $ak: k < n \ a \neq 0$

**shows** *similar-mat (mult-col-div-row a k A) A*  
 ⟨*proof*⟩

**lemma** *swap-cols-rows-similar*: **assumes** *A: A ∈ carrier-mat n n and kl: k < n l < n*  
**shows** *similar-mat (swap-cols-rows k l A) A*  
 ⟨*proof*⟩

**lemma** *swapcols-carrier[simp]*: *(swapcols l k A ∈ carrier-mat n m) = (A ∈ carrier-mat n m)*  
 ⟨*proof*⟩

**fun** *swap-row-to-front* :: *'a mat ⇒ nat ⇒ 'a mat where*  
*swap-row-to-front A 0 = A*  
 | *swap-row-to-front A (Suc I) = swap-row-to-front (swaprrows I (Suc I) A) I*

**fun** *swap-col-to-front* :: *'a mat ⇒ nat ⇒ '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 ∈ carrier-mat n m ⇒ I < n ⇒ swap-row-to-front A I =*  
*mat n m (λ (i,j). if i = 0 then A \$\$ (I,j)*  
*else if i ≤ I then A \$\$ (i - 1, j) else A \$\$ (i,j))*  
 ⟨*proof*⟩

**lemma** *swap-col-to-front-result*: *A ∈ carrier-mat n m ⇒ J < m ⇒ swap-col-to-front A J =*  
*mat n m (λ (i,j). if j = 0 then A \$\$ (i,J)*  
*else if j ≤ J then A \$\$ (i, j-1) else A \$\$ (i,j))*  
 ⟨*proof*⟩

**lemma** *swapcols-is-transp-swap-rows*: **assumes** *A: A ∈ carrier-mat n m k < m l < m*  
**shows** *swapcols k l A = transpose-mat (swaprrows k l (transpose-mat A))*  
 ⟨*proof*⟩

**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**

*det* A = (if dim-row A = dim-col A then  $(\sum p \in \{p. p \text{ permutes } \{0 \dots \dim\text{-row } A\}\})$ .

*signof* p \*  $(\prod i = 0 \dots \dim\text{-row } A. A \text{ \$\$ } (i, p \ i))$  else 0)

**lemma**(in *ring-hom*) *hom-signof*[*simp*]: *hom* (*signof* p) = *signof* p

*<proof>*

**lemma**(in *comm-ring-hom*) *hom-det*[*simp*]: *det* (*map-mat* *hom* A) = *hom* (*det* A)

*<proof>*

**lemma** *det-def'*: A  $\in$  *carrier-mat* n n  $\implies$

*det* A =  $(\sum p \in \{p. p \text{ permutes } \{0 \dots n\}\})$ .

*signof* p \*  $(\prod i = 0 \dots n. A \text{ \$\$ } (i, p \ i))$  *<proof>*

**lemma** *det-smult*[*simp*]: *det* (a  $\cdot_m$  A) = a  $\wedge$  dim-col A \* *det* A

*<proof>*

**lemma** *det-transpose*: **assumes** A: A  $\in$  *carrier-mat* n n

**shows** *det* (*transpose-mat* A) = *det* A

*<proof>*

**lemma** *det-col*:

**assumes** A: A  $\in$  *carrier-mat* n n

**shows** *det* A =  $(\sum p \mid p \text{ permutes } \{0 \dots n\}. \text{signof } p * (\prod j < n. A \text{ \$\$ } (p \ j, j)))$

(is - = (sum ( $\lambda p. - * ?\text{prod } p$ ) ?P))

*<proof>*

**lemma** *mat-det-left-def*: **assumes** A: A  $\in$  *carrier-mat* n n

**shows** *det* A =  $(\sum p \in \{p. p \text{ permutes } \{0 \dots \dim\text{-row } A\}\}. \text{signof } p * (\prod i = 0 \dots \dim\text{-row } A. A \text{ \$\$ } (p \ i, i)))$

*<proof>*

**lemma** *det-upper-triangular*:

**assumes** *ut*: *upper-triangular* A

**and** *m*: A  $\in$  *carrier-mat* n n

**shows**  $\det A = \text{prod-list } (\text{diag-mat } A)$   
 ⟨proof⟩

**lemma** *det-one[simp]*:  $\det (1_m \ n) = 1$   
 ⟨proof⟩

**lemma** *det-zero[simp]*: **assumes**  $n > 0$  **shows**  $\det (0_m \ n \ n) = 0$   
 ⟨proof⟩

**lemma** *det-dim-zero[simp]*:  $A \in \text{carrier-mat } 0 \ 0 \implies \det A = 1$   
 ⟨proof⟩

**lemma** *det-lower-triangular*:  
**assumes**  $ld: \bigwedge i \ j. i < j \implies j < n \implies A \ \$\$ (i, j) = 0$   
**and**  $m: A \in \text{carrier-mat } n \ n$   
**shows**  $\det A = \text{prod-list } (\text{diag-mat } A)$   
 ⟨proof⟩

**lemma** *det-permute-rows*: **assumes**  $A: A \in \text{carrier-mat } n \ n$   
**and**  $p: p \text{ permutes } \{0 \ ..< (n \ :: \ \text{nat})\}$   
**shows**  $\det (\text{mat } n \ n \ (\lambda (i, j). A \ \$\$ (p \ i, j))) = \text{signof } p * \det A$   
 ⟨proof⟩

**lemma** *det-multrow-mat*: **assumes**  $k: k < n$   
**shows**  $\det (\text{multrow-mat } n \ k \ a) = a$   
 ⟨proof⟩

**lemma** *swap-rows-mat-eq-permute*:  
 $k < n \implies l < n \implies \text{swaprows-mat } n \ k \ l = \text{mat } n \ n \ (\lambda(i, j). 1_m \ n \ \$\$ (\text{transpose } k \ l \ i, j))$   
 ⟨proof⟩

**lemma** *det-swaprows-mat*: **assumes**  $k: k < n$  **and**  $l: l < n$  **and**  $kl: k \neq l$   
**shows**  $\det (\text{swaprows-mat } n \ k \ l) = - 1$   
 ⟨proof⟩

**lemma** *det-addrow-mat*:  
**assumes**  $l: k \neq l$   
**shows**  $\det (\text{addrow-mat } n \ a \ k \ l) = 1$   
 ⟨proof⟩

The following proof is new, as it does not use  $2 \neq 0$  as in Multivariate-Analysis.

**lemma** *det-identical-rows*:  
**assumes**  $A: A \in \text{carrier-mat } n \ n$   
**and**  $ij: i \neq j$   
**and**  $i: i < n$  **and**  $j: j < n$   
**and**  $r: \text{row } A \ i = \text{row } A \ j$   
**shows**  $\det A = 0$



*<proof>*

**lemma** *det-row-0*: **assumes**  $k: k < n$   
**and**  $c: c \in \{0 \dots n\} \rightarrow \text{carrier-vec } n$   
**shows**  $\det(\text{mat}_r \ n \ n \ (\lambda i. \text{if } i = k \text{ then } 0_v \ n \ \text{else } c \ i)) = 0$   
*<proof>*

**lemma** *det-row-add*:  
**assumes**  $abc: a \ k \in \text{carrier-vec } n \ b \ k \in \text{carrier-vec } n \ c \in \{0 \dots n\} \rightarrow \text{carrier-vec } n$   
**and**  $k: k < n$   
**shows**  $\det(\text{mat}_r \ n \ n \ (\lambda i. \text{if } i = k \text{ then } a \ i + b \ i \ \text{else } c \ i)) =$   
 $\det(\text{mat}_r \ n \ n \ (\lambda i. \text{if } i = k \text{ then } a \ i \ \text{else } c \ i)) +$   
 $\det(\text{mat}_r \ n \ n \ (\lambda i. \text{if } i = k \text{ then } b \ i \ \text{else } c \ i))$   
(**is** *?lhs = ?rhs*)  
*<proof>*

**lemma** *det-linear-row-finsum*:  
**assumes**  $fS: \text{finite } S$  **and**  $c: c \in \{0 \dots n\} \rightarrow \text{carrier-vec } n$  **and**  $k: k < n$   
**and**  $a: a \ k \in S \rightarrow \text{carrier-vec } n$   
**shows**  $\det(\text{mat}_r \ n \ n \ (\lambda i. \text{if } i = k \text{ then } \text{finsum-vec } \text{TYPE}('a :: \text{comm-ring-1}) \ n$   
 $(a \ i) \ S \ \text{else } c \ i)) =$   
 $\text{sum } (\lambda j. \det(\text{mat}_r \ n \ n \ (\lambda i. \text{if } i = k \text{ then } a \ i \ j \ \text{else } c \ i))) \ S$   
*<proof>*

**lemma** *det-linear-rows-finsum-lemma*:  
**assumes**  $fS: \text{finite } S$   
**and**  $fT: \text{finite } T$  **and**  $c: c \in \{0 \dots n\} \rightarrow \text{carrier-vec } n$   
**and**  $T: T \subseteq \{0 \dots n\}$   
**and**  $a: a \in T \rightarrow S \rightarrow \text{carrier-vec } n$   
**shows**  $\det(\text{mat}_r \ n \ n \ (\lambda i. \text{if } i \in T \text{ then } \text{finsum-vec } \text{TYPE}('a :: \text{comm-ring-1}) \ n$   
 $(a \ i) \ S \ \text{else } c \ i)) =$   
 $\text{sum } (\lambda f. \det(\text{mat}_r \ n \ n \ (\lambda i. \text{if } i \in T \text{ then } a \ i \ (f \ i) \ \text{else } c \ i)))$   
 $\{f. (\forall i \in T. f \ i \in S) \wedge (\forall i. i \notin T \longrightarrow f \ i = i)\}$   
*<proof>*

**lemma** *det-linear-rows-sum*:  
**assumes**  $fS: \text{finite } S$   
**and**  $a: a \in \{0 \dots n\} \rightarrow S \rightarrow \text{carrier-vec } n$   
**shows**  $\det(\text{mat}_r \ n \ n \ (\lambda i. \text{finsum-vec } \text{TYPE}('a :: \text{comm-ring-1}) \ n \ (a \ i) \ S)) =$   
 $\text{sum } (\lambda f. \det(\text{mat}_r \ n \ n \ (\lambda i. a \ i \ (f \ i))))$   
 $\{f. (\forall i \in \{0 \dots n\}. f \ i \in S) \wedge (\forall i. i \notin \{0 \dots n\} \longrightarrow f \ i = i)\}$   
*<proof>*

**lemma** *det-rows-mul*:  
**assumes**  $a: a \in \{0 \dots n\} \rightarrow \text{carrier-vec } n$   
**shows**  $\det(\text{mat}_r \ n \ n \ (\lambda i. c \ i \cdot_v \ a \ i)) =$

$\text{prod } c \{0..<n\} * \text{det}(\text{mat}_r \ n \ n \ (\lambda \ i. \ a \ i))$   
 $\langle \text{proof} \rangle$

**lemma** *mat-mul-finsum-alt*:

**assumes**  $A: A \in \text{carrier-mat } nr \ n$  **and**  $B: B \in \text{carrier-mat } n \ nc$   
**shows**  $A * B = \text{mat}_r \ nr \ nc \ (\lambda \ i. \ \text{finsum-vec } \text{TYPE}('a :: \text{semiring-0}) \ nc \ (\lambda k. \ A$   
 $\text{\$\$ } (i,k) \cdot_v \ \text{row } B \ k) \ \{0 \ ..< n\})$   
 $\langle \text{proof} \rangle$

**lemma** *det-mult*:

**assumes**  $A: A \in \text{carrier-mat } n \ n$  **and**  $B: B \in \text{carrier-mat } n \ n$   
**shows**  $\text{det} (A * B) = \text{det } A * \text{det} (B :: 'a :: \text{comm-ring-1 } \text{mat})$   
 $\langle \text{proof} \rangle$

**lemma** *unit-imp-det-non-zero*: **assumes**  $A \in \text{Units} (\text{ring-mat } \text{TYPE}('a :: \text{comm-ring-1})$   
 $n \ b)$

**shows**  $\text{det } A \neq 0$   
 $\langle \text{proof} \rangle$

The following proof is based on the Gauss-Jordan algorithm.

**lemma** *det-non-zero-imp-unit*: **assumes**  $A: A \in \text{carrier-mat } n \ n$

**and**  $dA: \text{det } A \neq (0 :: 'a :: \text{field})$   
**shows**  $A \in \text{Units} (\text{ring-mat } \text{TYPE}('a) \ n \ b)$   
 $\langle \text{proof} \rangle$

**lemma** *mat-mult-left-right-inverse*: **assumes**  $A: (A :: 'a :: \text{field } \text{mat}) \in \text{carrier-mat}$   
 $n \ n$

**and**  $B: B \in \text{carrier-mat } n \ n$  **and**  $AB: A * B = 1_m \ n$   
**shows**  $B * A = 1_m \ n$   
 $\langle \text{proof} \rangle$

**lemma** *det-zero-imp-zero-row*: **assumes**  $A: (A :: 'a :: \text{field } \text{mat}) \in \text{carrier-mat } n$   
 $n$

**and**  $\text{det}: \text{det } A = 0$   
**shows**  $\exists P. P \in \text{Units} (\text{ring-mat } \text{TYPE}('a) \ n \ b) \wedge \text{row} (P * A) (n - 1) = 0_v$   
 $n \wedge 0 < n$   
 $\wedge \text{row-echelon-form} (P * A)$   
 $\langle \text{proof} \rangle$

**lemma** *det-0-iff-vec-prod-zero-field*: **assumes**  $A: (A :: 'a :: \text{field } \text{mat}) \in \text{carrier-mat}$   
 $n \ n$

**shows**  $\text{det } A = 0 \iff (\exists v. v \in \text{carrier-vec } n \wedge v \neq 0_v \ n \wedge A *_v v = 0_v \ n)$  (**is**  
 $?l = (\exists v. ?P \ v)$ )  
 $\langle \text{proof} \rangle$

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**  $A: (A :: 'a :: \text{idom } \text{mat}) \in \text{carrier-mat } n$

$n$   
**shows**  $\det A = 0 \iff (\exists v. v \in \text{carrier-vec } n \wedge v \neq 0_v \ n \wedge A *_{v} v = 0_v \ n)$   
 <proof>

**lemma** *det-0-negate*: **assumes**  $A: (A :: 'a :: \text{field mat}) \in \text{carrier-mat } n \ n$   
**shows**  $(\det (- A) = 0) = (\det A = 0)$   
 <proof>

**lemma** *det-multrow*:  
**assumes**  $k: k < n$  **and**  $A: A \in \text{carrier-mat } n \ n$   
**shows**  $\det (\text{multrow } k \ a \ A) = a * \det A$   
 <proof>

**lemma** *det-multrow-div*:  
**assumes**  $k: k < n$  **and**  $A: A \in \text{carrier-mat } n \ n$  **and**  $a0: a \neq 0$   
**shows**  $\det (\text{multrow } k \ a \ A :: 'a :: \text{idom-divide mat}) \ \text{div } a = \det A$   
 <proof>

**lemma** *det-addrow*:  
**assumes**  $l: l < n$  **and**  $k: k \neq l$  **and**  $A: A \in \text{carrier-mat } n \ n$   
**shows**  $\det (\text{addrow } a \ k \ l \ A) = \det A$   
 <proof>

**lemma** *det-swaprows*:  
**assumes**  $*$ :  $k < n$   $l < n$  **and**  $k: k \neq l$  **and**  $A: A \in \text{carrier-mat } n \ n$   
**shows**  $\det (\text{swaprows } k \ l \ A) = - \det A$   
 <proof>

**lemma** *det-similar*: **assumes** *similar-mat*  $A \ B$   
**shows**  $\det A = \det B$   
 <proof>

**lemma** *det-four-block-mat-upper-right-zero-col*: **assumes**  $A1: A1 \in \text{carrier-mat } n \ n$   
**and**  $A20: A2 = (0_m \ n \ 1)$  **and**  $A3: A3 \in \text{carrier-mat } 1 \ n$   
**and**  $A4: A4 \in \text{carrier-mat } 1 \ 1$   
**shows**  $\det (\text{four-block-mat } A1 \ A2 \ A3 \ A4) = \det A1 * \det A4$  (**is**  $\det ?A = -$ )  
 <proof>

**lemma** *det-swap-initial-rows*: **assumes**  $A: A \in \text{carrier-mat } m \ m$   
**and**  $lt: k + n \leq m$   
**shows**  $\det A = (- 1) \wedge (k * n) * \det (\text{mat } m \ m \ (\lambda(i, j). A \ \$\$ (\text{if } i < n \ \text{then } i + k \ \text{else if } i < k + n \ \text{then } i - n \ \text{else } i, j)))$   
 <proof>

**lemma** *det-swap-rows*: **assumes**  $A: A \in \text{carrier-mat } (k + n) \ (k + n)$   
**shows**  $\det A = (-1) \wedge (k * n) * \det (\text{mat } (k + n) \ (k + n) \ (\lambda(i, j). A \ \$\$ (\text{if } i < k \ \text{then } i + n \ \text{else } i - k, j)))$

*<proof>*

**lemma** *det-swap-final-rows*: **assumes**  $A: A \in \text{carrier-mat } m \ m$

**and**  $m: m = l + k + n$

**shows**  $\det A = (-1)^{\wedge(k * n)} *$

$\det (\text{mat } m \ m \ (\lambda(i, j). A \ \$\$ (\text{if } i < l \text{ then } i \text{ else if } i < l + n \text{ then } i + k \text{ else } i - n, j)))$

**(is - = - \* det ?M)**

*<proof>*

**lemma** *det-swap-final-cols*: **assumes**  $A: A \in \text{carrier-mat } m \ m$

**and**  $m: m = l + k + n$

**shows**  $\det A = (-1)^{\wedge(k * n)} *$

$\det (\text{mat } m \ m \ (\lambda(i, j). A \ \$\$ (i, \text{if } j < l \text{ then } j \text{ else if } j < l + n \text{ then } j + k \text{ else } j - n)))$

*<proof>*

**lemma** *det-swap-initial-cols*: **assumes**  $A: A \in \text{carrier-mat } m \ m$

**and**  $lt: k + n \leq m$

**shows**  $\det A = (-1)^{\wedge(k * n)} *$

$\det (\text{mat } m \ m \ (\lambda(i, j). A \ \$\$ (i, \text{if } j < n \text{ then } j + k \text{ else if } j < k + n \text{ then } j - n \text{ else } j)))$

*<proof>*

**lemma** *det-swap-cols*: **assumes**  $A: A \in \text{carrier-mat } (k + n) \ (k + n)$

**shows**  $\det A = (-1)^{\wedge(k * n)} * \det (\text{mat } (k + n) \ (k + n) \ (\lambda(i, j).$

$A \ \$\$ (i, (\text{if } j < k \text{ then } j + n \text{ else } j - k))))$  **(is - = - \* det ?B)**

*<proof>*

**lemma** *det-four-block-mat-upper-right-zero*: **fixes**  $A1 :: 'a :: \text{idom mat}$

**assumes**  $A1: A1 \in \text{carrier-mat } n \ n$

**and**  $A20: A2 = (0_m \ n \ m)$  **and**  $A3: A3 \in \text{carrier-mat } m \ n$

**and**  $A4: A4 \in \text{carrier-mat } m \ m$

**shows**  $\det (\text{four-block-mat } A1 \ A2 \ A3 \ A4) = \det A1 * \det A4$

*<proof>*

**lemma** *det-swapcols*:

**assumes**  $*$ :  $k < n \ l < n \ k \neq l$  **and**  $A: A \in \text{carrier-mat } n \ n$

**shows**  $\det (\text{swapcols } k \ l \ A) = - \det A$

*<proof>*

**lemma** *swap-row-to-front-det*:  $A \in \text{carrier-mat } n \ n \implies I < n \implies \det (\text{swap-row-to-front } A \ I)$

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

*<proof>*

**lemma** *swap-col-to-front-det*:  $A \in \text{carrier-mat } n \ n \implies I < n \implies \det (\text{swap-col-to-front } A \ I)$

=  $(-1)^{\wedge I} * \det A$   
 ⟨proof⟩

**lemma** *swap-row-to-front-four-block*: **assumes**  $A1: A1 \in \text{carrier-mat } n \ m1$   
**and**  $A2: A2 \in \text{carrier-mat } n \ m2$   
**and**  $A3: A3 \in \text{carrier-mat } 1 \ m1$   
**and**  $A4: A4 \in \text{carrier-mat } 1 \ m2$   
**shows** *swap-row-to-front* (*four-block-mat*  $A1 \ A2 \ A3 \ A4$ )  $n = \text{four-block-mat } A3 \ A4 \ A1 \ A2$   
 ⟨proof⟩

**lemma** *swap-col-to-front-four-block*: **assumes**  $A1: A1 \in \text{carrier-mat } n1 \ m$   
**and**  $A2: A2 \in \text{carrier-mat } n1 \ 1$   
**and**  $A3: A3 \in \text{carrier-mat } n2 \ m$   
**and**  $A4: A4 \in \text{carrier-mat } n2 \ 1$   
**shows** *swap-col-to-front* (*four-block-mat*  $A1 \ A2 \ A3 \ A4$ )  $m = \text{four-block-mat } A2 \ A1 \ A4 \ A3$   
 ⟨proof⟩

**lemma** *det-four-block-mat-lower-right-zero-col*: **assumes**  $A1: A1 \in \text{carrier-mat } 1 \ n$   
**and**  $A2: A2 \in \text{carrier-mat } 1 \ 1$   
**and**  $A3: A3 \in \text{carrier-mat } n \ n$   
**and**  $A40: A4 = (0_m \ n \ 1)$   
**shows** *det* (*four-block-mat*  $A1 \ A2 \ A3 \ A4$ ) =  $(-1)^{\wedge n} * \det A2 * \det A3$  (**is det**  $?A = -$ )  
 ⟨proof⟩

**lemma** *det-four-block-mat-lower-left-zero-col*: **assumes**  $A1: A1 \in \text{carrier-mat } 1 \ 1$   
**and**  $A2: A2 \in \text{carrier-mat } 1 \ n$   
**and**  $A30: A3 = (0_m \ n \ 1)$   
**and**  $A4: A4 \in \text{carrier-mat } n \ n$   
**shows** *det* (*four-block-mat*  $A1 \ A2 \ A3 \ A4$ ) = *det*  $A1 * \det A4$  (**is det**  $?A = -$ )  
 ⟨proof⟩

**lemma** *det-addcol[simp]*:  
**assumes**  $l: l < n$  **and**  $k: k \neq l$  **and**  $A: A \in \text{carrier-mat } n \ n$   
**shows** *det* (*addcol*  $a \ k \ l \ A$ ) = *det*  $A$   
 ⟨proof⟩

**definition** *insert-index*  $i \equiv \lambda i'. \text{if } i' < i \text{ then } i' \text{ else } \text{Suc } i'$

**definition** *delete-index*  $i \equiv \lambda i'. \text{if } i' < i \text{ then } i' \text{ else } i' - \text{Suc } 0$

**lemma** *insert-index[simp]*:  
 $i' < i \implies \text{insert-index } i \ i' = i'$   
 $i' \geq i \implies \text{insert-index } i \ i' = \text{Suc } i'$   
 ⟨proof⟩

**lemma** *delete-insert-index*[simp]:

*delete-index*  $i$  (*insert-index*  $i$   $i'$ ) =  $i'$   
(proof)

**lemma** *insert-delete-index*:

**assumes**  $i' \neq i$   
**shows** *insert-index*  $i$  (*delete-index*  $i$   $i'$ ) =  $i'$   
(proof)

**definition** *delete-dom*  $p$   $i \equiv \lambda i'. p$  (*insert-index*  $i$   $i'$ )

**definition** *delete-ran*  $p$   $j \equiv \lambda i. \text{delete-index } j$  ( $p$   $i$ )

**definition** *permutation-delete*  $p$   $i = \text{delete-ran}$  (*delete-dom*  $p$   $i$ ) ( $p$   $i$ )

**definition** *insert-ran*  $p$   $j \equiv \lambda i. \text{insert-index } j$  ( $p$   $i$ )

**definition** *insert-dom*  $p$   $i$   $j \equiv$

$\lambda i'. \text{if } i' < i \text{ then } p$   $i'$  **else if**  $i' = i$  **then**  $j$  **else**  $p$  ( $i' - 1$ )

**definition** *permutation-insert*  $i$   $j$   $p \equiv \text{insert-dom}$  (*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* ( $i::\text{nat}$ )  $j$   $p$   $i = j$   
(proof)

**lemma** *permutation-insert-base*:

**assumes**  $p$ :  $p$  *permutes*  $\{0..<n\}$   
**shows** *permutation-insert*  $n$   $n$   $p = p$   
(proof)

**lemma** *permutation-insert-row-step*:

$\langle \text{permutation-insert } (\text{Suc } i) j p \circ \text{transpose } i (\text{Suc } i) = \text{permutation-insert } i j p \rangle$   
(**is**  $\langle ?l = ?r \rangle$ )  
(proof)

**lemma** *permutation-insert-column-step*:

**assumes**  $p$ :  $p$  *permutes*  $\{0..<n\}$  **and**  $j < n$   
**shows** *transpose*  $j$  (*Suc*  $j$ )  $\circ$  *permutation-insert*  $i$  (*Suc*  $j$ )  $p = \text{permutation-insert}$   
 $i$   $j$   $p$   
(**is**  $?l = ?r$ )

*<proof>*

**lemma** *delete-dom-image:*

**assumes**  $i: i \in \{0..<Suc\ n\}$  (**is**  $- \in ?N$ )

**assumes** *iff*:  $\forall i' \in ?N. f\ i' = f\ i \longrightarrow i' = i$

**shows**  $delete\_dom\ f\ i\ '\{0..<n\} = f\ '\ ?N - \{f\ i\}$  (**is**  $?L = ?R$ )

*<proof>*

**lemma** *delete-ran-image:*

**assumes**  $j: j \in \{0..<Suc\ n\}$  (**is**  $- \in ?N$ )

**assumes** *fmng*:  $f\ '\{0..<n\} = ?N - \{j\}$

**shows**  $delete\_ran\ f\ j\ '\{0..<n\} = \{0..<n\}$  (**is**  $?L = ?R$ )

*<proof>*

**lemma** *delete-index-inj-on:*

**assumes**  $iS: i \notin S$

**shows**  $inj\_on\ (delete\_index\ i)\ S$

*<proof>*

**lemma** *insert-index-inj-on:*

**shows**  $inj\_on\ (insert\_index\ i)\ S$

*<proof>*

**lemma** *delete-dom-inj-on:*

**assumes**  $i: i \in \{0..<Suc\ n\}$  (**is**  $- \in ?N$ )

**assumes** *inj*:  $inj\_on\ f\ ?N$

**shows**  $inj\_on\ (delete\_dom\ f\ i)\ \{0..<n\}$

*<proof>*

**lemma** *delete-ran-inj-on:*

**assumes**  $j: j \in \{0..<Suc\ n\}$  (**is**  $- \in ?N$ )

**assumes** *img*:  $f\ '\{0..<n\} = ?N - \{j\}$

**shows**  $inj\_on\ (delete\_ran\ f\ j)\ \{0..<n\}$

*<proof>*

**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>*

**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>*

**lemma** *permutation-insert-delete:*

**assumes**  $p$ :  $p$  permutes  $\{0..<Suc\ n\}$   
**and**  $i$ :  $i < Suc\ n$   
**shows**  $permutation-insert\ i\ (p\ i)$  ( $permutation-delete\ p\ i$ ) =  $p$   
(is ?l = -)  
<proof>

**lemma** *insert-index-exclude*[simp]:  
 $insert-index\ i\ i' \neq i$  <proof>

**lemma** *insert-index-image*:  
**assumes**  $i$ :  $i < Suc\ n$   
**shows**  $insert-index\ i\ ' \{0..<n\} = \{0..<Suc\ n\} - \{i\}$  (is ?L = ?R)  
<proof>

**lemma** *insert-ran-image*:  
**assumes**  $j$ :  $j < Suc\ n$   
**assumes**  $img$ :  $f\ ' \{0..<n\} = \{0..<n\}$   
**shows**  $insert-ran\ f\ j\ ' \{0..<n\} = \{0..<Suc\ n\} - \{j\}$  (is ?L = ?R)  
<proof>

**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>

**lemma** *insert-ran-inj-on*:  
**assumes**  $inj$ :  $inj-on\ f\ \{0..<n\}$  **and**  $j$ :  $j < Suc\ n$   
**shows**  $inj-on\ (insert-ran\ f\ j)\ \{0..<n\}$  (is  $inj-on\ ?f\ -$ )  
<proof>

**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**  $inj-on\ (insert-dom\ f\ i\ j)\ ?N$   
<proof>

**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>

**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)  
<proof>



**lemma** *permutation-fix*:

**assumes**  $i: i < \text{Suc } n$  **and**  $j: j < \text{Suc } n$   
**shows**  $\{ p. p \text{ permutes } \{0..<\text{Suc } n\} \wedge p \ i = j \} =$   
 $\text{permutation-insert } i \ j \ ' \ \{ q. q \text{ permutes } \{0..<n\} \}$   
(**is**  $?L = ?R$ )  
(*proof*)

**lemma** *permutation-split-ran*:

**assumes**  $j: j \in S$   
**shows**  $\{ p. p \text{ permutes } S \} = (\bigcup i \in S. \{ p. p \text{ permutes } S \wedge p \ i = j \})$   
(**is**  $?L = ?R$ )  
(*proof*)

**lemma** *permutation-disjoint-dom*:

**assumes**  $i: i \in S$  **and**  $i': i' \in S$  **and**  $j: j \in S$  **and**  $ii': i \neq i'$   
**shows**  $\{ p. p \text{ permutes } S \wedge p \ i = j \} \cap \{ p. p \text{ permutes } S \wedge p \ i' = j \} = \{\}$   
(**is**  $?L \cap ?R = \{\}$ )  
(*proof*)

**lemma** *permutation-disjoint-ran*:

**assumes**  $i: i \in S$  **and**  $j: j \in S$  **and**  $j': j' \in S$  **and**  $jj': j \neq j'$   
**shows**  $\{ p. p \text{ permutes } S \wedge p \ i = j \} \cap \{ p. p \text{ permutes } S \wedge p \ i = j' \} = \{\}$   
(**is**  $?L \cap ?R = \{\}$ )  
(*proof*)

**lemma** *permutation-insert-inj-on*:

**assumes**  $i < \text{Suc } n$   
**assumes**  $j < \text{Suc } n$   
**shows**  $\text{inj-on } (\text{permutation-insert } i \ j) \ \{ q. q \text{ permutes } \{0..<n\} \}$   
(**is**  $\text{inj-on } ?f \ ?S$ )  
(*proof*)

**lemma** *signof-permutation-insert*:

**assumes**  $p: p \text{ permutes } \{0..<n\}$  **and**  $i: i < \text{Suc } n$  **and**  $j: j < \text{Suc } n$   
**shows**  $\text{signof } (\text{permutation-insert } i \ j \ p) = (-1::'a::\text{ring-1})^{i+j} * \text{signof } p$   
(*proof*)

**lemma** *foo*:

**assumes**  $i: i < \text{Suc } n$  **and**  $j: j < \text{Suc } n$   
**assumes**  $q: q \text{ permutes } \{0..<n\}$   
**shows**  $\{(i', \text{permutation-insert } i \ j \ q \ i') \mid i'. i' \in \{0..<\text{Suc } n\} - \{i\}\} =$   
 $\{( \text{insert-index } i \ i'', \text{insert-index } j \ (q \ i'') \mid i''. i'' < n \}$  (**is**  $?L = ?R$ )  
(*proof*)

**definition** *mat-delete*  $A \ i \ j \equiv$

$\text{mat } (\text{dim-row } A - 1) \ (\text{dim-col } A - 1) \ (\lambda(i',j').$   
 $A \ \$\$ \ (\text{if } i' < i \ \text{then } i' \ \text{else } \text{Suc } i', \ \text{if } j' < j \ \text{then } j' \ \text{else } \text{Suc } j'))$

**lemma** *mat-delete-dim[simp]*:

*dim-row* (mat-delete  $A$   $i$   $j$ ) = *dim-row*  $A$  - 1

*dim-col* (mat-delete  $A$   $i$   $j$ ) = *dim-col*  $A$  - 1

*<proof>*

**lemma** *mat-delete-carrier*:

**assumes**  $A$ :  $A \in \text{carrier-mat } m \ n$

**shows** mat-delete  $A$   $i$   $j \in \text{carrier-mat } (m-1) \ (n-1)$  *<proof>*

**lemma** *mat-delete-index*:

**assumes**  $A$ :  $A \in \text{carrier-mat } (\text{Suc } n) \ (\text{Suc } n)$

**and**  $i$ :  $i < \text{Suc } n$  **and**  $j$ :  $j < \text{Suc } n$

**and**  $i'$ :  $i' < n$  **and**  $j'$ :  $j' < n$

**shows**  $A$  \$\$ (insert-index  $i$   $i'$ , insert-index  $j$   $j'$ ) = mat-delete  $A$   $i$   $j$  \$\$ ( $i'$ ,  $j'$ )

*<proof>*

**definition** *cofactor*  $A$   $i$   $j$  =  $(-1)^{\wedge(i+j)} * \text{det} (\text{mat-delete } A \ i \ j)$

**lemma** *laplace-expansion-column*:

**assumes**  $A$ : ( $A$  :: 'a :: comm-ring-1 mat)  $\in \text{carrier-mat } n \ n$

**and**  $j$ :  $j < n$

**shows**  $\text{det } A = (\sum_{i < n}. A \ \$\$ (i,j) * \text{cofactor } A \ i \ j)$

*<proof>*

**lemma** *laplace-expansion-row*:

**assumes**  $A$ : ( $A$  :: 'a :: comm-ring-1 mat)  $\in \text{carrier-mat } n \ n$

**and**  $i$ :  $i < n$

**shows**  $\text{det } A = (\sum_{j < n}. A \ \$\$ (i,j) * \text{cofactor } A \ i \ j)$

*<proof>*

**lemma** *degree-det-le*: **assumes**  $\bigwedge i \ j. i < n \implies j < n \implies \text{degree } (A \ \$\$ (i,j)) \leq k$

**and**  $A$ :  $A \in \text{carrier-mat } n \ n$

**shows**  $\text{degree } (\text{det } A) \leq k * n$

*<proof>*

**lemma** *upper-triangular-imp-det-eq-0-iff*:

**fixes**  $A$  :: 'a :: idom mat

**assumes**  $A \in \text{carrier-mat } n \ n$  **and** upper-triangular  $A$

**shows**  $\text{det } A = 0 \longleftrightarrow 0 \in \text{set } (\text{diag-mat } A)$

*<proof>*

**lemma** *det-identical-columns*:

**assumes**  $A$ :  $A \in \text{carrier-mat } n \ n$

**and**  $ij$ :  $i \neq j$

**and**  $i$ :  $i < n$  **and**  $j$ :  $j < n$

**and**  $r$ :  $\text{col } A \ i = \text{col } A \ j$

**shows**  $\text{det } A = 0$

*<proof>*

**definition** *adj-mat* :: 'a :: comm-ring-1 mat  $\Rightarrow$  '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  $\in$  carrier-mat n n

**shows** *adj-mat* A  $\in$  carrier-mat n n

A \* *adj-mat* A = det A  $\cdot$   $\mathbf{1}_m$  n

*adj-mat* A \* A = det A  $\cdot$   $\mathbf{1}_m$  n

*<proof>*

**definition** *replace-col* A b k = mat (dim-row A) (dim-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** det (replace-col A (A \*\_v x) k) = x \$ k \* det A

*<proof>*

end

## 10 Code Equations for Determinants

We compute determinants on arbitrary rings by applying elementary row-operations 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 row-operations 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  $\times$  'a)list  $\Rightarrow$  nat

**definition** *det-selection-fun* :: 'a det-selection-fun  $\Rightarrow$  bool **where**

*det-selection-fun* f = ( $\forall$  xs. xs  $\neq$  []  $\longrightarrow$  f xs  $\in$  fst ' set xs)

**lemma** *det-selection-funD*:  $\text{det-selection-fun } f \implies xs \neq [] \implies f\ xs \in \text{fst } \text{' set } xs$   
 ⟨proof⟩

**definition** *mute-fun* :: ( $'a :: \text{comm-ring-1} \implies 'a \implies 'a \times 'a \times 'a$ )  $\implies \text{bool}$  **where**  
*mute-fun*  $f = (\forall\ x\ y\ x'\ y'\ g.\ f\ x\ y = (x',y',g) \longrightarrow y \neq 0$   
 $\longrightarrow x = x' * g \wedge y * x' = x * y')$

**context**

**fixes** *sel-fun* ::  $'a :: \text{idom-divide det-selection-fun}$   
**begin**

## 10.1 Properties of triangular matrices

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

**definition** *triangular-column*:: $\text{nat} \implies 'a\ \text{mat} \implies \text{bool}$   
**where** *triangular-column*  $j\ A \equiv \forall i.\ j < i \longrightarrow i < \text{dim-row } A \longrightarrow A\ \$\$ (i,j) = 0$

**lemma** *triangular-columnD* [*dest*]:  
*triangular-column*  $j\ A \implies j < i \implies i < \text{dim-row } A \implies A\ \$\$ (i,j) = 0$   
 ⟨proof⟩

**lemma** *triangular-columnI* [*intro*]:  
 $(\bigwedge i.\ j < i \implies i < \text{dim-row } A \implies A\ \$\$ (i,j) = 0) \implies \text{triangular-column } j\ A$   
 ⟨proof⟩

The following predicate states that the first  $k$  columns satisfy triangularity.

**definition** *triangular-to*::  $\text{nat} \implies 'a\ \text{mat} \implies \text{bool}$   
**where** *triangular-to*  $k\ A == \forall j.\ j < k \longrightarrow \text{triangular-column } j\ A$

**lemma** *triangular-to-triangular*:  $\text{upper-triangular } A = \text{triangular-to } (\text{dim-row } A)\ A$   
 ⟨proof⟩

**lemma** *triangular-toD* [*dest*]:  
*triangular-to*  $k\ A \implies j < k \implies j < i \implies i < \text{dim-row } A \implies A\ \$\$ (i,j) = 0$   
 ⟨proof⟩

**lemma** *triangular-toI* [*intro*]:  
 $(\bigwedge i.\ j < k \implies j < i \implies i < \text{dim-row } A \implies A\ \$\$ (i,j) = 0) \implies \text{triangular-to } k\ A$   
 ⟨proof⟩

**lemma** *triangle-growth*:  
**assumes** *tri*:*triangular-to*  $k\ A$   
**and** *col*:*triangular-column*  $k\ A$   
**shows** *triangular-to*  $(\text{Suc } k)\ A$   
 ⟨proof⟩

**lemma** *triangle-trans*:  $\text{triangular-to } k \ A \implies k > k' \implies \text{triangular-to } k' \ A$   
 ⟨proof⟩

## 10.2 Algorithms for Triangulization

**context**

**fixes**  $mf :: 'a \Rightarrow 'a \Rightarrow 'a \times 'a \times 'a$

**begin**

**private fun** *mute* ::  $'a \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \times 'a \text{ mat} \Rightarrow 'a \times 'a \text{ mat}$  **where**  
 $\text{mute } A \text{--ll } k \ l \ (r, A) = (\text{let } p = A \ \$\$ \ (k, l) \ \text{in if } p = 0 \ \text{then } (r, A) \ \text{else}$   
 $\text{case } mf \ A \text{--ll } p \ \text{of } (q', p', g) \Rightarrow$   
 $(r * q', \text{addrow } (-p') \ k \ l \ (\text{multrow } k \ q' \ A)))$

**lemma** *mute-preserves-dimensions*:

**assumes**  $\text{mute } q \ k \ l \ (r, A) = (r', A')$

**shows** [*simp*]:  $\text{dim-row } A' = \text{dim-row } A$  **and** [*simp*]:  $\text{dim-col } A' = \text{dim-col } A$   
 ⟨proof⟩

Algorithm *mute*  $k \ l$  makes  $k$ -th row  $l$ -th column element to 0.

**lemma** *mute-makes-0* :

**assumes** *mute-fun*:  $\text{mute-fun } mf$

**assumes**  $\text{mute } (A \ \$\$ \ (l, l)) \ k \ l \ (r, A) = (r', A')$

$l < \text{dim-row } A$

$l < \text{dim-col } A$

$k < \text{dim-row } A$

$k \neq l$

**shows**  $A' \ \$\$ \ (k, l) = 0$

⟨proof⟩

It will not touch unexpected rows.

**lemma** *mute-preserves*:

$\text{mute } q \ k \ l \ (r, A) = (r', A') \implies$

$i < \text{dim-row } A \implies$

$j < \text{dim-col } A \implies$

$l < \text{dim-row } A \implies$

$k < \text{dim-row } A \implies$

$i \neq k \implies$

$A' \ \$\$ \ (i, j) = A \ \$\$ \ (i, j)$

⟨proof⟩

It preserves 0s in the touched row.

**lemma** *mute-preserves-0*:

$\text{mute } q \ k \ l \ (r, A) = (r', A') \implies$

$i < \text{dim-row } A \implies$

$j < \text{dim-col } A \implies$

$l < \text{dim-row } A \implies$

$k < \text{dim-row } A \implies$

$A \text{ \#\# } (i,j) = 0 \implies$   
 $A \text{ \#\# } (l,j) = 0 \implies$   
 $A' \text{ \#\# } (i,j) = 0$   
 <proof>

Hence, it will respect partially triangular matrix.

**lemma** *mute-preserves-triangle*:  
**assumes**  $rA' : \text{mute } q \ k \ l \ (r,A) = (r',A')$   
**and**  $\text{tri}A : \text{triangular-to } l \ A$   
**and**  $lk : l < k$   
**and**  $kr : k < \text{dim-row } A$   
**and**  $lr : l < \text{dim-row } A$   
**and**  $lc : l < \text{dim-col } A$   
**shows**  $\text{triangular-to } l \ A'$   
 <proof>

Recursive application of *mute*

**private fun**  $\text{sub1} :: 'a \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \times 'a \ \text{mat} \Rightarrow 'a \times 'a \ \text{mat}$   
**where**  $\text{sub1 } q \ 0 \ l \ rA = rA$   
 $|\ \text{sub1 } q \ (\text{Suc } k) \ l \ rA = \text{mute } q \ (l + \text{Suc } k) \ l \ (\text{sub1 } q \ k \ l \ rA)$

**lemma** *sub1-preserves-dimensions*[simp]:  
 $\text{sub1 } q \ k \ l \ (r,A) = (r',A') \implies \text{dim-row } A' = \text{dim-row } A$   
 $\text{sub1 } q \ k \ l \ (r,A) = (r',A') \implies \text{dim-col } A' = \text{dim-col } A$   
 <proof>

**lemma** *sub1-closed* [simp]:  
 $\text{sub1 } q \ k \ l \ (r,A) = (r',A') \implies A \in \text{carrier-mat } m \ n \implies A' \in \text{carrier-mat } m \ n$   
 <proof>

**lemma** *sub1-preserves-diagonal*:  
**assumes**  $\text{sub1 } q \ k \ l \ (r,A) = (r',A')$   
**and**  $l < \text{dim-col } A$   
**and**  $k + l < \text{dim-row } A$   
**shows**  $A' \text{ \#\# } (l,l) = A \text{ \#\# } (l,l)$   
 <proof>

Triangularity is respected by *sub1*.

**lemma** *sub1-preserves-triangle*:  
**assumes**  $\text{sub1 } q \ k \ l \ (r,A) = (r',A')$   
**and**  $\text{tri} : \text{triangular-to } l \ A$   
**and**  $lr : l < \text{dim-row } A$   
**and**  $lc : l < \text{dim-col } A$   
**and**  $lkr : l + k < \text{dim-row } A$   
**shows**  $\text{triangular-to } l \ A'$   
 <proof>

**context**  
**assumes**  $mf : \text{mute-fun } mf$

```

begin
lemma sub1-makes-0s:
  assumes sub1 (A $$ (l,l)) k l (r,A) = (r',A')
  and lr: l < dim-row A
  and lc: l < dim-col A
  and li: l < i
  and i ≤ k + l
  and k + l < dim-row A
  shows A' $$ (i,l) = 0
⟨proof⟩

lemma sub1-triangularizes-column:
  assumes rA': sub1 (A $$ (l,l)) (dim-row A - Suc l) l (r,A) = (r',A')
  and tri:triangular-to l A
  and r: dim-row A > 0
  and lr: l < dim-row A
  and lc: l < dim-col A
  shows triangular-column l A'
⟨proof⟩

```

The algorithm *sub1* increases the number of columns that form triangle.

```

lemma sub1-grows-triangle:
  assumes rA': sub1 (A $$ (l,l)) (dim-row A - Suc l) l (r,A) = (r',A')
  and r: dim-row A > 0
  and tri:triangular-to l A
  and lr: l < dim-row A
  and lc: l < dim-col A
  shows triangular-to (Suc l) A'
⟨proof⟩
end

```

### 10.3 Finding Non-Zero Elements

```

private definition find-non0 :: nat ⇒ 'a mat ⇒ nat option where
  find-non0 l A = (let is = [Suc l ..< dim-row A];
    Ais = filter (λ (i,Ail). Ail ≠ 0) (map (λ i. (i, A $$ (i,l))) is)
    in case Ais of [] ⇒ None | - ⇒ 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) ≠ 0 l < m m < dim-row A
⟨proof⟩

```

If *find-non0 l A* fails, then *A* is already triangular to *l*-th column.

```

lemma find-non0-all0:
  find-non0 l A = None ⇒ triangular-column l A
⟨proof⟩

```

## 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 => nat => 'a × 'a mat => 'a × 'a mat
  where sub2 d l (r,A) = (
    case find-non0 l A of None => (r,A)
    | Some m => let A' = swaprows m l A in sub1 (A' $$ (l,l)) (d - Suc l) l (-r,
A'))
```

```
lemma sub2-preserves-dimensions[simp]:
  assumes rA': sub2 d l (r,A) = (r',A')
  shows dim-row A' = dim-row A ∧ dim-col A' = dim-col A
⟨proof⟩
```

```
lemma sub2-closed [simp]:
  sub2 d l (r,A) = (r',A') => A ∈ carrier-mat m n => A' ∈ carrier-mat m n
⟨proof⟩
```

```
context
  assumes sel-fun: det-selection-fun sel-fun
begin
```

```
lemma sub2-preserves-triangle:
  assumes rA': sub2 d l (r,A) = (r',A')
  and tri: triangular-to l A
  and lc: l < dim-col A
  and ld: l < d
  and dr: d ≤ dim-row A
  shows triangular-to l A'
⟨proof⟩
```

```
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⟩
end
```

## 10.5 Recursive Triangulization of Columns

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

```
private fun sub3 :: nat => nat => 'a × '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]:  
 $sub3\ d\ l\ (r,A) = (r',A') \implies dim\text{-row}\ A' = dim\text{-row}\ A$   
 $sub3\ d\ l\ (r,A) = (r',A') \implies dim\text{-col}\ A' = dim\text{-col}\ A$   
 ⟨proof⟩

**lemma** *sub3-closed*[simp]:  
 $sub3\ k\ l\ (r,A) = (r',A') \implies A \in carrier\text{-mat}\ m\ n \implies A' \in carrier\text{-mat}\ m\ n$   
 ⟨proof⟩

**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',A'*)  
**and**  $l \leq dim\text{-row}\ A$   
**and**  $l \leq dim\text{-col}\ A$   
**shows** *triangular-to l A'*  
 ⟨proof⟩

## 10.6 Triangulization

**definition** *triangulize* :: *'a mat*  $\Rightarrow$  *'a*  $\times$  *'a mat*  
**where** *triangulize A* = *sub3* (*dim-row A*) (*dim-row A*) (*1,A*)

**lemma** *triangulize-preserves-dimensions*[simp]:  
 $triangulize\ A = (r',A') \implies dim\text{-row}\ A' = dim\text{-row}\ A$   
 $triangulize\ A = (r',A') \implies dim\text{-col}\ A' = dim\text{-col}\ A$   
 ⟨proof⟩

**lemma** *triangulize-closed*[simp]:  
 $triangulize\ A = (r',A') \implies A \in carrier\text{-mat}\ m\ n \implies A' \in carrier\text{-mat}\ m\ n$   
 ⟨proof⟩

**context**  
**assumes** *mf*: *mute-fun mf*  
**and** *sel-fun*: *det-selection-fun sel-fun*  
**begin**

**theorem** *triangulized*:  
**assumes**  $A \in carrier\text{-mat}\ n\ n$   
**and** *triangulize A* = (*r',A'*)  
**shows** *upper-triangular A'*  
 ⟨proof⟩

## 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** *rA'*: *sub1 q k l (r, A)* = (*r',A'*)

**and**  $r0: r \neq 0$   
**and**  $All: q \neq 0$   
**and**  $k + l < \dim\text{-row } A$   
**and**  $lc: l < \dim\text{-col } A$   
**shows**  $r' \neq 0$   
 $\langle \text{proof} \rangle$

The algorithm *sub2* will not require such a condition.

**lemma** *sub2-divisor* [*simp*]:  
**assumes**  $rA': \text{sub2 } k \ l \ (r, A) = (r', A')$   
**and**  $lk: l < k$   
**and**  $kr: k \leq \dim\text{-row } A$   
**and**  $lc: l < \dim\text{-col } A$   
**and**  $r0: r \neq 0$   
**shows**  $r' \neq 0$   
 $\langle \text{proof} \rangle$

**lemma** *sub3-divisor* [*simp*]:  
**assumes**  $\text{sub3 } d \ l \ (r, A) = (r'', A'')$   
**and**  $l \leq d$   
**and**  $d \leq \dim\text{-row } A$   
**and**  $l \leq \dim\text{-col } A$   
**and**  $r0: r \neq 0$   
**shows**  $r'' \neq 0$   
 $\langle \text{proof} \rangle$

**theorem** *triangulize-divisor*:  
**assumes**  $A: A \in \text{carrier-mat } d \ d$   
**shows**  $\text{triangulize } A = (r', A') \implies r' \neq 0$   
 $\langle \text{proof} \rangle$

## 10.8 Determinant Preservation Results

For each sub-algorithm  $f$ , we show  $f(r, A) = (r', A')$  implies  $r * \det A' = r' * \det A$ .

**lemma** *mute-det*:  
**assumes**  $A \in \text{carrier-mat } n \ n$   
**and**  $rA': \text{mute } q \ k \ l \ (r, A) = (r', A')$   
**and**  $k < n$   
**and**  $l < n$   
**and**  $k \neq l$   
**shows**  $r * \det A' = r' * \det A$   
 $\langle \text{proof} \rangle$

**lemma** *sub1-det*:  
**assumes**  $A: A \in \text{carrier-mat } n \ n$   
**and**  $\text{sub1: } \text{sub1 } q \ k \ l \ (r, A) = (r'', A'')$   
**and**  $r0: r \neq 0$   
**and**  $All0: q \neq 0$

**and**  $l: l + k < n$   
**shows**  $r * \det A'' = r'' * \det A$   
 $\langle proof \rangle$

**lemma** *sub2-det*:  
**assumes**  $A: A \in \text{carrier-mat } d \ d$   
**and**  $rA': \text{sub2 } d \ l \ (r, A) = (r', A')$   
**and**  $r0: r \neq 0$   
**and**  $ld: l < d$   
**shows**  $r * \det A' = r' * \det A$   
 $\langle proof \rangle$

**lemma** *sub3-det*:  
**assumes**  $A: A \in \text{carrier-mat } d \ d$   
**and**  $\text{sub3 } d \ l \ (r, A) = (r'', A'')$   
**and**  $r0: r \neq 0$   
**and**  $l \leq d$   
**shows**  $r * \det A'' = r'' * \det A$   
 $\langle proof \rangle$

**theorem** *triangulize-det*:  
**assumes**  $A: A \in \text{carrier-mat } d \ d$   
**and**  $rA': \text{triangulize } A = (r', A')$   
**shows**  $\det A * r' = \det A'$   
 $\langle proof \rangle$   
**end**

## 10.9 Determinant Computation

**definition** *det-code* ::  $'a \ \text{mat} \Rightarrow 'a \ \text{where}$   
 $\text{det-code } A = (\text{if } \text{dim-row } A = \text{dim-col } A \ \text{then}$   
 $\text{case } \text{triangulize } A \ \text{of } (m, A') \Rightarrow$   
 $\text{prod-list } (\text{diag-mat } A') \ \text{div } m$   
 $\text{else } 0)$

**lemma** *det-code[simp]*: **assumes**  $\text{sel-fun}: \text{det-selection-fun } \text{sel-fun}$   
**and**  $\text{mf}: \text{mute-fun } \text{mf}$   
**shows**  $\text{det-code } A = \det A$   
 $\langle proof \rangle$

**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* ::  $'a :: \text{comm-ring-1} \Rightarrow 'a \Rightarrow 'a \times 'a \times 'a \ \text{where}$   
 $\text{trivial-mute-fun } x \ y = (x, y, 1)$

**lemma** *trivial-mute-fun*[simp,intro]: *mute-fun trivial-mute-fun*  
 ⟨proof⟩

**definition** *fst-sel-fun* :: 'a *det-selection-fun* **where**  
*fst-sel-fun* x = *fst* (hd x)

**lemma** *fst-sel-fun*[simp]: *det-selection-fun fst-sel-fun*  
 ⟨proof⟩

**context**

**fixes** *measure* :: 'a ⇒ nat

**begin**

**private fun** *select-min-main* **where**

*select-min-main* m i ((j,p) # xs) = (let n = *measure* p in if n < m then *select-min-main* n j xs

else *select-min-main* m i xs)

| *select-min-main* m i [] = i

**definition** *select-min* :: (nat × 'a) list ⇒ nat **where**  
*select-min* xs = (case xs of ((i,p) # ys) ⇒ (*select-min-main* (*measure* p) i ys))

**lemma** *select-min*[simp]: *det-selection-fun select-min*  
 ⟨proof⟩

**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  
 ⟨proof⟩

But we also provide specialiced functions for more specific carriers.

**definition** *field-mute-fun* :: 'a :: *field* ⇒ 'a ⇒ 'a × 'a × 'a **where**  
*field-mute-fun* x y = (x/y,1,y)

**lemma** *field-mute-fun*[simp,intro]: *mute-fun field-mute-fun*  
 ⟨proof⟩

**definition** *det-field* :: 'a :: *field mat* ⇒ 'a **where**  
*det-field* A = *det-code fst-sel-fun field-mute-fun* A

**lemma** *det-field*[simp]: *det-field* = *det*  
 ⟨proof⟩

**definition** *gcd-mute-fun* :: 'a :: *ring-gcd* ⇒ 'a ⇒ 'a × 'a × 'a **where**  
*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*  
 ⟨proof⟩

**definition** *det-int* :: *int mat* ⇒ *int* **where**  
*det-int* *A* = *det-code* (*select-min* ( $\lambda x. \text{nat } (\text{abs } x)$ )) *gcd-mute-fun* *A*

**lemma** *det-int[simp]*: *det-int* = *det*  
 ⟨*proof*⟩

**definition** *det-field-poly* :: '*a* :: {*field,field-gcd*} *poly mat* ⇒ '*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*  
 ⟨*proof*⟩

**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* :: *show vec* ⇒ *shows* **where**  
*shows-vec* *v* ≡ *shows* (*list-of-vec* *v*)

**instantiation** *vec* :: (*show*) *show*  
**begin**

**definition** *shows-prec* *p* (*v* :: '*a vec*) ≡ *shows-vec* *v*

**definition** *shows-list* (*vs* :: '*a vec list*) ≡ *shows-sep* *shows-vec* (*shows* " , ") *vs*

**instance**

⟨*proof*⟩

**end**

**definition** *shows-mat* :: '*a* :: *show mat* ⇒ *shows* **where**  
*shows-mat* *A* ≡ *shows* (*mat-to-list* *A*)

**instantiation** *mat* :: (*show*) *show*  
**begin**

**definition** *shows-prec* *p* (*A* :: '*a mat*) ≡ *shows-mat* *A*

**definition** *shows-list* (*As* :: '*a mat list*) ≡ *shows-sep* *shows-mat* (*shows* " , ") *As*

**instance**

*<proof>*  
**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  $\Rightarrow$  'a vec  $\Rightarrow$  'a  $\Rightarrow$  bool **where**  
*eigenvector* A v k = (v  $\in$  carrier-vec (dim-row A)  $\wedge$  v  $\neq$  0\_v (dim-row A)  $\wedge$  A \*\_v v = k \*\_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  $\widehat{m}$  i \*\_v v = k  $\widehat{i}$  \*\_v v

*<proof>*

**definition** *eigenvalue* :: 'a :: comm-ring-1 mat  $\Rightarrow$  'a  $\Rightarrow$  bool **where**

*eigenvalue* A k = ( $\exists$  v. *eigenvector* A v k)

**definition** *char-matrix* :: 'a :: field mat  $\Rightarrow$  'a  $\Rightarrow$  'a mat **where**

*char-matrix* A e = A + ((-e) \*\_m (1\_m (dim-row A)))

**lemma** *char-matrix-closed[simp]*: A  $\in$  carrier-mat n n  $\implies$  *char-matrix* A e  $\in$  carrier-mat n n

*<proof>*

**lemma** *eigenvector-char-matrix*: **assumes** A: (A :: 'a :: field mat)  $\in$  carrier-mat n n

**shows** *eigenvector* A v e = (v  $\in$  carrier-vec n  $\wedge$  v  $\neq$  0\_v n  $\wedge$  *char-matrix* A e \*\_v v = 0\_v n)

*<proof>*

**lemma eigenvalue-char-matrix:** **assumes**  $A: (A :: 'a :: \text{field mat}) \in \text{carrier-mat } n$

**shows**  $\text{eigenvalue } A \ e = (\exists v. v \in \text{carrier-vec } n \wedge v \neq 0_v \ n \wedge \text{char-matrix } A \ e$   
 $*_v \ v = 0_v \ n)$   
 ⟨proof⟩

**definition find-eigenvector**  $:: 'a::\text{field mat} \Rightarrow 'a \Rightarrow 'a \text{ vec}$  **where**

$\text{find-eigenvector } A \ e =$   
 $\text{find-base-vector } (\text{fst } (\text{gauss-jordan } (\text{char-matrix } A \ e) \ (0_m \ (\text{dim-row } A) \ 0)))$

**lemma find-eigenvector:** **assumes**  $A: A \in \text{carrier-mat } n \ n$

**and**  $ev: \text{eigenvalue } A \ e$

**shows**  $\text{eigenvector } A \ (\text{find-eigenvector } A \ e) \ e$   
 ⟨proof⟩

**lemma eigenvalue-imp-nonzero-dim:** **assumes**  $A \in \text{carrier-mat } n \ n$

**and**  $\text{eigenvalue } A \ ev$

**shows**  $n > 0$   
 ⟨proof⟩

**lemma eigenvalue-det:** **assumes**  $A: (A :: 'a :: \text{field mat}) \in \text{carrier-mat } n \ n$  **shows**

$\text{eigenvalue } A \ e = (\text{det } (\text{char-matrix } A \ e) = 0)$   
 ⟨proof⟩

**definition char-poly-matrix**  $:: 'a :: \text{comm-ring-1 mat} \Rightarrow 'a \text{ poly mat}$  **where**

$\text{char-poly-matrix } A = (([:0,1:] \cdot_m \ 1_m \ (\text{dim-row } A)) + \text{map-mat } (\lambda a. [: - a :])$   
 $A)$

**lemma char-poly-matrix-closed[simp]:**  $A \in \text{carrier-mat } n \ n \Longrightarrow \text{char-poly-matrix}$   
 $A \in \text{carrier-mat } n \ n$

⟨proof⟩

**definition char-poly**  $:: 'a :: \text{comm-ring-1 mat} \Rightarrow 'a \text{ poly}$  **where**

$\text{char-poly } A = (\text{det } (\text{char-poly-matrix } A))$

**lemmas**  $\text{char-poly-defs} = \text{char-poly-def } \text{char-poly-matrix-def}$

**lemma (in comm-ring-hom) char-poly-matrix-hom:** **assumes**  $A: A \in \text{carrier-mat}$   
 $n \ n$

**shows**  $\text{char-poly-matrix } (\text{mat}_h \ A) = \text{map-mat } (\text{map-poly } \text{hom}) \ (\text{char-poly-matrix}$   
 $A)$   
 ⟨proof⟩

**lemma (in comm-ring-hom) char-poly-hom:** **assumes**  $A: A \in \text{carrier-mat } n \ n$

**shows**  $\text{char-poly } (\text{map-mat } \text{hom } A) = \text{map-poly } \text{hom } (\text{char-poly } A)$   
 ⟨proof⟩

**context**  $\text{inj-comm-ring-hom}$

**begin**

**lemma** *eigenvector-hom*: **assumes**  $A: A \in \text{carrier-mat } n \ n$   
**and**  $ev: \text{eigenvector } A \ v \ ev$   
**shows**  $\text{eigenvector } (\text{mat}_h \ A) \ (\text{vec}_h \ v) \ (\text{hom } ev)$   
*<proof>*

**lemma** *eigenvalue-hom*: **assumes**  $A: A \in \text{carrier-mat } n \ n$   
**and**  $ev: \text{eigenvalue } A \ ev$   
**shows**  $\text{eigenvalue } (\text{mat}_h \ A) \ (\text{hom } ev)$   
*<proof>*

**lemma** *eigenvector-hom-rev*: **assumes**  $A: A \in \text{carrier-mat } n \ n$   
**and**  $ev: \text{eigenvector } (\text{mat}_h \ A) \ (\text{vec}_h \ v) \ (\text{hom } ev)$   
**shows**  $\text{eigenvector } A \ v \ ev$   
*<proof>*

**end**

**lemma** *poly-det-cong*: **assumes**  $A: A \in \text{carrier-mat } n \ n$   
**and**  $B: B \in \text{carrier-mat } n \ n$   
**and**  $\text{poly}: \bigwedge i \ j. i < n \implies j < n \implies \text{poly } (B \ \S\$(i,j)) \ k = A \ \S\$(i,j)$   
**shows**  $\text{poly } (\text{det } B) \ k = \text{det } A$   
*<proof>*

**lemma** *char-poly-matrix*: **assumes**  $A: (A :: 'a :: \text{field mat}) \in \text{carrier-mat } n \ n$   
**shows**  $\text{poly } (\text{char-poly } A) \ k = \text{det } (- (\text{char-matrix } A \ k))$  *<proof>*

**lemma** *eigenvalue-root-char-poly*: **assumes**  $A: (A :: 'a :: \text{field mat}) \in \text{carrier-mat } n \ n$   
**shows**  $\text{eigenvalue } A \ k \longleftrightarrow \text{poly } (\text{char-poly } A) \ k = 0$   
*<proof>*

**context**

**fixes**  $A :: 'a :: \text{comm-ring-1 mat}$  **and**  $n :: \text{nat}$   
**assumes**  $A: A \in \text{carrier-mat } n \ n$   
**and**  $ut: \text{upper-triangular } A$

**begin**

**lemma** *char-poly-matrix-upper-triangular*:  $\text{upper-triangular } (\text{char-poly-matrix } A)$   
*<proof>*

**lemma** *char-poly-upper-triangular*:  
 $\text{char-poly } A = (\prod a \leftarrow \text{diag-mat } A. [:- a, 1:])$   
*<proof>*

**end**

**lemma** *map-poly-mult*: **assumes**  $A: A \in \text{carrier-mat } nr \ n$   
**and**  $B: B \in \text{carrier-mat } n \ nc$



**shows**

$map\text{-}mat (\lambda a. [: a :]) (A * B) = map\text{-}mat (\lambda a. [: a :]) A * map\text{-}mat (\lambda a. [: a :]) B$  (**is** *?id*)  
 $map\text{-}mat (\lambda a. [: a :] * p) (A * B) = map\text{-}mat (\lambda a. [: a :] * p) A * map\text{-}mat (\lambda a. [: a :]) B$  (**is** *?left*)  
 $map\text{-}mat (\lambda a. [: a :] * p) (A * B) = map\text{-}mat (\lambda a. [: a :]) A * map\text{-}mat (\lambda a. [: a :] * p) B$  (**is** *?right*)  
(*proof*)

**lemma** *char-poly-similar*: **assumes** *similar-mat A (B :: 'a :: comm-ring-1 mat)*  
**shows** *char-poly A = char-poly B*  
(*proof*)

**lemma** *degree-signof-mult[simp]*: *degree (signof p \* q) = degree q*  
(*proof*)

**lemma** *degree-monic-char-poly*: **assumes** *A: A ∈ carrier-mat n n*  
**shows** *degree (char-poly A) = n ∧ coeff (char-poly A) n = 1*  
(*proof*)

**lemma** *char-poly-factorized*: **fixes** *A :: complex mat*  
**assumes** *A: A ∈ carrier-mat n n*  
**shows**  $\exists as. char\text{-}poly A = (\prod a \leftarrow as. [: - a, 1:]) \wedge length\ as = n$   
(*proof*)

**lemma** *char-poly-four-block-zeros-col*: **assumes** *A1: A1 ∈ carrier-mat 1 1*  
**and** *A2: A2 ∈ carrier-mat 1 n and A3: A3 ∈ carrier-mat n n*  
**shows** *char-poly (four-block-mat A1 A2 (0<sub>m</sub> n 1) A3) = char-poly A1 \* char-poly A3*  
(**is** *char-poly ?A = ?cp1 \* ?cp3*)  
(*proof*)

**lemma** *char-poly-transpose-mat[simp]*: **assumes** *A: A ∈ carrier-mat n n*  
**shows** *char-poly (transpose-mat A) = char-poly A*  
(*proof*)

**lemma** *pderiv-char-poly*: **fixes** *A :: 'a :: idom mat*  
**assumes** *A: A ∈ carrier-mat n n*  
**shows** *pderiv (char-poly A) = ( $\sum i < n. char\text{-}poly (mat\text{-}delete A i i)$ )*  
(*proof*)

**lemma** *char-poly-0-column*: **fixes** *A :: 'a :: idom mat*  
**assumes** *0:  $\bigwedge j. j < n \implies A \$\$ (j, i) = 0$*   
**and** *A: A ∈ carrier-mat n n*  
**and** *i: i < n*  
**shows** *char-poly A = monom 1 1 \* char-poly (mat-delete A i i)*  
(*proof*)

**definition** *mat-erase :: 'a :: zero mat  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  'a mat* **where**

*mat-erase*  $A\ i\ j = \text{Matrix.mat } (\text{dim-row } A)\ (\text{dim-col } A)$   
 $(\lambda\ (i',j').\ \text{if } i' = i \vee j' = j\ \text{then } 0\ \text{else } A\ \text{\$\$ } (i',j'))$

**lemma** *mat-erase-carrier[simp]*:  $(\text{mat-erase } A\ i\ j) \in \text{carrier-mat } nr\ nc \iff A \in \text{carrier-mat } nr\ nc$   
 $\langle \text{proof} \rangle$

**lemma** *pderiv-char-poly-mat-erase*: **fixes**  $A :: 'a :: \text{idom mat}$   
**assumes**  $A: A \in \text{carrier-mat } n\ n$   
**shows**  $\text{monom } 1\ 1\ * \text{pderiv } (\text{char-poly } A) = (\sum\ i < n.\ \text{char-poly } (\text{mat-erase } A\ i\ i))$   
 $\langle \text{proof} \rangle$

**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*  $:: \text{nat} \Rightarrow 'a :: \{\text{zero,one}\} \Rightarrow 'a\ \text{mat}$  **where**  
 $\text{jordan-block } n\ a = \text{mat } n\ n\ (\lambda\ (i,j).\ \text{if } i = j\ \text{then } a\ \text{else if } \text{Suc } i = j\ \text{then } 1\ \text{else } 0)$

**lemma** *jordan-block-index[simp]*:  $i < n \implies j < n \implies$   
 $\text{jordan-block } n\ a\ \text{\$\$ } (i,j) = (\text{if } i = j\ \text{then } a\ \text{else if } \text{Suc } i = j\ \text{then } 1\ \text{else } 0)$   
 $\text{dim-row } (\text{jordan-block } n\ k) = n$   
 $\text{dim-col } (\text{jordan-block } n\ k) = n$   
 $\langle \text{proof} \rangle$

**lemma** *jordan-block-carrier[simp]*:  $\text{jordan-block } n\ k \in \text{carrier-mat } n\ n$   
 $\langle \text{proof} \rangle$

**lemma** *jordan-block-char-poly*:  $\text{char-poly } (\text{jordan-block } n\ a) = [:-a, 1:]^{\wedge} n$   
 $\langle \text{proof} \rangle$

**lemma** *jordan-block-pow-carrier[simp]*:  
 $\text{jordan-block } n\ a\ \widehat{m}\ r \in \text{carrier-mat } n\ n\ \langle \text{proof} \rangle$

**lemma** *jordan-block-pow-dim[simp]*:  
 $\text{dim-row } (\text{jordan-block } n\ a\ \widehat{m}\ r) = n\ \text{dim-col } (\text{jordan-block } n\ a\ \widehat{m}\ r) = n\ \langle \text{proof} \rangle$

**lemma** *jordan-block-pow*:  $(\text{jordan-block } n \ (a :: 'a :: \text{comm-ring-1})) \widehat{m} \ r =$   
 $\text{mat } n \ n \ (\lambda \ (i,j). \text{if } i \leq j \text{ then of-nat } (r \text{ choose } (j - i)) * a \widehat{(r + i - j)} \text{ else } 0)$   
 ⟨proof⟩

**definition** *jordan-matrix* ::  $(\text{nat} \times 'a :: \{\text{zero,one}\})\text{list} \Rightarrow 'a \text{ mat}$  **where**  
*jordan-matrix*  $n\text{-as} = \text{diag-block-mat} \ (\text{map} \ (\lambda \ (n,a). \text{jordan-block } n \ a) \ n\text{-as})$

**lemma** *jordan-matrix-dim[simp]*:  
 $\text{dim-row} \ (\text{jordan-matrix } n\text{-as}) = \text{sum-list} \ (\text{map } \text{fst} \ n\text{-as})$   
 $\text{dim-col} \ (\text{jordan-matrix } n\text{-as}) = \text{sum-list} \ (\text{map } \text{fst} \ n\text{-as})$   
 ⟨proof⟩

**lemma** *jordan-matrix-carrier[simp]*:  
 $\text{jordan-matrix } n\text{-as} \in \text{carrier-mat} \ (\text{sum-list} \ (\text{map } \text{fst} \ n\text{-as})) \ (\text{sum-list} \ (\text{map } \text{fst} \ n\text{-as}))$   
 ⟨proof⟩

**lemma** *jordan-matrix-upper-triangular*:  $i < \text{sum-list} \ (\text{map } \text{fst} \ n\text{-as})$   
 $\implies j < i \implies \text{jordan-matrix } n\text{-as} \ \$\$ \ (i,j) = 0$   
 ⟨proof⟩

**lemma** *jordan-matrix-pow*:  $(\text{jordan-matrix } n\text{-as}) \widehat{m} \ r =$   
 $\text{diag-block-mat} \ (\text{map} \ (\lambda \ (n,a). \ (\text{jordan-block } n \ a) \widehat{m} \ r) \ n\text{-as})$   
 ⟨proof⟩

**lemma** *jordan-matrix-char-poly*:  
 $\text{char-poly} \ (\text{jordan-matrix } n\text{-as}) = (\prod (n, a) \leftarrow n\text{-as}. \text{[: - } a, 1:] \widehat{n})$   
 ⟨proof⟩

**definition** *jordan-nf* ::  $'a :: \text{semiring-1} \text{ mat} \Rightarrow (\text{nat} \times 'a)\text{list} \Rightarrow \text{bool}$  **where**  
*jordan-nf*  $A \ n\text{-as} \equiv (0 \notin \text{fst } ' \text{ set } n\text{-as} \wedge \text{similar-mat } A \ (\text{jordan-matrix } n\text{-as}))$

**lemma** *jordan-nf-powE*: **assumes**  $A: A \in \text{carrier-mat } n \ n$  **and**  $\text{jnf}: \text{jordan-nf } A \ n\text{-as}$

**obtains**  $P \ Q$  **where**  $P \in \text{carrier-mat } n \ n$   $Q \in \text{carrier-mat } n \ n$  **and**  
 $\text{char-poly } A = (\prod (na, a) \leftarrow n\text{-as}. \text{[: - } a, 1:] \widehat{na})$   
 $\bigwedge k. A \widehat{m} \ k = P * (\text{jordan-matrix } n\text{-as}) \widehat{m} \ k * Q$   
 ⟨proof⟩

**lemma** *choose-poly-bound*: **assumes**  $i \leq d$   
**shows**  $r \text{ choose } i \leq \text{max } 1 \ (r \widehat{d})$   
 ⟨proof⟩

**context**  
**fixes**  $b :: 'a :: \text{archimedean-field}$   
**assumes**  $b: 0 < b \ b < 1$   
**begin**

**lemma** *poly-exp-constant-bound*:  $\exists p. \forall x. c * b \widehat{x} * \text{of-nat } x \widehat{\text{deg}} \leq p$

*<proof>*

**lemma** *poly-exp-max-constant-bound*:  $\exists p. \forall x. c * b^x * \max 1 (of\text{-}nat\ x \ \widehat{deg}) \leq p$   
*<proof>*  
**end**

**context**

**fixes**  $a :: 'a :: real\text{-}normed\text{-}field$

**begin**

**lemma** *jordan-block-bound*:

**assumes**  $i: i < n$  **and**  $j: j < n$

**shows**  $norm ((jordan\text{-}block\ n\ a\ \widehat{m}\ k)\ \S\S\ (i,j))$

$\leq norm\ a^{\widehat{(k+i-j)}} * \max 1 (of\text{-}nat\ k^{\widehat{(n-1)}})$

(**is** *?lhs*  $\leq$  *?rhs*)

*<proof>*

**lemma** *jordan-block-poly-bound*:

**assumes**  $i: i < n$  **and**  $j: j < n$  **and**  $a: norm\ a = 1$

**shows**  $norm ((jordan\text{-}block\ n\ a\ \widehat{m}\ k)\ \S\S\ (i,j)) \leq \max 1 (of\text{-}nat\ k^{\widehat{(n-1)}})$

(**is** *?lhs*  $\leq$  *?rhs*)

*<proof>*

**theorem** *jordan-block-constant-bound*: **assumes**  $a: norm\ a < 1$

**shows**  $\exists p. \forall i\ j\ k. i < n \longrightarrow j < n \longrightarrow norm ((jordan\text{-}block\ n\ a\ \widehat{m}\ k)\ \S\S\ (i,j)) \leq p$

*<proof>*

**definition** *norm-bound*  $:: 'a\ mat \Rightarrow real \Rightarrow bool$  **where**

$norm\text{-}bound\ A\ b \equiv \forall i\ j. i < dim\text{-}row\ A \longrightarrow j < dim\text{-}col\ A \longrightarrow norm (A\ \S\S\ (i,j)) \leq b$

**lemma** *norm-boundI[intro]*:

**assumes**  $\bigwedge i\ j. i < dim\text{-}row\ A \Longrightarrow j < dim\text{-}col\ A \Longrightarrow norm (A\ \S\S\ (i,j)) \leq b$

**shows**  $norm\text{-}bound\ A\ b$

*<proof>*

**lemma** *jordan-block-constant-bound2*:

$\exists p. norm (a :: 'a :: real\text{-}normed\text{-}field) < 1 \longrightarrow$

$(\forall i\ j\ k. i < n \longrightarrow j < n \longrightarrow norm ((jordan\text{-}block\ n\ a\ \widehat{m}\ k)\ \S\S\ (i,j)) \leq p)$

*<proof>*

**lemma** *jordan-matrix-poly-bound2*:

**fixes**  $n\text{-}as :: (nat \times 'a)\ list$

**assumes**  $n\text{-}as: \bigwedge n\ a. (n,a) \in set\ n\text{-}as \Longrightarrow n > 0 \Longrightarrow norm\ a \leq 1$

**and**  $N: \bigwedge n\ a. (n,a) \in set\ n\text{-}as \Longrightarrow norm\ a = 1 \Longrightarrow n \leq N$

**shows**  $\exists c1. \forall k. \forall e \in elements\text{-}mat (jordan\text{-}matrix\ n\text{-}as\ \widehat{m}\ k).$

$norm\ e \leq c1 + of\text{-}nat\ k^{\widehat{(N-1)}}$

*<proof>*

**lemma** *norm-bound-bridge*:

$\forall e \in \text{elements-mat } A. \text{norm } e \leq b \implies \text{norm-bound } A \ b$

*<proof>*

**lemma** *norm-bound-mult*: **assumes**  $A1: A1 \in \text{carrier-mat } nr \ n$

**and**  $A2: A2 \in \text{carrier-mat } n \ nc$

**and**  $b1: \text{norm-bound } A1 \ b1$

**and**  $b2: \text{norm-bound } A2 \ b2$

**shows**  $\text{norm-bound } (A1 * A2) \ (b1 * b2 * \text{of-nat } n)$

*<proof>*

**lemma** *norm-bound-max*:  $\text{norm-bound } A \ (\text{Max } \{\text{norm } (A \ \&\& \ (i,j)) \mid i \ j. \ i < \text{dim-row } A \wedge j < \text{dim-col } A\})$

(**is**  $\text{norm-bound } A \ (\text{Max } ?\text{norms})$ )

*<proof>*

**lemma** *jordan-matrix-poly-bound*: **fixes**  $n\text{-as} :: (\text{nat} \times 'a)\text{list}$

**assumes**  $n\text{-as}: \bigwedge n \ a. (n,a) \in \text{set } n\text{-as} \implies n > 0 \implies \text{norm } a \leq 1$

**and**  $N: \bigwedge n \ a. (n,a) \in \text{set } n\text{-as} \implies \text{norm } a = 1 \implies n \leq N$

**shows**  $\exists c1. \forall k. \text{norm-bound } (\text{jordan-matrix } n\text{-as} \ \widehat{m} \ k) \ (c1 + \text{of-nat } k \ \widehat{(N - 1)})$

*<proof>*

**lemma** *jordan-nf-matrix-poly-bound*: **fixes**  $n\text{-as} :: (\text{nat} \times 'a)\text{list}$

**assumes**  $A: A \in \text{carrier-mat } n \ n$

**and**  $n\text{-as}: \bigwedge n \ a. (n,a) \in \text{set } n\text{-as} \implies n > 0 \implies \text{norm } a \leq 1$

**and**  $N: \bigwedge n \ a. (n,a) \in \text{set } n\text{-as} \implies \text{norm } a = 1 \implies n \leq N$

**and**  $jnf: \text{jordan-nf } A \ n\text{-as}$

**shows**  $\exists c1 \ c2. \forall k. \text{norm-bound } (A \ \widehat{m} \ k) \ (c1 + c2 * \text{of-nat } k \ \widehat{(N - 1)})$

*<proof>*

**end**

**context**

**fixes**  $f\text{-ty} :: 'a :: \text{field itself}$

**begin**

**lemma** *char-matrix-jordan-block*:  $\text{char-matrix } (\text{jordan-block } n \ a) \ b = (\text{jordan-block } n \ (a - b))$

*<proof>*

**lemma** *diag-jordan-block-pow*:  $\text{diag-mat } (\text{jordan-block } n \ (a :: 'a) \ \widehat{m} \ k) = \text{replicate } n \ (a \ \widehat{k})$

*<proof>*

**lemma** *jordan-block-zero-pow*:  $(\text{jordan-block } n \ (0 :: 'a) \ \widehat{m} \ k =$

$(\text{mat } n \ n \ (\lambda \ (i,j). \ \text{if } j \geq i \wedge j - i = k \ \text{then } 1 \ \text{else } 0))$

*<proof>*

**end**

**lemma** *jordan-matrix-concat-diag-block-mat*: *jordan-matrix* (concat *jbs*) = *diag-block-mat*  
 (map *jordan-matrix jbs*)  
 ⟨*proof*⟩

**lemma** *jordan-nf-diag-block-mat*: **assumes** *Ms*:  $\bigwedge A \text{ jbs. } (A, \text{jbs}) \in \text{set } Ms \implies$   
*jordan-nf A jbs*  
**shows** *jordan-nf* (*diag-block-mat* (map *fst Ms*)) (concat (map *snd Ms*))  
 ⟨*proof*⟩

**lemma** *jordan-nf-char-poly*: **assumes** *jordan-nf A n-as*  
**shows** *char-poly A* =  $(\prod (n, a) \leftarrow n\text{-as. } [- a, 1:] \wedge n)$   
 ⟨*proof*⟩

**lemma** *jordan-nf-block-size-order-bound*: **assumes** *jnf: jordan-nf A n-as*  
**and** *mem*:  $(n, a) \in \text{set } n\text{-as}$   
**shows**  $n \leq \text{order } a$  (*char-poly A*)  
 ⟨*proof*⟩

**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*⟩

**lemma** *jordan-matrix-Cons*: *jordan-matrix* (*Cons* (*n, a*) *n-as*) = *four-block-mat*  
 (*jordan-block n a*) ( $0_m n$  (*sum-list* (map *fst n-as*)))  
 ( $0_m$  (*sum-list* (map *fst n-as*)) *n*) (*jordan-matrix n-as*)  
 ⟨*proof*⟩

**lemma** *similar-mat-jordan-matrix-smult*: **fixes** *n-as* ::  $(\text{nat} \times 'a :: \text{field}) \text{ list}$   
**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*⟩

**lemma** *jordan-nf-smult*: **fixes** *k* :: 'a :: field  
**assumes** *jn*: *jordan-nf A n-as*  
**and** *k*:  $k \neq 0$   
**shows** *jordan-nf* ( $k \cdot_m A$ ) (map  $(\lambda (n, a). (n, k * a))$  *n-as*)  
 ⟨*proof*⟩

**lemma** *jordan-nf-order*: **assumes** *jordan-nf A n-as*  
**shows** *order a* (*char-poly A*) = *sum-list* (map *fst* (*filter*  $(\lambda na. \text{snd } na = a)$  *n-as*))  
 ⟨*proof*⟩

### 13.1 Application for Complexity

```

lemma factored-char-poly-norm-bound: assumes  $A: A \in \text{carrier-mat } n \ n$ 
  and linear-factors:  $\text{char-poly } A = (\prod (a :: 'a :: \text{real-normed-field}) \leftarrow \text{as. } [:- a, 1:])$ 
  and jnf-exists:  $\exists \text{ n-as. jordan-nf } A \text{ n-as}$ 
  and le-1:  $\bigwedge a. a \in \text{set as} \implies \text{norm } a \leq 1$ 
  and le-N:  $\bigwedge a. a \in \text{set as} \implies \text{norm } a = 1 \implies \text{length } (\text{filter } ((=) a) \text{ as}) \leq N$ 
  shows  $\exists c1 \ c2. \forall k. \text{norm-bound } (A \widehat{m} k) (c1 + c2 * \text{of-nat } k \widehat{ } (N - 1))$ 
  <proof>

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  $f :: 'a \Rightarrow 'a \Rightarrow 'a$  and  $A :: 'a \text{ set}$ 
  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  $A = (\text{UNIV} :: 'a \text{ set})$ 
  shows comp-fun-commute  $f$ 
  <proof>

```

```

lemma fun-left-comm:
  assumes  $y \in A$  and  $x \in A$  and  $z \in A$  shows  $f \ y (f \ x \ z) = f \ x (f \ y \ z)$ 
  <proof>

```

```

lemma commute-left-comp:
  assumes  $y \in A$  and  $x \in A$  and  $z \in A$  and  $g \in A \rightarrow A$ 
  shows  $f \ y (f \ x (g \ z)) = f \ x (f \ y (g \ z))$ 
  <proof>

```

**lemma** *fold-graph-finite*:

**assumes** *fold-graph f z B y*

**shows** *finite B*

*<proof>*

**lemma** *fold-graph-closed*:

**assumes** *fold-graph f z B y* **and**  $B \subseteq A$  **and**  $z \in A$

**shows**  $y \in A$

*<proof>*

**lemma** *fold-graph-insertE-aux*:

*fold-graph f z B y*  $\implies a \in B \implies z \in A$

$\implies B \subseteq A$

$\implies \exists y'. y = f a y' \wedge \text{fold-graph } f z (B - \{a\}) y' \wedge y' \in A$

*<proof>*

**lemma** *fold-graph-insertE*:

**assumes** *fold-graph f z (insert x B) v* **and**  $x \notin B$  **and**  $\text{insert } x B \subseteq A$  **and**  $z \in A$

**obtains**  $y$  **where**  $v = f x y$  **and** *fold-graph f z B y*

*<proof>*

**lemma** *fold-graph-determ*: *fold-graph f z B x*  $\implies$  *fold-graph f z B y*  $\implies B \subseteq A$

$\implies z \in A \implies y = x$

*<proof>*

**lemma** *fold-equality*: *fold-graph f z B y*  $\implies B \subseteq A \implies z \in A \implies \text{Finite-Set.fold } f z B = y$

*<proof>*

**lemma** *fold-graph-fold*:

**assumes**  $f$ : *finite B* **and**  $BA$ :  $B \subseteq A$  **and**  $z$ :  $z \in A$

**shows** *fold-graph f z B (Finite-Set.fold f z B)*

*<proof>*

**lemma** *fold-insert [simp]*:

**assumes** *finite B* **and**  $x \notin B$  **and**  $BA$ :  $\text{insert } x B \subseteq A$  **and**  $z$ :  $z \in A$

**shows**  $\text{Finite-Set.fold } f z (\text{insert } x B) = f x (\text{Finite-Set.fold } f z B)$

*<proof>*

**end**

**lemma** *fold-cong*:

**assumes**  $f$ : *comp-fun-commute-on f A* **and**  $g$ : *comp-fun-commute-on g A*

**and** *finite S*

**and**  $\text{cong}$ :  $\bigwedge x. x \in S \implies f x = g x$

**and**  $s = t$  **and**  $S = T$

**and**  $SA$ :  $S \subseteq A$  **and**  $s$ :  $s \in A$

**shows**  $\text{Finite-Set.fold } f s S = \text{Finite-Set.fold } g t T$



*<proof>*

**context** *comp-fun-commute-on*  
**begin**

**lemma** *comp-fun-Pi*:  $(\lambda x. f x \rightsquigarrow g x) \in A \rightarrow A \rightarrow A$   
*<proof>*

**lemma** *comp-fun-commute-funpow*: *comp-fun-commute-on*  $(\lambda x. f x \rightsquigarrow g x) A$   
*<proof>*

**lemma** *fold-mset-add-mset*:  
  **assumes** *MA*: *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>*  
**end**

**lemma** *Diff-not-in*:  $a \notin A - \{a\}$  *<proof>*

**context** *abelian-group* **begin**

**lemma** *finsum-restrict*:  
  **assumes** *fA*:  $f : A \rightarrow \text{carrier } G$   
  **and** *restr*: *restrict f A = restrict g A*  
  **shows** *finsum G f A = finsum G g A*  
*<proof>*

**lemma** *minus-nonzero*:  $x : \text{carrier } G \implies x \neq \mathbf{0} \implies \ominus x \neq \mathbf{0}$   
*<proof>*

**end**

**lemma** (**in** *ordered-comm-monoid-add*) *positive-sum*:  
  **assumes** *X* : *finite X*  
  **and** *f* :  $X \rightarrow \{ y :: 'a. y \geq 0 \}$   
  **shows**  $\text{sum } f X \geq 0 \wedge (\text{sum } f X = 0 \longrightarrow f ' X \subseteq \{0\})$   
*<proof>*

**lemma** *insert-union*:  $\text{insert } x X = X \cup \{x\}$  *<proof>*

**context** *vectorspace* **begin**

**lemmas** *lincomb-insert2* = *lincomb-insert*[*unfolded insert-union*[*symmetric*]]

**lemma** *lincomb-restrict*:

**assumes**  $U: U \subseteq \text{carrier } V$   
**and**  $a: a : U \rightarrow \text{carrier } K$   
**and**  $\text{restr}: \text{restrict } a \ U = \text{restrict } b \ U$   
**shows**  $\text{lincomb } a \ U = \text{lincomb } b \ U$

*<proof>*

**lemma** *lindep-span*:

**assumes**  $U: U \subseteq \text{carrier } V$  **and**  $\text{fin}U: \text{finite } U$   
**shows**  $\text{lin-dep } U = (\exists u \in U. u \in \text{span } (U - \{u\}))$  (**is**  $?l = ?r$ )

*<proof>*

**lemma** *not-lindepD*:

**assumes**  $\sim \text{lin-dep } S$   
**and**  $\text{finite } A \ A \subseteq S \ f : A \rightarrow \text{carrier } K \ \text{lincomb } f \ A = \text{zero } V$   
**shows**  $f : A \rightarrow \{\text{zero } K\}$

*<proof>*

**lemma** *span-mem*:

**assumes**  $E: E \subseteq \text{carrier } V$  **and**  $uE: u : E$  **shows**  $u : \text{span } E$

*<proof>*

**lemma** *lincomb-distrib*:

**assumes**  $U: U \subseteq \text{carrier } V$   
**and**  $a: a : U \rightarrow \text{carrier } K$   
**and**  $c: c : \text{carrier } K$   
**shows**  $c \odot_V \text{lincomb } a \ U = \text{lincomb } (\lambda u. c \otimes_K a \ u) \ U$   
(**is**  $- = \text{lincomb } ?b \ U$ )

*<proof>*

**lemma** *span-swap*:

**assumes**  $\text{fin}E[\text{simp}]: \text{finite } E$   
**and**  $E[\text{simp}]: E \subseteq \text{carrier } V$   
**and**  $u[\text{simp}]: u : \text{carrier } V$   
**and**  $uE: u \notin \text{span } E$   
**and**  $v[\text{simp}]: v : \text{carrier } V$   
**and**  $uEv: u : \text{span } (\text{insert } v \ E)$   
**shows**  $\text{span } (\text{insert } u \ E) \subseteq \text{span } (\text{insert } v \ E)$  (**is**  $?L \subseteq ?R$ )

*<proof>*

**lemma** *basis-swap*:

**assumes**  $\text{fin}E[\text{simp}]: \text{finite } E$   
**and**  $u[\text{simp}]: u : \text{carrier } V$   
**and**  $uE[\text{simp}]: u \notin E$   
**and**  $b: \text{basis } (\text{insert } u \ E)$   
**and**  $v[\text{simp}]: v : \text{carrier } V$

**and**  $uEv: u : \text{span } (\text{insert } v E)$   
**shows**  $\text{basis } (\text{insert } v E)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{span-empty}: \text{span } \{\} = \{\text{zero } V\}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{span-self}: \text{assumes } [simp]: v : \text{carrier } V$  **shows**  $v : \text{span } \{v\}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{span-zero}: \text{zero } V : \text{span } U$   $\langle \text{proof} \rangle$

**definition**  $\text{emb where } \text{emb } f D x = (\text{if } x : D \text{ then } f x \text{ else } \text{zero } K)$

**lemma**  $\text{emb-carrier}[simp]: f : D \rightarrow R \implies \text{emb } f D : D \rightarrow R$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{emb-restrict}: \text{restrict } (\text{emb } f D) D = \text{restrict } f D$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{emb-zero}: \text{emb } f D : X - D \rightarrow \{\text{zero } K\}$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{lincomb-clean}$ :  
**assumes**  $A: A \subseteq \text{carrier } V$   
**and**  $Z: Z \subseteq \text{carrier } V$   
**and**  $\text{fin}A: \text{finite } A$   
**and**  $\text{fin}Z: \text{finite } Z$   
**and**  $aA: a : A \rightarrow \text{carrier } K$   
**and**  $aZ: a : Z \rightarrow \{\text{zero } K\}$   
**shows**  $\text{lincomb } a (A \cup Z) = \text{lincomb } a A$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{span-add1}$ :  
**assumes**  $U: U \subseteq \text{carrier } V$  **and**  $v: v : \text{span } U$  **and**  $w: w : \text{span } U$   
**shows**  $v \oplus_V w : \text{span } U$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{span-neg}$ :  
**assumes**  $U: U \subseteq \text{carrier } V$  **and**  $vU: v : \text{span } U$   
**shows**  $\ominus_V v : \text{span } U$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{span-closed}[simp]: U \subseteq \text{carrier } V \implies v : \text{span } U \implies v : \text{carrier } V$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{span-add}$ :  
**assumes**  $U: U \subseteq \text{carrier } V$  **and**  $vU: v : \text{span } U$  **and**  $w[simp]: w : \text{carrier } V$   
**shows**  $w : \text{span } U \longleftrightarrow v \oplus_V w : \text{span } U$  (**is**  $?L \longleftrightarrow ?R$ )

*<proof>*

**lemma** *lincomb-union*:

**assumes**  $U: U \subseteq \text{carrier } V$   
**and**  $U'[simp]: U' \subseteq \text{carrier } V$   
**and**  $disj: U \cap U' = \{\}$   
**and**  $fnU: \text{finite } U$   
**and**  $fnU': \text{finite } U'$   
**and**  $a: a : U \cup U' \rightarrow \text{carrier } K$   
**shows**  $\text{lincomb } a (U \cup U') = \text{lincomb } a U \oplus_V \text{lincomb } a U'$   
*<proof>*

**lemma** *span-union1*:

**assumes**  $U: U \subseteq \text{carrier } V$  **and**  $U': U' \subseteq \text{carrier } V$  **and**  $UU': \text{span } U = \text{span } U'$   
**and**  $W: W \subseteq \text{carrier } V$  **and**  $W': W' \subseteq \text{carrier } V$  **and**  $WW': \text{span } W = \text{span } W'$   
**shows**  $\text{span } (U \cup W) \subseteq \text{span } (U' \cup W')$  (**is**  $?L \subseteq ?R$ )  
*<proof>*

**lemma** *span-Un*:

**assumes**  $U: U \subseteq \text{carrier } V$  **and**  $U': U' \subseteq \text{carrier } V$  **and**  $UU': \text{span } U = \text{span } U'$   
**and**  $W: W \subseteq \text{carrier } V$  **and**  $W': W' \subseteq \text{carrier } V$  **and**  $WW': \text{span } W = \text{span } W'$   
**shows**  $\text{span } (U \cup W) = \text{span } (U' \cup W')$  (**is**  $?L = ?R$ )  
*<proof>*

**lemma** *lincomb-zero*:

**assumes**  $U: U \subseteq \text{carrier } V$  **and**  $a: a : U \rightarrow \{\text{zero } K\}$   
**shows**  $\text{lincomb } a U = \text{zero } V$   
*<proof>*

**end**

**context** *module*

**begin**

**lemma** *lincomb-empty[simp]*:  $\text{lincomb } a \{\} = \mathbf{0}_M$   
*<proof>*

**end**

**context** *linear-map*

**begin**

**interpretation** *Ker*: *vectorspace*  $K$  ( $V.\text{vs } \text{ker}T$ )  
*<proof>*

**interpretation** *im: vectorspace K (W.vs imT)*  
⟨proof⟩

**lemma** *inj-imp-Ker0:*  
**assumes** *inj-on T (carrier V)*  
**shows** *carrier (V.vs kerT) = {0<sub>V</sub>}*  
⟨proof⟩

**lemma** *Ke0-imp-inj:*  
**assumes** *c: carrier (V.vs kerT) = {0<sub>V</sub>}*  
**shows** *inj-on T (carrier V)*  
⟨proof⟩

**corollary** *Ke0-iff-inj: inj-on T (carrier V) = (carrier (V.vs kerT) = {0<sub>V</sub>})*  
⟨proof⟩

**lemma** *inj-imp-dim-ker0:*  
**assumes** *inj-on T (carrier V)*  
**shows** *vectorspace.dim K (V.vs kerT) = 0*  
⟨proof⟩

**lemma** *surj-imp-imT-carrier:*  
**assumes** *surj: T' (carrier V) = carrier W*  
**shows** *(imT) = carrier W*  
⟨proof⟩

**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⟩

**lemma** *lincomb-linear-image:*  
**assumes** *inj-T: inj-on T (carrier V)*  
**assumes** *A-in-V: A ⊆ carrier V* **and** *a: a ∈ (T'A) → carrier K*  
**assumes** *f: finite A*  
**shows** *W.module.lincomb a (T'A) = T (V.module.lincomb (a ∘ T) A)*  
⟨proof⟩

**lemma** *surj-fin-dim:*  
**assumes** *fd: V.fin-dim* **and** *surj: T' (carrier V) = carrier W*  
**shows** *image-fin-dim: W.fin-dim*  
⟨proof⟩

**lemma** *linear-inj-image-is-basis*:  
**assumes** *inj-T*: *inj-on T (carrier V)* **and** *surj*:  $T'(\text{carrier } V) = \text{carrier } W$   
**and** *basis-B*:  $V.\text{basis } B$   
**and** *fin-dim-V*:  $V.\text{fin-dim}$   
**shows**  $W.\text{basis } (T'B)$   
 $\langle \text{proof} \rangle$

**end**

**lemma** (**in** *vectorspace*) *dim1I*:  
**assumes** *gen-set*  $\{v\}$   
**assumes**  $v \neq \mathbf{0}_V$   $v \in \text{carrier } V$   
**shows**  $\text{dim} = 1$   
 $\langle \text{proof} \rangle$

**lemma** (**in** *vectorspace*) *dim0I*:  
**assumes** *gen-set*  $\{\mathbf{0}_V\}$   
**shows**  $\text{dim} = 0$   
 $\langle \text{proof} \rangle$

**lemma** (**in** *vectorspace*) *dim-le1I*:  
**assumes** *gen-set*  $\{v\}$   
**assumes**  $v \in \text{carrier } V$   
**shows**  $\text{dim} \leq 1$   
 $\langle \text{proof} \rangle$

**definition** *find-indices* **where** *find-indices*  $x \text{ } xs \equiv [i \leftarrow [0..<\text{length } xs]. \text{xs}!i = x]$

**lemma** *find-indices-Nil* [*simp*]:  
 $\text{find-indices } x \ [] = []$   
 $\langle \text{proof} \rangle$

**lemma** *find-indices-Cons*:  
 $\text{find-indices } x \ (y\#\text{ys}) = (\text{if } x = y \text{ then } \text{Cons } 0 \text{ else } \text{id}) (\text{map } \text{Suc } (\text{find-indices } x \ \text{ys}))$   
 $\langle \text{proof} \rangle$

**lemma** *find-indices-snoc* [*simp*]:  
 $\text{find-indices } x \ (\text{ys}@[y]) = \text{find-indices } x \ \text{ys} \ @ \ (\text{if } x = y \text{ then } [\text{length } \text{ys}] \text{ else } [])$   
 $\langle \text{proof} \rangle$

**lemma** *mem-set-find-indices* [*simp*]:  $i \in \text{set } (\text{find-indices } x \ \text{xs}) \longleftrightarrow i < \text{length } \text{xs} \wedge \text{xs}!i = x$   
 $\langle \text{proof} \rangle$

**lemma** *distinct-find-indices*:  $\text{distinct } (\text{find-indices } x \ \text{xs})$   
 $\langle \text{proof} \rangle$

**context** *abelian-monoid* **begin**

**definition** *sumlist*  
where  $sumlist\ xs \equiv foldr\ (\oplus)\ xs\ \mathbf{0}$

**lemma** [*simp*]:  
shows *sumlist-Cons*:  $sumlist\ (x\#\ xs) = x \oplus sumlist\ xs$   
and *sumlist-Nil*:  $sumlist\ [] = \mathbf{0}$   
(*proof*)

**lemma** *sumlist-carrier* [*simp*]:  
assumes  $set\ xs \subseteq carrier\ G$  shows  $sumlist\ xs \in carrier\ G$   
(*proof*)

**lemma** *sumlist-neutral*:  
assumes  $set\ xs \subseteq \{\mathbf{0}\}$  shows  $sumlist\ xs = \mathbf{0}$   
(*proof*)

**lemma** *sumlist-append*:  
assumes  $set\ xs \subseteq carrier\ G$  and  $set\ ys \subseteq carrier\ G$   
shows  $sumlist\ (xs\ @\ ys) = sumlist\ xs \oplus sumlist\ ys$   
(*proof*)

**lemma** *sumlist-snoc*:  
assumes  $set\ xs \subseteq carrier\ G$  and  $x \in carrier\ G$   
shows  $sumlist\ (xs\ @\ [x]) = sumlist\ xs \oplus x$   
(*proof*)

**lemma** *sumlist-as-finsum*:  
assumes  $set\ xs \subseteq carrier\ G$  and *distinct*  $xs$  shows  $sumlist\ xs = (\bigoplus_{x \in set\ xs} x)$   
(*proof*)

**lemma** *sumlist-map-as-finsum*:  
assumes  $f : set\ xs \rightarrow carrier\ G$  and *distinct*  $xs$   
shows  $sumlist\ (map\ f\ xs) = (\bigoplus_{x \in set\ xs} f\ x)$   
(*proof*)

**definition** *summset* where  $summset\ M \equiv fold-mset\ (\oplus)\ \mathbf{0}\ M$

**lemma** *summset-empty* [*simp*]:  $summset\ \{\#\} = \mathbf{0}$  (*proof*)

**lemma** *fold-mset-add-carrier*:  $a \in carrier\ G \implies set-mset\ M \subseteq carrier\ G \implies fold-mset\ (\oplus)\ a\ M \in carrier\ G$   
(*proof*)

**lemma** *summset-carrier*[*intro*]:  $set-mset\ M \subseteq carrier\ G \implies summset\ M \in carrier\ G$   
(*proof*)

**lemma** *summset-add-mset*[simp]:  
**assumes**  $a \in \text{carrier } G$  **and**  $MG: \text{set-mset } M \subseteq \text{carrier } G$   
**shows**  $\text{summset } (\text{add-mset } a \ M) = a \oplus \text{summset } M$   
 $\langle \text{proof} \rangle$

**lemma** *sumlist-as-summset*:  
**assumes**  $\text{set } xs \subseteq \text{carrier } G$  **shows**  $\text{sumlist } xs = \text{summset } (\text{mset } xs)$   
 $\langle \text{proof} \rangle$

**lemma** *sumlist-rev*:  
**assumes**  $\text{set } xs \subseteq \text{carrier } G$   
**shows**  $\text{sumlist } (\text{rev } xs) = \text{sumlist } xs$   
 $\langle \text{proof} \rangle$

**lemma** *sumlist-as-fold*:  
**assumes**  $\text{set } xs \subseteq \text{carrier } G$   
**shows**  $\text{sumlist } xs = \text{fold } (\oplus) \ xs \ \mathbf{0}$   
 $\langle \text{proof} \rangle$

**end**

**context** *Module.module* **begin**

**definition** *lincomb-list*  
**where**  $\text{lincomb-list } c \ vs = \text{sumlist } (\text{map } (\lambda i. \ c \ i \ \odot_M \ vs \ ! \ i) \ [0..<\text{length } vs])$

**lemma** *lincomb-list-carrier*:  
**assumes**  $\text{set } vs \subseteq \text{carrier } M$  **and**  $c : \{0..<\text{length } vs\} \rightarrow \text{carrier } R$   
**shows**  $\text{lincomb-list } c \ vs \in \text{carrier } M$   
 $\langle \text{proof} \rangle$

**lemma** *lincomb-list-Nil* [simp]:  $\text{lincomb-list } c \ [] = \mathbf{0}_M$   
 $\langle \text{proof} \rangle$

**lemma** *lincomb-list-Cons* [simp]:  
 $\text{lincomb-list } c \ (v\#\vs) = c \ 0 \ \odot_M \ v \oplus_M \ \text{lincomb-list } (c \ o \ \text{Suc}) \ vs$   
 $\langle \text{proof} \rangle$

**lemma** *lincomb-list-eq-0*:  
**assumes**  $\bigwedge i. \ i < \text{length } vs \implies c \ i \ \odot_M \ vs \ ! \ i = \mathbf{0}_M$   
**shows**  $\text{lincomb-list } c \ vs = \mathbf{0}_M$   
 $\langle \text{proof} \rangle$

**definition** *mk-coeff* **where**  $\text{mk-coeff } vs \ c \ v \equiv R.\text{sumlist } (\text{map } c \ (\text{find-indices } v \ vs))$

**lemma** *mk-coeff-carrier*:  
**assumes**  $c : \{0..<\text{length } vs\} \rightarrow \text{carrier } R$  **shows**  $\text{mk-coeff } vs \ c \ w \in \text{carrier } R$   
 $\langle \text{proof} \rangle$



**lemma** *mk-coeff-Cons*:  
**assumes**  $c : \{0..<\text{length } (v\#vs)\} \rightarrow \text{carrier } R$   
**shows**  $\text{mk-coeff } (v\#vs) \ c = (\lambda w. (\text{if } w = v \text{ then } c \ 0 \ \text{else } \mathbf{0}) \oplus \text{mk-coeff } vs \ (c \ o \ \text{Suc}) \ w)$   
 $\langle \text{proof} \rangle$

**lemma** *mk-coeff-0[simp]*:  
**assumes**  $v \notin \text{set } vs$   
**shows**  $\text{mk-coeff } vs \ c \ v = \mathbf{0}$   
 $\langle \text{proof} \rangle$

**lemma** *lincomb-list-as-lincomb*:  
**assumes**  $vs\text{-}M: \text{set } vs \subseteq \text{carrier } M$  **and**  $c: c : \{0..<\text{length } vs\} \rightarrow \text{carrier } R$   
**shows**  $\text{lincomb-list } c \ vs = \text{lincomb } (\text{mk-coeff } vs \ c) \ (\text{set } vs)$   
 $\langle \text{proof} \rangle$

**definition** *span-list*  $vs \equiv \{\text{lincomb-list } c \ vs \mid c. c : \{0..<\text{length } vs\} \rightarrow \text{carrier } R\}$

**lemma** *in-span-listI*:  
**assumes**  $c : \{0..<\text{length } vs\} \rightarrow \text{carrier } R$  **and**  $v = \text{lincomb-list } c \ vs$   
**shows**  $v \in \text{span-list } vs$   
 $\langle \text{proof} \rangle$

**lemma** *in-span-listE*:  
**assumes**  $v \in \text{span-list } vs$   
**and**  $\bigwedge c. c : \{0..<\text{length } vs\} \rightarrow \text{carrier } R \implies v = \text{lincomb-list } c \ vs \implies \text{thesis}$   
**shows** *thesis*  
 $\langle \text{proof} \rangle$

**lemmas**  $\text{lincomb-insert2} = \text{lincomb-insert}[\text{unfolded insert-union}[\text{symmetric}]]$

**lemma** *lincomb-zero*:  
**assumes**  $U: U \subseteq \text{carrier } M$  **and**  $a: a : U \rightarrow \{\text{zero } R\}$   
**shows**  $\text{lincomb } a \ U = \text{zero } M$   
 $\langle \text{proof} \rangle$

**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 ≡ (| carrier = UNIV, mult = (*), one = 1, zero = 0, add = (+) |)

interpretation class-semiring: semiring class-ring :: 'a :: 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⟩

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 ring) = uminus
  and [class-ring-simps]: a-minus (class-ring :: 'a ring) = minus
  and pow (class-ring :: 'a ring) = (∧)
  and finsum (class-ring :: 'a ring) = sum
⟨proof⟩

interpretation class-crng: crng class-ring :: 'a :: comm-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 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 [class-ring-simps]: finprod class-ring = prod

```

*<proof>*

**definition** *div0* :: 'a :: {one,plus,times,zero} **where**  
  *div0*  $\equiv$  *m-inv* (*class-ring* :: 'a ring) 0

**lemma** *class-field: field* (*class-ring* :: 'a :: field ring) (**is field** ?r)  
*<proof>*

**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* = ( $\wedge$ )  
  **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>*

**lemmas** *matrix-vs-simps* = *module-mat-simps class-ring-simps*

**definition** *class-field* :: 'a :: field ring  
  **where** [*class-ring-simps*]: *class-field*  $\equiv$  *class-ring*

**locale** *matrix-ring* =

**fixes** *n* :: nat  
  **and** *field-type* :: 'a :: field itself

**begin**

**abbreviation** *R* **where** *R*  $\equiv$  *ring-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<sub>m</sub>* *n*  
  **and** *zero R* = *0<sub>m</sub>* *n n*

*<proof>*

**end**

**lemma** *matrix-vs: vectorspace* (*class-ring* :: 'a :: field ring) (*module-mat* TYPE('a)  
*nr nc*)

*<proof>*

```
locale vec-module =  
  fixes f-ty::'a::comm-ring-1 itself  
  and n::nat  
begin
```

```
abbreviation V where V  $\equiv$  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 =  $0_v$  n  
  and module.smult V =  $(\cdot_v)$   
  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) =  $(\hat{\cdot})$   
  and finsum (class-ring :: 'a ring) = sum  
  and finprod (class-ring :: 'a ring) = prod  
  and  $\bigwedge X. X \subseteq \text{UNIV} = \text{True}$   
  and  $\bigwedge x. x \in \text{UNIV} = \text{True}$   
  and  $\bigwedge a A. a \in A \rightarrow \text{UNIV} \equiv \text{True}$   
  and  $\bigwedge P. P \wedge \text{True} \equiv P$   
  and  $\bigwedge P. (\text{True} \implies P) \equiv \text{Trueprop } P$   
<proof>
```

```
end
```

```
locale matrix-vs =  
  fixes nr :: nat  
  and nc :: nat  
  and field-type :: 'a :: field itself  
begin
```

```
abbreviation V where V  $\equiv$  module-mat TYPE('a) nr nc
```

```
sublocale
```

```
  vectorspace class-ring V  
  rewrites carrier V = carrier-mat nr nc  
  and add V = (+)  
  and mult V = (*)  
  and one V =  $1_m$  nr  
  and zero V =  $0_m$  nr nc  
  and smult V =  $(\cdot_m)$ 
```

```

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) = ( $\wedge$ )
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)
  <proof>
end

lemma vec-module: module (class-ring :: 'a :: field ring) (module-vec TYPE('a) n)
  <proof>

lemma vec-vs: vectorspace (class-ring :: 'a :: field ring) (module-vec TYPE('a) n)
  <proof>

locale vec-space =
  fixes f-ty::'a::field itself
  and n::nat
begin

  sublocale vec-module f-ty n<proof>

  sublocale vectorspace class-ring V
  rewrites cV[simp]: carrier V = carrier-vec n
  and [simp]: add V = (+)
  and [simp]: zero V =  $0_v$  n
  and [simp]: smult V = ( $\cdot_v$ )
  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) = ( $\wedge$ )
  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)
  <proof>

lemma finsum-vec[simp]: finsum-vec TYPE('a) n = finsum V
  <proof>

```

**lemma** *finsum-scalar-prod-sum*:

**assumes**  $f: f : U \rightarrow \text{carrier-vec } n$

**and**  $w: w: \text{carrier-vec } n$

**shows**  $\text{finsum } V f U \cdot w = \text{sum } (\lambda u. f u \cdot w) U$

*<proof>*

**lemma** *vec-neg[simp]*: **assumes**  $x : \text{carrier-vec } n$  **shows**  $\ominus_V x = - x$

*<proof>*

**lemma** *finsum-dim*:

$\text{finite } A \implies f \in A \rightarrow \text{carrier-vec } n \implies \text{dim-vec } (\text{finsum } V f A) = n$   
*<proof>*

**lemma** *lincomb-dim*:

**assumes**  $\text{fin}: \text{finite } X$

**and**  $X: X \subseteq \text{carrier-vec } n$

**shows**  $\text{dim-vec } (\text{lincomb } a X) = n$

*<proof>*

**lemma** *finsum-index*:

**assumes**  $i: i < n$

**and**  $f: f \in X \rightarrow \text{carrier-vec } n$

**and**  $X: X \subseteq \text{carrier-vec } n$

**shows**  $\text{finsum } V f X \$ i = \text{sum } (\lambda x. f x \$ i) X$

*<proof>*

**lemma** *lincomb-index*:

**assumes**  $i: i < n$

**and**  $X: X \subseteq \text{carrier-vec } n$

**shows**  $\text{lincomb } a X \$ i = \text{sum } (\lambda x. a x * x \$ i) X$

*<proof>*

**lemma** *append-insert*:  $\text{set } (xs @ [x]) = \text{insert } x (\text{set } xs)$  *<proof>*

**lemma** *lincomb-units*:

**assumes**  $i: i < n$

**shows**  $\text{lincomb } a (\text{set } (\text{unit-vecs } n)) \$ i = a (\text{unit-vec } n i)$

*<proof>*

**lemma** *lincomb-coordinates*:

**assumes**  $v: v : \text{carrier-vec } n$

**defines**  $a \equiv (\lambda u. v \$ (\text{THE } i. u = \text{unit-vec } n i))$

**shows**  $\text{lincomb } a (\text{set } (\text{unit-vecs } n)) = v$

*<proof>*

**lemma** *span-unit-vecs-is-carrier*:  $\text{span } (\text{set } (\text{unit-vecs } n)) = \text{carrier-vec } n$  (**is** ?L = ?R)

*<proof>*

```

lemma fin-dim[simp]: fin-dim
  ⟨proof⟩

lemma unit-vecs-basis: basis (set (unit-vecs n)) ⟨proof⟩

lemma unit-vecs-length[simp]: length (unit-vecs n) = n
  ⟨proof⟩

lemma unit-vecs-distinct: distinct (unit-vecs n)
  ⟨proof⟩

lemma dim-is-n: dim = n
  ⟨proof⟩

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  $\equiv$  ring-mat TYPE('a) nc nr
end

context vec-space
begin
lemma fin-dim-span:
assumes finite A A  $\subseteq$  carrier V
shows vectorspace.fin-dim class-ring (vs (span A))
  ⟨proof⟩

lemma fin-dim-span-cols:
assumes A  $\in$  carrier-mat n nc
shows vectorspace.fin-dim class-ring (vs (span (set (cols A))))
  ⟨proof⟩
end

context vec-module
begin

lemma lincomb-list-as-mat-mult:
assumes  $\forall w \in$  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⟩

lemma lincomb-vec-diff-add:
assumes A: A  $\subseteq$  carrier-vec n
and BA: B  $\subseteq$  A and fin-A: finite A

```

**and**  $f: A \rightarrow UNIV$  **shows**  $lincomb\ f\ A = lincomb\ f\ (A-B) + lincomb\ f\ B$   
 ⟨proof⟩

**lemma** *dim-sumlist*:

**assumes**  $\forall x \in set\ xs.\ dim-vec\ x = n$   
**shows**  $dim-vec\ (M.sumlist\ xs) = n$  ⟨proof⟩

**lemma** *sumlist-nth*:

**assumes**  $\forall x \in set\ xs.\ dim-vec\ x = n$  **and**  $i < n$   
**shows**  $(M.sumlist\ xs)\ \$\ i = sum\ (\lambda j.\ (xs\ !\ j)\ \$\ i)\ \{0..<length\ xs\}$   
 ⟨proof⟩

**lemma** *lincomb-as-lincomb-list-distinct*:

**assumes**  $s: set\ ws \subseteq carrier-vec\ n$  **and**  $d: distinct\ ws$   
**shows**  $lincomb\ f\ (set\ ws) = lincomb-list\ (\lambda i.\ f\ (ws\ !\ i))\ ws$   
 ⟨proof⟩

**end**

**locale** *idom-vec* = *vec-module f-ty* **for** *f-ty* :: 'a :: *idom* *itself*  
**begin**

**lemma** *lin-dep-cols-imp-det-0'*:

**fixes**  $ws$   
**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**  $det\ A = 0$   
 ⟨proof⟩

**lemma** *lin-dep-cols-imp-det-0*:

**assumes**  $A: A \in carrier-mat\ n\ n$  **and**  $ld: lin-dep\ (set\ (cols\ A))$   
**shows**  $det\ A = 0$   
 ⟨proof⟩

**corollary** *lin-dep-rows-imp-det-0*:

**assumes**  $A: A \in carrier-mat\ n\ n$  **and**  $ld: lin-dep\ (set\ (rows\ A))$   
**shows**  $det\ A = 0$   
 ⟨proof⟩

**lemma** *det-not-0-imp-lin-indpt-rows*:

**assumes**  $A: A \in carrier-mat\ n\ n$  **and**  $det: det\ A \neq 0$   
**shows**  $lin-indpt\ (set\ (rows\ A))$   
 ⟨proof⟩

**lemma** *upper-triangular-imp-lin-indpt-rows*:

**assumes**  $A: A \in carrier-mat\ n\ n$   
**and**  $tri: upper-triangular\ A$   
**and**  $diag: 0 \notin set\ (diag-mat\ A)$



**shows** *lin-indpt* (*set (rows A)*)  
 ⟨*proof*⟩

**lemma** *lincomb-as-lincomb-list*:  
**fixes** *ws f*  
**assumes** *s: set ws ⊆ carrier-vec n*  
**shows** *lincomb f (set ws) = lincomb-list (λi. if ∃j<i. ws!i = ws!j then 0 else f (ws ! i)) ws*  
 ⟨*proof*⟩

**lemma** *span-list-as-span*:  
**assumes** *set vs ⊆ carrier-vec n*  
**shows** *span-list vs = span (set vs)*  
 ⟨*proof*⟩

**lemma** *in-spanI[intro]*:  
**assumes** *v = lincomb a A finite A A ⊆ W*  
**shows** *v ∈ span W*  
 ⟨*proof*⟩

**lemma** *in-spanE*:  
**assumes** *v ∈ span W*  
**shows**  $\exists a A. v = \text{lincomb } a A \wedge \text{finite } A \wedge A \subseteq W$   
 ⟨*proof*⟩

**declare** *in-own-span[intro]*

**lemma** *smult-in-span*:  
**assumes** *W ⊆ carrier-vec n and insp: x ∈ span W*  
**shows** *c ·<sub>v</sub> x ∈ span W*  
 ⟨*proof*⟩

**lemma** *span-subsetI*: **assumes** *ws: ws ⊆ carrier-vec n*  
*us ⊆ span ws*  
**shows** *span us ⊆ span ws*  
 ⟨*proof*⟩

**end**

**context** *vec-space* **begin**  
**sublocale** *idom-vec*⟨*proof*⟩

**lemma** *sumlist-in-span*: **assumes** *W: W ⊆ carrier-vec n*  
**shows**  $(\bigwedge x. x \in \text{set } xs \implies x \in \text{span } W) \implies \text{sumlist } xs \in \text{span } W$   
 ⟨*proof*⟩

**lemma** *span-span[simp]*:  
**assumes** *W ⊆ carrier-vec n*

**shows**  $\text{span} (\text{span } W) = \text{span } W$   
 ⟨proof⟩

**lemma** *upper-triangular-imp-basis*:  
**assumes**  $A: A \in \text{carrier-mat } n \ n$   
**and**  $\text{tri}: \text{upper-triangular } A$   
**and**  $\text{diag}: 0 \notin \text{set} (\text{diag-mat } A)$   
**shows**  $\text{basis} (\text{set} (\text{rows } A))$   
 ⟨proof⟩

**lemma** *fin-dim-span-rows*:  
**assumes**  $A: A \in \text{carrier-mat } nr \ n$   
**shows**  $\text{vectorspace.fin-dim class-ring} (\text{vs} (\text{span} (\text{set} (\text{rows } A))))$   
 ⟨proof⟩

**definition** *row-space*  $B = \text{span} (\text{set} (\text{rows } B))$   
**definition** *col-space*  $B = \text{span} (\text{set} (\text{cols } B))$

**lemma** *row-space-eq-col-space-transpose*:  
**shows**  $\text{row-space } A = \text{col-space } A^T$   
 ⟨proof⟩

**lemma** *col-space-eq-row-space-transpose*:  
**shows**  $\text{col-space } A = \text{row-space } A^T$   
 ⟨proof⟩

**lemma** *col-space-eq*:  
**assumes**  $A: A \in \text{carrier-mat } n \ nc$   
**shows**  $\text{col-space } A = \{y \in \text{carrier-vec} (\text{dim-col } A). \exists x \in \text{carrier-vec} (\text{dim-row } A). A *_v x = y\}$   
 ⟨proof⟩

**lemma** *vector-space-row-space*:  
**assumes**  $A: A \in \text{carrier-mat } nr \ n$   
**shows**  $\text{vectorspace class-ring} (\text{vs} (\text{row-space } A))$   
 ⟨proof⟩

**lemma** *row-space-eq*:  
**assumes**  $A: A \in \text{carrier-mat } nr \ n$   
**shows**  $\text{row-space } A = \{w \in \text{carrier-vec} (\text{dim-col } A). \exists y \in \text{carrier-vec} (\text{dim-row } A). A^T *_v y = w\}$   
 ⟨proof⟩

**lemma** *row-space-is-preserved*:  
**assumes**  $\text{inv-}P: \text{invertible-mat } P$  **and**  $P: P \in \text{carrier-mat } m \ m$  **and**  $A: A \in \text{carrier-mat } m \ n$   
**shows**  $\text{row-space} (P * A) = \text{row-space } A$

*<proof>*

**end**

**context** *vec-module* **begin**

**lemma** *R-sumlist[simp]*:  $R.sumlist = sum-list$

*<proof>*

**lemma** *sumlist-dim*: **assumes**  $\bigwedge x. x \in set\ xs \implies x \in carrier-vec\ n$

**shows**  $dim-vec\ (sumlist\ xs) = n$

*<proof>*

**lemma** *sumlist-vec-index*: **assumes**  $\bigwedge x. x \in set\ xs \implies x \in carrier-vec\ n$

**and**  $i < n$

**shows**  $sumlist\ xs\ \$\ i = sum-list\ (map\ (\lambda x. x\ \$\ i)\ xs)$

*<proof>*

**lemma** *scalar-prod-left-sum-distrib*:

**assumes**  $vs: \bigwedge v. v \in set\ vvs \implies v \in carrier-vec\ n$  **and**  $w: w \in carrier-vec\ n$

**shows**  $sumlist\ vvs \cdot w = sum-list\ (map\ (\lambda v. v \cdot w)\ vvs)$

*<proof>*

**lemma** *scalar-prod-right-sum-distrib*:

**assumes**  $vs: \bigwedge v. v \in set\ vvs \implies v \in carrier-vec\ n$  **and**  $w: w \in carrier-vec\ n$

**shows**  $w \cdot sumlist\ vvs = sum-list\ (map\ (\lambda v. w \cdot v)\ vvs)$

*<proof>*

**lemma** *lincomb-list-add-vec-2*: **assumes**  $us: set\ us \subseteq carrier-vec\ n$

**and**  $x: x = lincomb-list\ lc\ (us\ [i := us\ !\ i + c \cdot_v\ us\ !\ j])$

**and**  $i: j < length\ us\ i < length\ us\ i \neq j$

**shows**  $x = lincomb-list\ (lc\ (j := lc\ j + lc\ i * c))\ us\ (is - = ?x)$

*<proof>*

**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**  $x = lincomb-list\ (lc\ (j := lc\ j - lc\ i * c))\ (us\ [i := us\ !\ i + c \cdot_v\ us\ !\ j])\ (is - = ?x)$

*<proof>*

**end**

**context** *vec-space*

**begin**

**lemma** *add-vec-span*: **assumes**  $us: set\ us \subseteq carrier-vec\ n$

**and**  $i: j < length\ us\ i < length\ us\ i \neq j$

**shows**  $span\ (set\ us) = span\ (set\ (us\ [i := us\ !\ i + c \cdot_v\ us\ !\ j]))\ (is - = span\ (set\ ?us))$

*<proof>*

*<proof>*

**lemma** *prod-in-span[intro!]*:

**assumes**  $b \in \text{carrier-vec } n \ S \subseteq \text{carrier-vec } n \ a = 0 \vee b \in \text{span } S$   
**shows**  $a \cdot_v b \in \text{span } S$

*<proof>*

**lemma** *det-nonzero-congruence*:

**assumes**  $eq: A * M = B * M$  **and**  $det: det (M::'a \text{ mat}) \neq 0$

**and**  $M: M \in \text{carrier-mat } n \ n$  **and**  $carr: A \in \text{carrier-mat } n \ n \ B \in \text{carrier-mat } n \ n$

**shows**  $A = B$

*<proof>*

**lemma** *mat-of-rows-mult-as-finsum*:

**assumes**  $v \in \text{carrier-vec } (length \ lst) \ \wedge \ i. \ i < length \ lst \implies \ lst \ ! \ i \in \text{carrier-vec } n$

**defines**  $f \ l \equiv \text{sum } (\lambda \ i. \ \text{if } l = lst \ ! \ i \ \text{then } v \ \$ \ i \ \text{else } 0) \ \{0..<length \ lst\}$

**shows**  $\text{mat-of-cols-mult-as-finsum: mat-of-cols } n \ lst * _v \ v = \text{lincomb } f \ (\text{set } lst)$

*<proof>*

**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*  $\equiv$

$\forall i < length \ vs. \ \forall j < length \ vs. \ vs \ ! \ i \cdot c \ vs \ ! \ j = 0 \iff i \neq j$

**lemma** *corthogonalD[elim]*:

$corthogonal \ vs \implies i < length \ vs \implies j < length \ vs \implies$

$vs \ ! \ i \cdot c \ vs \ ! \ j = 0 \iff i \neq j$

*<proof>*

**lemma** *corthogonalI[intro]*:

$(\bigwedge i j. i < \text{length } vs \implies j < \text{length } vs \implies vs ! i \cdot c \ vs ! j = 0 \iff i \neq j) \implies$   
*corthogonal vs*  
 <proof>

**lemma** *corthogonal-distinct*: *corthogonal us*  $\implies$  *distinct us*  
 <proof>

**lemma** *corthogonal-sort*:  
**assumes** *dist'*: *distinct us'*  
**and** *mem*: *set us = set us'*  
**shows** *corthogonal us*  $\implies$  *corthogonal us'*  
 <proof>

## 16.2 The Algorithm

**fun** *adjuster* :: *nat*  $\Rightarrow$  '*a* :: *conjugatable-field vec*  $\Rightarrow$  '*a vec list*  $\Rightarrow$  '*a vec*  
**where** *adjuster n w []* =  $0_v \ n$   
 | *adjuster n w (u # us)* =  $-(w \cdot c \ u) / (u \cdot c \ u) \cdot_v \ u + \text{adjuster } n \ w \ us$

The following formulation is easier to analyze, but outputs of the sub-routine 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*  $\Rightarrow$  '*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 # ws)* =  
 (let *u = adjuster n w us + w* in  
*u # gram-schmidt-sub2 n (u # ws) ws*)

**lemma** *gram-schmidt-sub-eq*:  
*rev (gram-schmidt-sub n us ws)* = *rev us @ gram-schmidt-sub2 n us ws*  
 <proof>

**lemma** *gram-schmidt-code*[code]:  
*gram-schmidt n ws* = *gram-schmidt-sub2 n [] ws*  
 <proof>

## 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**  $U: \text{set } us \subseteq \text{carrier-vec } n$   
**and**  $\text{dist: distinct } (us :: 'a \text{ vec list})$   
**shows**  $\text{adjuster } n \ w \ us = \text{finsum } V \ (\lambda u. -(w \cdot c \ u)/(u \cdot c \ u) \cdot_v \ u) \ (\text{set } us)$   
 $\langle \text{proof} \rangle$

**lemma** *adjuster-lincomb*:  
**assumes**  $w: (w :: 'a \text{ vec}) : \text{carrier-vec } n$   
**and**  $us: \text{set } (us :: 'a \text{ vec list}) \subseteq \text{carrier-vec } n$   
**and**  $\text{dist: distinct } us$   
**shows**  $\text{adjuster } n \ w \ us = \text{lincomb } (\lambda u. -(w \cdot c \ u)/(u \cdot c \ u)) \ (\text{set } us)$   
 $(\text{is } - = \text{lincomb } ?a \ -)$   
 $\langle \text{proof} \rangle$

**lemma** *adjuster-in-span*:  
**assumes**  $w: (w :: 'a \text{ vec}) : \text{carrier-vec } n$   
**and**  $us: \text{set } (us :: 'a \text{ vec list}) \subseteq \text{carrier-vec } n$   
**and**  $\text{dist: distinct } us$   
**shows**  $\text{adjuster } n \ w \ us : \text{span } (\text{set } us)$   
 $\langle \text{proof} \rangle$

**lemma** *adjuster-carrier[simp]*:  
**assumes**  $w: (w :: 'a \text{ vec}) : \text{carrier-vec } n$   
**and**  $us: \text{set } (us :: 'a \text{ vec list}) \subseteq \text{carrier-vec } n$   
**and**  $\text{dist: distinct } us$   
**shows**  $\text{adjuster } n \ w \ us : \text{carrier-vec } n$   
 $\langle \text{proof} \rangle$

**lemma** *adjust-not-in-span*:  
**assumes**  $w[\text{simp}]: (w :: 'a \text{ vec}) : \text{carrier-vec } n$   
**and**  $us: \text{set } (us :: 'a \text{ vec list}) \subseteq \text{carrier-vec } n$   
**and**  $\text{dist: distinct } us$   
**and**  $\text{ind: } w \notin \text{span } (\text{set } us)$   
**shows**  $\text{adjuster } n \ w \ us + w \notin \text{span } (\text{set } us)$   
 $\langle \text{proof} \rangle$

**lemma** *adjust-not-mem*:  
**assumes**  $w[\text{simp}]: (w :: 'a \text{ vec}) : \text{carrier-vec } n$   
**and**  $us: \text{set } (us :: 'a \text{ vec list}) \subseteq \text{carrier-vec } n$   
**and**  $\text{dist: distinct } us$   
**and**  $\text{ind: } w \notin \text{span } (\text{set } us)$   
**shows**  $\text{adjuster } n \ w \ us + w \notin \text{set } us$   
 $\langle \text{proof} \rangle$

**lemma** *adjust-in-span*:  
**assumes**  $w[\text{simp}]: (w :: 'a \text{ vec}) : \text{carrier-vec } n$   
**and**  $us: \text{set } (us :: 'a \text{ vec list}) \subseteq \text{carrier-vec } n$   
**and**  $\text{dist: distinct } us$

**shows**  $\text{adjuster } n \ w \ us + w : \text{span } (\text{insert } w \ (\text{set } us))$  (**is**  $?v + - : \text{span } ?U$ )  
 ⟨proof⟩

**lemma** *adjust-not-lindep*:

**assumes**  $w[\text{simp}]$ :  $(w :: 'a \ \text{vec}) : \text{carrier-vec } n$   
**and**  $us$ :  $\text{set } (us :: 'a \ \text{vec list}) \subseteq \text{carrier-vec } n$   
**and**  $dist$ :  $\text{distinct } us$   
**and**  $wus$ :  $w \notin \text{span } (\text{set } us)$   
**and**  $ind$ :  $\sim \text{lin-dep } (\text{set } us)$   
**shows**  $\sim \text{lin-dep } (\text{insert } (\text{adjuster } n \ w \ us + w) \ (\text{set } us))$   
 (**is**  $\sim - (\text{insert } ?v -)$ )  
 ⟨proof⟩

**lemma** *adjust-preserves-span*:

**assumes**  $w[\text{simp}]$ :  $(w :: 'a \ \text{vec}) : \text{carrier-vec } n$   
**and**  $us$ :  $\text{set } (us :: 'a \ \text{vec list}) \subseteq \text{carrier-vec } n$   
**and**  $dist$ :  $\text{distinct } us$   
**shows**  $w : \text{span } (\text{set } us) \longleftrightarrow \text{adjuster } n \ w \ us + w : \text{span } (\text{set } us)$   
 (**is**  $- \longleftrightarrow ?v + - : -$ )  
 ⟨proof⟩

**lemma** *in-span-adjust*:

**assumes**  $w[\text{simp}]$ :  $(w :: 'a \ \text{vec}) : \text{carrier-vec } n$   
**and**  $us$ :  $\text{set } (us :: 'a \ \text{vec list}) \subseteq \text{carrier-vec } n$   
**and**  $dist$ :  $\text{distinct } us$   
**shows**  $w : \text{span } (\text{insert } (\text{adjuster } n \ w \ us + w) \ (\text{set } us))$   
 (**is**  $- : \text{span } (\text{insert } ?v -)$ )  
 ⟨proof⟩

**lemma** *adjust-zero*:

**assumes**  $U$ :  $\text{set } (us :: 'a \ \text{vec list}) \subseteq \text{carrier-vec } n$   
**and**  $orth$ :  $\text{corthogonal } us$   
**and**  $w[\text{simp}]$ :  $w : \text{carrier-vec } n$   
**and**  $i$ :  $i < \text{length } us$   
**shows**  $(\text{adjuster } n \ w \ us + w) \cdot c \ us!i = 0$   
 ⟨proof⟩

**lemma** *adjust-nonzero*:

**assumes**  $U$ :  $\text{set } (us :: 'a \ \text{vec list}) \subseteq \text{carrier-vec } n$   
**and**  $dist$ :  $\text{distinct } us$   
**and**  $w[\text{simp}]$ :  $w : \text{carrier-vec } n$   
**and**  $wsU$ :  $w \notin \text{span } (\text{set } us)$   
**shows**  $\text{adjuster } n \ w \ us + w \neq 0_v \ n$  (**is**  $?a + - \neq -$ )  
 ⟨proof⟩

**lemma** *adjust-orthogonal*:

**assumes**  $U$ :  $\text{set } (us :: 'a \ \text{vec list}) \subseteq \text{carrier-vec } n$   
**and**  $orth$ :  $\text{corthogonal } us$   
**and**  $w[\text{simp}]$ :  $w : \text{carrier-vec } n$

**and**  $wsU: w \notin \text{span } (\text{set } us)$   
**shows**  $\text{corthogonal } ((\text{adjuster } n \ w \ us + w) \# \ us)$   
 (**is**  $\text{corthogonal } (?aw \# \ -)$ )  
 $\langle \text{proof} \rangle$

**lemma** *gram-schmidt-sub-span*:  
**assumes**  $w[\text{simp}]: w : \text{carrier-vec } n$   
**and**  $us: \text{set } us \subseteq \text{carrier-vec } n$   
**and**  $dist: \text{distinct } us$   
**shows**  $\text{span } (\text{set } ((\text{adjuster } n \ w \ us + w) \# \ us)) = \text{span } (\text{set } (w \# \ us))$   
 (**is**  $\text{span } (\text{set } (?v \# \ -)) = \text{span } ?wU$ )  
 $\langle \text{proof} \rangle$

**lemma** *gram-schmidt-sub-result*:  
**assumes**  $\text{gram-schmidt-sub } n \ us \ ws = us'$   
**and**  $\text{set } ws \subseteq \text{carrier-vec } n$   
**and**  $\text{set } us \subseteq \text{carrier-vec } n$   
**and**  $\text{distinct } (us \ @ \ ws)$   
**and**  $\sim \text{lin-dep } (\text{set } (us \ @ \ ws))$   
**and**  $\text{corthogonal } us$   
**shows**  $\text{set } us' \subseteq \text{carrier-vec } n \wedge$   
 $\text{distinct } us' \wedge$   
 $\text{corthogonal } us' \wedge$   
 $\text{span } (\text{set } (us \ @ \ ws)) = \text{span } (\text{set } us') \wedge \text{length } us' = \text{length } us + \text{length } ws$   
 $\langle \text{proof} \rangle$

**lemma** *gram-schmidt-hd* [*simp*]:  
**assumes** [*simp*]:  $w : \text{carrier-vec } n$  **shows**  $\text{hd } (\text{gram-schmidt } n \ (w \# \ ws)) = w$   
 $\langle \text{proof} \rangle$

**theorem** *gram-schmidt-result*:  
**assumes**  $ws: \text{set } ws \subseteq \text{carrier-vec } n$   
**and**  $dist: \text{distinct } ws$   
**and**  $ind: \sim \text{lin-dep } (\text{set } ws)$   
**and**  $us: us = \text{gram-schmidt } n \ ws$   
**shows**  $\text{span } (\text{set } ws) = \text{span } (\text{set } us)$   
**and**  $\text{corthogonal } us$   
**and**  $\text{set } us \subseteq \text{carrier-vec } n$   
**and**  $\text{length } us = \text{length } ws$   
**and**  $\text{distinct } us$

$\langle \text{proof} \rangle$

**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  $A = PBQ$ ,  $B$  is upper-triangular and  $PQ = 1$ . 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  $\Rightarrow$  'a vec

**where** *vec-inv* v = 1 / (v · c v) ·<sub>v</sub> conjugate v

**lemma** *vec-inv-closed[simp]*: v ∈ carrier-vec n  $\implies$  *vec-inv* v ∈ carrier-vec n

*<proof>*

**lemma** *vec-inv-dim[simp]*: dim-vec (*vec-inv* v) = dim-vec v

*<proof>*

**lemma** *vec-inv[simp]*:

**assumes** v: v : carrier-vec n

**and** v0: (v::'a::conjugatable-ordered-field vec)  $\neq$  0<sub>v</sub> n

**shows** *vec-inv* v · v = 1

*<proof>*

**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 vs)

(is *inverts-mat* ?W ?V)

*<proof>*

**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**  $\text{mat-adjoint } A \equiv \text{mat-of-rows } (\text{dim-row } A) (\text{map conjugate } (\text{cols } A))$

**definition**  $\text{corthogonal-mat} :: 'a::\text{conjugatable-field mat} \Rightarrow \text{bool}$

**where**  $\text{corthogonal-mat } A \equiv$   
 $\text{let } B = \text{mat-adjoint } A * A \text{ in}$   
 $\text{diagonal-mat } B \wedge (\forall i < \text{dim-col } A. B \$\$ (i,i) \neq 0)$

**lemma**  $\text{corthogonal-matD}[\text{elim}]$ :

**assumes**  $\text{orth}: \text{corthogonal-mat } A$   
**and**  $i: i < \text{dim-col } A$   
**and**  $j: j < \text{dim-col } A$   
**shows**  $(\text{col } A \ i \cdot \text{col } A \ j = 0) = (i \neq j)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{corthogonal-matI}[\text{intro}]$ :

**assumes**  $(\bigwedge i \ j. i < \text{dim-col } A \Longrightarrow j < \text{dim-col } A \Longrightarrow (\text{col } A \ i \cdot \text{col } A \ j = 0) = (i \neq j))$   
**shows**  $\text{corthogonal-mat } A$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{corthogonal-inv-result}$ :

**assumes**  $o: \text{corthogonal-mat } (A::'a::\text{conjugatable-field mat})$   
**shows**  $\text{inverts-mat } (\text{corthogonal-inv } A) \ A$   
 $\langle \text{proof} \rangle$

extends a vector to a basis

**definition**  $\text{basis-completion} :: 'a::\text{ring-1 vec} \Rightarrow 'a \text{ vec list}$  **where**

$\text{basis-completion } v \equiv \text{let}$   
 $n = \text{dim-vec } v;$   
 $\text{drop-index} = \text{hd } ([i \ . \ i < - [0..<n], v \$ i \neq 0]);$   
 $\text{vs} = [\text{unit-vec } n \ i. \ i < - [0..<n], i \neq \text{drop-index}]$   
 $\text{in } v \# \text{vs}$

**lemma**  $(\text{in vec-space}) \text{basis-completion}$ : **fixes**  $v :: 'a :: \text{field vec}$

**assumes**  $v: v \in \text{carrier-vec } n$   
**and**  $v0: v \neq 0_v \ n$   
**shows**  
 $\text{basis } (\text{set } (\text{basis-completion } v))$   
 $\text{set } (\text{basis-completion } v) \subseteq \text{carrier-vec } n$   
 $\text{span } (\text{set } (\text{basis-completion } v)) = \text{carrier-vec } n$   
 $\text{distinct } (\text{basis-completion } v)$   
 $\neg \text{lin-dep } (\text{set } (\text{basis-completion } v))$   
 $\text{length } (\text{basis-completion } v) = n$   
 $\text{hd } (\text{basis-completion } v) = v$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{orthogonal-mat-of-cols}$ :

**assumes**  $W: \text{set } ws \subseteq \text{carrier-vec } n$   
**and**  $\text{orth}: \text{corthogonal } ws$

**and**  $len: length\ ws = n$   
**shows** *corthogonal-mat* (*mat-of-cols*  $n\ ws$ ) (**is** *corthogonal-mat* ? $W$ )  
*<proof>*

**lemma** *corthogonal-col-ev-0*: **fixes**  $A :: 'a :: conjugatable-ordered-field\ mat$   
**assumes**  $A: A \in carrier-mat\ n\ n$   
**and**  $v: v \in carrier-vec\ n$   
**and**  $v0: v \neq 0_v\ n$   
**and** *eigen[simp]*:  $A *_v\ v = e *_v\ v$   
**and**  $n: n \neq 0$   
**and** *hdws*:  $hd\ ws = v$   
**and**  $ws: set\ ws \subseteq carrier-vec\ n\ corthogonal\ ws\ length\ ws = n$   
**defines**  $W == mat-of-cols\ n\ ws$   
**defines**  $W' == corthogonal-inv\ W$   
**defines**  $A' == W' * A * W$   
**shows**  $col\ A'\ 0 = vec\ n\ (\lambda\ i.\ if\ i = 0\ then\ e\ else\ 0)$   
*<proof>*

Schur decomposition

**fun** *schur-decomposition* ::  $'a::conjugatable-field\ mat \Rightarrow 'a\ list \Rightarrow 'a\ mat \times 'a\ mat$   
 $\times 'a\ mat$  **where**  
*schur-decomposition*  $A\ [] = (A,\ 1_m\ (dim-row\ A),\ 1_m\ (dim-row\ A))$   
| *schur-decomposition*  $A\ (e\ \#\ es) = (let$   
 $n = dim-row\ A;$   
 $n1 = n - 1;$   
 $v = find-eigenvector\ A\ e;$   
 $ws = gram-schmidt\ n\ (basis-completion\ v);$   
 $W = mat-of-cols\ n\ ws;$   
 $W' = corthogonal-inv\ W;$   
 $A' = W' * A * W;$   
 $(A1,A2,A0,A3) = split-block\ A'\ 1\ 1;$   
 $(B,P,Q) = schur-decomposition\ A3\ es;$   
 $z-row = (0_m\ 1\ n1);$   
 $z-col = (0_m\ n1\ 1);$   
 $one-1 = 1_m\ 1$   
**in** (*four-block-mat*  $A1\ (A2 * P)\ A0\ B,$   
 $W * four-block-mat\ one-1\ z-row\ z-col\ P,$   
 $four-block-mat\ one-1\ z-row\ z-col\ Q * W')$ )

**theorem** *schur-decomposition*:

**assumes**  $A: (A::'a::conjugatable-ordered-field\ mat) \in carrier-mat\ n\ n$   
**and**  $c: char-poly\ A = (\prod\ (e :: 'a) \leftarrow es.\ [-\ e,\ 1:])$   
**and**  $B: schur-decomposition\ A\ es = (B,P,Q)$   
**shows** *similar-mat-wit*  $A\ B\ P\ Q \wedge upper-triangular\ B \wedge diag-mat\ B = es$   
*<proof>*

**definition** *schur-upper-triangular* ::  $'a::conjugatable-field\ mat \Rightarrow 'a\ list \Rightarrow 'a\ mat$   
**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) ∈ carrier-mat *n n*

**and linear**: char-poly *A* = (∏ *a* ← es. [:- *a*, 1:])

**defines** *B*: *B* ≡ schur-upper-triangular *A es*

**shows** *B* ∈ carrier-mat *n n* upper-triangular *B* similar-mat *A B*

⟨proof⟩

**lemma** *schur-decomposition-exists*: **assumes** *A*: *A* ∈ carrier-mat *n n*

**and linear**: char-poly *A* = (∏ (*a* :: '*a* :: conjugatable-ordered-field) ← es. [:- *a*, 1:])

**shows** ∃ *B* ∈ carrier-mat *n n*. upper-triangular *B* ∧ similar-mat *A B*

⟨proof⟩

**lemma** *char-poly-0-block*: **fixes** *A* :: '*a* :: conjugatable-ordered-field mat

**assumes** *A*: *A* = four-block-mat *B C* (*0<sub>m</sub>* *m n*) *D*

**and linear***B*: ∃ es. char-poly *B* = (∏ *a* ← es. [:- *a*, 1:])

**and linear***D*: ∃ es. char-poly *D* = (∏ *a* ← es. [:- *a*, 1:])

**and** *B*: *B* ∈ carrier-mat *n n*

**and** *C*: *C* ∈ carrier-mat *n m*

**and** *D*: *D* ∈ carrier-mat *m m*

**shows** char-poly *A* = char-poly *B* \* char-poly *D*

⟨proof⟩

**lemma** *char-poly-0-block'*: **fixes** *A* :: '*a* :: conjugatable-ordered-field mat

**assumes** *A*: *A* = four-block-mat *B* (*0<sub>m</sub>* *n m*) *C D*

**and linear***B*: ∃ es. char-poly *B* = (∏ *a* ← es. [:- *a*, 1:])

**and linear***D*: ∃ es. char-poly *D* = (∏ *a* ← es. [:- *a*, 1:])

**and** *B*: *B* ∈ carrier-mat *n n*

**and** *C*: *C* ∈ carrier-mat *m n*

**and** *D*: *D* ∈ carrier-mat *m m*

**shows** char-poly *A* = char-poly *B* \* char-poly *D*

⟨proof⟩

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  $A_{ij}$  with EV  $a$  left of it, and EV  $b$  below of it, perform *add-col-sub-row*  $(A_{ij} / (b - a)) i j$ . 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*  $(i + 2) j$ , ... *swap-cols-rows*  $(j - 1) j$ . Afterwards row  $j$  will be at row  $i + 1$ , and rows  $i + 1, \dots, j - 1$  will be moved to  $i + 2, \dots, j$ . 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  $A_{ik}$  in rows of form  $0 \dots 0 \text{ ev } 1 \ 0 \dots 0 \ A_{ik}$ : *add-col-sub-row*  $(- A_{ik}) (i + 1) k$ . Perform elimination by increasing  $i$ .
  - b) Figure out largest JB (of  $n - 1 \times n - 1$  sub-matrix) with lowest row of form  $0 \dots 0 \text{ ev } 0 \dots 0 \ A_{lk}$  where  $A_{lk} \neq 0$ , and set  $x := A_{lk}$ .
  - c) If such a JB does not exist, continue with next column. Otherwise, eliminate all other non-zero-entries  $y := A_{ik}$  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*  $(y / x) (i - 2) (l - 2)$ , ... where the number of steps is determined by the size of the JB left-above of  $A_{ik}$ . Perform an iteration over  $i$ .
  - d) Normalize value in row  $l$  to 1: *mult-col-div-row*  $((1::'a) / x) k$ .
  - e) Move the 1 down from row  $l$  to row  $k - 1$ : *swap-cols-rows*  $(l + 1) k$ , *swap-cols-rows*  $(l + 2) k$ , ..., *swap-cols-rows*  $(k - 1) k$ .

## 18.2 Real Algorithm

**fun** *lookup-ev* :: 'a ⇒ nat ⇒ 'a mat ⇒ 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 ⇒ nat ⇒ 'a mat ⇒ 'a mat **where**

*swap-cols-rows-block* i j A = (if i < j then

*swap-cols-rows-block* (Suc i) j (*swap-cols-rows* i j A) else A)

⟨proof⟩

**termination** ⟨proof⟩

**fun** *identify-block* :: 'a :: one mat ⇒ nat ⇒ nat **where**

*identify-block* A 0 = 0

| *identify-block* A (Suc i) = (if A \$\$ (i,Suc i) = 1 then  
*identify-block* A i else (Suc i))

**function** *identify-blocks-main* :: 'a :: ring-1 mat ⇒ nat ⇒ (nat × nat) list ⇒ (nat × 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)  
)

⟨proof⟩

**definition** *identify-blocks* :: 'a :: ring-1 mat ⇒ nat ⇒ (nat × nat) list **where**

*identify-blocks* A i = *identify-blocks-main* A i []

**fun** *find-largest-block* :: nat × nat ⇒ (nat × nat) list ⇒ nat × nat **where**

*find-largest-block* block [] = block

| *find-largest-block* (m-start,m-end) ((i-start,i-end) # blocks) =  
(if i-end - i-start ≥ 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 ⇒ nat ⇒ 'a mat ⇒ nat option **where**

*lookup-other-ev* ev 0 A = None

| *lookup-other-ev* ev (Suc i) A = (if A \$\$ (i,i) ≠ ev then Some i else *lookup-other-ev* ev i A)

**partial-function** (*tailrec*) *partition-ev-blocks* :: 'a mat ⇒ 'a mat list ⇒ 'a mat list **where**

[code]: *partition-ev-blocks* A bs = (let n = dim-row A in

if n = 0 then bs

else (case *lookup-other-ev* (A \$\$ (n-1, n-1)) (n-1) A of  
None ⇒ A # bs

| Some i ⇒ case *split-block* A (Suc i) (Suc i) of (UL,-,LR) ⇒ *partition-ev-blocks* UL (LR # bs)))

```

context
  fixes n :: nat
  and ty :: 'a :: field itself
begin

function step-1-main :: nat ⇒ nat ⇒ 'a mat ⇒ 'a mat where
  step-1-main i j A = (if j ≥ n then A else if i = 0 then step-1-main (j+1) (j+1)
  A
  else let
    i' = i - 1;
    ev-left = A $$ (i',i');
    ev-below = A $$ (j,j);
    aij = A $$ (i',j);
    B = if (ev-left ≠ ev-below ∧ aij ≠ 0) then add-col-sub-row (aij / (ev-below
  - ev-left)) i' j A else A
  in step-1-main i' j B)
  ⟨proof⟩
termination ⟨proof⟩

function step-2-main :: nat ⇒ 'a mat ⇒ 'a mat where
  step-2-main j A = (if j ≥ n then A
  else
  let ev = A $$ (j,j);
  B = (case lookup-ev ev j A of
  None ⇒ A
  | Some i ⇒ swap-cols-rows-block (Suc i) j A
  )
  in step-2-main (Suc j) B)
  ⟨proof⟩
termination ⟨proof⟩

fun step-3-a :: nat ⇒ nat ⇒ 'a mat ⇒ 'a mat where
  step-3-a 0 j A = A
  | step-3-a (Suc i) j A = (let
  aij = A $$ (i,j);
  B = (if A $$ (i,i+1) = 1 ∧ aij ≠ 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 ⇒ nat ⇒ nat ⇒ nat ⇒ 'a mat ⇒ 'a mat where
  step-3-c-inner-loop val l i 0 A = A
  | step-3-c-inner-loop val l i (Suc k) A = step-3-c-inner-loop val (l - 1) (i - 1) k
  (add-col-sub-row val i l A)

fun step-3-c :: 'a ⇒ nat ⇒ nat ⇒ (nat × nat)list ⇒ 'a mat ⇒ 'a mat where
  step-3-c x l k [] A = A
  | step-3-c x l k ((i-begin,i-end) # blocks) A = (
  let
  B = (if i-end = l then A else

```

*step-3-c-inner-loop* ( $A \ \$\$ (i\text{-end},k) / x$ )  $l$   $i\text{-end}$  ( $\text{Suc } i\text{-end} - i\text{-begin}$ )  $A$ )  
in *step-3-c*  $x$   $l$   $k$  blocks  $B$ )

**function** *step-3-main* ::  $\text{nat} \Rightarrow 'a \text{ mat} \Rightarrow 'a \text{ mat}$  **where**

*step-3-main*  $k$   $A = (\text{if } k \geq n \text{ then } A$

else let

$B = \text{step-3-a } (k-1) k A$ ; — 3-a

$\text{all-blocks} = \text{identify-blocks } B k$ ;

$\text{blocks} = \text{filter } (\lambda \text{ block. } B \ \$\$ (\text{snd } \text{block},k) \neq 0) \ \text{all-blocks}$ ;

$F = (\text{if } \text{blocks} = [] \text{ — column } k \text{ has only } 0\text{s}$

then  $B$

else let

$(l\text{-start},l) = \text{find-largest-block } (\text{hd } \text{blocks}) (\text{tl } \text{blocks})$ ; — 3-b

$x = B \ \$\$ (l,k)$ ;

$C = \text{step-3-c } x l k \text{ blocks } B$ ; — 3-c

$D = \text{mult-col-div-row } (\text{inverse } x) k C$ ; — 3-d

$E = \text{swap-cols-rows-block } (\text{Suc } l) k D$  — 3-e

in  $E$ )

in *step-3-main* ( $\text{Suc } k$ )  $F$ )

$\langle \text{proof} \rangle$

**termination**  $\langle \text{proof} \rangle$

**end**

**definition** *step-1* ::  $'a :: \text{field mat} \Rightarrow 'a \text{ mat}$  **where**

*step-1*  $A = \text{step-1-main } (\text{dim-row } A) 0 0 A$

**definition** *step-2* ::  $'a :: \text{field mat} \Rightarrow 'a \text{ mat}$  **where**

*step-2*  $A = \text{step-2-main } (\text{dim-row } A) 0 A$

**definition** *step-3* ::  $'a :: \text{field mat} \Rightarrow 'a \text{ mat}$  **where**

*step-3*  $A = \text{step-3-main } (\text{dim-row } A) 1 A$

**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* ::  $\text{nat} \Rightarrow 'a :: \text{one mat} \Rightarrow (\text{nat} \times 'a) \text{ list}$  **where**

*jnf-vector-main* 0  $A = []$

| *jnf-vector-main* ( $\text{Suc } i\text{-end}$ )  $A = (\text{let}$

$i\text{-start} = \text{identify-block } A i\text{-end}$

in *jnf-vector-main*  $i\text{-start}$   $A @ [(\text{Suc } i\text{-end} - i\text{-start}, A \ \$\$ (i\text{-start},i\text{-start}))]$ )

$\langle \text{proof} \rangle$

**definition** *jnf-vector* ::  $'a :: \text{one mat} \Rightarrow (\text{nat} \times 'a) \text{ list}$  **where**

*jnf-vector*  $A = \text{jnf-vector-main } (\text{dim-row } A) A$

**definition** *triangular-to-jnf-vector* ::  $'a :: \text{field mat} \Rightarrow (\text{nat} \times 'a) \text{ list}$  **where**



*triangular-to-jnf-vector*  $A \equiv \text{let } B = \text{step-2 } (\text{step-1 } A)$   
*in concat (map (jnf-vector o step-3) (partition-ev-blocks B []))*

### 18.3 Preservation of Dimensions

**lemma** *swap-cols-rows-block-dims-main*:

$\text{dim-row } (\text{swap-cols-rows-block } i \ j \ A) = \text{dim-row } A \wedge \text{dim-col } (\text{swap-cols-rows-block } i \ j \ A) = \text{dim-col } A$   
 ⟨proof⟩

**lemma** *swap-cols-rows-block-dims[simp]*:

$\text{dim-row } (\text{swap-cols-rows-block } i \ j \ A) = \text{dim-row } A$   
 $\text{dim-col } (\text{swap-cols-rows-block } i \ j \ A) = \text{dim-col } A$   
 $A \in \text{carrier-mat } n \ n \implies \text{swap-cols-rows-block } i \ j \ A \in \text{carrier-mat } n \ n$   
 ⟨proof⟩

**lemma** *step-1-main-dims-main*:

$\text{dim-row } (\text{step-1-main } n \ i \ j \ A) = \text{dim-row } A \wedge \text{dim-col } (\text{step-1-main } n \ i \ j \ A) = \text{dim-col } A$   
 ⟨proof⟩

**lemma** *step-1-main-dims[simp]*:

$\text{dim-row } (\text{step-1-main } n \ i \ j \ A) = \text{dim-row } A$   
 $\text{dim-col } (\text{step-1-main } n \ i \ j \ A) = \text{dim-col } A$   
 ⟨proof⟩

**lemma** *step-2-main-dims-main*:

$\text{dim-row } (\text{step-2-main } n \ j \ A) = \text{dim-row } A \wedge \text{dim-col } (\text{step-2-main } n \ j \ A) = \text{dim-col } A$   
 ⟨proof⟩

**lemma** *step-2-main-dims[simp]*:

$\text{dim-row } (\text{step-2-main } n \ j \ A) = \text{dim-row } A$   
 $\text{dim-col } (\text{step-2-main } n \ j \ A) = \text{dim-col } A$   
 ⟨proof⟩

**lemma** *step-3-a-dims-main*:

$\text{dim-row } (\text{step-3-a } i \ j \ A) = \text{dim-row } A \wedge \text{dim-col } (\text{step-3-a } i \ j \ A) = \text{dim-col } A$   
 ⟨proof⟩

**lemma** *step-3-a-dims[simp]*:

$\text{dim-row } (\text{step-3-a } i \ j \ A) = \text{dim-row } A$   
 $\text{dim-col } (\text{step-3-a } i \ j \ A) = \text{dim-col } A$   
 ⟨proof⟩

**lemma** *step-3-c-inner-loop-dims-main*:

$\text{dim-row } (\text{step-3-c-inner-loop } \text{val } l \ i \ j \ A) = \text{dim-row } A \wedge \text{dim-col } (\text{step-3-c-inner-loop } \text{val } l \ i \ j \ A) = \text{dim-col } A$   
 ⟨proof⟩

**lemma** *step-3-c-inner-loop-dims*[simp]:

$$\begin{aligned} \dim\text{-row } (\text{step-3-c-inner-loop val } l \ i \ j \ A) &= \dim\text{-row } A \\ \dim\text{-col } (\text{step-3-c-inner-loop val } l \ i \ j \ A) &= \dim\text{-col } A \\ \langle \text{proof} \rangle \end{aligned}$$

**lemma** *step-3-c-dims-main*:

$$\begin{aligned} \dim\text{-row } (\text{step-3-c } x \ l \ k \ i \ A) &= \dim\text{-row } A \wedge \dim\text{-col } (\text{step-3-c } x \ l \ k \ i \ A) = \dim\text{-col } \\ A \\ \langle \text{proof} \rangle \end{aligned}$$

**lemma** *step-3-c-dims*[simp]:

$$\begin{aligned} \dim\text{-row } (\text{step-3-c } x \ l \ k \ i \ A) &= \dim\text{-row } A \\ \dim\text{-col } (\text{step-3-c } x \ l \ k \ i \ A) &= \dim\text{-col } A \\ \langle \text{proof} \rangle \end{aligned}$$

**lemma** *step-3-main-dims-main*:

$$\begin{aligned} \dim\text{-row } (\text{step-3-main } n \ k \ A) &= \dim\text{-row } A \wedge \dim\text{-col } (\text{step-3-main } n \ k \ A) = \\ \dim\text{-col } A \\ \langle \text{proof} \rangle \end{aligned}$$

**lemma** *step-3-main-dims*[simp]:

$$\begin{aligned} \dim\text{-row } (\text{step-3-main } n \ j \ A) &= \dim\text{-row } A \\ \dim\text{-col } (\text{step-3-main } n \ j \ A) &= \dim\text{-col } A \\ \langle \text{proof} \rangle \end{aligned}$$

**lemma** *triangular-to-jnf-steps-dims*[simp]:

$$\begin{aligned} \dim\text{-row } (\text{step-1 } A) &= \dim\text{-row } A \\ \dim\text{-col } (\text{step-1 } A) &= \dim\text{-col } A \\ \dim\text{-row } (\text{step-2 } A) &= \dim\text{-row } A \\ \dim\text{-col } (\text{step-2 } A) &= \dim\text{-col } A \\ \dim\text{-row } (\text{step-3 } A) &= \dim\text{-row } A \\ \dim\text{-col } (\text{step-3 } A) &= \dim\text{-col } A \\ \langle \text{proof} \rangle \end{aligned}$$

## 18.4 Properties of Auxiliary Algorithms

**lemma** *lookup-ev-Some*:

$$\begin{aligned} \text{lookup-ev ev } j \ A = \text{Some } i &\implies \\ i < j \wedge A \ \$\$ (i,i) = \text{ev} \wedge (\forall k. i < k \implies k < j \implies A \ \$\$ (k,k) \neq \text{ev}) \\ \langle \text{proof} \rangle \end{aligned}$$

**lemma** *lookup-ev-None*:  $\text{lookup-ev ev } j \ A = \text{None} \implies i < j \implies A \ \$\$ (i,i) \neq \text{ev}$

$\langle \text{proof} \rangle$

**lemma** *swap-cols-rows-block-index*[simp]:

$$\begin{aligned} i < \dim\text{-row } A &\implies i < \dim\text{-col } A \implies j < \dim\text{-row } A \implies j < \dim\text{-col } A \\ \implies \text{low} \leq \text{high} &\implies \text{high} < \dim\text{-row } A \implies \text{high} < \dim\text{-col } A \\ \implies \text{swap-cols-rows-block } \text{low } \text{high } A \ \$\$ (i,j) &= A \ \$\$ \end{aligned}$$

(if  $i = \text{low}$  then  $\text{high}$  else if  $i > \text{low} \wedge i \leq \text{high}$  then  $i - 1$  else  $i$ ,  
 if  $j = \text{low}$  then  $\text{high}$  else if  $j > \text{low} \wedge j \leq \text{high}$  then  $j - 1$  else  $j$ )  
 <proof>

**lemma** *find-largest-block-main*: **assumes** *find-largest-block block blocks = (m-b, m-e)*  
**shows**  $(m-b, m-e) \in \text{insert block (set blocks)}$   
 $\wedge (\forall b \in \text{insert block (set blocks)}. m-e - m-b \geq \text{snd } b - \text{fst } b)$   
 <proof>

**lemma** *find-largest-block*: **assumes**  $bl: \text{blocks} \neq []$   
**and** *find*: *find-largest-block (hd blocks) (tl blocks) = (m-begin, m-end)*  
**shows**  $(m\text{-begin}, m\text{-end}) \in \text{set blocks}$   
 $\wedge i\text{-begin } i\text{-end}. (i\text{-begin}, i\text{-end}) \in \text{set blocks} \implies m\text{-end} - m\text{-begin} \geq i\text{-end} - i\text{-begin}$   
 <proof>

**context**  
**fixes**  $ev :: 'a :: \text{one}$   
**and**  $A :: 'a \text{ mat}$   
**begin**

**lemma** *identify-block-main*: **assumes** *identify-block A j = i*  
**shows**  $i \leq j \wedge (i = 0 \vee A \text{ $$ } (i - 1, i) \neq 1) \wedge (\forall k. i \leq k \longrightarrow k < j \longrightarrow A \text{ $$ } (k, \text{Suc } k) = 1)$   
 (is ?P j)  
 <proof>

**lemma** *identify-block-le*: *identify-block A i ≤ i*  
 <proof>  
**end**

**lemma** *identify-block*: **assumes** *identify-block A j = i*  
**shows**  $i \leq j$   
 $i = 0 \vee A \text{ $$ } (i - 1, i) \neq 1$   
 $i \leq k \implies k < j \implies A \text{ $$ } (k, \text{Suc } k) = 1$   
 <proof>

**lemmas** *identify-block-le' = identify-block(1)*

**lemma** *identify-block-le-rev*:  $j = \text{identify-block } A \ i \implies j \leq i$   
 <proof>

**termination** *identify-blocks-main* <proof>

**termination** *jnf-vector-main* <proof>

**lemma** *identify-blocks-main*: **assumes**  $(i\text{-start}, i\text{-end}) \in \text{set } (\text{identify-blocks-main } A \text{ } i \text{ list})$

**and**  $\bigwedge i\text{-s } i\text{-e}. (i\text{-s}, i\text{-e}) \in \text{set list} \implies i\text{-s} \leq i\text{-e} \wedge i\text{-e} < k$

**and**  $i \leq k$

**shows**  $i\text{-start} \leq i\text{-end} \wedge i\text{-end} < k$  *<proof>*

**lemma** *identify-blocks*: **assumes**  $(i\text{-start}, i\text{-end}) \in \text{set } (\text{identify-blocks } B \text{ } k)$

**shows**  $i\text{-start} \leq i\text{-end} \wedge i\text{-end} < k$

*<proof>*

## 18.5 Proving Similarity

**context**

**begin**

**private lemma** *swap-cols-rows-block-similar*: **assumes**  $A \in \text{carrier-mat } n \text{ } n$

**and**  $j < n$  **and**  $i \leq j$

**shows** *similar-mat* (*swap-cols-rows-block*  $i \text{ } j \text{ } A$ )  $A$

*<proof>* **lemma** *step-1-main-similar*:  $i \leq j \implies A \in \text{carrier-mat } n \text{ } n \implies \text{similar-mat } (\text{step-1-main } n \text{ } i \text{ } j \text{ } A) \text{ } A$

*<proof>* **lemma** *step-2-main-similar*:  $A \in \text{carrier-mat } n \text{ } n \implies \text{similar-mat } (\text{step-2-main } n \text{ } j \text{ } A) \text{ } A$

*<proof>* **lemma** *step-3-a-similar*:  $A \in \text{carrier-mat } n \text{ } n \implies i < j \implies j < n \implies \text{similar-mat } (\text{step-3-a } i \text{ } j \text{ } A) \text{ } A$

*<proof>* **lemma** *step-3-c-inner-loop-similar*:

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

*similar-mat* (*step-3-c-inner-loop*  $\text{val } l \text{ } i \text{ } k \text{ } A$ )  $A$

*<proof>* **lemma** *step-3-c-similar*:

$A \in \text{carrier-mat } n \text{ } n \implies l < k \implies k < n$

$\implies (\bigwedge i\text{-begin } i\text{-end}. (i\text{-begin}, i\text{-end}) \in \text{set blocks} \implies i\text{-end} \leq k \wedge i\text{-end} - i\text{-begin} \leq l)$

$\implies \text{similar-mat } (\text{step-3-c } x \text{ } l \text{ } k \text{ blocks } A) \text{ } A$

*<proof>* **lemma** *step-3-main-similar*:  $A \in \text{carrier-mat } n \text{ } n \implies k > 0 \implies \text{similar-mat } (\text{step-3-main } n \text{ } k \text{ } A) \text{ } A$

*<proof>*

**lemma** *step-1-similar*:  $A \in \text{carrier-mat } n \text{ } n \implies \text{similar-mat } (\text{step-1 } A) \text{ } A$

*<proof>*

**lemma** *step-2-similar*:  $A \in \text{carrier-mat } n \text{ } n \implies \text{similar-mat } (\text{step-2 } A) \text{ } A$

*<proof>*

**lemma** *step-3-similar*:  $A \in \text{carrier-mat } n \text{ } n \implies \text{similar-mat } (\text{step-3 } A) \text{ } A$

*<proof>*

**end**

## 18.6 Invariants for Proving that Result is in JNF

**context**

**fixes**  $n :: \text{nat}$   
**and**  $ty :: 'a :: \text{field itself}$   
**begin**

**definition**  $\text{uppert} :: 'a \text{ mat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}$  **where**  
 $\text{uppert } A \ i \ j \equiv j < i \longrightarrow A \ \$\$ \ (i,j) = 0$

**definition**  $\text{diff-ev} :: 'a \text{ mat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}$  **where**  
 $\text{diff-ev } A \ i \ j \equiv i < j \longrightarrow A \ \$\$ \ (i,i) \neq A \ \$\$ \ (j,j) \longrightarrow A \ \$\$ \ (i,j) = 0$

**definition**  $\text{ev-blocks-part} :: \text{nat} \Rightarrow 'a \text{ mat} \Rightarrow \text{bool}$  **where**  
 $\text{ev-blocks-part } m \ A \equiv \forall \ i \ j \ k. \ i < j \longrightarrow j < k \longrightarrow k < m \longrightarrow A \ \$\$ \ (k,k) = A \ \$\$ \ (i,i) \longrightarrow A \ \$\$ \ (j,j) = A \ \$\$ \ (i,i)$

**definition**  $\text{ev-blocks} :: 'a \text{ mat} \Rightarrow \text{bool}$  **where**  
 $\text{ev-blocks} \equiv \text{ev-blocks-part } n$

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  $\text{local.uppert}$ ,  $\text{local.diff-ev}$ , and  $\text{local.ev-blocks}$ , and the column at  $k$  will have one of the following properties, which are ensured in the different phases of step 3.

**private definition**  $\text{one-zero} :: 'a \text{ mat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}$  **where**  
 $\text{one-zero } A \ i \ j \equiv$   
 $(\text{Suc } i < j \longrightarrow A \ \$\$ \ (i, \text{Suc } i) = 1 \longrightarrow A \ \$\$ \ (i,j) = 0) \wedge$   
 $(j < i \longrightarrow A \ \$\$ \ (i,j) = 0) \wedge$   
 $(i < j \longrightarrow A \ \$\$ \ (i,i) \neq A \ \$\$ \ (j,j) \longrightarrow A \ \$\$ \ (i,j) = 0)$

**private definition**  $\text{single-non-zero} :: \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \Rightarrow 'a \text{ mat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}$  **where**  
 $\text{single-non-zero} \equiv \lambda \ l \ k \ x. (\lambda \ A \ i \ j. (i \notin \{k,l\} \longrightarrow A \ \$\$ \ (i,k) = 0) \wedge A \ \$\$ \ (l,k) = x)$

**private definition**  $\text{single-one} :: \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ mat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}$  **where**  
 $\text{single-one} \equiv \lambda \ l \ k. (\lambda \ A \ i \ j. (i \notin \{k,l\} \longrightarrow A \ \$\$ \ (i,k) = 0) \wedge A \ \$\$ \ (l,k) = 1)$

**private definition**  $\text{lower-one} :: \text{nat} \Rightarrow 'a \text{ mat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}$  **where**  
 $\text{lower-one } k \ A \ i \ j = (j = k \longrightarrow$   
 $(A \ \$\$ \ (i,j) = 0 \vee i = j \vee (A \ \$\$ \ (i,j) = 1 \wedge j = \text{Suc } i \wedge A \ \$\$ \ (i,i) = A \ \$\$ \ (j,j))))$

**definition**  $\text{jb} :: 'a \text{ mat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}$  **where**

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

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 3b are the same as one would identify for the matrices in the upcoming steps 3c and 3d.

**definition** *same-diag* :: 'a mat  $\Rightarrow$  'a mat  $\Rightarrow$  bool **where**  
*same-diag* A B  $\equiv \forall\ i < n. A\ \$\$ (i,i) = B\ \$\$ (i,i)$

**private definition** *same-upto* :: nat  $\Rightarrow$  'a mat  $\Rightarrow$  'a mat  $\Rightarrow$  bool **where**  
*same-upto* j A B  $\equiv \forall\ i' j'. i' < n \longrightarrow j' < j \longrightarrow A\ \$\$ (i',j') = B\ \$\$ (i',j')$

Definitions stating where the properties hold

**definition** *inv-all* :: ('a mat  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  bool)  $\Rightarrow$  '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)  $\Rightarrow$  '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* :: ('a mat  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  bool)  $\Rightarrow$  '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* :: ('a mat  $\Rightarrow$  nat  $\Rightarrow$  bool)  $\Rightarrow$  'a mat  $\Rightarrow$  nat  $\Rightarrow$  bool **where**  
*inv-from-bot* p A mi  $\equiv \forall\ i. i \geq mi \longrightarrow i < n \longrightarrow p\ A\ i$

Auxiliary Lemmas on Handling, Comparing, and Accessing Invariants

**lemma** *jb-imp-uppert*:  $jb\ A\ i\ j \Longrightarrow uppert\ A\ i\ j$

*<proof>* **lemma** *ev-blocks-partD*:

*ev-blocks-part* m A  $\Longrightarrow i < j \Longrightarrow j < k \Longrightarrow k < m \Longrightarrow A\ \$\$ (k,k) = A\ \$\$ (i,i) \Longrightarrow A\ \$\$ (j,j) = A\ \$\$ (i,i)$

*<proof>* **lemma** *ev-blocks-part-leD*:

**assumes** *ev-blocks-part* m A

$i \leq j \leq k \ k < m \ A\ \$\$ (k,k) = A\ \$\$ (i,i)$

**shows**  $A\ \$\$ (j,j) = A\ \$\$ (i,i)$

*<proof>* **lemma** *ev-blocks-partI*:

**assumes**  $\bigwedge i j k. i < j \implies j < k \implies k < m \implies A \text{ \textasciitilde\textasciitilde } (k,k) = A \text{ \textasciitilde\textasciitilde } (i,i) \implies A \text{ \textasciitilde\textasciitilde } (j,j) = A \text{ \textasciitilde\textasciitilde } (i,i)$   
**shows** *ev-blocks-part m A*  
 (proof) **lemma** *ev-blocksD*:  
*ev-blocks A  $\implies i < j \implies j < k \implies k < n \implies A \text{ \textasciitilde\textasciitilde } (k,k) = A \text{ \textasciitilde\textasciitilde } (i,i) \implies A \text{ \textasciitilde\textasciitilde } (j,j) = A \text{ \textasciitilde\textasciitilde } (i,i)$*   
 (proof) **lemma** *ev-blocks-leD*:  
*ev-blocks A  $\implies i \leq j \implies j \leq k \implies k < n \implies A \text{ \textasciitilde\textasciitilde } (k,k) = A \text{ \textasciitilde\textasciitilde } (i,i) \implies A \text{ \textasciitilde\textasciitilde } (j,j) = A \text{ \textasciitilde\textasciitilde } (i,i)$*   
 (proof)

**lemma** *inv-allD*: *inv-all p A  $\implies i < n \implies j < n \implies p A i j$*   
 (proof) **lemma** *inv-allI*: **assumes**  $\bigwedge i j. i < n \implies j < n \implies p A i j$   
**shows** *inv-all p A*  
 (proof) **lemma** *inv-partI*: **assumes**  $\bigwedge i j. i < n \implies j < n \implies j < m-j \vee j = m-j \wedge i \geq m-i \implies p A i j$   
**shows** *inv-part p A m-i m-j*  
 (proof) **lemma** *inv-partD*: **assumes** *inv-part p A m-i m-j i < n j < n*  
**shows**  $j < m-j \implies p A i j$   
**and**  $j = m-j \implies i \geq m-i \implies p A i j$   
**and**  $j < m-j \vee j = m-j \wedge i \geq m-i \implies p A i j$   
 (proof) **lemma** *inv-uptoI*: **assumes**  $\bigwedge i j. i < n \implies j < n \implies j < m \implies p A i j$   
**shows** *inv-upto p A m*  
 (proof) **lemma** *inv-uptoD*: **assumes** *inv-upto p A m i < n j < n j < m*  
**shows**  $p A i j$   
 (proof) **lemma** *inv-upto-Suc*: **assumes** *inv-upto p A m*  
**and**  $\bigwedge i. i < n \implies p A i m$   
**shows** *inv-upto p A (Suc m)*  
 (proof) **lemma** *inv-upto-mono*: **assumes**  $\bigwedge i j. i < n \implies j < k \implies p A i j \implies q A i j$   
**shows** *inv-upto p A k  $\implies$  inv-upto q A k*  
 (proof) **lemma** *inv-fromI*: **assumes**  $\bigwedge i j. i < n \implies j < n \implies j > m \implies p A i j$   
**shows** *inv-from p A m*  
 (proof) **lemma** *inv-fromD*: **assumes** *inv-from p A m i < n j < n j > m*  
**shows**  $p A i j$   
 (proof) **lemma** *inv-atI[intro]*: **assumes**  $\bigwedge i. i < n \implies p A i m$   
**shows** *inv-at p A m*  
 (proof) **lemma** *inv-atD*: **assumes** *inv-at p A m i < n*  
**shows**  $p A i m$   
 (proof) **lemma** *inv-all-imp-inv-part*:  $m i \leq n \implies m-j \leq n \implies \text{inv-all } p A \implies \text{inv-part } p A m-i m-j$   
 (proof) **lemma** *inv-all-eq-inv-part*: *inv-all p A = inv-part p A n n*  
 (proof) **lemma** *inv-part-0-Suc*:  $m-j < n \implies \text{inv-part } p A 0 m-j = \text{inv-part } p A n (\text{Suc } m-j)$   
 (proof) **lemma** *inv-all-uppertD*: *inv-all uppert A  $\implies j < i \implies i < n \implies A \text{ \textasciitilde\textasciitilde } (i,j) = 0$*   
 (proof) **lemma** *inv-all-diff-evD*: *inv-all diff-ev A  $\implies i < j \implies j < n$*

$\implies A \text{ \&\& } (i, i) \neq A \text{ \&\& } (j, j) \implies A \text{ \&\& } (i, j) = 0$   
 ⟨proof⟩ **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**  $A \text{ \&\& } (i, j) = 0$   
 ⟨proof⟩ **lemma** *inv-from-bot-step*:  $p \ A \ i \implies \text{inv-from-bot } p \ A \ (\text{Suc } i) \implies \text{inv-from-bot } p \ A \ i$   
 ⟨proof⟩ **lemma** *same-diag-refl[simp]*:  $\text{same-diag } A \ A$  ⟨proof⟩ **lemma** *same-diag-trans*:  
 $\text{same-diag } A \ B \implies \text{same-diag } B \ C \implies \text{same-diag } A \ C$   
 ⟨proof⟩ **lemma** *same-diag-ev-blocks*:  $\text{same-diag } A \ B \implies \text{ev-blocks } A \implies \text{ev-blocks } B$   
 ⟨proof⟩ **lemma** *same-uptoI[intro]*: **assumes**  $\bigwedge i' j'. i' < n \implies j' < j \implies A \text{ \&\& } (i', j') = B \text{ \&\& } (i', j')$   
**shows**  $\text{same-upto } j \ A \ B$   
 ⟨proof⟩ **lemma** *same-uptoD[dest]*: **assumes**  $\text{same-upto } j \ A \ B \ i' < n \ j' < j$   
**shows**  $A \text{ \&\& } (i', j') = B \text{ \&\& } (i', j')$   
 ⟨proof⟩ **lemma** *same-upto-refl[simp]*:  $\text{same-upto } j \ A \ A$  ⟨proof⟩ **lemma** *same-upto-trans*:  
 $\text{same-upto } j \ A \ B \implies \text{same-upto } j \ B \ C \implies \text{same-upto } j \ A \ C$   
 ⟨proof⟩ **lemma** *same-upto-inv-upto-jb*:  $\text{same-upto } j \ A \ B \implies \text{inv-upto } j \ A \ j \implies \text{inv-upto } j \ B \ j$   
 ⟨proof⟩

**lemma** *jb-imp-diff-ev*:  $\text{jb } A \ i \ j \implies \text{diff-ev } A \ i \ j$   
 ⟨proof⟩ **lemma** *ev-blocks-diag*:  
 $\text{same-diag } A \ B \implies \text{ev-blocks } B \implies \text{ev-blocks } A$   
 ⟨proof⟩ **lemma** *inv-all-imp-inv-from*:  $\text{inv-all } p \ A \implies \text{inv-from } p \ A \ k$   
 ⟨proof⟩ **lemma** *inv-all-imp-inv-at*:  $\text{inv-all } p \ A \implies k < n \implies \text{inv-at } p \ A \ k$   
 ⟨proof⟩ **lemma** *inv-from-upto-at-all*:  
**assumes**  $\text{inv-upto } j \ B \ A \ k \ \text{inv-from } \text{diff-ev } A \ k \ \text{inv-from } \text{uppert } A \ k \ \text{inv-at } p \ A \ k$   
**and**  $\bigwedge i. i < n \implies p \ A \ i \ k \implies \text{diff-ev } A \ i \ k \wedge \text{uppert } A \ i \ k$   
**shows**  $\text{inv-all } \text{diff-ev } A \ \text{inv-all } \text{uppert } A$   
 ⟨proof⟩ **lemma** *lower-one-diff-uppert*:  
 $i < n \implies \text{lower-one } k \ B \ i \ k \implies \text{diff-ev } B \ i \ k \wedge \text{uppert } B \ i \ k$   
 ⟨proof⟩

**definition** *ev-block* ::  $\text{nat} \Rightarrow 'a \ \text{mat} \Rightarrow \text{bool}$  **where**  
 $\bigwedge n. \text{ev-block } n \ A = (\forall i \ j. i < n \longrightarrow j < n \longrightarrow A \text{ \&\& } (i, i) = A \text{ \&\& } (j, j))$

**lemma** *ev-blockD*:  $\bigwedge n. \text{ev-block } n \ A \implies i < n \implies j < n \implies A \text{ \&\& } (i, i) = A \text{ \&\& } (j, j)$   
 ⟨proof⟩

**lemma** *same-diag-ev-block*:  $\text{same-diag } A \ B \implies \text{ev-block } n \ A \implies \text{ev-block } n \ B$   
 ⟨proof⟩



## 18.7 Alternative Characterization of *identify-blocks* in Presence of *local.ev-block*

**private lemma** *identify-blocks-main-iff*: **assumes** \*:  $k \leq k'$   
 $k' \neq k \longrightarrow k > 0 \longrightarrow A \ \$\$ (k - 1, k) \neq 1$  **and**  $k' < n$   
**shows**  $set (identify-blocks-main A k list) =$   
 $set list \cup \{(i,j) \mid i j. i \leq j \wedge j < k \wedge (\forall l. i \leq l \longrightarrow l < j \longrightarrow A \ \$\$ (l, Suc l) = 1)$   
 $\wedge (Suc j \neq k' \longrightarrow A \ \$\$ (j, Suc j) \neq 1) \wedge (i > 0 \longrightarrow A \ \$\$ (i - 1, i) \neq 1)\}$  (**is**  
 $- = - \cup ?ss A k$ )  
 $\langle proof \rangle$  **lemma** *identify-blocks-iff*: **assumes**  $k < n$   
**shows**  $set (identify-blocks A k) =$   
 $\{(i,j) \mid i j. i \leq j \wedge j < k \wedge (\forall l. i \leq l \longrightarrow l < j \longrightarrow A \ \$\$ (l, Suc l) = 1)$   
 $\wedge (Suc j \neq k \longrightarrow A \ \$\$ (j, Suc j) \neq 1) \wedge (i > 0 \longrightarrow A \ \$\$ (i - 1, i) \neq 1)\}$   
 $\langle proof \rangle$  **lemma** *identify-blocksD*: **assumes**  $k < n$  **and**  $(i,j) \in set (identify-blocks A k)$   
**shows**  $i \leq j j < k$   
 $\bigwedge l. i \leq l \Longrightarrow l < j \Longrightarrow A \ \$\$ (l, Suc l) = 1$   
 $Suc j \neq k \Longrightarrow A \ \$\$ (j, Suc j) \neq 1$   
 $i > 0 \Longrightarrow A \ \$\$ (i - 1, i - 1) \neq A \ \$\$ (k,k) \vee A \ \$\$ (i - 1, i) \neq 1$   
 $\langle proof \rangle$  **lemma** *identify-blocksI*: **assumes** *inv*:  $k < n$   
 $i \leq j j < k \bigwedge l. i \leq l \Longrightarrow l < j \Longrightarrow A \ \$\$ (l, Suc l) = 1$   
 $Suc j \neq k \Longrightarrow A \ \$\$ (j, Suc j) \neq 1 i > 0 \Longrightarrow A \ \$\$ (i - 1, i) \neq 1$   
**shows**  $(i,j) \in set (identify-blocks A k)$   
 $\langle proof \rangle$  **lemma** *identify-blocks-rev*: **assumes**  $A \ \$\$ (i, Suc i) = 0 \wedge Suc i < k \vee$   
 $Suc i = k$   
**and** *inv*:  $k < n$   
**shows**  $(identify-block A i, i) \in set (identify-blocks A k)$   
 $\langle proof \rangle$

## 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** *ijk*:  $i < j j < n k < n$   
**shows**  $add-col-sub-row a i j A \ \$\$ (k,k) = A \ \$\$ (k,k)$   
 $\langle proof \rangle$  **lemma** *add-col-sub-row-diff-ev-part-old*: **assumes**  $A: A \in carrier-mat n n$   
**and** *ij*:  $i \leq j i \neq 0 i < n j < n i' < n j' < n$   
**and** *choice*:  $j' < j \vee j' = j \wedge i' \geq i$   
**and** *old*: *inv-part diff-ev A i j*  
**and** *ut*: *inv-all uppert A*  
**shows**  $diff-ev (add-col-sub-row a (i - 1) j A) i' j'$   
 $\langle proof \rangle$  **lemma** *add-col-sub-row-uppert*: **assumes**  $A \in carrier-mat n n$   
**and**  $i < j$   
**and**  $j < n$   
**and** *inv*: *inv-all uppert (A :: 'a mat)*  
**shows** *inv-all uppert (add-col-sub-row a i j A)*  
 $\langle proof \rangle$  **lemma** *step-1-main-inv*:  $i \leq j$   
 $\Longrightarrow A \in carrier-mat n n$   
 $\Longrightarrow inv-all uppert A$

$\implies \text{inv-part diff-ev } A \ i \ j$   
 $\implies \text{inv-all uppert } (\text{step-1-main } n \ i \ j \ A) \wedge \text{inv-all diff-ev } (\text{step-1-main } n \ i \ j \ A)$   
 ⟨proof⟩ **lemma** *step-2-main-inv*:  $A \in \text{carrier-mat } n \ n$   
 $\implies \text{inv-all uppert } A$   
 $\implies \text{inv-all diff-ev } A$   
 $\implies \text{ev-blocks-part } j \ A$   
 $\implies \text{inv-all uppert } (\text{step-2-main } n \ j \ A) \wedge \text{inv-all diff-ev } (\text{step-2-main } n \ j \ A)$   
 $\wedge \text{ev-blocks } (\text{step-2-main } n \ j \ A)$   
 ⟨proof⟩ **lemma** *add-col-sub-row-same-upto*: **assumes**  $i < j \ j < n \ A \in \text{carrier-mat } n \ n$   
 $\text{inv-upto uppert } A \ j$   
**shows** *same-upto*  $j \ A \ (\text{add-col-sub-row } v \ i \ j \ A) \ j$   
 ⟨proof⟩ **lemma** *add-col-sub-row-inv-from-uppert*: **assumes**  $*$ :  $\text{inv-from uppert } A$   
 $j$   
**and**  $*$ :  $A \in \text{carrier-mat } n \ n \ i < n \ i < j \ j < n$   
**shows** *inv-from uppert*  $(\text{add-col-sub-row } v \ i \ j \ A) \ j$   
 ⟨proof⟩ **lemma** *step-3-a-inv*:  $A \in \text{carrier-mat } n \ n$   
 $\implies i < j \implies j < n$   
 $\implies \text{inv-upto } j \ b \ A \ j$   
 $\implies \text{inv-from uppert } A \ j$   
 $\implies \text{inv-from-bot } (\lambda \ A \ i. \text{one-zero } A \ i \ j) \ A \ i$   
 $\implies \text{ev-block } n \ A$   
 $\implies \text{inv-from uppert } (\text{step-3-a } i \ j \ A) \ j$   
 $\wedge \text{inv-upto } j \ b \ (\text{step-3-a } i \ j \ A) \ j$   
 $\wedge \text{inv-at one-zero } (\text{step-3-a } i \ j \ A) \ j \wedge \text{same-diag } A \ (\text{step-3-a } i \ j \ A)$   
 ⟨proof⟩ **lemma** *identify-block-cong*: **assumes**  $su$ : *same-upto*  $k \ A \ B$  **and**  $kn$ :  $k < n$   
**shows**  $i < k \implies \text{identify-block } A \ i = \text{identify-block } B \ i$   
 ⟨proof⟩ **lemma** *identify-blocks-main-cong*:  
 $k < n \implies \text{same-upto } k \ A \ B \implies \text{identify-blocks-main } A \ k \ xs = \text{identify-blocks-main } B \ k \ xs$   
 ⟨proof⟩ **lemma** *identify-blocks-cong*:  
 $k < n \implies \text{same-diag } A \ B \implies \text{same-upto } k \ A \ B \implies \text{identify-blocks } A \ k = \text{identify-blocks } B \ k$   
 ⟨proof⟩ **lemma** *inv-from-upto-at-all-ev-block*:  
**assumes**  $jb$ : *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**  $p$ :  $\bigwedge i. i < n \implies p \ A \ i \ k \implies \text{uppert } A \ i \ k$   
**and**  $k$ :  $k < n$   
**shows** *inv-all uppert*  $A$   
 ⟨proof⟩

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**  $inv$ :  
 $\text{inv-upto } j \ b \ A \ k$   
 $\text{inv-from uppert } A \ k$   
 $\text{inv-at one-zero } A \ k$   
 $\text{ev-block } n \ A$

**and**  $k: k < n$   
**and**  $A: A \in \text{carrier-mat } n \ n$   
**and**  $lb: (lb, l) \in \text{set } (\text{identify-blocks } A \ k)$   
**and**  $ib\text{-block}: (i\text{-begin}, i\text{-end}) \in \text{set } (\text{identify-blocks } A \ k)$   
**and**  $il: i\text{-end} \neq l$   
**and**  $large: l - lb \geq i\text{-end} - i\text{-begin}$   
**and**  $Alk: A \ \$\$ (l, k) \neq 0$   
**shows**  $\text{step-3-c-inner-loop } (A \ \$\$ (i\text{-end}, k) / A \ \$\$ (l, k)) \ l \ i\text{-end} \ (\text{Suc } i\text{-end} - i\text{-begin}) \ A =$   
 $\text{mat } n \ n$   
 $(\lambda(i, j). \text{if } (i, j) = (i\text{-end}, k) \text{ then } 0$   
 $\quad \text{else if } i\text{-begin} \leq i \wedge i \leq i\text{-end} \wedge k < j \text{ then } A \ \$\$ (i, j) - A \ \$\$ (i\text{-end},$   
 $k) / A \ \$\$ (l, k) * A \ \$\$ (l + i - i\text{-end}, j)$   
 $\quad \text{else } A \ \$\$ (i, j)) \ (\text{is } ?L = ?R)$   
 $\langle \text{proof} \rangle$  **lemma**  $\text{step-3-c-inv}: A \in \text{carrier-mat } n \ n$   
 $\implies k < n$   
 $\implies (lb, l) \in \text{set } (\text{identify-blocks } A \ k)$   
 $\implies \text{inv-upto } jb \ A \ k$   
 $\implies \text{inv-from } \text{uppert } A \ k$   
 $\implies \text{inv-at one-zero } A \ k$   
 $\implies \text{ev-block } n \ A$   
 $\implies \text{set } bs \subseteq \text{set } (\text{identify-blocks } A \ k)$   
 $\implies (\bigwedge be. be \notin \text{snd } ' \text{set } bs \implies be \notin \{l, k\} \implies be < n \implies A \ \$\$ (be, k) = 0)$   
 $\implies (\bigwedge bb \ be. (bb, be) \in \text{set } bs \implies be - bb \leq l - lb) \text{ — largest block}$   
 $\implies x = A \ \$\$ (l, k)$   
 $\implies x \neq 0$   
 $\implies \text{inv-all } \text{uppert } (\text{step-3-c } x \ l \ k \ bs \ A)$   
 $\quad \wedge \text{same-diag } A \ (\text{step-3-c } x \ l \ k \ bs \ A)$   
 $\quad \wedge \text{same-upto } k \ A \ (\text{step-3-c } x \ l \ k \ bs \ A)$   
 $\quad \wedge \text{inv-at } (\text{single-non-zero } l \ k \ x) \ (\text{step-3-c } x \ l \ k \ bs \ A) \ k$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{step-3-main-inv}: A \in \text{carrier-mat } n \ n$   
 $\implies k > 0$   
 $\implies \text{inv-all } \text{uppert } A$   
 $\implies \text{ev-block } n \ A$   
 $\implies \text{inv-upto } jb \ A \ k$   
 $\implies \text{inv-all } jb \ (\text{step-3-main } n \ k \ A) \wedge \text{same-diag } A \ (\text{step-3-main } n \ k \ A)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{step-1-2-inv}$ :  
**assumes**  $A: A \in \text{carrier-mat } n \ n$   
**and**  $\text{upper-t}: \text{upper-triangular } A$   
**and**  $\text{Bid}: B = \text{step-2 } (\text{step-1 } A)$   
**shows**  $\text{inv-all } \text{uppert } B \ \text{inv-all } \text{diff-ev } B \ \text{ev-blocks } B$   
 $\langle \text{proof} \rangle$

**definition**  $\text{inv-all}' :: ('a \ \text{mat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow \text{bool}) \Rightarrow 'a \ \text{mat} \Rightarrow \text{bool}$  **where**  
 $\text{inv-all}' \ p \ A \equiv \forall \ i \ j. i < \text{dim-row } A \longrightarrow j < \text{dim-row } A \longrightarrow p \ A \ i \ j$

**private lemma** *lookup-other-ev-None*: **assumes** *lookup-other-ev ev k A = None*  
**and**  $i < k$   
**shows**  $A \text{ \$\$ } (i,i) = \text{ev}$   
 $\langle \text{proof} \rangle$  **lemma** *lookup-other-ev-Some*: **assumes** *lookup-other-ev ev k A = Some*  
 $i$   
**shows**  $i < k \wedge A \text{ \$\$ } (i,i) \neq \text{ev} \wedge (\forall j. i < j \wedge j < k \longrightarrow A \text{ \$\$ } (j,j) = \text{ev})$   
 $\langle \text{proof} \rangle$

**lemma** *partition-jb*: **assumes**  $A: (A :: 'a \text{ mat}) \in \text{carrier-mat } n \ n$   
**and** *inv*: *inv-all uppert A inv-all diff-ev A ev-blocks A*  
**and** *part*: *partition-ev-blocks A [] = bs*  
**shows**  $A = \text{diag-block-mat } bs \ \bigwedge \ B. B \in \text{set } bs \implies \text{inv-all}' \ \text{uppert } B \ \wedge \ \text{ev-block}$   
 $(\text{dim-col } B) \ B \ \wedge \ \text{dim-row } B = \text{dim-col } B$   
 $\langle \text{proof} \rangle$

**lemma** *uppert-to-jb*: **assumes** *ut*: *inv-all uppert A* **and**  $A \in \text{carrier-mat } n \ n$   
**shows** *inv-upto jb A 1*  
 $\langle \text{proof} \rangle$

**lemma** *jnf-vector*: **assumes**  $A: A \in \text{carrier-mat } n \ n$   
**and** *jb*:  $\bigwedge i \ j. i < n \implies j < n \implies \text{jb } A \ i \ j$   
**and** *evb*: *ev-block n A*  
**shows** *jordan-matrix (jnf-vector A) = (A :: 'a mat)*  
 $0 \notin \text{fst ' set (jnf-vector A)}$   
 $\langle \text{proof} \rangle$

**end**

**lemma** *triangular-to-jnf-vector*:  
**assumes**  $A: A \in \text{carrier-mat } n \ n$   
**and** *upper-t*: *upper-triangular A*  
**shows** *jordan-nf A (triangular-to-jnf-vector A)*  
 $\langle \text{proof} \rangle$

**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 = ( $\prod$  a  $\leftarrow$  es. [:- a, 1:])  
**shows** *jordan-nf* A (*jordan-nf-via-factored-charpoly* A es)  
*<proof>*

**lemma** *jordan-nf-exists*: **assumes** A: A  $\in$  carrier-mat n n  
**and** *linear*: char-poly A = ( $\prod$  (a :: 'a :: conjugatable-ordered-field)  $\leftarrow$  as. [:- a, 1:])  
**shows**  $\exists$  n-as. *jordan-nf* A n-as  
*<proof>*

**lemma** *jordan-nf-iff-linear-factorization*: **fixes** A :: 'a :: conjugatable-ordered-field mat  
**assumes** A: A  $\in$  carrier-mat n n  
**shows** ( $\exists$  n-as. *jordan-nf* A n-as) = ( $\exists$  as. char-poly A = ( $\prod$  a  $\leftarrow$  as. [:- a, 1:]))  
(is ?l = ?r)  
*<proof>*

**lemma** *similar-iff-same-jordan-nf*: **fixes** A :: complex mat  
**assumes** A: A  $\in$  carrier-mat n n **and** B: B  $\in$  carrier-mat n n  
**shows** *similar-mat* A B = (*jordan-nf* A = *jordan-nf* B)  
*<proof>*

**lemma** *order-char-poly-smult*: **fixes** A :: complex mat  
**assumes** A: A  $\in$  carrier-mat n n  
**and** k: k  $\neq$  0  
**shows** order x (char-poly (k  $\cdot_m$  A)) = order (x / k) (char-poly A)  
*<proof>*

## 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 = ( $\prod$  (a :: 'a :: {conjugatable-ordered-field, real-normed-field})  $\leftarrow$  as. [:- a, 1:])  
**and** le-1:  $\bigwedge$  a. a  $\in$  set as  $\implies$  norm a  $\leq$  1

**and** *le-N*:  $\bigwedge a. a \in \text{set } as \implies \text{norm } a = 1 \implies \text{length } (\text{filter } ((=) a) as) \leq N$   
**shows**  $\exists c1\ c2. \forall k. \text{norm-bound } (A \hat{=}^m k) (c1 + c2 * \text{of-nat } k \hat{=} (N - 1))$   
*<proof>*

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*:  $\bigwedge a. a \in \text{set } (\text{diag-mat } A) \implies \text{norm } a \leq 1$   
**and** *le-N*:  $\bigwedge a. a \in \text{set } (\text{diag-mat } A) \implies \text{norm } a = 1 \implies \text{length } (\text{filter } ((=) a) (\text{diag-mat } A)) \leq N$   
**shows**  $\exists c1\ c2. \forall k. \text{norm-bound } (A \hat{=}^m k) (c1 + c2 * \text{of-nat } k \hat{=} (N - 1))$   
*<proof>*

If we have an upper triangular matrix *A* then we can compute a JNF-vector of it. If this vector does not contain entries  $(n, ev)$  with *ev* being larger 1, then the growth rate of  $A^k$  can be restricted by  $\mathcal{O}(k^{N-1})$  where *N* is the maximal value for *n*, where  $(n, |ev| = 1)$  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** *le-1*:  $\bigwedge n\ a. (n, a) \in \text{set } (\text{triangular-to-jnf-vector } A) \implies \text{norm } a \leq 1$   
**and** *le-N*:  $\bigwedge n\ a. (n, a) \in \text{set } (\text{triangular-to-jnf-vector } A) \implies \text{norm } a = 1 \implies n \leq N$   
**shows**  $\exists c1\ c2. \forall k. \text{norm-bound } (A \hat{=}^m k) (c1 + c2 * \text{of-nat } k \hat{=} (N - 1))$   
*<proof>*

**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] 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 [
  [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 :: rat]])
```

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

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* ≡  
*dim-row A* < *strassen-constant* ∨  
*dim-col A* < *strassen-constant* ∨  
*dim-col B* < *strassen-constant*

We have to make a case analysis on whether all dimensions are even.

**definition** *strassen-even* *A B* ≡ *even (dim-row A)* ∧ *even (dim-col A)* ∧ *even (dim-col B)*

And then we can define the algorithm.

**function** *strassen-mat-mult* :: '*a* :: *ring mat* ⇒ '*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 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);*  
*M4 = strassen-mat-mult A4 (B3 - B1);*  
*M5 = strassen-mat-mult (A1 + A2) B4;*  
*M6 = strassen-mat-mult (A3 - A1) (B1 + B2);*  
*M7 = strassen-mat-mult (A2 - A4) (B3 + B4);*  
*C1 = M1 + M4 - M5 + M7;*  
*C2 = M3 + M5;*  
*C3 = M2 + M4;*  
*C4 = M1 - M2 + M3 + M6*  
*in four-block-mat C1 C2 C3 C4 else*  
*let*  
*nr' = (nr div 2) \* 2;*  
*n' = (n div 2) \* 2;*  
*nc' = (nc div 2) \* 2;*  
*(A1,A2,A3,A4) = split-block A nr' n';*  
*(B1,B2,B3,B4) = split-block B n' nc';*  
*C1 = strassen-mat-mult A1 B1 + A2 \* B3;*  
*C2 = A1 \* B2 + A2 \* B4;*  
*C3 = A3 \* B1 + A4 \* B3;*  
*C4 = A3 \* B2 + A4 \* B4*



*in four-block-mat C1 C2 C3 C4)*  
 ⟨proof⟩

For termination, we use the following measure.

**definition** *strassen-measure*  $\equiv \lambda (A,B). (\text{dim-row } A + \text{dim-col } A + \text{dim-col } B)$   
 $+ (\text{dim-row } A + \text{dim-col } A + \text{dim-col } B) + (\text{if strassen-even } A \ B \ \text{then } 0 \ \text{else } 1)$

**lemma** *strassen-measure-add[simp]*:

*strassen-measure*  $(A + B, C) = \text{strassen-measure } (B, C)$   
*strassen-measure*  $(A, B + C) = \text{strassen-measure } (A, C)$   
*strassen-measure*  $(A - B, C) = \text{strassen-measure } (B, C)$   
*strassen-measure*  $(A, B - C) = \text{strassen-measure } (A, C)$   
*strassen-measure*  $(- A, B) = \text{strassen-measure } (A, B)$   
*strassen-measure*  $(A, - B) = \text{strassen-measure } (A, B)$   
 ⟨proof⟩

**lemma** *strassen-measure-div-2: assumes*  $(A1, A2, A3, A4) = \text{split-block } A (\text{dim-row } A \ \text{div } 2)$   $(\text{dim-col } A \ \text{div } 2)$

$(B1, B2, B3, B4) = \text{split-block } B (\text{dim-col } A \ \text{div } 2) (\text{dim-col } B \ \text{div } 2)$

**and** *large*:  $\neg \text{strassen-too-small } A \ B$

**shows**

*strassen-measure*  $(A1, B4) < \text{strassen-measure } (A, B)$   
*strassen-measure*  $(A1, B2) < \text{strassen-measure } (A, B)$   
*strassen-measure*  $(A2, B4) < \text{strassen-measure } (A, B)$   
*strassen-measure*  $(A3, B2) < \text{strassen-measure } (A, B)$   
*strassen-measure*  $(A4, B1) < \text{strassen-measure } (A, B)$   
*strassen-measure*  $(A4, B3) < \text{strassen-measure } (A, B)$   
*strassen-measure*  $(A4, B4) < \text{strassen-measure } (A, B)$

⟨proof⟩

**lemma** *strassen-measure-odd: assumes*  $(A1, A2, A3, A4) = \text{split-block } A ((\text{dim-row } A \ \text{div } 2) * 2) ((\text{dim-col } A \ \text{div } 2) * 2)$

**and**  $(B1, B2, B3, B4) = \text{split-block } B ((\text{dim-col } A \ \text{div } 2) * 2) ((\text{dim-col } B \ \text{div } 2) * 2)$

**and** *odd*:  $\neg \text{strassen-even } A \ B$

**shows** *strassen-measure*  $(A1, B1) < \text{strassen-measure } (A, B)$

⟨proof⟩

**termination** ⟨proof⟩

**lemma** *strassen-mat-mult*:

*dim-col*  $A = \text{dim-row } B \implies \text{strassen-mat-mult } A \ B = A * B$

⟨proof⟩

**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  $?A * ?B \equiv \text{mat } (\text{dim-row } ?A) (\text{dim-col } ?B) (\lambda(i, j). \text{row } ?A \ i \cdot \text{col } ?B \ j)$  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*  $A \ B = \text{mat } (\text{dim-row } A) (\text{dim-col } B) (\lambda \ (i,j). \text{row } A \ i \cdot \text{col } B \ j)$

*<proof>*

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]*:  $A * B = (\text{if } \text{dim-col } A = \text{dim-row } B \text{ then } \text{strassen-mat-mult } A \ B \text{ else } \text{basic-mat-mult } A \ B)$

*<proof>*

**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  $\Rightarrow$  'a mat  $\Rightarrow$  bool (**infix**  $\geq_m$  50) **where**  
A  $\geq_m$  B = ( $\forall i < \text{dim-row } A. \forall j < \text{dim-col } A. A \text{ \$\$ } (i,j) \geq B \text{ \$\$ } (i,j)$ )

**lemma** *mat-geI[intro]*: **assumes** A  $\in$  *carrier-mat nr nc*

$\bigwedge i j. i < nr \implies j < nc \implies A \text{ \$\$ } (i,j) \geq B \text{ \$\$ } (i,j)$

**shows** A  $\geq_m$  B

*<proof>*

**lemma** *mat-geD[dest]*: **assumes** A  $\geq_m$  B **and**  $i < \text{dim-row } A$   $j < \text{dim-col } A$

**shows** A  $\text{ \$\$ } (i,j) \geq B \text{ \$\$ } (i,j)$

*<proof>*

**definition** *mat-gt* :: ('a  $\Rightarrow$  'a  $\Rightarrow$  bool)  $\Rightarrow$  nat  $\Rightarrow$  'a mat  $\Rightarrow$  'a mat  $\Rightarrow$  bool **where**

*mat-gt gt sd* A B = (A  $\geq_m$  B  $\wedge$  ( $\exists i < \text{sd}. \exists j < \text{sd}. \text{gt } (A \text{ \$\$ } (i,j)) (B \text{ \$\$ } (i,j))$ ))

**lemma** *mat-gtI[intro]*: **assumes** A  $\geq_m$  B

**and**  $i < \text{sd } j < \text{sd } \text{gt } (A \text{ \$\$ } (i,j)) (B \text{ \$\$ } (i,j))$

**shows** *mat-gt gt sd* A B

*<proof>*

**lemma** *mat-gtD[dest]*: **assumes** *mat-gt gt sd* A B

**shows** A  $\geq_m$  B  $\exists i < \text{sd}. \exists j < \text{sd}. \text{gt } (A \text{ \$\$ } (i,j)) (B \text{ \$\$ } (i,j))$

*<proof>*

**definition** *mat-max* :: 'a mat  $\Rightarrow$  'a mat  $\Rightarrow$  'a mat (*max<sub>m</sub>*) **where**

*max<sub>m</sub>* A B = *mat* (*dim-row* A) (*dim-col* A) ( $\lambda ij. \text{max } (A \text{ \$\$ } ij) (B \text{ \$\$ } ij)$ )

**lemma** *mat-max-carrier[simp]*:

*max<sub>m</sub>* A B  $\in$  *carrier-mat* (*dim-row* A) (*dim-col* A)

*<proof>*

**lemma** *mat-max-closed[intro]*:

A  $\in$  *carrier-mat nr nc*  $\implies$  B  $\in$  *carrier-mat nr nc*  $\implies$  *max<sub>m</sub>* A B  $\in$  *carrier-mat nr nc*

*<proof>*

**lemma** *mat-max-index*:

**assumes**  $i < \text{dim-row } A$   $j < \text{dim-col } A$

**shows** (*mat-max* A B)  $\text{ \$\$ } (i,j) = \text{max } (A \text{ \$\$ } (i,j)) (B \text{ \$\$ } (i,j))$

*<proof>*

**definition** (**in** *zero*) *mat-default* :: 'a  $\Rightarrow$  nat  $\Rightarrow$  'a mat (*default<sub>m</sub>*) **where**

*default<sub>m</sub>* d n = *mat* n n ( $\lambda (i,j). \text{if } i = j \text{ then } d \text{ else } 0$ )

**lemma** *mat-default-carrier[simp]*: *default<sub>m</sub>* d n  $\in$  *carrier-mat n n*

*<proof>*  
**end**

**definition** *mat-mono* :: ('a ⇒ bool) ⇒ nat ⇒ 'a mat ⇒ bool  
**where** *mat-mono* P sd A = (∀ j < sd. ∃ i < sd. P (A \$\$ (i,j)))

**context** *non-strict-order*  
**begin**

**lemma** *mat-ge-trans*: **assumes**  $A \geq_m B$   $B \geq_m C$   
**and**  $A \in \text{carrier-mat nr nc}$   $B \in \text{carrier-mat nr nc}$   
**shows**  $A \geq_m C$   
*<proof>*

**lemma** *mat-ge-refl*:  $A \geq_m A$   
*<proof>*

**lemma** *mat-max-comm*:  $A \in \text{carrier-mat nr nc} \implies B \in \text{carrier-mat nr nc} \implies$   
 $\text{max}_m A B = \text{max}_m B A$   
*<proof>*

**lemma** *mat-max-ge*:  $\text{max}_m A B \geq_m A$   
*<proof>*

**lemma** *mat-max-ge-0*:  $A \in \text{carrier-mat nr nc} \implies B \in \text{carrier-mat nr nc} \implies A$   
 $\geq_m B \implies \text{max}_m A B = A$   
*<proof>*

**lemma** *mat-max-mono*:  $A \geq_m B \implies$   
 $A \in \text{carrier-mat nr nc} \implies B \in \text{carrier-mat nr nc} \implies C \in \text{carrier-mat nr nc}$   
 $\implies$   
 $\text{max}_m C A \geq_m \text{max}_m C B$   
*<proof>*

**end**

**lemma** *mat-plus-left-mono*:  $A \geq_m (B :: 'a :: \text{ordered-ab-semigroup mat})$   
 $\implies A \in \text{carrier-mat nr nc} \implies B \in \text{carrier-mat nr nc} \implies C \in \text{carrier-mat nr}$   
 $\text{nc}$   
 $\implies A + C \geq_m B + C$   
*<proof>*

**lemma** *mat-plus-right-mono*:  $B \geq_m (C :: 'a :: \text{ordered-ab-semigroup mat})$   
 $\implies A \in \text{carrier-mat nr nc} \implies B \in \text{carrier-mat nr nc} \implies C \in \text{carrier-mat nr}$   
 $\text{nc}$   
 $\implies A + B \geq_m A + C$   
*<proof>*

**lemma** *plus-mono*:  $x_1 \geq (x_2 :: 'a :: \text{ordered-ab-semigroup}) \implies$   
 $y_1 \geq y_2 \implies x_1 + y_1 \geq x_2 + y_2$

*<proof>*

Since one cannot use  $(\bigwedge i. i \in ?K \implies ?f i \leq ?g i) \implies \text{sum } ?f ?K \leq \text{sum } ?g ?K$  (it requires other class constraints like *order*), we make our own copy of this fact.

**lemma** *sum-mono-ge*:

**assumes** *ge*:  $\bigwedge i. i \in K \implies f (i::'a) \geq ((g i)::('b::\text{ordered-semiring-0}))$

**shows**  $(\sum i \in K. f i) \geq (\sum i \in K. g i)$

*<proof>*

**lemma** (in *one-mono-ordered-semiring-1*) *sum-mono-gt*:

**assumes** *le*:  $\bigwedge i. i \in K \implies f (i::'b) \geq ((g i)::'a)$

**and** *i*:  $i \in K$

**and** *gt*:  $f i \succ g i$

**and** *K*: *finite K*

**shows**  $(\sum i \in K. f i) \succ (\sum i \in K. g i)$

*<proof>*

**lemma** *scalar-left-mono*: **assumes**

*u*  $\in$  *carrier-vec n* *v*  $\in$  *carrier-vec n* *w*  $\in$  *carrier-vec n*

**and**  $\bigwedge i. i < n \implies u \$ i \geq v \$ i$

**and**  $\bigwedge i. i < n \implies w \$ i \geq (0 :: 'a :: \text{ordered-semiring-0})$

**shows**  $u \cdot w \geq v \cdot w$  *<proof>*

**lemma** *scalar-right-mono*: **assumes**

*u*  $\in$  *carrier-vec n* *v*  $\in$  *carrier-vec n* *w*  $\in$  *carrier-vec n*

**and**  $\bigwedge i. i < n \implies v \$ i \geq w \$ i$

**and**  $\bigwedge i. i < n \implies u \$ i \geq (0 :: 'a :: \text{ordered-semiring-0})$

**shows**  $u \cdot v \geq u \cdot w$

*<proof>*

**lemma** *mat-mult-left-mono*: **assumes** *C0*:  $C \geq_m 0_m n n$

**and** *AB*:  $A \geq_m (B :: 'a :: \text{ordered-semiring-0 mat})$

**and** *carr*:  $A \in \text{carrier-mat } n n$   $B \in \text{carrier-mat } n n$   $C \in \text{carrier-mat } n n$

**shows**  $A * C \geq_m B * C$

*<proof>*

**lemma** *mat-mult-right-mono*: **assumes** *A0*:  $A \geq_m 0_m n n$

**and** *BC*:  $B \geq_m (C :: 'a :: \text{ordered-semiring-0 mat})$

**and** *carr*:  $A \in \text{carrier-mat } n n$   $B \in \text{carrier-mat } n n$   $C \in \text{carrier-mat } n n$

**shows**  $A * B \geq_m A * C$

*<proof>*

**lemma** *one-mat-ge-zero*:  $(1_m n :: 'a :: \text{ordered-semiring-1 mat}) \geq_m 0_m n n$

*<proof>*

**context** *order-pair*

**begin**

**lemma** *mat-ge-gt-trans*: **assumes** *sd*:  $sd \leq n$  **and** *AB*:  $A \geq_m B$  **and** *BC*: *mat-gt*

*gt sd B C*  
**and**  $A: A \in \text{carrier-mat } n \ n$  **and**  $B: B \in \text{carrier-mat } n \ n$   
**shows** *mat-gt gt sd A C*  
 ⟨*proof*⟩

**lemma** *mat-gt-ge-trans*: **assumes**  $sd: sd \leq n$  **and**  $AB: \text{mat-gt gt sd } A \ B$  **and**  $BC: B \geq_m C$   
**and**  $A: A \in \text{carrier-mat } n \ n$  **and**  $B: B \in \text{carrier-mat } n \ n$   
**shows** *mat-gt gt sd A C*  
 ⟨*proof*⟩

**lemma** *mat-gt-imp-mat-ge*:  $\text{mat-gt gt sd } A \ B \implies A \geq_m B$   
 ⟨*proof*⟩

**lemma** *mat-gt-trans*: **assumes**  $sd: sd \leq n$  **and**  $AB: \text{mat-gt gt sd } A \ B$  **and**  $BC: \text{mat-gt gt sd } B \ C$   
**and**  $A: A \in \text{carrier-mat } n \ n$  **and**  $B: B \in \text{carrier-mat } n \ n$   
**shows** *mat-gt gt sd A C*  
 ⟨*proof*⟩

**lemma** *mat-default-ge-0*:  $\text{default}_m \ \text{default } n \geq_m \ 0_m \ n \ n$   
 ⟨*proof*⟩  
**end**

**definition** *mat-ordered-semiring* ::  $\text{nat} \Rightarrow \text{nat} \Rightarrow ('a :: \text{ordered-semiring-1} \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow 'b \Rightarrow ('a \ \text{mat}, 'b) \ \text{ordered-semiring-scheme}$  **where**  
 $\text{mat-ordered-semiring } n \ sd \ gt \ b \equiv \text{ring-mat } \text{TYPE}('a) \ n \ (\langle$   
 $\text{ordered-semiring.geq} = (\geq_m),$   
 $gt = \text{mat-gt gt sd},$   
 $max = \text{max}_m,$   
 $\dots = b \rangle)$

**lemma** (**in** *one-mono-ordered-semiring-1*) *mat-ordered-semiring*: **assumes**  $sd-n: sd \leq n$   
**shows** *ordered-semiring*  
 $(\text{mat-ordered-semiring } n \ sd \ (\succ) \ b :: ('a \ \text{mat}, 'b) \ \text{ordered-semiring-scheme})$   
**(is** *ordered-semiring ?R*)  
 ⟨*proof*⟩

**context** *weak-SN-strict-mono-ordered-semiring-1*  
**begin**

**lemma** *weak-mat-gt-mono*: **assumes**  $sd-n: sd \leq n$  **and**  
 $\text{orient}: \bigwedge A \ B. A \in \text{carrier-mat } n \ n \implies B \in \text{carrier-mat } n \ n \implies (A, B) \in \text{set } ABs \implies \text{mat-gt weak-gt sd } A \ B$   
**shows**  $\exists \ gt. \text{SN-strict-mono-ordered-semiring-1 default gt mono} \wedge$   
 $(\forall A \ B. A \in \text{carrier-mat } n \ n \longrightarrow B \in \text{carrier-mat } n \ n \longrightarrow (A, B) \in \text{set } ABs \longrightarrow \text{mat-gt gt sd } A \ B)$   
 ⟨*proof*⟩

**end**

**lemma** *sum-mat-mono*:

**assumes**  $A: A \in \text{carrier-mat } nr \ nc$  **and**  $B: B \in \text{carrier-mat } nr \ nc$

**and**  $AB: A \geq_m (B :: 'a :: \text{ordered-semiring-0 mat})$

**shows**  $\text{sum-mat } A \geq \text{sum-mat } B$

*<proof>*

**context** *one-mono-ordered-semiring-1*

**begin**

**lemma** *sum-mat-mono-gt*:

**assumes**  $sd \leq n$

**and**  $A: A \in \text{carrier-mat } n \ n$  **and**  $B: B \in \text{carrier-mat } n \ n$

**and**  $AB: \text{mat-gt } (\succ) \ sd \ A \ (B :: 'a \ \text{mat})$

**shows**  $\text{sum-mat } A \succ \text{sum-mat } B$

*<proof>*

**lemma** *mat-plus-gt-left-mono*: **assumes**  $sd\text{-}n: sd \leq n$  **and**  $gt: \text{mat-gt } (\succ) \ sd \ A \ B$

**and**  $A: A \in \text{carrier-mat } n \ n$  **and**  $B: B \in \text{carrier-mat } n \ n$  **and**  $C: C \in \text{carrier-mat } n \ n$

**shows**  $\text{mat-gt } (\succ) \ sd \ (A + C) \ (B + C)$

*<proof>*

**lemma** *mat-gt-ge-mono*:  $sd \leq n \implies \text{mat-gt } gt \ sd \ A \ B \implies$

$\text{mat-gt } gt \ sd \ C \ D \implies$

$A \in \text{carrier-mat } n \ n \implies$

$B \in \text{carrier-mat } n \ n \implies$

$C \in \text{carrier-mat } n \ n \implies$

$D \in \text{carrier-mat } n \ n \implies$

$\text{mat-gt } gt \ sd \ (A + C) \ (B + D)$

*<proof>*

**lemma** *mat-default-gt-mat0*: **assumes**  $sd\text{-}pos: sd > 0$  **and**  $sd\text{-}n: sd \leq n$

**shows**  $\text{mat-gt } (\succ) \ sd \ (\text{default}_m \ \text{default } n) \ (0_m \ n \ n)$

*<proof>*

**end**

**context** *SN-one-mono-ordered-semiring-1*

**begin**

**abbreviation**  $\text{mat-s} :: 'a \ \text{mat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \ \text{mat} \Rightarrow \text{bool} \ ((- \succ_m \ - \ -))$   
[51,51,51,51] 50)

**where**  $A \succ_m \ n \ sd \ B \equiv (A \in \text{carrier-mat } n \ n \wedge B \in \text{carrier-mat } n \ n \wedge B \geq_m \ 0_m \ n \ n \wedge \text{mat-gt } (\succ) \ sd \ A \ B)$

**lemma** *mat-gt-SN*: **assumes**  $sd\text{-}n: sd \leq n$  **shows**  $SN \ \{(m1, m2) . m1 \succ_m \ n \ sd \ m2\}$

*<proof>*  
**end**

**context** *SN-strict-mono-ordered-semiring-1*  
**begin**

**lemma** *mat-mono*: **assumes** *sd-n*:  $sd \leq n$  **and** *A*:  $A \in \text{carrier-mat } n \ n$  **and** *B*:  $B \in \text{carrier-mat } n \ n$  **and** *C*:  $C \in \text{carrier-mat } n \ n$   
**and** *gt*: *mat-gt* ( $\succ$ ) *sd* *B* *C* **and** *gez*:  $A \succeq_m 0_m \ n \ n$  **and** *mmono*: *mat-mono mono sd A*  
**shows** *mat-gt* ( $\succ$ ) *sd* ( $A * B$ ) ( $A * C$ ) (**is** *mat-gt* - - ?*AB* ?*AC*)  
*<proof>*  
**end**

**definition** *mat-comp-all* ::  $('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow 'a \ \text{mat} \Rightarrow 'a \ \text{mat} \Rightarrow \text{bool}$   
**where** *mat-comp-all* *r* *A* *B* =  
 $(\forall \ i < \text{dim-row } A. \forall \ j < \text{dim-col } A. \ r \ (A \ \text{\$}\$ \ (i,j)) \ (B \ \text{\$}\$ \ (i,j)))$

**lemma** *mat-comp-allI*:  
**assumes**  $A \in \text{carrier-mat } nr \ nc$   $B \in \text{carrier-mat } nr \ nc$   
**and**  $\bigwedge \ i \ j. \ i < nr \Longrightarrow j < nc \Longrightarrow r \ (A \ \text{\$}\$ \ (i,j)) \ (B \ \text{\$}\$ \ (i,j))$   
**shows** *mat-comp-all* *r* *A* *B*  
*<proof>*

**lemma** *mat-comp-allE*:  
**assumes** *mat-comp-all* *r* *A* *B*  
**and**  $A \in \text{carrier-mat } nr \ nc$   $B \in \text{carrier-mat } nr \ nc$   
**shows**  $\bigwedge \ i \ j. \ i < nr \Longrightarrow j < nc \Longrightarrow r \ (A \ \text{\$}\$ \ (i,j)) \ (B \ \text{\$}\$ \ (i,j))$   
*<proof>*

**context** *weak-SN-both-mono-ordered-semiring-1*  
**begin**

**abbreviation** *weak-mat-gt-arc* ::  $'a \ \text{mat} \Rightarrow 'a \ \text{mat} \Rightarrow \text{bool}$   
**where** *weak-mat-gt-arc*  $\equiv \text{mat-comp-all } \text{weak-gt}$

**lemma** *weak-mat-gt-both-mono*:  
**assumes** *ABs*:  $\text{set } ABs \subseteq \text{carrier-mat } n \ n \times \text{carrier-mat } n \ n$   
**and** *orient*:  $\forall (A,B) \in \text{set } ABs. \ \text{weak-mat-gt-arc } A \ B$   
**shows**  $\exists \ \text{gt}. \ \text{SN-both-mono-ordered-semiring-1 default gt arc-pos} \wedge$   
 $(\forall (A,B) \in \text{set } ABs. \ \text{mat-comp-all gt } A \ B)$   
*<proof>*  
**end**

**definition** *mat-both-ordered-semiring* ::  $\text{nat} \Rightarrow ('a :: \text{ordered-semiring-1} \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow 'b \Rightarrow ('a \ \text{mat}, 'b) \ \text{ordered-semiring-scheme}$  **where**  
*mat-both-ordered-semiring* *n* *gt* *b*  $\equiv \text{ring-mat } \text{TYPE}('a) \ n \ (\text{ordered-semiring.geq} = \text{mat-ge},$   
 $\text{gt} = \text{mat-comp-all } \text{gt},$



$max = mat-max,$   
... = b)

**definition**  $mat-arc-posI :: ('a \Rightarrow bool) \Rightarrow 'a\ mat \Rightarrow bool$   
**where**  $mat-arc-posI\ ap\ A \equiv ap\ (A\ \$\$ (0,0))$

**context**  $both-mono-ordered-semiring-1$   
**begin**

**abbreviation**  $mat-gt-arc :: 'a\ mat \Rightarrow 'a\ mat \Rightarrow bool$   
**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**  $A: A \in carrier-mat\ nr\ nc$   
**and**  $B: B \in carrier-mat\ nr\ nc$   
**shows**  $mat-max\ A\ B = A$   
(*proof*)

**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**  $C: C \in carrier-mat\ nr\ nc$   
**shows**  $mat-gt-arc\ A\ C$   
(*proof*)

**lemma**  $mat-gt-arc-compat$ :  
**assumes**  $ge: mat-ge\ A\ B$   
**and**  $gt: mat-gt-arc\ B\ C$   
**and**  $A: A \in carrier-mat\ nr\ nc$   
**and**  $B: B \in carrier-mat\ nr\ nc$   
**and**  $C: C \in carrier-mat\ nr\ nc$   
**shows**  $mat-gt-arc\ A\ C$   
(*proof*)

**lemma**  $mat-gt-arc-compat2$ :  
**assumes**  $gt: mat-gt-arc\ A\ B$   
**and**  $ge: mat-ge\ B\ C$   
**and**  $A: A \in carrier-mat\ nr\ nc$   
**and**  $B: B \in carrier-mat\ nr\ nc$   
**and**  $C: C \in carrier-mat\ nr\ nc$   
**shows**  $mat-gt-arc\ A\ C$   
(*proof*)

**lemma** *mat-gt-arc-imp-mat-ge*:  
**assumes** *gt*: *mat-gt-arc* *A B*  
**and** *A*: *A*  $\in$  *carrier-mat* *nr nc*  
**and** *B*: *B*  $\in$  *carrier-mat* *nr nc*  
**shows** *mat-ge* *A B*  
 $\langle$ *proof* $\rangle$

**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*)  
 $\langle$ *proof* $\rangle$

**lemma** *mat0-leastI*:  
**assumes** *A*: *A*  $\in$  *carrier-mat* *nr nc*  
**shows** *mat-gt-arc* *A* ( $0_m$  *nr nc*)  
 $\langle$ *proof* $\rangle$

**lemma** *mat0-leastII*:  
**assumes** *gt*: *mat-gt-arc* ( $0_m$  *nr nc*) *A*  
**and** *A*: *A*  $\in$  *carrier-mat* *nr nc*  
**shows** *A* =  $0_m$  *nr nc*  
 $\langle$ *proof* $\rangle$

**lemma** *mat0-leastIII*:  
**assumes** *A*: *A*  $\in$  *carrier-mat* *nr nc*  
**shows** *mat-ge* *A* ( $(0_m$  *nr nc*) :: '*a mat*)  
 $\langle$ *proof* $\rangle$

**lemma** *mat-max-0-id*: **fixes** *A* :: '*a mat*  
**assumes** *A*: *A*  $\in$  *carrier-mat* *nr nc*  
**shows** *mat-max* ( $0_m$  *nr nc*) *A* = *A*  
 $\langle$ *proof* $\rangle$

**lemma** *mat-arc-pos-one*:  
**assumes**  $n0$ :  $n > 0$   
**shows** *mat-arc-posI* *arc-pos* ( $1_m$   $n$ )  
 $\langle$ *proof* $\rangle$

**lemma** *mat-arc-pos-zero*:  
**assumes**  $n0$ :  $n > 0$   
**shows**  $\neg$  *mat-arc-posI* *arc-pos* ( $0_m$   $n$   $n$ )  
 $\langle$ *proof* $\rangle$

**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*) ∈ *carrier-mat nr nc*  
**and** *B*: (*B*::'*a* *mat*) ∈ *carrier-mat nr nc*  
**and** *C*: (*C*::'*a* *mat*) ∈ *carrier-mat nr nc*  
**and** *D*: (*D*::'*a* *mat*) ∈ *carrier-mat nr nc*  
**shows** *mat-gt-arc* (*A* + *C*) (*B* + *D*) (**is** *mat-gt-arc* ?*AC* ?*BD*)  
 ⟨*proof*⟩

**definition** *vec-comp-all* :: ('*a* ⇒ '*a* ⇒ *bool*) ⇒ '*a* *vec* ⇒ '*a* *vec* ⇒ *bool*  
**where** *vec-comp-all* *r v w* ≡ ∀ *i* < *dim-vec v*. *r* (*v* \$ *i*) (*w* \$ *i*)

**lemma** *vec-comp-allI*:  
**assumes** ∧*i*. *i* < *dim-vec v* ⇒ *r* (*v* \$ *i*) (*w* \$ *i*)  
**shows** *vec-comp-all* *r v w*  
 ⟨*proof*⟩

**lemma** *vec-comp-allE*:  
*vec-comp-all* *r v w* ⇒ *i* < *dim-vec v* ⇒ *r* (*v* \$ *i*) (*w* \$ *i*)  
 ⟨*proof*⟩

**lemma** *scalar-prod-left-mono*:  
**assumes** *u*: *u* ∈ *carrier-vec n*  
**and** *v*: *v* ∈ *carrier-vec n*  
**and** *w*: *w* ∈ *carrier-vec n*  
**and** *uv*: *vec-comp-all* *gt u v*  
**shows** *scalar-prod u w* > *scalar-prod v w*  
 ⟨*proof*⟩

**lemma** *scalar-prod-right-mono*:  
**assumes** *u*: *u* ∈ *carrier-vec n*  
**and** *v*: *v* ∈ *carrier-vec n*  
**and** *w*: *w* ∈ *carrier-vec n*  
**and** *vw*: *vec-comp-all* *gt v w*  
**shows** *scalar-prod u v* > *scalar-prod u w*  
 ⟨*proof*⟩

**lemma** *mat-gt-arc-mult-left-mono*:  
**assumes** *gt1*: *mat-gt-arc* *A B*  
**and** *A*: (*A*::'*a* *mat*) ∈ *carrier-mat nr n*  
**and** *B*: (*B*::'*a* *mat*) ∈ *carrier-mat nr n*  
**and** *C*: (*C*::'*a* *mat*) ∈ *carrier-mat n nc*  
**shows** *mat-gt-arc* (*A* \* *C*) (*B* \* *C*) (**is** *mat-gt-arc* ?*AC* ?*BC*)  
 ⟨*proof*⟩

**lemma** *mat-gt-arc-mult-right-mono*:  
**assumes** *gt1*: *mat-gt-arc* *B C*  
**and** *A*: (*A*::'*a* *mat*) ∈ *carrier-mat nr n*  
**and** *B*: (*B*::'*a* *mat*) ∈ *carrier-mat n nc*  
**and** *C*: (*C*::'*a* *mat*) ∈ *carrier-mat n nc*

**shows**  $mat\text{-}gt\text{-}arc (A * B) (A * C)$  (**is**  $mat\text{-}gt\text{-}arc ?AB ?AC$ )  
 ⟨*proof*⟩

**lemma**  $mat\text{-}arc\text{-}pos\text{-}plus$ :  
**assumes**  $n0: n > 0$   
**and**  $A: A \in carrier\text{-}mat\ n\ n$   
**and**  $B: B \in carrier\text{-}mat\ n\ n$   
**and**  $arc\text{-}pos: mat\text{-}arc\text{-}pos\ A$   
**shows**  $mat\text{-}arc\text{-}pos (A + B)$   
 ⟨*proof*⟩

**lemma**  $scalar\text{-}prod\text{-}split\text{-}head$ : **assumes**  
 $A \in carrier\text{-}mat\ n\ n\ B \in carrier\text{-}mat\ n\ n\ n > 0$   
**shows**  $row\ A\ 0 \cdot col\ B\ 0 = A\ \$\$ (0,0) * B\ \$\$ (0,0) + (\sum\ i = 1..<n.\ A\ \$\$ (0,$   
 $i) * B\ \$\$ (i, 0))$   
 ⟨*proof*⟩

**lemma**  $mat\text{-}arc\text{-}pos\text{-}mult$ :  
**assumes**  $n0: n > 0$   
**and**  $A: A \in carrier\text{-}mat\ n\ n$   
**and**  $B: B \in carrier\text{-}mat\ n\ n$   
**and**  $apA: mat\text{-}arc\text{-}pos\ A$   
**and**  $apB: mat\text{-}arc\text{-}pos\ B$   
**shows**  $mat\text{-}arc\text{-}pos (A * B)$   
 ⟨*proof*⟩

**lemma**  $mat\text{-}arc\text{-}pos\text{-}mat\text{-}default$ :  
**assumes**  $n0: n > 0$  **shows**  $mat\text{-}arc\text{-}pos (mat\text{-}default\ default\ n)$   
 ⟨*proof*⟩

**lemma**  $mat\text{-}not\text{-}all\text{-}ge$ :  
**assumes**  $n\text{-}pos: n > 0$   
**and**  $A: A \in carrier\text{-}mat\ n\ n$   
**and**  $B: B \in carrier\text{-}mat\ n\ n$   
**and**  $apB: mat\text{-}arc\text{-}pos\ B$   
**shows**  $\exists C. C \in carrier\text{-}mat\ n\ n \wedge mat\text{-}ge\ C\ (0_m\ n\ n) \wedge mat\text{-}arc\text{-}pos\ C \wedge \neg$   
 $mat\text{-}ge\ A\ (B * C)$   
 ⟨*proof*⟩

**end**

**context**  $SN\text{-}both\text{-}mono\text{-}ordered\text{-}semiring\text{-}1$   
**begin**

**lemma**  $mat\text{-}gt\text{-}arc\text{-}SN$ :  
**assumes**  $n\text{-}pos: n > 0$   
**shows**  $SN\ \{(A,B) \in carrier\text{-}mat\ n\ n \times carrier\text{-}mat\ n\ n.\ mat\text{-}arc\text{-}pos\ B \wedge$   
 $mat\text{-}gt\text{-}arc\ A\ B\}$

```

    (is SN ?rel)
  ⟨proof⟩

```

```

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 ⇒ 'b :: floor-ceiling +

```

```

  assumes hom-le: hom  $x \leq z \implies x \leq \text{of-int } \lceil z \rceil$ 

```

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

```

class real-embedding = linordered-idom +

```

```

  fixes real-of :: 'a ⇒ 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  $x \leq z \implies x \leq \text{of-int } \lceil z \rceil$ 

```

```

interpretation real-embedding: ord-ring-hom (real-of :: 'a :: real-embedding ⇒
real)

```

```

  ⟨proof⟩

```

```

instantiation real :: real-embedding

```

```

begin

```

```

definition real-of-real :: real ⇒ real where

```

```

  real-of-real x = x

```

```

instance

```

```

  ⟨proof⟩

```

```

end

```

**instantiation** *int* :: *real-embedding*  
**begin**

**definition** *real-of-int* :: *int*  $\Rightarrow$  *real* **where**  
*real-of-int* *x* = *x*

**instance**  
*<proof>*  
**end**

**lemma** *real-of-rat-ineq*: **assumes** *real-of-rat* *x*  $\leq$  *z*  
**shows** *x*  $\leq$  *of-int* [*z*]  
*<proof>*

**instantiation** *rat* :: *real-embedding*  
**begin**

**definition** *real-of-rat* :: *rat*  $\Rightarrow$  *real* **where**  
*real-of-rat* *x* = *of-rat* *x*

**instance**  
*<proof>*  
**end**

**abbreviation** *mat-real* (*mat<sub>R</sub>*) **where** *mat<sub>R</sub>*  $\equiv$  *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*  $\Rightarrow$  '*a*  $\Rightarrow$  *nat*  $\Rightarrow$  *bool*

**where**

*deriv-bound* *r* *a* *n*  $\longleftrightarrow$   $\neg$  ( $\exists$  *b*. (*a*, *b*)  $\in$  *r*  $\rightsquigarrow$  *Suc* *n*)

**lemma** *deriv-boundI* [*intro?*]:

( $\bigwedge$  *m*. *n* < *m*  $\implies$  (*a*, *b*)  $\in$  *r*  $\rightsquigarrow$  *m*  $\implies$  *False*)  $\implies$  *deriv-bound* *r* *a* *n*

*<proof>*

**lemma** *deriv-boundE*:

```

assumes deriv-bound r a n
  and ( $\bigwedge b m. n < m \implies (a, b) \in r \rightsquigarrow m \implies \text{False}$ )  $\implies P$ 
shows P
<proof>

lemma deriv-bound-iff:
  deriv-bound r a n  $\iff (\forall b m. n < m \longrightarrow (a, b) \notin r \rightsquigarrow m)$ 
<proof>

lemma deriv-bound-empty [simp]:
  deriv-bound {} a n
<proof>

lemma deriv-bound-mono:
  assumes  $m \leq n$  and deriv-bound r a m
  shows deriv-bound r a n
<proof>

lemma deriv-bound-image:
  assumes b: deriv-bound r' (f a) n
  and step:  $\bigwedge a b. (a, b) \in r \implies (f a, f b) \in r'^+$ 
  shows deriv-bound r a n
<proof>

lemma deriv-bound-subset:
  assumes  $r \subseteq r'^+$ 
  and b: deriv-bound r' a n
  shows deriv-bound r a n
<proof>

lemma deriv-bound-SN-on:
  assumes deriv-bound r a n
  shows SN-on r {a}
<proof>

lemma deriv-bound-steps:
  assumes  $(a, b) \in r \rightsquigarrow n$ 
  and deriv-bound r a m
  shows  $n \leq m$ 
<proof>
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 <proof>
instance int  :: large-real-ordered-semiring-1 <proof>
instance rat  :: large-real-ordered-semiring-1 <proof>

    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 de-
fault gt
  for gt :: 'a :: large-ordered-semiring-1  $\Rightarrow$  'a  $\Rightarrow$  bool (infix  $\succ$  50) and default ::
'a +
  fixes bound :: 'a  $\Rightarrow$  nat
  assumes bound-mono:  $\bigwedge a b. a \geq b \implies bound\ a \geq bound\ b$ 
  and bound-plus:  $\bigwedge a b. bound\ (a + b) \leq bound\ a + bound\ b$ 
  and bound-plus-of-nat:  $\bigwedge a n. a \geq 0 \implies bound\ (a + of\ nat\ n) = bound\ a +$ 
bound (of-nat n)
  and bound-zero[simp]: bound 0 = 0
  and bound-one: bound 1  $\geq$  1
  and bound:  $\bigwedge a. deriv\ bound\ \{(a,b). b \geq 0 \wedge a \succ b\}\ a\ (bound\ a)$ 
begin

lemma bound-linear:  $\exists c. \forall n. bound\ (of\ nat\ n) \leq c * n$ 
<proof>

lemma bound-of-nat-times: bound (of-nat n * v)  $\leq$  n * bound v
<proof>

lemma bound-mult-of-nat: bound (a * of-nat n)  $\leq$  bound a * bound (of-nat n)
<proof>

lemma bound-pow-of-nat: bound (a * of-nat n ^ deg)  $\leq$  bound a * of-nat n ^ deg
<proof>
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 |
  shows-arctic (MinInfty) = shows "'-inf'"

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 a r @ s
  <proof>

definition shows-list (as :: arctic list) = showsp-list shows-prec 0 as

instance
  <proof>

end

instantiation arctic-delta :: (show) show
begin

fun shows-arctic-delta :: 'a arctic-delta  $\Rightarrow$  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-artic-delta-append [show-law-simps]:
  shows-prec d (a :: 'a arctic-delta) (r @ s) = shows-prec d a r @ s
  <proof>

definition shows-list (ps :: 'a arctic-delta list) = showsp-list shows-prec 0 ps

instance
  <proof>

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 = SN-one-mono-ordered-semiring-1 d gt
  for gt :: 'a :: {show,ordered-semiring-1}  $\Rightarrow$  'a  $\Rightarrow$  bool (infix  $\succ$  50) and d :: 'a

locale mono-matrix-carrier = complexity-one-mono-ordered-semiring-1 gt d bound
  for gt :: 'a :: {show,large-real-ordered-semiring-1}  $\Rightarrow$  'a  $\Rightarrow$  bool (infix  $\succ$  50) and
  d :: 'a
  and bound :: 'a  $\Rightarrow$  nat
+ fixes mono :: 'a  $\Rightarrow$  bool
  assumes mono:  $\bigwedge x y z. \text{mono } x \Rightarrow y \succ z \Rightarrow x \geq 0 \Rightarrow x * y \succ x * z$ 
```

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}  $\Rightarrow$  'a  $\Rightarrow$  bool
  and default :: 'a
  and mono :: 'a  $\Rightarrow$  bool
  assumes weak-gt-mono:  $\forall x y. (x,y) \in \text{set } xys \longrightarrow \text{weak-gt } x y$ 
   $\implies \exists gt \text{ bound. } \text{mono-matrix-carrier } gt \text{ default bound mono} \wedge (\forall x y. (x,y) \in$ 
   $\text{set } xys \longrightarrow gt \ x \ y)$ 
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 sd-n:  $sd \leq n$  and
  orient:  $\bigwedge A B. A \in \text{carrier-mat } n \ n \implies B \in \text{carrier-mat } n \ n \implies (A,B) \in \text{set}$ 
   $ABs \implies \text{weak-mat-gt } sd \ A \ B$ 
  shows  $\exists gt \text{ bound. } \text{mono-matrix-carrier } gt \text{ default bound mono}$ 
   $\wedge (\forall A B. A \in \text{carrier-mat } n \ n \longrightarrow B \in \text{carrier-mat } n \ n \longrightarrow (A, B) \in \text{set } ABs$ 
   $\longrightarrow \text{mat-gt } gt \ sd \ A \ B)$ 
  <proof>
end
```

```
sublocale mono-matrix-carrier  $\subseteq$  SN-strict-mono-ordered-semiring-1 d gt mono
  <proof>
```

**sublocale** *mono-matrix-carrier*  $\subseteq$  *matrix-carrier*  $\langle$ *proof* $\rangle$

## 27.2 The Integers as Carrier

**lemma** *int-complexity*:

*mono-matrix-carrier* ( $(>) :: \text{int} \Rightarrow \text{int} \Rightarrow \text{bool}$ ) 1 *nat int-mono*  
 $\langle$ *proof* $\rangle$

**lemma** *int-weak-complexity*:

*weak-complexity-linear-poly-order-carrier* ( $>$ ) 1 *int-mono*  
 $\langle$ *proof* $\rangle$

## 27.3 The Rational and Real Numbers as Carrier

**definition** *delta-bound* :: 'a :: floor-ceiling  $\Rightarrow$  'a  $\Rightarrow$  nat

**where**

*delta-bound* d x = nat (ceiling (x \* of-int (ceiling (1 / d))))

**lemma** *delta-complexity*:

**assumes** d0: d > 0 **and** d1: d  $\leq$  def

**shows** *mono-matrix-carrier* (delta-gt d) def (delta-bound d) *delta-mono*  
 $\langle$ *proof* $\rangle$

**lemma** *delta-weak-complexity-carrier*:

**assumes** d0: def > 0

**shows** *weak-complexity-linear-poly-order-carrier* ( $>$ ) def *delta-mono*  
 $\langle$ *proof* $\rangle$

## 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*  $\langle$ *proof* $\rangle$

**lemma** *arctic-weak-carrier*:

*weak-SN-both-mono-ordered-semiring-1* ( $>$ ) 1 *pos-arctic*  
 $\langle$ *proof* $\rangle$

**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  $\Rightarrow$  'a vec set **where**

*mat-kernel* A = { v . v  $\in$  carrier-vec (dim-col A)  $\wedge$  A \*<sub>v</sub> v = 0<sub>v</sub> (dim-row A) }

**lemma** *mat-kernelI*: **assumes** A  $\in$  carrier-mat nr nc v  $\in$  carrier-vec nc A \*<sub>v</sub> v = 0<sub>v</sub> nr

**shows** v  $\in$  *mat-kernel* A

*<proof>*

**lemma** *mat-kernelD*: **assumes** A  $\in$  carrier-mat nr nc v  $\in$  *mat-kernel* A

**shows** v  $\in$  carrier-vec nc A \*<sub>v</sub> v = 0<sub>v</sub> nr

*<proof>*

**lemma** *mat-kernel*: **assumes** A  $\in$  carrier-mat nr nc

**shows** *mat-kernel* A = {v. v  $\in$  carrier-vec nc  $\wedge$  A \*<sub>v</sub> v = 0<sub>v</sub> nr }

*<proof>*

**lemma** *mat-kernel-carrier*:

**assumes** A  $\in$  carrier-mat nr nc **shows** *mat-kernel* A  $\subseteq$  carrier-vec nc

*<proof>*

**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  $\subseteq$  *mat-kernel* (B \* A)

*<proof>*

**lemma** *mat-kernel-smult*: **assumes** A: A  $\in$  carrier-mat nr nc

**and** v: v  $\in$  *mat-kernel* A

**shows**  $a \cdot_v v \in \text{mat-kernel } A$   
 ⟨proof⟩

**lemma** *mat-kernel-mult-eq*: **assumes**  $A: A \in \text{carrier-mat } nr \ nc$   
**and**  $B: B \in \text{carrier-mat } nr \ nr$   
**and**  $C: C \in \text{carrier-mat } nr \ nr$   
**and** *inv*:  $C * B = 1_m \ nr$   
**shows**  $\text{mat-kernel } (B * A) = \text{mat-kernel } A$   
 ⟨proof⟩

**locale** *kernel* =  
**fixes**  $nr :: nat$   
**and**  $nc :: nat$   
**and**  $A :: 'a :: \text{field mat}$   
**assumes**  $A: A \in \text{carrier-mat } nr \ nc$   
**begin**

**sublocale** *NC*: *vec-space*  $TYPE('a) \ nc$  ⟨proof⟩

**abbreviation**  $VK \equiv NC.V(\text{carrier} := \text{mat-kernel } A)$

**sublocale** *Ker*: *vectorspace class-ring*  $VK$   
**rewrites** *carrier*  $VK = \text{mat-kernel } A$   
**and** [*simp*]: *add*  $VK = (+)$   
**and** [*simp*]: *zero*  $VK = 0_v \ nc$   
**and** [*simp*]: *module.smult*  $VK = (\cdot_v)$   
**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) = *minus*  
**and** *pow* (*class-ring* :: 'a ring) =  $(\wedge)$   
**and** *finsum* (*class-ring* :: 'a ring) = *sum*  
**and** *finprod* (*class-ring* :: 'a ring) = *prod*  
**and** *m-inv* (*class-ring* :: 'a ring)  $x = (\text{if } x = 0 \text{ then } \text{div0} \text{ else } \text{inverse } x)$   
 ⟨proof⟩

**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*  $\equiv Ker.gen-set$

**lemma** *finsum-same*:  
**assumes**  $f : S \rightarrow \text{mat-kernel } A$

**shows**  $\text{finsum } VK \ f \ S = \text{finsum } NC.V \ f \ S$   
 ⟨proof⟩

**lemma** *lincomb-same*:

**assumes**  $S\text{-kernel}: S \subseteq \text{mat-kernel } A$   
**shows**  $\text{lincomb } a \ S = NC.\text{lincomb } a \ S$   
 ⟨proof⟩

**lemma** *span-same*:

**assumes**  $S\text{-kernel}: S \subseteq \text{mat-kernel } A$   
**shows**  $\text{span } S = NC.\text{span } S$   
 ⟨proof⟩

**lemma** *lindep-same*:

**assumes**  $S\text{-kernel}: S \subseteq \text{mat-kernel } A$   
**shows**  $\text{Ker.lin-dep } S = NC.\text{lin-dep } S$   
 ⟨proof⟩

**lemma** *lincomb-index*:

**assumes**  $i: i < nc$   
**and**  $Xk: X \subseteq \text{mat-kernel } A$   
**shows**  $\text{lincomb } a \ X \ \$ \ i = \text{sum } (\lambda x. a \ x * x \ \$ \ i) \ X$   
 ⟨proof⟩

**end**

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

**and**  $A: A \in \text{carrier-mat } nr \ nc$  **shows**  
 $\text{set } (\text{find-base-vectors } A) \subseteq \text{mat-kernel } A$   
 $0_v \ nc \notin \text{set } (\text{find-base-vectors } A)$   
 $\text{kernel.basis } nc \ A \ (\text{set } (\text{find-base-vectors } A))$   
 $\text{card } (\text{set } (\text{find-base-vectors } A)) = nc - \text{card } \{ i. i < nr \wedge \text{row } A \ i \neq 0_v \ nc \}$   
 $\text{length } (\text{pivot-positions } A) = \text{card } \{ i. i < nr \wedge \text{row } A \ i \neq 0_v \ nc \}$   
 $\text{kernel.dim } nc \ A = nc - \text{card } \{ i. i < nr \wedge \text{row } A \ i \neq 0_v \ nc \}$   
 ⟨proof⟩

**definition** *kernel-dim* :: 'a :: field mat  $\Rightarrow$  nat **where**

[code del]:  $\text{kernel-dim } A = \text{kernel.dim } (\text{dim-col } A) \ A$

**lemma** (in *kernel*) *kernel-dim [simp]*:  $\text{kernel-dim } A = \text{dim}$  ⟨proof⟩

**lemma** *kernel-dim-code*[code]:

$\text{kernel-dim } A = \text{dim-col } A - \text{length } (\text{pivot-positions } (\text{gauss-jordan-single } A))$   
 ⟨proof⟩

**lemma** *kernel-one-mat*: **fixes**  $A :: 'a :: \text{field mat}$  **and**  $n :: \text{nat}$

**defines**  $A: A \equiv 1_m \ n$

**shows**  
 $\text{kernel.dim } n \ A = 0$   
 $\text{kernel.basis } n \ A \ \{\}$   
 ⟨proof⟩

**lemma** *kernel-upper-triangular*: **assumes**  $A: A \in \text{carrier-mat } n \ n$   
**and**  $ut: \text{upper-triangular } A$  **and**  $0: 0 \notin \text{set } (\text{diag-mat } A)$   
**shows**  $\text{kernel.dim } n \ A = 0$   $\text{kernel.basis } n \ A \ \{\}$   
 ⟨proof⟩

**lemma** *kernel-basis-exists*: **assumes**  $A: A \in \text{carrier-mat } nr \ nc$   
**shows**  $\exists B. \text{finite } B \wedge \text{kernel.basis } nc \ A \ B$   
 ⟨proof⟩

**lemma** *mat-kernel-mult-right-gen-set*: **assumes**  $A: A \in \text{carrier-mat } nr \ nc$   
**and**  $B: B \in \text{carrier-mat } nc \ nc$   
**and**  $C: C \in \text{carrier-mat } nc \ nc$   
**and**  $inv: B * C = 1_m \ nc$   
**and**  $gen\text{-set}: \text{kernel.gen-set } nc \ (A * B) \ gen$  **and**  $gen: gen \subseteq \text{mat-kernel } (A * B)$   
**shows**  $\text{kernel.gen-set } nc \ A \ ((*_v) \ B) \ 'gen) \ (*_v) \ B \ 'gen \subseteq \text{mat-kernel } A \ \text{card}$   
 $((*_v) \ B) \ 'gen) = \text{card } gen$   
 ⟨proof⟩

**lemma** *mat-kernel-mult-right-basis*: **assumes**  $A: A \in \text{carrier-mat } nr \ nc$   
**and**  $B: B \in \text{carrier-mat } nc \ nc$   
**and**  $C: C \in \text{carrier-mat } nc \ nc$   
**and**  $inv: B * C = 1_m \ nc$   
**and**  $fin: \text{finite } gen$   
**and**  $basis: \text{kernel.basis } nc \ (A * B) \ gen$   
**shows**  $\text{kernel.basis } nc \ A \ ((*_v) \ B) \ 'gen)$   
 $\text{card } ((*_v) \ B) \ 'gen) = \text{card } gen$   
 ⟨proof⟩

**lemma** *mat-kernel-dim-mult-eq-right*: **assumes**  $A: A \in \text{carrier-mat } nr \ nc$   
**and**  $B: B \in \text{carrier-mat } nc \ nc$   
**and**  $C: C \in \text{carrier-mat } nc \ nc$   
**and**  $BC: B * C = 1_m \ nc$   
**shows**  $\text{kernel.dim } nc \ (A * B) = \text{kernel.dim } nc \ A$   
 ⟨proof⟩

**locale** *vardim* =  
**fixes**  $f\text{-ty} :: 'a :: \text{field itself}$   
**begin**

**abbreviation**  $M == \lambda k. \text{module-vec } \text{TYPE}('a) \ k$

**abbreviation**  $\text{span} == \lambda k. \text{LinearCombinations.module.span class-ring } (M k)$

**abbreviation**  $\text{lincomb} == \lambda k. \text{module.lincomb } (M k)$

**abbreviation**  $\text{lin-dep} == \lambda k. \text{module.lin-dep class-ring } (M k)$

**abbreviation**  $\text{padr } m v == v @_v 0_v m$

**definition**  $\text{unpadr } m v == \text{vec } (\text{dim-vec } v - m) (\lambda i. v \$ i)$

**abbreviation**  $\text{padl } m v == 0_v m @_v v$

**definition**  $\text{unpadl } m v == \text{vec } (\text{dim-vec } v - m) (\lambda i. v \$ (m+i))$

**lemma**  $\text{unpadr-padr[simp]}: \text{unpadr } m (\text{padr } m v) = v \langle \text{proof} \rangle$

**lemma**  $\text{unpadl-padl[simp]}: \text{unpadl } m (\text{padl } m v) = v \langle \text{proof} \rangle$

**lemma**  $\text{padr-unpadr[simp]}: v : \text{padr } m 'U \implies \text{padr } m (\text{unpadr } m v) = v \langle \text{proof} \rangle$

**lemma**  $\text{padl-unpadl[simp]}: v : \text{padl } m 'U \implies \text{padl } m (\text{unpadl } m v) = v \langle \text{proof} \rangle$

**lemma**  $\text{padr-image}$ :

**assumes**  $U \subseteq \text{carrier-vec } n$  **shows**  $\text{padr } m 'U \subseteq \text{carrier-vec } (n + m)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{padl-image}$ :

**assumes**  $U \subseteq \text{carrier-vec } n$  **shows**  $\text{padl } m 'U \subseteq \text{carrier-vec } (m + n)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{padr-inj}$ :

**shows**  $\text{inj-on } (\text{padr } m) (\text{carrier-vec } n :: 'a \text{ vec set})$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{padl-inj}$ :

**shows**  $\text{inj-on } (\text{padl } m) (\text{carrier-vec } n :: 'a \text{ vec set})$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{lincomb-pad}$ :

**fixes**  $m n a$

**assumes**  $U: (U :: 'a \text{ vec set}) \subseteq \text{carrier-vec } n$

**and**  $\text{fin}U: \text{finite } U$

**defines**  $\text{goal pad unpad } W == \text{pad } m (\text{lincomb } n a W) = \text{lincomb } (n+m) (a o \text{unpad } m) (\text{pad } m 'W)$

**shows**  $\text{goal padr unpadr } U$  **(is ?R)** **and**  $\text{goal padl unpadl } U$  **(is ?L)**  
 $\langle \text{proof} \rangle$

**lemma**  $\text{span-pad}$ :

**assumes**  $U: (U :: 'a \text{ vec set}) \subseteq \text{carrier-vec } n$

**defines**  $\text{goal pad } m == \text{pad } m ' \text{span } n U = \text{span } (n+m) (\text{pad } m 'U)$

**shows**  $\text{goal padr } m \text{ goal padl } m$

$\langle \text{proof} \rangle$

**lemma**  $\text{kernel-padr}$ :

**assumes**  $aA: a : \text{mat-kernel } (A :: 'a :: \text{field mat})$

**and**  $A: A : \text{carrier-mat } nr1 \ nc1$

**and**  $B: B : \text{carrier-mat } nr1 \ nc2$



**and**  $D: D \in \text{carrier-mat } nr2 \ nc2$   
**shows**  $\text{padr } nc2 \ a : \text{mat-kernel } (\text{four-block-mat } A \ B \ (0_m \ nr2 \ nc1) \ D) \ (\text{is } - : \text{mat-kernel } ?ABCD)$   
 <proof>

**lemma** *kernel-padl*:

**assumes**  $dD: d \in \text{mat-kernel } (D :: 'a :: \text{field mat})$   
**and**  $A: A \in \text{carrier-mat } nr1 \ nc1$   
**and**  $C: C \in \text{carrier-mat } nr2 \ nc1$   
**and**  $D: D \in \text{carrier-mat } nr2 \ nc2$   
**shows**  $\text{padl } nc1 \ d \in \text{mat-kernel } (\text{four-block-mat } A \ (0_m \ nr1 \ nc2) \ C \ D) \ (\text{is } - \in \text{mat-kernel } ?ABCD)$   
 <proof>

**lemma** *mat-kernel-split*:

**assumes**  $A: A \in \text{carrier-mat } n \ n$   
**and**  $D: D \in \text{carrier-mat } m \ m$   
**and**  $kAD: k \in \text{mat-kernel } (\text{four-block-mat } A \ (0_m \ n \ m) \ (0_m \ m \ n) \ D)$   
 ( $\text{is } - \in \text{mat-kernel } ?A00D$ )  
**shows**  $\text{vec-first } k \ n \in \text{mat-kernel } A \ (\text{is } ?a \in -)$   
**and**  $\text{vec-last } k \ m \in \text{mat-kernel } D \ (\text{is } ?d \in -)$   
 <proof>

**lemma** *padr-padl-eq*:

**assumes**  $v: v \in \text{carrier-vec } n$   
**shows**  $\text{padr } m \ v = \text{padl } n \ u \iff v = 0_v \ n \wedge u = 0_v \ m$   
 <proof>

**lemma** *pad-disjoint*:

**assumes**  $A: A \subseteq \text{carrier-vec } n$  **and**  $A0: 0_v \ n \notin A$  **and**  $B: B \subseteq \text{carrier-vec } m$   
**shows**  $\text{padr } m \ 'A \cap \text{padl } n \ 'B = \{\}$  ( $\text{is } ?A \cap ?B = -$ )  
 <proof>

**lemma** *padr-padl-lindep*:

**assumes**  $A: A \subseteq \text{carrier-vec } n$  **and**  $liA: \sim \text{lin-dep } n \ A$   
**and**  $B: B \subseteq \text{carrier-vec } m$  **and**  $liB: \sim \text{lin-dep } m \ B$   
**shows**  $\sim \text{lin-dep } (n+m) \ (\text{padr } m \ 'A \cup \text{padl } n \ 'B) \ (\text{is } \sim \text{lin-dep } - \ ( ?A \cup ?B))$   
 <proof>

**end**

**lemma** *kernel-four-block-0-mat*:

**assumes**  $Adef: (A :: 'a::\text{field mat}) = \text{four-block-mat } B \ (0_m \ n \ m) \ (0_m \ m \ n) \ D$   
**and**  $B: B \in \text{carrier-mat } n \ n$   
**and**  $D: D \in \text{carrier-mat } m \ m$   
**shows**  $\text{kernel.dim } (n + m) \ A = \text{kernel.dim } n \ B + \text{kernel.dim } m \ D$   
 <proof>

**lemma** *similar-mat-wit-kernel-dim*: **assumes**  $A: A \in \text{carrier-mat } n \ n$   
**and** *wit*: *similar-mat-wit*  $A \ B \ P \ Q$   
**shows**  $\text{kernel.dim } n \ A = \text{kernel.dim } n \ B$   
 $\langle \text{proof} \rangle$

**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*  $A \ B \ P \ Q$

**shows** *similar-mat-wit*  $(\text{char-matrix } A \ ev) (\text{char-matrix } B \ ev) \ P \ Q$

$\langle \text{proof} \rangle$

**context** **fixes**  $ty :: 'a :: \text{field itself}$

**begin**

**lemma** *dim-kernel-non-zero-jordan-block-pow*: **assumes**  $a: a \neq 0$

**shows**  $\text{kernel.dim } n \ (\text{jordan-block } n \ (a :: 'a) \ \widehat{m} \ k) = 0$

$\langle \text{proof} \rangle$

**lemma** *dim-kernel-zero-jordan-block-pow*:

$\text{kernel.dim } n \ ((\text{jordan-block } n \ (0 :: 'a)) \ \widehat{m} \ k) = \min k \ n \ (\text{is } \text{kernel.dim } - \ ?A = \ ?c)$

$\langle \text{proof} \rangle$

**definition** *dim-gen-eigenspace*  $:: 'a \ \text{mat} \Rightarrow 'a \Rightarrow \text{nat} \Rightarrow \text{nat}$  **where**

$\text{dim-gen-eigenspace } A \ ev \ k = \text{kernel-dim } ((\text{char-matrix } A \ ev) \ \widehat{m} \ k)$

**lemma** *dim-gen-eigenspace-jordan-matrix*:

$\text{dim-gen-eigenspace } (\text{jordan-matrix } n\text{-as}) \ ev \ k$

$= (\sum n \leftarrow \text{map fst } [(n, e) \leftarrow n\text{-as} \ . \ e = ev]. \ \min k \ n)$

$\langle \text{proof} \rangle$

**lemma** *dim-gen-eigenspace-similar*: **assumes** *sim*: *similar-mat*  $A \ B$

**shows**  $\text{dim-gen-eigenspace } A = \text{dim-gen-eigenspace } B$

$\langle \text{proof} \rangle$

**lemma** *dim-gen-eigenspace*: **assumes** *jordan-nf A n-as*  
**shows** *dim-gen-eigenspace A ev k*  
 $= (\sum n \leftarrow \text{map fst } [(n, e) \leftarrow n\text{-as} . e = \text{ev}]. \text{min } k \ n)$   
*<proof>*

**definition** *compute-nr-of-jordan-blocks* :: *'a mat*  $\Rightarrow$  *'a*  $\Rightarrow$  *nat*  $\Rightarrow$  *nat* **where**  
*compute-nr-of-jordan-blocks A ev k* =  $2 * \text{dim-gen-eigenspace } A \text{ ev } k -$   
 $\text{dim-gen-eigenspace } A \text{ ev } (k - 1) - \text{dim-gen-eigenspace } A \text{ ev } (\text{Suc } k)$

This lemma finally shows uniqueness of JNFs. Take an arbitrary JNF of a matrix  $A$ , (encoded by the list of Jordan-blocks  $n\text{-as}$ ), then then number of occurrences of each Jordan-Block in  $n\text{-as}$  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* =  $\text{length } (\text{filter } ((=) (k, \text{ev})) \ n\text{-as})$   
*<proof>*

**definition** *compute-set-of-jordan-blocks* :: *'a mat*  $\Rightarrow$  *'a*  $\Rightarrow$   $(\text{nat} \times 'a)\text{list}$  **where**  
*compute-set-of-jordan-blocks A ev*  $\equiv$  *let*  
 $k = \text{Polynomial.order ev } (\text{char-poly } A);$   
 $as = \text{map } (\text{dim-gen-eigenspace } A \text{ ev}) [0 ..< \text{Suc } (\text{Suc } k)];$   
 $cards = \text{map } (\lambda k. (k, 2 * as ! k - as ! (k - 1) - as ! \text{Suc } k)) [1 ..< \text{Suc } k]$   
*in*  $\text{map } (\lambda (k, c). (k, \text{ev})) (\text{filter } (\lambda (k, c). c \neq 0) \ cards)$

**lemma** *compute-set-of-jordan-blocks*: **assumes** *jnf: jordan-nf A n-as*  
**shows**  $\text{set } (\text{compute-set-of-jordan-blocks } A \text{ ev}) = \text{set } n\text{-as} \cap \text{UNIV} \times \{\text{ev}\}$  **(is**  
 $?C = ?N')$   
*<proof>*

**lemma** *jordan-nf-unique*: **assumes** *jordan-nf (A :: 'a mat) n-as* **and** *jordan-nf A m-bs*  
**shows**  $\text{set } n\text{-as} = \text{set } m\text{-bs}$   
*<proof>*

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*  $A = \text{Collect (eigenvalue } A)$

**lemma** *spectrum-root-char-poly*: **assumes**  $A: (A :: 'a :: \text{field mat}) \in \text{carrier-mat } n \ n$

**shows**  $\text{spectrum } A = \{k. \text{poly (char-poly } A) \ k = 0\}$

*<proof>*

**lemma** *card-finite-spectrum*: **assumes**  $A: (A :: 'a :: \text{field mat}) \in \text{carrier-mat } n \ n$

**shows**  $\text{finite (spectrum } A) \ \text{card (spectrum } A) \leq n$

*<proof>*

**lemma** *spectrum-non-empty*: **assumes**  $A: (A :: \text{complex mat}) \in \text{carrier-mat } n \ n$

**and**  $n: n > 0$

**shows**  $\text{spectrum } A \neq \{\}$

*<proof>*

**definition** *spectral-radius*  $:: \text{complex mat} \Rightarrow \text{real}$  **where**

$\text{spectral-radius } A = \text{Max (norm ' spectrum } A)$

**lemma** *spectral-radius-mem-max*: **assumes**  $A: A \in \text{carrier-mat } n \ n$

**and**  $n: n > 0$

**shows**  $\text{spectral-radius } A \in \text{norm ' spectrum } A$  (**is ?one**)

$a \in \text{norm ' spectrum } A \Rightarrow a \leq \text{spectral-radius } A$

*<proof>*

If spectral radius is at most 1, and JNF exists, then we have polynomial growth.

**lemma** *spectral-radius-jnf-norm-bound-le-1*: **assumes**  $A: A \in \text{carrier-mat } n \ n$

**and**  $\text{sr-1: spectral-radius } A \leq 1$

**and**  $\text{jnfxists: } \exists \text{ n-as. jordan-nf } A \ \text{n-as}$

**shows**  $\exists \ c1 \ c2. \forall \ k. \text{norm-bound (} A \ \widehat{\ }_m \ k) (c1 + c2 * \text{of-nat } k \ \widehat{\ } (n - 1))$

*<proof>*

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**  $A: A \in \text{carrier-mat } n \ n$

**and**  $\text{sr-1: spectral-radius } A < 1$

**and**  $\text{jnfxists: } \exists \ \text{n-as. jordan-nf } A \ \text{n-as}$

**shows**  $\exists \ c. \forall \ k. \text{norm-bound (} A \ \widehat{\ }_m \ k) \ c$

*<proof>*

If spectral radius is larger than 1, then we have exponential growth.

**lemma** *spectral-radius-gt-1*: **assumes**  $A: A \in \text{carrier-mat } n \ n$

**and**  $n: n > 0$

**and** *sr-1: spectral-radius*  $A > 1$   
**shows**  $\exists v c. v \in \text{carrier-vec } n \wedge \text{norm } c > 1 \wedge v \neq 0_v \wedge A \hat{\ }_m k *_v v = c \hat{\ }_k$   
 $\cdot_v v$   
 $\langle \text{proof} \rangle$

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**  $A: (A :: \text{complex mat}) \in \text{carrier-mat } n \ n$   
**and** *sr-1: spectral-radius*  $A \leq 1$   
**shows**  $\exists c1 \ c2. \forall k. \text{norm-bound } (A \hat{\ }_m k) (c1 + c2 * \text{of-nat } k \hat{\ }^{(n-1)})$   
 $\langle \text{proof} \rangle$

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**  $A: (A :: \text{complex mat}) \in \text{carrier-mat } n \ n$   
**and** *sr-1: spectral-radius*  $A < 1$   
**shows**  $\exists c. \forall k. \text{norm-bound } (A \hat{\ }_m k) c$   
 $\langle \text{proof} \rangle$

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

**lemma** *spectral-radius-poly-bound*: **fixes**  $A :: \text{complex mat}$   
**assumes**  $A: A \in \text{carrier-mat } n \ n$   
**and** *sr-1: spectral-radius*  $A \leq 1$   
**and** *eq-1*:  $\bigwedge ev \ k. \text{poly } (\text{char-poly } A) \ ev = 0 \implies \text{norm } ev = 1 \implies \text{Polynomial.order } ev (\text{char-poly } A) \leq d$   
**shows**  $\exists c1 \ c2. \forall k. \text{norm-bound } (A \hat{\ }_m k) (c1 + c2 * \text{of-nat } k \hat{\ }^{(d-1)})$   
 $\langle \text{proof} \rangle$

**end**

## 31 Missing Lemmas of List

**theory** *DL-Missing-List*  
**imports** *Main*  
**begin**

**lemma** *nth-map-zip*:  
**assumes**  $i < \text{length } xs$   
**assumes**  $i < \text{length } ys$   
**shows**  $\text{map } f (\text{zip } xs \ ys) ! i = f (xs ! i, ys ! i)$   
 $\langle \text{proof} \rangle$

**lemma** *nth-map-zip2*:  
**assumes**  $i < \text{length } (\text{map } f (\text{zip } xs \ ys))$   
**shows**  $\text{map } f (\text{zip } xs \ ys) ! i = f (xs ! i, ys ! i)$

*<proof>*

**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 ∈ *set* xs  
**shows** *find-first* a xs < *length* xs  
*<proof>*

**lemma** *nth-find-first*:  
**assumes** a ∈ *set* xs  
**shows** xs ! (*find-first* a xs) = a  
*<proof>*

**lemma** *find-first-unique*:  
**assumes** *distinct* xs  
**and** i < *length* xs  
**shows** *find-first* (xs ! i) xs = i  
*<proof>*

**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 ⊆ *carrier* V  
**and** *finite* S  
**and** *vectorspace.dim* K (*span-vs* S) = *card* S  
**shows** *lin-indpt* S  
*<proof>*

**lemma** (**in** *vectorspace*) *dim-span*:  
**assumes** S ⊆ *carrier* V  
**and** *finite* S  
**and** *maximal* U (λT. T ⊆ S ∧ *lin-indpt* T)  
**shows** *vectorspace.dim* K (*span-vs* S) = *card* U  
*<proof>*

**definition** (**in** *vec-space*) *rank* :: 'a mat ⇒ nat  
**where** *rank* A = *vectorspace.dim class-ring* (*span-vs* (*set* (*cols* A)))

**lemma** (*in vec-space*) *rank-card-indpt*:  
**assumes**  $A \in \text{carrier-mat } n \text{ } nc$   
**assumes** *maximal*  $S (\lambda T. T \subseteq \text{set } (\text{cols } A) \wedge \text{lin-indpt } T)$   
**shows**  $\text{rank } A = \text{card } S$   
*<proof>*

**lemma** *maximal-exists-superset*:  
**assumes** *finite*  $S$   
**assumes** *maxc*:  $\bigwedge A. P A \implies A \subseteq S \text{ and } P B$   
**shows**  $\exists A. \text{finite } A \wedge \text{maximal } A P \wedge B \subseteq A$   
*<proof>*

**lemma** (*in vec-space*) *rank-ge-card-indpt*:  
**assumes**  $A \in \text{carrier-mat } n \text{ } nc$   
**assumes**  $U \subseteq \text{set } (\text{cols } A)$   
**assumes** *lin-indpt*  $U$   
**shows**  $\text{rank } A \geq \text{card } U$   
*<proof>*

**lemma** (*in vec-space*) *lin-indpt-full-rank*:  
**assumes**  $A \in \text{carrier-mat } n \text{ } nc$   
**assumes** *distinct*  $(\text{cols } A)$   
**assumes** *lin-indpt*  $(\text{set } (\text{cols } A))$   
**shows**  $\text{rank } A = nc$   
*<proof>*

**lemma** (*in vec-space*) *rank-le-nc*:  
**assumes**  $A \in \text{carrier-mat } n \text{ } nc$   
**shows**  $\text{rank } A \leq nc$   
*<proof>*

**lemma** (*in vec-space*) *full-rank-lin-indpt*:  
**assumes**  $A \in \text{carrier-mat } n \text{ } nc$   
**assumes**  $\text{rank } A = nc$   
**assumes** *distinct*  $(\text{cols } A)$   
**shows** *lin-indpt*  $(\text{set } (\text{cols } A))$   
*<proof>*

**lemma** (*in vec-space*) *mat-mult-eq-lincomb*:  
**assumes**  $A \in \text{carrier-mat } n \text{ } nc$   
**assumes** *distinct*  $(\text{cols } A)$   
**shows**  $A *_v (\text{vec } nc (\lambda i. a (\text{col } A \ i))) = \text{lincomb } a (\text{set } (\text{cols } A))$   
*<proof>*

**lemma** (*in vec-space*) *lincomb-eq-mat-mult*:  
**assumes**  $A \in \text{carrier-mat } n \text{ } nc$   
**assumes**  $v \in \text{carrier-vec } nc$

**assumes** *distinct* (*cols A*)  
**shows** *lincomb* ( $\lambda a. v \ \$ \ \text{find-first } a \ (\text{cols } A)$ ) (*set* (*cols A*)) = (*A* \*<sub>v</sub> *v*)  
⟨*proof*⟩

**lemma** (**in** *vec-space*) *lin-depI*:  
**assumes** *A* ∈ *carrier-mat* *n nc*  
**assumes** *v* ∈ *carrier-vec* *nc v ≠ 0<sub>v</sub> nc A* \*<sub>v</sub> *v* = *0<sub>v</sub> n*  
**assumes** *distinct* (*cols A*)  
**shows** *lin-dep* (*set* (*cols A*))  
⟨*proof*⟩

**lemma** (**in** *vec-space*) *lin-depE*:  
**assumes** *A* ∈ *carrier-mat* *n nc*  
**assumes** *lin-dep* (*set* (*cols A*))  
**assumes** *distinct* (*cols A*)  
**obtains** *v* **where** *v* ∈ *carrier-vec* *nc v ≠ 0<sub>v</sub> nc A* \*<sub>v</sub> *v* = *0<sub>v</sub> n*  
⟨*proof*⟩

**lemma** (**in** *vec-space*) *non-distinct-low-rank*:  
**assumes** *A* ∈ *carrier-mat* *n n*  
**and**  $\neg$  *distinct* (*cols A*)  
**shows** *rank A* < *n*  
⟨*proof*⟩

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* ∈ *carrier-mat* *n n*  
**and** *det A* = *0*  
**shows** *rank A* < *n*  
⟨*proof*⟩

**lemma** *det-identical-cols*:  
**assumes** *A*: *A* ∈ *carrier-mat* *n n*  
**and** *ij*: *i* ≠ *j*  
**and** *i*: *i* < *n* **and** *j*: *j* < *n*  
**and** *r*: *col A i* = *col A j*  
**shows** *det A* = *0*  
⟨*proof*⟩

**lemma** (**in** *vec-space*) *low-rank-det-zero*:  
**assumes** *A* ∈ *carrier-mat* *n n*  
**and** *det A* ≠ *0*  
**shows** *rank A* = *n*  
⟨*proof*⟩

**lemma** (**in** *vec-space*) *det-rank-iff*:  
**assumes** *A* ∈ *carrier-mat* *n n*  
**shows** *det A* ≠ *0*  $\longleftrightarrow$  *rank A* = *n*



*<proof>*

### 33 Subadditivity of rank

Subadditivity is the property of rank, that  $\text{rank}(A + B) \leq \text{rank} A + \text{rank} B$ .

**lemma** (in *Module.module*) *lincomb-add*:

**assumes** *finite* ( $b1 \cup b2$ )

**assumes**  $b1 \cup b2 \subseteq \text{carrier } M$

**assumes**  $x1 = \text{lincomb } a1 \ b1 \ a1 \in (b1 \rightarrow \text{carrier } R)$

**assumes**  $x2 = \text{lincomb } a2 \ b2 \ a2 \in (b2 \rightarrow \text{carrier } R)$

**assumes**  $x = x1 \oplus_M x2$

**shows**  $\text{lincomb } (\lambda v. (\lambda v. \text{if } v \in b1 \text{ then } a1 \ v \ \text{else } \mathbf{0}) \ v \oplus (\lambda v. \text{if } v \in b2 \text{ then } a2 \ v \ \text{else } \mathbf{0}) \ v) \ (b1 \cup b2) = x$

*<proof>*

**lemma** (in *vectorspace*) *dim-subadditive*:

**assumes** *subspace*  $K \ W1 \ V$

**and** *vectorspace.fin-dim*  $K \ (vs \ W1)$

**assumes** *subspace*  $K \ W2 \ V$

**and** *vectorspace.fin-dim*  $K \ (vs \ W2)$

**shows**  $\text{vectorspace.dim } K \ (vs \ (\text{subspace-sum } W1 \ W2)) \leq \text{vectorspace.dim } K \ (vs \ W1) + \text{vectorspace.dim } K \ (vs \ W2)$

*<proof>*

**lemma** (in *Module.module*) *nested-submodules*:

**assumes** *submodule*  $R \ W \ M$

**assumes** *submodule*  $R \ X \ M$

**assumes**  $X \subseteq W$

**shows** *submodule*  $R \ X \ (md \ W)$

*<proof>*

**lemma** (in *vectorspace*) *nested-subspaces*:

**assumes** *subspace*  $K \ W \ V$

**assumes** *subspace*  $K \ X \ V$

**assumes**  $X \subseteq W$

**shows** *subspace*  $K \ X \ (vs \ W)$

*<proof>*

**lemma** (in *vectorspace*) *subspace-dim*:

**assumes** *subspace*  $K \ X \ V$  *fin-dim* *vectorspace.fin-dim*  $K \ (vs \ X)$

**shows**  $\text{vectorspace.dim } K \ (vs \ X) \leq \text{dim}$

*<proof>*

**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>*

**lemma** (*in vec-space*) *rank-subadditive:*  
**assumes**  $A \in \text{carrier-mat } n \text{ } nc$   
**assumes**  $B \in \text{carrier-mat } n \text{ } nc$   
**shows**  $\text{rank } (A + B) \leq \text{rank } A + \text{rank } B$   
*<proof>*

**lemma** (*in vec-space*) *span-zero: span {zero V} = {zero V}*  
*<proof>*

**lemma** (*in vec-space*) *dim-zero-vs: vectorspace.dim class-ring (span-vs {}) = 0*  
*<proof>*

**lemma** (*in vec-space*) *rank-0I: rank (0<sub>m</sub> n nc) = 0*  
*<proof>*

**lemma** (*in vec-space*) *rank-le-1-product-entries:*  
**fixes**  $f g :: \text{nat} \Rightarrow 'a$   
**assumes**  $A \in \text{carrier-mat } n \text{ } nc$   
**assumes**  $\bigwedge r \ c. r < \text{dim-row } A \implies c < \text{dim-col } A \implies A \ \$\$ (r,c) = f \ r * g \ c$   
**shows**  $\text{rank } A \leq 1$   
*<proof>*

**end**

## 34 Missing Lemmas of Sublist

**theory** *DL-Missing-Sublist*  
**imports** *Main*  
**begin**

**lemma** *nths-only-one:*  
**assumes**  $\{i. i < \text{length } xs \wedge i \in I\} = \{j\}$   
**shows**  $\text{nths } xs \ I = [xs!j]$   
*<proof>*

**lemma** *nths-replicate:*  
 $\text{nths } (\text{replicate } n \ x) \ A = (\text{replicate } (\text{card } \{i. i < n \wedge i \in A\}) \ x)$   
*<proof>*

**lemma** *length-nths-even:*  
**assumes** *even (length xs)*  
**shows**  $\text{length } (\text{nths } xs \ (\text{Collect } \text{even})) = \text{length } (\text{nths } xs \ (\text{Collect } \text{odd}))$   
*<proof>*

**lemma** *nths-map:*

$nths (map f xs) A = map f (nths xs A)$   
*<proof>*

## 35 Pick

**fun** *pick* :: *nat set*  $\Rightarrow$  *nat*  $\Rightarrow$  *nat* **where**  
*pick S 0* = (*LEAST a. a*  $\in$  *S*) |  
*pick S (Suc 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>*

**lemma** *pick-mono-inf*:  
**assumes** *infinite S*  
**shows**  $m < n \implies pick S m < pick S n$   
*<proof>*

**lemma** *pick-eq-iff-inf*:  
**assumes** *infinite S*  
**shows**  $x = y \longleftrightarrow pick S x = pick S y$   
*<proof>*

**lemma** *card-le-pick-inf*:  
**assumes** *infinite S*  
**and** *pick S n*  $\geq$  *i*  
**shows**  $card \{a \in S. a < i\} \leq n$   
*<proof>*

**lemma** *card-pick-inf*:  
**assumes** *infinite S*  
**shows**  $card \{a \in S. a < pick S n\} = n$   
*<proof>*

**lemma**  
**assumes**  $n < card S$   
**shows**  
  *pick-in-set-le*: *pick S n*  $\in$  *S* **and**  
  *card-pick-le*:  $card \{a \in S. a < pick S n\} = n$  **and**  
  *pick-mono-le*:  $m < n \implies pick S m < pick S n$   
*<proof>*

**lemma** *card-le-pick-le*:  
**assumes**  $n < card S$   
**and** *pick S n*  $\geq$  *i*  
**shows**  $card \{a \in S. a < i\} \leq n$   
*<proof>*

**lemma**  
**assumes**  $n < \text{card } S \vee \text{infinite } S$   
**shows**  
*pick-in-set*:  $\text{pick } S \ n \in S$  **and**  
*card-le-pick*:  $i \leq \text{pick } S \ n \implies \text{card } \{a \in S. a < i\} \leq n$  **and**  
*card-pick*:  $\text{card } \{a \in S. a < \text{pick } S \ n\} = n$  **and**  
*pick-mono*:  $m < n \implies \text{pick } S \ m < \text{pick } S \ n$   
*<proof>*

**lemma** *pick-card*:  
 $\text{pick } I \ (\text{card } \{a \in I. a < i\}) = (\text{LEAST } a. a \in I \wedge a \geq i)$   
*<proof>*

**lemma** *pick-card-in-set*:  $i \in I \implies \text{pick } I \ (\text{card } \{a \in I. a < i\}) = i$   
*<proof>*

## 36 Sublist

**lemma** *nth-nths-card*:  
**assumes**  $j < \text{length } xs$   
**and**  $j \in J$   
**shows**  $\text{nths } xs \ J \ ! \ \text{card } \{j0. j0 < j \wedge j0 \in J\} = xs!j$   
*<proof>*

**lemma** *pick-reduce-set*:  
**assumes**  $i < \text{card } \{a. a < m \wedge a \in I\}$   
**shows**  $\text{pick } I \ i = \text{pick } \{a. a < m \wedge a \in I\} \ i$   
*<proof>*

**lemma** *nth-nths*:  
**assumes**  $i < \text{card } \{i. i < \text{length } xs \wedge i \in I\}$   
**shows**  $\text{nths } xs \ I \ ! \ i = xs \ ! \ \text{pick } I \ i$   
*<proof>*

**lemma** *pick-UNIV*:  $\text{pick } \text{UNIV} \ j = j$   
*<proof>*

**lemma** *pick-le*:  
**assumes**  $n < \text{card } \{a. a < i \wedge a \in S\}$   
**shows**  $\text{pick } S \ n < i$   
*<proof>*

**lemma** *prod-list-complementary-nthss*:  
**fixes**  $f :: 'a \Rightarrow 'b :: \text{comm-monoid-mult}$   
**shows**  $\text{prod-list } (\text{map } f \ xs) = \text{prod-list } (\text{map } f \ (\text{nths } xs \ A)) * \text{prod-list } (\text{map } f \ (\text{nths } xs \ (-A)))$   
*<proof>*

**lemma** *nths-zip*:  $\text{nths } (\text{zip } xs \ ys) \ I = \text{zip } (\text{nths } xs \ I) \ (\text{nths } ys \ I)$

*<proof>*

### 37 weave

**definition** *weave* :: nat set  $\Rightarrow$  'a list  $\Rightarrow$  'a list  $\Rightarrow$  'a list **where**  
*weave* A xs ys = map ( $\lambda i$ . if  $i \in A$  then  $xs!(\text{card } \{a \in A. a < i\})$  else  $ys!(\text{card } \{a \in -A. a < i\})$ ) [0.. $\text{length } xs + \text{length } ys$ ]

**lemma** *length-weave*:

**shows**  $\text{length } (\text{weave } A \text{ } xs \text{ } ys) = \text{length } xs + \text{length } ys$

*<proof>*

**lemma** *nth-weave*:

**assumes**  $i < \text{length } (\text{weave } A \text{ } xs \text{ } ys)$

**shows**  $\text{weave } A \text{ } xs \text{ } ys ! i = (\text{if } i \in A \text{ then } xs!(\text{card } \{a \in A. a < i\}) \text{ else } ys!(\text{card } \{a \in -A. a < i\}))$

*<proof>*

**lemma** *weave-append1*:

**assumes**  $\text{length } xs + \text{length } ys \in A$

**assumes**  $\text{length } xs = \text{card } \{a \in A. a < \text{length } xs + \text{length } ys\}$

**shows**  $\text{weave } A \text{ } (xs @ [x]) \text{ } ys = \text{weave } A \text{ } xs \text{ } ys @ [x]$

*<proof>*

**lemma** *weave-append2*:

**assumes**  $\text{length } xs + \text{length } ys \notin A$

**assumes**  $\text{length } ys = \text{card } \{a \in -A. a < \text{length } xs + \text{length } ys\}$

**shows**  $\text{weave } A \text{ } xs \text{ } (ys @ [y]) = \text{weave } A \text{ } xs \text{ } ys @ [y]$

*<proof>*

**lemma** *nths-nth*:

**assumes**  $n \in A \text{ } n < \text{length } xs$

**shows**  $nths \text{ } xs \text{ } A ! (\text{card } \{i. i < n \wedge i \in A\}) = xs ! n$

*<proof>*

**lemma** *list-all2-nths*:

**assumes**  $list\text{-all2 } P \text{ } (nths \text{ } xs \text{ } A) \text{ } (nths \text{ } ys \text{ } A)$

**and**  $list\text{-all2 } P \text{ } (nths \text{ } xs \text{ } (-A)) \text{ } (nths \text{ } ys \text{ } (-A))$

**shows**  $list\text{-all2 } P \text{ } xs \text{ } ys$

*<proof>*

**lemma** *nths-weave*:

**assumes**  $\text{length } xs = \text{card } \{a \in A. a < \text{length } xs + \text{length } ys\}$

**assumes**  $\text{length } ys = \text{card } \{a \in (-A). a < \text{length } xs + \text{length } ys\}$

**shows**  $nths \text{ } (\text{weave } A \text{ } xs \text{ } ys) \text{ } A = xs \wedge nths \text{ } (\text{weave } A \text{ } xs \text{ } ys) \text{ } (-A) = ys$

*<proof>*

**lemma** *set-weave*:

**assumes**  $\text{length } xs = \text{card } \{a \in A. a < \text{length } xs + \text{length } ys\}$

**assumes**  $\text{length } ys = \text{card } \{a \in -A. a < \text{length } xs + \text{length } ys\}$   
**shows**  $\text{set } (\text{weave } A \text{ } xs \text{ } ys) = \text{set } xs \cup \text{set } ys$   
 $\langle \text{proof} \rangle$

**lemma** *weave-complementary-nthss[simp]*:  
 $\text{weave } A \text{ } (\text{nths } xs \text{ } A) \text{ } (\text{nths } xs \text{ } (-A)) = xs$   
 $\langle \text{proof} \rangle$

**lemma** *length-nths'*:  $\text{length } (\text{nths } xs \text{ } I) = \text{card } \{i \in I. i < \text{length } xs\}$   
 $\langle \text{proof} \rangle$

**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**  
 $\text{submatrix } A \text{ } I \text{ } J = \text{mat } (\text{card } \{i. i < \text{dim-row } A \wedge i \in I\}) \text{ } (\text{card } \{j. j < \text{dim-col } A \wedge j \in J\}) \text{ } (\lambda(i,j). A \text{ } \$\$ \text{ } (\text{pick } I \text{ } i, \text{pick } J \text{ } j))$

**lemma** *dim-submatrix*:  $\text{dim-row } (\text{submatrix } A \text{ } I \text{ } J) = \text{card } \{i. i < \text{dim-row } A \wedge i \in I\}$   
 $\text{dim-col } (\text{submatrix } A \text{ } I \text{ } J) = \text{card } \{j. j < \text{dim-col } A \wedge j \in J\}$   
 $\langle \text{proof} \rangle$

**lemma** *submatrix-index*:  
**assumes**  $i < \text{card } \{i. i < \text{dim-row } A \wedge i \in I\}$   
**assumes**  $j < \text{card } \{j. j < \text{dim-col } A \wedge j \in J\}$   
**shows**  $\text{submatrix } A \text{ } I \text{ } J \text{ } \$\$ \text{ } (i, j) = A \text{ } \$\$ \text{ } (\text{pick } I \text{ } i, \text{pick } J \text{ } j)$   
 $\langle \text{proof} \rangle$

**lemma** *set-le-in*:  $\{a. a < n \wedge a \in I\} = \{a \in I. a < n\}$   $\langle \text{proof} \rangle$

**lemma** *submatrix-index-card*:  
**assumes**  $i < \text{dim-row } A \text{ } j < \text{dim-col } A \text{ } i \in I \text{ } j \in J$   
**shows**  $\text{submatrix } A \text{ } I \text{ } J \text{ } \$\$ \text{ } (\text{card } \{a \in I. a < i\}, \text{card } \{a \in J. a < j\}) = A \text{ } \$\$ \text{ } (i, j)$   
 $\langle \text{proof} \rangle$

**lemma** *submatrix-split*:  $\text{submatrix } A \text{ } I \text{ } J = \text{submatrix } (\text{submatrix } A \text{ } \text{UNIV } J) \text{ } I$   
 $\text{UNIV}$   
 $\langle \text{proof} \rangle$

**end**

## 40 Rank and Submatrices

**theory** *DL-Rank-Submatrix*

**imports** *DL-Rank DL-Submatrix Matrix*

**begin**

**lemma** *row-submatrix-UNIV*:

**assumes**  $i < \text{card } \{i. i < \text{dim-row } A \wedge i \in I\}$

**shows**  $\text{row } (\text{submatrix } A \ I \ UNIV) \ i = \text{row } A \ (\text{pick } I \ i)$

*<proof>*

**lemma** *distinct-cols-submatrix-UNIV*:

**assumes**  $\text{distinct } (\text{cols } (\text{submatrix } A \ I \ UNIV))$

**shows**  $\text{distinct } (\text{cols } A)$

*<proof>*

**lemma** *cols-submatrix-subset*:  $\text{set } (\text{cols } (\text{submatrix } A \ UNIV \ J)) \subseteq \text{set } (\text{cols } A)$

*<proof>*

**lemma** (**in** *vec-space*) *lin-dep-submatrix-UNIV*:

**assumes**  $A \in \text{carrier-mat } n \ nc$

**assumes**  $\text{lin-dep } (\text{set } (\text{cols } A))$

**assumes**  $\text{distinct } (\text{cols } (\text{submatrix } A \ I \ UNIV))$

**shows**  $\text{LinearCombinations.module.lin-dep class-ring } (\text{module-vec } TYPE('a) \ (\text{card } \{i. i < n \wedge i \in I\})) \ (\text{set } (\text{cols } (\text{submatrix } A \ I \ UNIV)))$

(**is**  $\text{LinearCombinations.module.lin-dep class-ring } ?M \ (\text{set } ?S')$ )

*<proof>*

**lemma** (**in** *vec-space*) *rank-gt-minor*:

**assumes**  $A \in \text{carrier-mat } n \ nc$

**assumes**  $\text{det } (\text{submatrix } A \ I \ J) \neq 0$

**shows**  $\text{card } \{j. j < nc \wedge j \in J\} \leq \text{rank } A$

*<proof>*

**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](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.