

# A verified algorithm for computing the Smith normal form of a matrix

Jose Divasón

March 19, 2025

## Abstract

This work presents a formal proof in Isabelle/HOL of an algorithm to transform a matrix into its Smith normal form, a canonical matrix form, in a general setting: the algorithm is parameterized by operations to prove its existence over elementary divisor rings, while execution is guaranteed over Euclidean domains. We also provide a formal proof on some results about the generality of this algorithm as well as the uniqueness of the Smith normal form.

Since Isabelle/HOL does not feature dependent types, the development is carried out switching conveniently between two different existing libraries: the Hermite normal form (based on HOL Analysis) and the Jordan normal form AFP entries. This permits to reuse results from both developments and it is done by means of the lifting and transfer package together with the use of local type definitions.

## Contents

<b>1</b>	<b>Definition of Smith normal form in HOL Analysis</b>	<b>1</b>
1.1	Definitions . . . . .	1
1.2	Basic properties . . . . .	2
<b>2</b>	<b>Algorithm to transform a diagonal matrix into its Smith normal form</b>	<b>4</b>
2.1	Implementation of the algorithm . . . . .	4
2.2	Code equations to get an executable version . . . . .	6
2.3	Examples of execution . . . . .	6
2.4	Soundness of the algorithm . . . . .	7
2.5	Implementation and formal proof of the matrices $P$ and $Q$ which transform the input matrix by means of elementary operations. . . . .	30
2.6	The final soundness theorem . . . . .	46
<b>3</b>	<b>A new bridge to convert theorems from JNF to HOL Analysis and vice-versa, based on the <i>mod-type</i> class</b>	<b>46</b>

<b>4</b>	<b>Missing results</b>	<b>59</b>
4.1	Miscellaneous lemmas . . . . .	60
4.2	Transfer rules for the HMA_Connect file of the Perron-Frobenius development . . . . .	60
4.3	Lemmas obtained from HOL Analysis using local type definitions . . . . .	61
4.4	Lemmas about matrices, submatrices and determinants . . .	62
4.5	Lemmas about <i>sorted lists</i> , <i>insort</i> and <i>pick</i> . . . . .	79
<b>5</b>	<b>The Cauchy–Binet formula</b>	<b>85</b>
5.1	Previous missing results about <i>pick</i> and <i>insert</i> . . . . .	86
5.2	Start of the proof . . . . .	90
5.3	Final theorem . . . . .	115
<b>6</b>	<b>Definition of Smith normal form in JNF</b>	<b>117</b>
<b>7</b>	<b>Some theorems about rings and ideals</b>	<b>121</b>
7.1	Missing properties on ideals . . . . .	121
7.2	An equivalent characterization of Bézout rings . . . . .	132
<b>8</b>	<b>Connection between <i>mod-ring</i> and <i>mod-type</i></b>	<b>137</b>
<b>9</b>	<b>Generality of the Algorithm to transform from diagonal to Smith normal form</b>	<b>138</b>
9.1	Proof of the $\Leftarrow$ implication in HA. . . . .	139
9.2	Trying to prove the $\Rightarrow$ implication in HA. . . . .	140
9.3	Proof of the $\Rightarrow$ implication in JNF. . . . .	140
9.4	Trying to transfer the $\Rightarrow$ implication to HA. . . . .	145
9.5	Transferring the $\Leftarrow$ implication from HA to JNF using transfer rules and local type definitions . . . . .	150
9.6	Final theorem in JNF . . . . .	153
<b>10</b>	<b>Uniqueness of the Smith normal form</b>	<b>153</b>
10.1	More specific results about submatrices . . . . .	155
10.2	On the minors of a diagonal matrix . . . . .	159
10.3	Relating minors and GCD . . . . .	169
10.4	Final theorem . . . . .	174
10.5	Uniqueness fixing a complete set of non-associates . . . . .	178
<b>11</b>	<b>The Cauchy–Binet formula in HOL Analysis</b>	<b>180</b>
11.1	Definition of submatrices in HOL Analysis . . . . .	180
11.2	Transferring the proof from JNF to HOL Analysis . . . . .	181

<b>12</b>	<b>Diagonalizing matrices in JNF and HOL Analysis</b>	<b>182</b>
12.1	Diagonalizing matrices in JNF . . . . .	182
12.2	Implementation and soundness result moved to HOL Analysis.	183
<b>13</b>	<b>Smith normal form algorithm based on two steps in HOL Analysis</b>	<b>184</b>
13.1	The implementation . . . . .	184
13.2	Soundness in HOL Analysis . . . . .	185
<b>14</b>	<b>Algorithm to transform a diagonal matrix into its Smith normal form in JNF</b>	<b>185</b>
14.1	Attempt with the third option: definitions and conditional transfer rules . . . . .	186
14.2	Attempt with the second option: implementation and soundness in JNF . . . . .	187
14.3	Applying local type definitions . . . . .	196
14.4	The final result . . . . .	200
<b>15</b>	<b>Smith normal form algorithm based on two steps in JNF</b>	<b>200</b>
15.1	Moving the result from HOL Analysis to JNF . . . . .	200
<b>16</b>	<b>A general algorithm to transform a matrix into its Smith normal form</b>	<b>201</b>
16.1	Previous definitions and lemmas . . . . .	202
16.2	Previous operations . . . . .	203
16.3	The implementation . . . . .	209
16.3.1	Case $1 \times n$ . . . . .	218
16.3.2	Case $n \times 1$ . . . . .	224
16.3.3	Case $2 \times n$ . . . . .	225
16.3.4	Case $n \times 2$ . . . . .	241
16.3.5	Case $m \times n$ . . . . .	241
16.4	Soundness theorem . . . . .	260
<b>17</b>	<b>The Smith normal form algorithm in HOL Analysis</b>	<b>261</b>
17.1	Transferring the result from JNF to HOL Analysis . . . . .	261
17.2	Soundness in HOL Analysis . . . . .	262
<b>18</b>	<b>Elementary divisor rings</b>	<b>265</b>
18.1	Previous definitions and basic properties of Hermite ring . . . . .	265
18.2	The class that represents elementary divisor rings . . . . .	267
18.3	Hermite ring implies Bézout ring . . . . .	267
18.4	Elementary divisor ring implies Hermite ring . . . . .	272
18.5	Characterization of Elementary divisor rings . . . . .	288
18.6	Final theorem . . . . .	294

<b>19 Executable Smith normal form algorithm over Euclidean domains</b>	<b>296</b>
19.1 Previous code equations . . . . .	297
19.2 An executable algorithm to transform $2 \times 2$ matrices into its Smith normal form in HOL Analysis . . . . .	297
19.3 An executable algorithm to transform $2 \times 2$ matrices into its Smith normal form in JNF . . . . .	306
19.4 An executable algorithm to transform $1 \times 2$ matrices into its Smith normal form . . . . .	307
19.5 The final executable algorithm to transform any matrix into its Smith normal form . . . . .	311
<b>20 A certified checker based on an external algorithm to compute Smith normal form</b>	<b>312</b>

## 1 Definition of Smith normal form in HOL Analysis

```

theory Smith-Normal-Form
  imports
    Hermite.Hermite
begin

```

### 1.1 Definitions

Definition of diagonal matrix

**definition** *isDiagonal-upt-k*  $A\ k = (\forall\ a\ b. (to\ nat\ a \neq to\ nat\ b \wedge (to\ nat\ a < k \vee (to\ nat\ b < k))) \longrightarrow A\ \$\ a\ \$\ b = 0)$

**definition** *isDiagonal*  $A = (\forall\ a\ b. to\ nat\ a \neq to\ nat\ b \longrightarrow A\ \$\ a\ \$\ b = 0)$

**lemma** *isDiagonal-intro*:

```

fixes  $A::'a::\{zero\} \wedge cols::mod\ type \wedge rows::mod\ type$ 
assumes  $\bigwedge a::'rows. \bigwedge b::'cols. to\ nat\ a = to\ nat\ b$ 
shows isDiagonal  $A$ 
using assms
unfolding isDiagonal-def by auto

```

Definition of Smith normal form up to position k. The element  $A_{k-1,k-1}$  does not need to divide  $A_{k,k}$  and  $A_{k,k}$  could have non-zero entries above and below.

**definition** *Smith-normal-form-upt-k*  $A\ k =$   
 $($   
 $(\forall\ a\ b. to\ nat\ a = to\ nat\ b \wedge to\ nat\ a + 1 < k \wedge to\ nat\ b + 1 < k \longrightarrow A\ \$\ a\ \$\ b\ dvd\ A\ \$\ (a+1)\ \$\ (b+1))$   
 $\wedge\ isDiagonal\text{-}upt\text{-}k\ A\ k$   
 $)$

**definition** *Smith-normal-form*  $A =$

(  
   $(\forall a b. \text{to-nat } a = \text{to-nat } b \wedge \text{to-nat } a + 1 < \text{nrows } A \wedge \text{to-nat } b + 1 < \text{ncols } A \longrightarrow A \$ a \$ b \text{ dvd } A \$ (a+1) \$ (b+1))$   
   $\wedge \text{isDiagonal } A$   
)

## 1.2 Basic properties

**lemma** *Smith-normal-form-min*:

*Smith-normal-form*  $A = \text{Smith-normal-form-upt-k } A \text{ (min (nrows } A) \text{ (ncols } A))$

**unfolding** *Smith-normal-form-def* *Smith-normal-form-upt-k-def* *nrows-def* *ncols-def*

**unfolding** *isDiagonal-upt-k-def* *isDiagonal-def*

**by** (*auto*, *smt* (*verit*) *Suc-le-eq* *le-trans* *less-le* *min.boundedI* *not-less-eq-eq* *suc-not-zero*

*to-nat-less-card* *to-nat-plus-one-less-card'*)

**lemma** *Smith-normal-form-upt-k-0[simp]*: *Smith-normal-form-upt-k*  $A \ 0$

**unfolding** *Smith-normal-form-upt-k-def*

**unfolding** *isDiagonal-upt-k-def* *isDiagonal-def*

**by** *auto*

**lemma** *Smith-normal-form-upt-k-intro*:

**assumes**  $(\bigwedge a b. \text{to-nat } a = \text{to-nat } b \wedge \text{to-nat } a + 1 < k \wedge \text{to-nat } b + 1 < k \implies A \$ a \$ b \text{ dvd } A \$ (a+1) \$ (b+1))$

**and**  $(\bigwedge a b. (\text{to-nat } a \neq \text{to-nat } b \wedge (\text{to-nat } a < k \vee (\text{to-nat } b < k))) \implies A \$ a \$ b = 0)$

**shows** *Smith-normal-form-upt-k*  $A \ k$

**unfolding** *Smith-normal-form-upt-k-def*

**unfolding** *isDiagonal-upt-k-def* *isDiagonal-def* **using** *assms* **by** *simp*

**lemma** *Smith-normal-form-upt-k-intro-alt*:

**assumes**  $(\bigwedge a b. \text{to-nat } a = \text{to-nat } b \wedge \text{to-nat } a + 1 < k \wedge \text{to-nat } b + 1 < k \implies A \$ a \$ b \text{ dvd } A \$ (a+1) \$ (b+1))$

**and** *isDiagonal-upt-k*  $A \ k$

**shows** *Smith-normal-form-upt-k*  $A \ k$

**using** *assms*

**unfolding** *Smith-normal-form-upt-k-def* **by** *auto*

**lemma** *Smith-normal-form-upt-k-condition1*:

**fixes**  $A::'a::\{\text{bezout-ring}\}^{\wedge} \text{cols}::\text{mod-type}^{\wedge} \text{rows}::\text{mod-type}$

**assumes** *Smith-normal-form-upt-k*  $A \ k$

**and**  $\text{to-nat } a = \text{to-nat } b$  **and**  $\text{to-nat } a + 1 < k$  **and**  $\text{to-nat } b + 1 < k$

**shows**  $A \$ a \$ b \text{ dvd } A \$ (a+1) \$ (b+1)$

**using** *assms* **unfolding** *Smith-normal-form-upt-k-def* **by** *auto*

**lemma** *Smith-normal-form-upt-k-condition2*:  
**fixes**  $A::'a::\{\text{bezout-ring}\}^{\wedge}\text{cols}::\text{mod-type}^{\wedge}\text{rows}::\text{mod-type}$   
**assumes** *Smith-normal-form-upt-k*  $A$   $k$   
**and**  $\text{to-nat } a \neq \text{to-nat } b$  **and**  $(\text{to-nat } a < k \vee \text{to-nat } b < k)$   
**shows**  $((A \$ a) \$ b) = 0$   
**using** *assms* **unfolding** *Smith-normal-form-upt-k-def*  
**unfolding** *isDiagonal-upt-k-def isDiagonal-def* **by** *auto*

**lemma** *Smith-normal-form-upt-k1-intro*:  
**fixes**  $A::'a::\{\text{bezout-ring}\}^{\wedge}\text{cols}::\text{mod-type}^{\wedge}\text{rows}::\text{mod-type}$   
**assumes**  $s$ : *Smith-normal-form-upt-k*  $A$   $k$   
**and**  $\text{cond1}$ :  $A \$ \text{from-nat } (k - 1) \$ \text{from-nat } (k-1) \text{ dvd } A \$ (\text{from-nat } k) \$$   
 $(\text{from-nat } k)$   
**and**  $\text{cond2a}$ :  $\forall a. \text{to-nat } a > k \longrightarrow A \$ a \$ \text{from-nat } k = 0$   
**and**  $\text{cond2b}$ :  $\forall b. \text{to-nat } b > k \longrightarrow A \$ \text{from-nat } k \$ b = 0$   
**shows** *Smith-normal-form-upt-k*  $A$   $(\text{Suc } k)$   
**proof** (*rule Smith-normal-form-upt-k-intro*)  
**fix**  $a::\text{rows}$  **and**  $b::\text{cols}$   
**assume**  $a$ :  $\text{to-nat } a \neq \text{to-nat } b \wedge (\text{to-nat } a < \text{Suc } k \vee \text{to-nat } b < \text{Suc } k)$   
**show**  $A \$ a \$ b = 0$   
**by** (*metis Smith-normal-form-upt-k-condition2 a*  
*assms(1) cond2a cond2b from-nat-to-nat-id less-SucE nat-neq-iff*)

**next**  
**fix**  $a::\text{rows}$  **and**  $b::\text{cols}$   
**assume**  $a$ :  $\text{to-nat } a = \text{to-nat } b \wedge \text{to-nat } a + 1 < \text{Suc } k \wedge \text{to-nat } b + 1 < \text{Suc } k$   
**show**  $A \$ a \$ b \text{ dvd } A \$ (a + 1) \$ (b + 1)$   
**by** (*metis (mono-tags, lifting) Smith-normal-form-upt-k-condition1 a add-diff-cancel-right'*  
*cond1*  
*from-nat-suc from-nat-to-nat-id less-SucE s*)

**qed**

**lemma** *Smith-normal-form-upt-k1-intro-diagonal*:  
**fixes**  $A::'a::\{\text{bezout-ring}\}^{\wedge}\text{cols}::\text{mod-type}^{\wedge}\text{rows}::\text{mod-type}$   
**assumes**  $s$ : *Smith-normal-form-upt-k*  $A$   $k$   
**and**  $d$ : *isDiagonal*  $A$   
**and**  $\text{cond1}$ :  $A \$ \text{from-nat } (k - 1) \$ \text{from-nat } (k-1) \text{ dvd } A \$ (\text{from-nat } k) \$$   
 $(\text{from-nat } k)$   
**shows** *Smith-normal-form-upt-k*  $A$   $(\text{Suc } k)$   
**proof** (*rule Smith-normal-form-upt-k-intro*)  
**fix**  $a::\text{rows}$  **and**  $b::\text{cols}$   
**assume**  $a$ :  $\text{to-nat } a = \text{to-nat } b \wedge \text{to-nat } a + 1 < \text{Suc } k \wedge \text{to-nat } b + 1 < \text{Suc } k$   
**show**  $A \$ a \$ b \text{ dvd } A \$ (a + 1) \$ (b + 1)$   
**by** (*metis (mono-tags, lifting) Smith-normal-form-upt-k-condition1 a*  
*add-diff-cancel-right' cond1 from-nat-suc from-nat-to-nat-id less-SucE s*)

**next**  
**show**  $\bigwedge a b. \text{to-nat } a \neq \text{to-nat } b \wedge (\text{to-nat } a < \text{Suc } k \vee \text{to-nat } b < \text{Suc } k) \implies A$   
 $\$ a \$ b = 0$

```
    using d isDiagonal-def by blast
  qed
```

```
end
```

## 2 Algorithm to transform a diagonal matrix into its Smith normal form

```
theory Diagonal-To-Smith
  imports Hermite.Hermite
         HOL-Types-To-Sets.Types-To-Sets
         Smith-Normal-Form
begin
```

```
lemma invertible-mat-1: invertible (mat (1::'a::comm-ring-1))
  unfolding invertible-iff-is-unit by simp
```

### 2.1 Implementation of the algorithm

```
type-synonym 'a bezout = 'a  $\Rightarrow$  'a  $\Rightarrow$  'a  $\times$  'a  $\times$  'a  $\times$  'a
```

```
hide-const Countable.from-nat
hide-const Countable.to-nat
```

The algorithm is based on the one presented by Bradley in his article entitled “Algorithms for Hermite and Smith normal matrices and linear diophantine equations”. Some improvements have been introduced to get a general version for any matrix (including non-square and singular ones).

I also introduced another improvement: the element in the position  $j$  does not need to be checked each time, since the element  $A_{ii}$  will already divide  $A_{jj}$  (where  $j \leq k$ ). The gcd will be placed in  $A_{ii}$ .

This function transforms the element  $A_{jj}$  in order to be divisible by  $A_{ii}$  (and it changes  $A_{ii}$  as well).

The use of *from-nat* and *to-nat* is mandatory since the same index  $i$  cannot be used for both rows and columns at the same time, since they could have different type, concretely, when the matrix is rectangular.

The following definition is valid, but since execution requires the trick of converting all operations in terms of rows, then we would be recalculating the Bézout coefficients each time.

Thus, the definition is parameterized by the necessary elements instead of the operation, to avoid recalculations.

**definition** *diagonal-step*  $A\ i\ j\ d\ v =$   
 $(\chi\ a\ b.\ \text{if } a = \text{from-nat } i \wedge b = \text{from-nat } i \text{ then } d \text{ else}$   
 $\text{if } a = \text{from-nat } j \wedge b = \text{from-nat } j$   
 $\text{then } v * (A \$ (\text{from-nat } j) \$ (\text{from-nat } j)) \text{ else } A \$ a \$ b)$

**fun** *diagonal-to-Smith-i* ::  
 $\text{nat list} \Rightarrow 'a::\{\text{bezout-ring}\}^{\wedge}\text{cols}::\text{mod-type}^{\wedge}\text{rows}::\text{mod-type} \Rightarrow \text{nat} \Rightarrow ('a\ \text{bezout})$   
 $\Rightarrow 'a^{\wedge}\text{cols}::\text{mod-type}^{\wedge}\text{rows}::\text{mod-type}$   
**where**  
 $\text{diagonal-to-Smith-i } []\ A\ i\ \text{bezout} = A\ |$   
 $\text{diagonal-to-Smith-i } (j\#\text{xs})\ A\ i\ \text{bezout} = ($   
 $\text{if } A \$ (\text{from-nat } i) \$ (\text{from-nat } i)\ \text{dvd } A \$ (\text{from-nat } j) \$ (\text{from-nat } j)$   
 $\text{then } \text{diagonal-to-Smith-i } \text{xs}\ A\ i\ \text{bezout}$   
 $\text{else let } (p, q, u, v, d) = \text{bezout } (A \$ \text{from-nat } i \$ \text{from-nat } i)\ (A \$ \text{from-nat } j \$$   
 $\text{from-nat } j);$   
 $A' = \text{diagonal-step } A\ i\ j\ d\ v$   
 $\text{in } \text{diagonal-to-Smith-i } \text{xs}\ A'\ i\ \text{bezout}$   
 $)$

**definition** *Diagonal-to-Smith-row-i*  $A\ i\ \text{bezout}$   
 $= \text{diagonal-to-Smith-i } [i+1..<\text{min } (\text{nrows } A)\ (\text{ncols } A)]\ A\ i\ \text{bezout}$

**fun** *diagonal-to-Smith-aux* ::  $'a::\{\text{bezout-ring}\}^{\wedge}\text{cols}::\text{mod-type}^{\wedge}\text{rows}::\text{mod-type}$   
 $\Rightarrow \text{nat list} \Rightarrow ('a\ \text{bezout}) \Rightarrow 'a^{\wedge}\text{cols}::\text{mod-type}^{\wedge}\text{rows}::\text{mod-type}$   
**where**  
 $\text{diagonal-to-Smith-aux } A\ []\ \text{bezout} = A\ |$   
 $\text{diagonal-to-Smith-aux } A\ (i\#\text{xs})\ \text{bezout}$   
 $= \text{diagonal-to-Smith-aux } (\text{Diagonal-to-Smith-row-i } A\ i\ \text{bezout})\ \text{xs}\ \text{bezout}$

The minimum arises to include the case of non-square matrices (we do not demand the input diagonal matrix to be square, just have zeros in non-diagonal entries).

This iteration does not need to be performed until the last element of the diagonal, because in the second-to-last step the matrix will be already in Smith normal form.

**definition** *diagonal-to-Smith*  $A\ \text{bezout}$   
 $= \text{diagonal-to-Smith-aux } A\ [0..<\text{min } (\text{nrows } A)\ (\text{ncols } A) - 1]\ \text{bezout}$

## 2.2 Code equations to get an executable version

**definition** *diagonal-step-row*

**where** *diagonal-step-row*  $A\ i\ j\ c\ v\ a = \text{vec-lambda } (\%b.\ \text{if } a = \text{from-nat } i \wedge b =$   
 $\text{from-nat } i \text{ then } c \text{ else}$   
 $\text{if } a = \text{from-nat } j \wedge b = \text{from-nat } j$   
 $\text{then } v * (A \$ (\text{from-nat } j) \$ (\text{from-nat } j)) \text{ else } A \$ a \$ b)$

**lemma** *diagonal-step-code* [code abstract]:



*vec-nth* (*diagonal-step-row* *A i j c v a*) = (%*b*. if *a* = *from-nat i*  $\wedge$  *b* = *from-nat i* then *c* else

*if a* = *from-nat j*  $\wedge$  *b* = *from-nat j*

then *v* \* (*A* \$ (*from-nat j*) \$ (*from-nat j*)) else *A* \$ *a* \$ *b*)

**unfolding** *diagonal-step-row-def* **by** *auto*

**lemma** *diagonal-step-code-nth* [*code abstract*]: *vec-nth* (*diagonal-step* *A i j c v*)  
= *diagonal-step-row* *A i j c v*

**unfolding** *diagonal-step-def* **unfolding** *diagonal-step-row-def* [*abs-def*]

**by** *auto*

Code equation to avoid recalculations when computing the Bezout coefficients.

**lemma** *euclid-ext2-code* [*code*]:

*euclid-ext2 a b* = (*let* ((*p,q*),*d*) = *euclid-ext a b* in (*p,q*, - *b div d*, *a div d*, *d*))

**unfolding** *euclid-ext2-def* *split-beta* *Let-def*

**by** *auto*

## 2.3 Examples of execution

**value** *let A* = *list-of-list-to-matrix* [[*12,0,0::int*],[*0,6,0::int*],[*0,0,2::int*]]::*int*<sup>3</sup><sup>3</sup>  
in *matrix-to-list-of-list* (*diagonal-to-Smith* *A euclid-ext2*)

Example obtained from: <https://math.stackexchange.com/questions/77063/how-do-i-get-this-matrix-in-smith-normal-form-and-is-smith-normal-form-unique>

**value** *let A* = *list-of-list-to-matrix*

[  
  [[-*3,1*:],*0,0,0*],  
  [*0*,[*1,1*:],*0,0*],  
  [*0,0*,[*1,1*:],*0*],  
  [*0,0,0*,[*1,1*:] ]::*rat poly*<sup>4</sup><sup>4</sup>

in *matrix-to-list-of-list* (*diagonal-to-Smith* *A euclid-ext2*)

Polynomial matrix

**value** *let A* = *list-of-list-to-matrix*

[  
  [[-*3,1*:],*0,0,0*],  
  [*0*,[*1,1*:],*0,0*],  
  [*0,0*,[*1,1*:],*0*],  
  [*0,0,0*,[*1,1*:] ],  
  [*0,0,0,0*] ]::*rat poly*<sup>4</sup><sup>5</sup>

in *matrix-to-list-of-list* (*diagonal-to-Smith* *A euclid-ext2*)

## 2.4 Soundness of the algorithm

**lemma** *nrows-diagonal-step* [*simp*]: *nrows* (*diagonal-step* *A i j c v*) = *nrows* *A*  
**by** (*simp add*: *nrows-def*)

**lemma** *ncols-diagonal-step* [*simp*]: *ncols* (*diagonal-step* *A i j c v*) = *ncols* *A*

by (simp add: ncols-def)

**context**

fixes bezout::'a::{bezout-ring}  $\Rightarrow$  'a  $\Rightarrow$  'a  $\times$  'a  $\times$  'a  $\times$  'a  $\times$  'a

assumes ib: is-bezout-ext bezout

**begin**

**lemma** split-beta-bezout: bezout a b =

(fst (bezout a b),

fst (snd (bezout a b)),

fst (snd (snd (bezout a b))),

fst (snd (snd (snd (bezout a b)))))

snd (snd (snd (snd (bezout a b)))))) **unfolding** split-beta **by** (auto simp add: split-beta)

The following lemma shows that *diagonal-to-Smith-i* preserves the previous element. We use the assumption  $to\_nat\ a = to\_nat\ b$  in order to ensure that we are treating with a diagonal entry. Since the matrix could be rectangular, the types of a and b can be different, and thus we cannot write either  $a = b$  or  $A\ \$\ a\ \$\ b$ .

**lemma** diagonal-to-Smith-i-preserves-previous-diagonal:

fixes A::'a::{bezout-ring}  $\wedge$  b::mod-type  $\wedge$  c::mod-type

assumes i-min:  $i < \min\ (nrows\ A)\ (ncols\ A)$

and to-nat a  $\notin$  set xs **and** to-nat a = to-nat b

and to-nat a  $\neq$  i

and elements-xs-range:  $\forall x. x \in set\ xs \longrightarrow x < \min\ (nrows\ A)\ (ncols\ A)$

shows (diagonal-to-Smith-i xs A i bezout)  $\$ a\ \$ b = A\ \$ a\ \$ b$

using assms

**proof** (induct xs A i bezout rule: diagonal-to-Smith-i.induct)

case (1 A i bezout)

then show ?case **by** auto

**next**

case (2 j xs A i bezout)

let ?Aii = A  $\$$  from-nat i  $\$$  from-nat i

let ?Ajj = A  $\$$  from-nat j  $\$$  from-nat j

let ?p=case bezout (A  $\$$  from-nat i  $\$$  from-nat i) (A  $\$$  from-nat j  $\$$  from-nat j)

of (p,q,u,v,d)  $\Rightarrow$  p

let ?q=case bezout (A  $\$$  from-nat i  $\$$  from-nat i) (A  $\$$  from-nat j  $\$$  from-nat j)

of (p,q,u,v,d)  $\Rightarrow$  q

let ?u=case bezout (A  $\$$  from-nat i  $\$$  from-nat i) (A  $\$$  from-nat j  $\$$  from-nat j)

of (p,q,u,v,d)  $\Rightarrow$  u

let ?v=case bezout (A  $\$$  from-nat i  $\$$  from-nat i) (A  $\$$  from-nat j  $\$$  from-nat j)

of (p,q,u,v,d)  $\Rightarrow$  v

let ?d=case bezout (A  $\$$  from-nat i  $\$$  from-nat i) (A  $\$$  from-nat j  $\$$  from-nat j)

of (p,q,u,v,d)  $\Rightarrow$  d

let ?A'=diagonal-step A i j ?d ?v

have pqvwd: (?p, ?q, ?u, ?v, ?d) = bezout (A  $\$$  from-nat i  $\$$  from-nat i) (A  $\$$  from-nat j  $\$$  from-nat j)

```

    by (simp add: split-beta)
show ?case
proof (cases ?Aii dvd ?Ajj)
  case True
  then show ?thesis
    using 2.hyps 2.premis by auto
next
  case False
  note i-min = 2(3)
  note hyp = 2(2)
  note i-notin = 2(4)
  note a-eq-b = 2.premis(3)
  note elements-xs = 2(7)
  note a-not-i = 2(6)
  have a-not-j: a ≠ from-nat j
    by (metis elements-xs i-notin list.set-intros(1) min-less-iff-conj nrows-def
to-nat-from-nat-id)
  have diagonal-to-Smith-i (j # xs) A i bezout = diagonal-to-Smith-i xs ?A' i
bezout
    using False by (auto simp add: split-beta)
  also have ... $ a $ b = ?A' $ a $ b
    by (rule hyp[OF False], insert i-notin i-min a-eq-b a-not-i pquvd elements-xs,
auto)
  also have ... = A $ a $ b
    unfolding diagonal-step-def
    using a-not-j a-not-i
    by (smt (verit, del-insts) i-min min-less-iff-conj nrows-def
to-nat-from-nat-id vec-lambda-unique)
  finally show ?thesis .
qed
qed

```

```

lemma diagonal-step-dvd1[simp]:
  fixes A::'a::{bezout-ring} ^b::mod-type ^c::mod-type and j i
  defines v==case bezout (A $ from-nat i $ from-nat i) (A $ from-nat j $ from-nat
j) of (p,q,u,v,d) ⇒ v
    and d==case bezout (A $ from-nat i $ from-nat i) (A $ from-nat j $ from-nat
j) of (p,q,u,v,d) ⇒ d
  shows diagonal-step A i j d v $ from-nat i $ from-nat i dvd A $ from-nat i $
from-nat i
    using ib unfolding is-bezout-ext-def diagonal-step-def v-def d-def
    by (auto simp add: split-beta)

```

```

lemma diagonal-step-dvd2[simp]:
  fixes A::'a::{bezout-ring} ^b::mod-type ^c::mod-type and j i
  defines v==case bezout (A $ from-nat i $ from-nat i) (A $ from-nat j $ from-nat
j) of (p,q,u,v,d) ⇒ v
    and d==case bezout (A $ from-nat i $ from-nat i) (A $ from-nat j $ from-nat
j) of (p,q,u,v,d) ⇒ d

```

**shows** *diagonal-step*  $A\ i\ j\ d\ v\ \$\ \text{from-nat}\ i\ \$\ \text{from-nat}\ i\ \text{dvd}\ A\ \$\ \text{from-nat}\ j\ \$\ \text{from-nat}\ j$   
**using** *ib unfolding is-bezout-ext-def diagonal-step-def v-def d-def*  
**by** (*auto simp add: split-beta*)

**end**

Once the step is carried out, the new element  $A'_{ii}$  will divide the element  $A_{ii}$

**lemma** *diagonal-to-Smith-i-dvd-ii:*

**fixes**  $A::\text{'a}::\{\text{bezout-ring}\}^{\wedge}b::\text{mod-type}^{\wedge}c::\text{mod-type}$

**assumes** *ib: is-bezout-ext bezout*

**shows** *diagonal-to-Smith-i xs A i bezout*  $\ \$\ \text{from-nat}\ i\ \$\ \text{from-nat}\ i\ \text{dvd}\ A\ \$\ \text{from-nat}\ i\ \$\ \text{from-nat}\ i$

**using** *ib*

**proof** (*induct xs A i bezout rule: diagonal-to-Smith-i.induct*)

**case** ( $1\ A\ i\ \text{bezout}$ )

**then show** *?case* **by** *auto*

**next**

**case** ( $2\ j\ xs\ A\ i\ \text{bezout}$ )

**let**  $?A_{ii} = A\ \$\ \text{from-nat}\ i\ \$\ \text{from-nat}\ i$

**let**  $?A_{jj} = A\ \$\ \text{from-nat}\ j\ \$\ \text{from-nat}\ j$

**let**  $?p = \text{case}\ \text{bezout}\ (A\ \$\ \text{from-nat}\ i\ \$\ \text{from-nat}\ i)\ (A\ \$\ \text{from-nat}\ j\ \$\ \text{from-nat}\ j)$   
*of*  $(p,q,u,v,d) \Rightarrow p$

**let**  $?q = \text{case}\ \text{bezout}\ (A\ \$\ \text{from-nat}\ i\ \$\ \text{from-nat}\ i)\ (A\ \$\ \text{from-nat}\ j\ \$\ \text{from-nat}\ j)$   
*of*  $(p,q,u,v,d) \Rightarrow q$

**let**  $?u = \text{case}\ \text{bezout}\ (A\ \$\ \text{from-nat}\ i\ \$\ \text{from-nat}\ i)\ (A\ \$\ \text{from-nat}\ j\ \$\ \text{from-nat}\ j)$   
*of*  $(p,q,u,v,d) \Rightarrow u$

**let**  $?v = \text{case}\ \text{bezout}\ (A\ \$\ \text{from-nat}\ i\ \$\ \text{from-nat}\ i)\ (A\ \$\ \text{from-nat}\ j\ \$\ \text{from-nat}\ j)$   
*of*  $(p,q,u,v,d) \Rightarrow v$

**let**  $?d = \text{case}\ \text{bezout}\ (A\ \$\ \text{from-nat}\ i\ \$\ \text{from-nat}\ i)\ (A\ \$\ \text{from-nat}\ j\ \$\ \text{from-nat}\ j)$   
*of*  $(p,q,u,v,d) \Rightarrow d$

**let**  $?A' = \text{diagonal-step}\ A\ i\ j\ ?d\ ?v$

**have**  $pqvd: (?p, ?q, ?u, ?v, ?d) = \text{bezout}\ (A\ \$\ \text{from-nat}\ i\ \$\ \text{from-nat}\ i)\ (A\ \$\ \text{from-nat}\ j\ \$\ \text{from-nat}\ j)$

**by** (*simp add: split-beta*)

**note**  $ib = 2.\text{prems}(1)$

**show** *?case*

**proof** (*cases ?A<sub>ii</sub> dvd ?A<sub>jj</sub>*)

**case** *True*

**then show** *?thesis*

**using**  $2.\text{hyps}(1)\ 2.\text{prems}$  **by** *auto*

**next**

**case** *False*

**note**  $hyp = 2.\text{hyps}(2)$

**have** *diagonal-to-Smith-i (j # xs) A i bezout = diagonal-to-Smith-i xs ?A' i bezout*

**using** *False* **by** (*auto simp add: split-beta*)

**also have**  $\dots\ \$\ \text{from-nat}\ i\ \$\ \text{from-nat}\ i\ \text{dvd}\ ?A'\ \$\ \text{from-nat}\ i\ \$\ \text{from-nat}\ i$

```

    by (rule hyp[OF False], insert pqvd ib, auto)
  also have ... dvd A $ from-nat i $ from-nat i
    unfolding diagonal-step-def using ib unfolding is-bezout-ext-def
    by (auto simp add: split-beta)
  finally show ?thesis .
qed
qed

```

Once the step is carried out, the new element  $A'_{ii}$  divides the rest of elements of the diagonal. This proof requires commutativity (already included in the type restriction *bezout-ring*).

**lemma** *diagonal-to-Smith-i-dvd-jj*:

**fixes**  $A::\{bezout-ring\}^{\wedge}b::mod\text{-}type^{\wedge}c::mod\text{-}type$

**assumes** *ib*: *is-bezout-ext bezout*

**and** *i-min*:  $i < \min(\text{nrows } A) (\text{ncols } A)$

**and** *elements-xs-range*:  $\forall x. x \in \text{set } xs \longrightarrow x < \min(\text{nrows } A) (\text{ncols } A)$

**and** *to-nat a*  $\in \text{set } xs$

**and** *to-nat a* = *to-nat b*

**and** *to-nat a*  $\neq i$

**and** *distinct xs*

**shows** (*diagonal-to-Smith-i xs A i bezout*) \$ (*from-nat i*) \$ (*from-nat i*)  
 dvd (*diagonal-to-Smith-i xs A i bezout*) \$ *a* \$ *b*

**using** *assms*

**proof** (*induct xs A i bezout rule: diagonal-to-Smith-i.induct*)

**case** (*1 A i*)

**then show** *?case* **by** *auto*

**next**

**case** (*2 j xs A i bezout*)

**let** *?Aii* =  $A \$ \text{from-nat } i \$ \text{from-nat } i$

**let** *?Ajj* =  $A \$ \text{from-nat } j \$ \text{from-nat } j$

**let** *?p=case bezout* ( $A \$ \text{from-nat } i \$ \text{from-nat } i$ ) ( $A \$ \text{from-nat } j \$ \text{from-nat } j$ )

*of* ( $p,q,u,v,d$ )  $\Rightarrow p$

**let** *?q=case bezout* ( $A \$ \text{from-nat } i \$ \text{from-nat } i$ ) ( $A \$ \text{from-nat } j \$ \text{from-nat } j$ )

*of* ( $p,q,u,v,d$ )  $\Rightarrow q$

**let** *?u=case bezout* ( $A \$ \text{from-nat } i \$ \text{from-nat } i$ ) ( $A \$ \text{from-nat } j \$ \text{from-nat } j$ )

*of* ( $p,q,u,v,d$ )  $\Rightarrow u$

**let** *?v=case bezout* ( $A \$ \text{from-nat } i \$ \text{from-nat } i$ ) ( $A \$ \text{from-nat } j \$ \text{from-nat } j$ )

*of* ( $p,q,u,v,d$ )  $\Rightarrow v$

**let** *?d=case bezout* ( $A \$ \text{from-nat } i \$ \text{from-nat } i$ ) ( $A \$ \text{from-nat } j \$ \text{from-nat } j$ )

*of* ( $p,q,u,v,d$ )  $\Rightarrow d$

**let** *?A'* = *diagonal-step A i j ?d ?v*

**have** *pqvvd*: (*?p*, *?q*, *?u*, *?v*, *?d*) = *bezout* ( $A \$ \text{from-nat } i \$ \text{from-nat } i$ ) ( $A \$ \text{from-nat } j \$ \text{from-nat } j$ )

**by** (*simp add: split-beta*)

**note** *ib* = *2.prem1*

**note** *to-nat-a-not-i* = *2(8)*

**note** *i-min* = *2(4)*

**note** *elements-xs* = *2.prem3*

**note** *a-eq-b* = *2.prem5*

```

note  $a\text{-in-}j\text{-xs} = 2(6)$ 
note  $\text{distinct} = 2(9)$ 
show  $?case$ 
proof ( $cases\ ?Aii\ dvd\ ?Ajj$ )
  case  $True$  note  $Aii\text{-}dvd\text{-}Ajj = True$ 
  show  $?thesis$ 
  proof ( $cases\ \text{to-nat}\ a = j$ )
    case  $True$ 
    have  $a: a = (\text{from-nat}\ j::'c)$  using  $True$  by  $auto$ 
    have  $b: b = (\text{from-nat}\ j::'b)$ 
      using  $True\ a\text{-eq-}b$  by  $auto$ 
    have  $\text{diagonal-to-Smith-}i\ (j\ \#\ xs)\ A\ i\ \text{bezout} = \text{diagonal-to-Smith-}i\ xs\ A\ i$ 
     $\text{bezout}$ 
      using  $Aii\text{-}dvd\text{-}Ajj$  by  $auto$ 
    also have  $\dots\ \$\ \text{from-nat}\ j\ \$\ \text{from-nat}\ j = A\ \$\ \text{from-nat}\ j\ \$\ \text{from-nat}\ j$ 
      proof ( $rule\ \text{diagonal-to-Smith-}i\text{-preserves-previous-diagonal}[OF\ \text{ib}\ i\text{-min}]$ )

    show  $\text{to-nat}\ (\text{from-nat}\ j::'c) \notin \text{set}\ xs$  using  $True\ a\text{-in-}j\text{-xs}\ \text{distinct}$  by  $auto$ 
    show  $\text{to-nat}\ (\text{from-nat}\ j::'c) = \text{to-nat}\ (\text{from-nat}\ j::'b)$ 
      by ( $\text{metis}\ True\ a\text{-eq-}b\ \text{from-nat-to-nat-id}$ )
    show  $\text{to-nat}\ (\text{from-nat}\ j::'c) \neq i$ 
      using  $True\ \text{to-nat-a-not-}i$  by  $auto$ 
    show  $\forall x. x \in \text{set}\ xs \longrightarrow x < \min\ (\text{nrows}\ A)\ (\text{ncols}\ A)$  using  $\text{elements-x}$ 
  by  $auto$ 
  qed
  finally have  $\text{diagonal-to-Smith-}i\ (j\ \#\ xs)\ A\ i\ \text{bezout}\ \$\ \text{from-nat}\ j\ \$\ \text{from-nat}\ j$ 
   $j$ 
     $= A\ \$\ \text{from-nat}\ j\ \$\ \text{from-nat}\ j$  .
  hence  $\text{diagonal-to-Smith-}i\ (j\ \#\ xs)\ A\ i\ \text{bezout}\ \$\ a\ \$\ b = ?Ajj$  unfolding  $a\ b$ 
  .
  moreover have  $\text{diagonal-to-Smith-}i\ (j\ \#\ xs)\ A\ i\ \text{bezout}\ \$\ \text{from-nat}\ i\ \$\ \text{from-nat}\ i\ dvd\ ?Aii$ 
    by ( $rule\ \text{diagonal-to-Smith-}i\text{-dvd-ii}[OF\ \text{ib}]$ )
  ultimately show  $?thesis$  using  $Aii\text{-}dvd\text{-}Ajj\ \text{dvd-trans}$  by  $auto$ 
next
  case  $False$ 
  have  $a\text{-in-x}$ :  $\text{to-nat}\ a \in \text{set}\ xs$  using  $False$  using  $2.\text{prems}(4)$  by  $auto$ 
  have  $\text{diagonal-to-Smith-}i\ (j\ \#\ xs)\ A\ i\ \text{bezout} = \text{diagonal-to-Smith-}i\ xs\ A\ i$ 
   $\text{bezout}$ 
    using  $True$  by  $auto$ 
  also have  $\dots\ \$\ (\text{from-nat}\ i)\ \$\ (\text{from-nat}\ i)\ dvd\ \text{diagonal-to-Smith-}i\ xs\ A\ i$ 
   $\text{bezout}\ \$\ a\ \$\ b$ 
    by ( $rule\ 2.\text{hyps}(1)[OF\ True\ \text{ib}\ i\text{-min}\ -\ a\text{-in-x}\ a\text{-eq-}b\ \text{to-nat-a-not-}i]$ )
    ( $\text{insert}\ \text{elements-x}\ \text{distinct},\ auto$ )
  finally show  $?thesis$  .
qed
next
  case  $False$  note  $Aii\text{-not-dvd-}Ajj = False$ 
  show  $?thesis$ 

```

```

proof (cases to-nat a ∈ set xs)
  case True note a-in-xs = True
  have diagonal-to-Smith-i (j # xs) A i bezout = diagonal-to-Smith-i xs ?A' i
  bezout
  using False by (auto simp add: split-beta)
  also have ... $ from-nat i $ from-nat i dvd diagonal-to-Smith-i xs ?A' i bezout
  $ a $ b
  by (rule 2.hyps(2)[OF False - - - - - a-in-xs a-eq-b to-nat-a-not-i ])
  (insert elements-xs distinct i-min ib pqwd, auto simp add: nrows-def
  ncols-def)
  finally show ?thesis .
next
  case False
  have to-nat-a-eq-j: to-nat a = j
  using False a-in-j-xs by auto
  have a: a = (from-nat j::'c) using to-nat-a-eq-j by auto
  have b: b = (from-nat j::'b) using to-nat-a-eq-j a-eq-b by auto
  have d-eq: diagonal-to-Smith-i (j # xs) A i bezout = diagonal-to-Smith-i xs
  ?A' i bezout
  using Aii-not-dvd-Ajj by (simp add: split-beta)
  also have ... $ a $ b = ?A' $ a $ b
  by (rule diagonal-to-Smith-i-preserves-previous-diagonal[OF ib - False a-eq-b
  to-nat-a-not-i])
  (insert i-min elements-xs ib, auto)
  finally have diagonal-to-Smith-i (j # xs) A i bezout $ a $ b = ?A' $ a $ b .
  moreover have diagonal-to-Smith-i (j # xs) A i bezout $ from-nat i $
  from-nat i
  dvd ?A' $ from-nat i $ from-nat i
  using d-eq diagonal-to-Smith-i-dvd-ii[OF ib] by simp
  moreover have ?A' $ from-nat i $ from-nat i dvd ?A' $ from-nat j $ from-nat
  j
  unfolding diagonal-step-def using ib unfolding is-bezout-ext-def split-beta
  by (auto, meson dvd-mult)+
  ultimately show ?thesis using dvd-trans a b by auto
qed
qed
qed

```

The step preserves everything that is not in the diagonal

**lemma** *diagonal-to-Smith-i-preserves-previous:*

```

fixes A::'a:: {bezout-ring} ^b::mod-type ^c::mod-type
assumes ib: is-bezout-ext bezout
  and i-min: i < min (nrows A) (ncols A)
  and a-not-b: to-nat a ≠ to-nat b
  and elements-xs-range: ∀ x. x ∈ set xs ⟶ x < min (nrows A) (ncols A)
shows (diagonal-to-Smith-i xs A i bezout) $ a $ b = A $ a $ b
using assms
proof (induct xs A i bezout rule: diagonal-to-Smith-i.induct)
case (1 A i)

```

```

then show ?case by auto
next
  case (2 j xs A i bezout)
  let ?Aii = A $ from-nat i $ from-nat i
  let ?Ajj = A $ from-nat j $ from-nat j
  let ?p=case bezout (A $ from-nat i $ from-nat i) (A $ from-nat j $ from-nat j)
  of (p,q,u,v,d) => p
  let ?q=case bezout (A $ from-nat i $ from-nat i) (A $ from-nat j $ from-nat j)
  of (p,q,u,v,d) => q
  let ?u=case bezout (A $ from-nat i $ from-nat i) (A $ from-nat j $ from-nat j)
  of (p,q,u,v,d) => u
  let ?v=case bezout (A $ from-nat i $ from-nat i) (A $ from-nat j $ from-nat j)
  of (p,q,u,v,d) => v
  let ?d=case bezout (A $ from-nat i $ from-nat i) (A $ from-nat j $ from-nat j)
  of (p,q,u,v,d) => d
  let ?A'=diagonal-step A i j ?d ?v
  have pqvud: (?p, ?q, ?u, ?v,?d) = bezout (A $ from-nat i $ from-nat i) (A $
  from-nat j $ from-nat j)
  by (simp add: split-beta)
  note ib = 2.prem(1)
  show ?case
  proof (cases ?Aii dvd ?Ajj)
    case True
    then show ?thesis
    using 2.hyps(1) 2.prem by auto
  next
  case False
  note hyp = 2.hyps(2)
  have a1: a = from-nat i → b ≠ from-nat i
  by (metis 2.prem a-not-b from-nat-not-eq min.strict-boundedE ncols-def
  nrows-def)
  have a2: a = from-nat j → b ≠ from-nat j
  by (metis 2.prem a-not-b list.set-intros(1) min-less-iff-conj
  ncols-def nrows-def to-nat-from-nat-id)
  have diagonal-to-Smith-i (j # xs) A i bezout = diagonal-to-Smith-i xs ?A' i
  bezout
  using False by (simp add: split-beta)
  also have ... $ a $ b = ?A' $ a $ b
  by (rule hyp[OF False], insert 2.prem ib pqvud, auto)
  also have ... = A $ a $ b unfolding diagonal-step-def using a1 a2 by auto
  finally show ?thesis .
qed
qed

```

**lemma** diagonal-step-preserves:

```

fixes A::'a::{times}^'b::mod-type^'c::mod-type
assumes ai: a ≠ i and aj: a ≠ j and a-min: a < min (nrows A) (ncols A)
and i-min: i < min (nrows A) (ncols A)

```



```

and j-min:  $j < \min (\text{nrows } A) (\text{ncols } A)$ 
shows diagonal-step  $A \ i \ j \ d \ v \ \$ \text{from-nat } a \ \$ \text{from-nat } b = A \ \$ \text{from-nat } a \ \$ \text{from-nat } b$ 
proof –
  have 1:  $(\text{from-nat } a :: 'c) \neq \text{from-nat } i$ 
    by (metis a-min ai from-nat-eq-imp-eq i-min min.strict-boundedE nrows-def)
  have 2:  $(\text{from-nat } a :: 'c) \neq \text{from-nat } j$ 
    by (metis a-min aj from-nat-eq-imp-eq j-min min.strict-boundedE nrows-def)
  show ?thesis
    using 1 2 unfolding diagonal-step-def by auto
qed

```

```

context GCD-ring
begin

```

```

lemma gcd-greatest:
  assumes is-gcd gcd' and  $c \text{ dvd } a$  and  $c \text{ dvd } b$ 
  shows  $c \text{ dvd } \text{gcd}' \ a \ b$ 
  using assms is-gcd-def by blast

```

**end**

This is a key lemma for the soundness of the algorithm.

```

lemma diagonal-to-Smith-i-dvd:
  fixes  $A :: 'a :: \{\text{bezout-ring}\}^{\wedge} b :: \text{mod-type}^{\wedge} c :: \text{mod-type}$ 
  assumes ib: is-bezout-ext bezout
  and i-min:  $i < \min (\text{nrows } A) (\text{ncols } A)$ 
  and elements-xs-range:  $\forall x. x \in \text{set } xs \longrightarrow x < \min (\text{nrows } A) (\text{ncols } A)$ 
  and  $\forall a \ b. \text{to-nat } a \in \text{insert } i (\text{set } xs) \wedge \text{to-nat } a = \text{to-nat } b \longrightarrow$ 
     $A \ \$ (\text{from-nat } c) \ \$ (\text{from-nat } c) \ \text{dvd } A \ \$ a \ \$ b$ 
  and  $c \notin (\text{set } xs)$  and  $c < \min (\text{nrows } A) (\text{ncols } A)$ 
  and distinct xs
  shows  $A \ \$ (\text{from-nat } c) \ \$ (\text{from-nat } c) \ \text{dvd}$ 
     $(\text{diagonal-to-Smith-}i \ xs \ A \ i \ \text{bezout}) \ \$ (\text{from-nat } i) \ \$ (\text{from-nat } i)$ 
  using assms
proof (induct xs A i bezout rule: diagonal-to-Smith-i.induct)
  case ( $1 \ A \ i$ )
  then show ?case
    by (auto simp add: ncols-def nrows-def to-nat-from-nat-id)
next
  case ( $2 \ j \ xs \ A \ i \ \text{bezout}$ )
  let  $?A_{ii} = A \ \$ \text{from-nat } i \ \$ \text{from-nat } i$ 
  let  $?A_{jj} = A \ \$ \text{from-nat } j \ \$ \text{from-nat } j$ 
  let  $?p = \text{case bezout } (A \ \$ \text{from-nat } i \ \$ \text{from-nat } i) \ (A \ \$ \text{from-nat } j \ \$ \text{from-nat } j)$ 
  of  $(p, q, u, v, d) \Rightarrow p$ 
  let  $?q = \text{case bezout } (A \ \$ \text{from-nat } i \ \$ \text{from-nat } i) \ (A \ \$ \text{from-nat } j \ \$ \text{from-nat } j)$ 
  of  $(p, q, u, v, d) \Rightarrow q$ 
  let  $?u = \text{case bezout } (A \ \$ \text{from-nat } i \ \$ \text{from-nat } i) \ (A \ \$ \text{from-nat } j \ \$ \text{from-nat } j)$ 
  of  $(p, q, u, v, d) \Rightarrow u$ 

```

```

let ?v=case bezout (A $ from-nat i $ from-nat i) (A $ from-nat j $ from-nat j)
of (p,q,u,v,d) => v
let ?d=case bezout (A $ from-nat i $ from-nat i) (A $ from-nat j $ from-nat j)
of (p,q,u,v,d) => d
let ?A'=diagonal-step A i j ?d ?v
have pqvd: (?p, ?q, ?u, ?v,?d) = bezout (A $ from-nat i $ from-nat i) (A $
from-nat j $ from-nat j)
by (simp add: split-beta)
note ib = 2.prem1
show ?case
proof (cases ?Aii dvd ?Ajj)
case True note Aii-dvd-Ajj = True
show ?thesis using True
using 2.hyps 2.prem1 by force
next
case False
let ?Acc = A $ from-nat c $ from-nat c
let ?D=diagonal-step A i j ?d ?v
note hyp = 2.hyps(2)
note dvd-condition = 2.prem4
note a-eq-b = 2.hyps
have 1: (from-nat c::'c) ≠ from-nat i
by (metis 2.prem1 False c insert-iff list.set-intros(1)
min.strict-boundedE ncols-def nrows-def to-nat-from-nat-id)
have 2: (from-nat c::'c) ≠ from-nat j
by (metis 2.prem1 c insertI1 list.simps(15) min-less-iff-conj nrows-def
to-nat-from-nat-id)
have ?D $ from-nat c $ from-nat c = ?Acc
unfolding diagonal-step-def using 1 2 by auto
have aux: ?D $ from-nat c $ from-nat c dvd ?D $ a $ b
if a-in-set: to-nat a ∈ insert i (set xs) and ab: to-nat a = to-nat b for a b
proof -
have Acc-dvd-Aii: ?Acc dvd ?Aii
by (metis 2.prem2 2.prem4 insert-iff min.strict-boundedE
ncols-def nrows-def to-nat-from-nat-id)
moreover have Acc-dvd-Ajj: ?Acc dvd ?Ajj
by (metis 2.prem3 2.prem4 insert-iff list.set-intros(1)
min-less-iff-conj ncols-def nrows-def to-nat-from-nat-id)
ultimately have Acc-dvd-gcd: ?Acc dvd ?d
by (metis (mono-tags, lifting) ib is-gcd-def is-gcd-is-bezout-ext)
show ?thesis
using 1 2 Acc-dvd-Ajj Acc-dvd-Aii Acc-dvd-gcd a-in-set ab dvd-condition
unfolding diagonal-step-def by auto
qed
have ?A' $ from-nat c $ from-nat c = A $ from-nat c $ from-nat c
unfolding diagonal-step-def using 1 2 by auto
moreover have ?A' $ from-nat c $ from-nat c
dvd diagonal-to-Smith-i xs ?A' i bezout $ from-nat i $ from-nat i
by (rule hyp[OF False - - - - - ib])

```

(insert nrows-def ncols-def 2.premis 2.hyps aux pquvd, auto)  
**ultimately show** ?thesis **using** False **by** (auto simp add: split-beta)  
**qed**  
**qed**

**lemma** *diagonal-to-Smith-i-dvd2*:  
**fixes**  $A::'a::\{\text{bezout-ring}\}^{\wedge}b::\text{mod-type}^{\wedge}c::\text{mod-type}$   
**assumes** *ib*: is-bezout-ext bezout  
**and** *i-min*:  $i < \min(\text{nrows } A) (\text{ncols } A)$   
**and** *elements-xs-range*:  $\forall x. x \in \text{set } xs \longrightarrow x < \min(\text{nrows } A) (\text{ncols } A)$   
**and** *dvd-condition*:  $\forall a b. \text{to-nat } a \in \text{insert } i (\text{set } xs) \wedge \text{to-nat } a = \text{to-nat } b \longrightarrow$   
 $A \$ (\text{from-nat } c) \$ (\text{from-nat } c) \text{dvd } A \$ a \$ b$   
**and** *c-notin*:  $c \notin (\text{set } xs)$   
**and** *c*:  $c < \min(\text{nrows } A) (\text{ncols } A)$   
**and** *distinct*: distinct *xs*  
**and** *ab*:  $\text{to-nat } a = \text{to-nat } b$   
**and** *a-in*:  $\text{to-nat } a \in \text{insert } i (\text{set } xs)$   
**shows**  $A \$ (\text{from-nat } c) \$ (\text{from-nat } c) \text{dvd } (\text{diagonal-to-Smith-i } xs \ A \ i \ \text{bezout}) \$$   
 $a \$ b$   
**proof** (cases  $a = \text{from-nat } i$ )  
**case** True  
**hence**  $b = \text{from-nat } i$   
**by** (metis *ab from-nat-to-nat-id i-min min-less-iff-conj nrows-def to-nat-from-nat-id*)  
**show** ?thesis **by** (unfold True *b*, rule *diagonal-to-Smith-i-dvd*, insert *assms*, auto)  
**next**  
**case** False  
**have** *ai*:  $\text{to-nat } a \neq i$  **using** False **by** auto  
**hence** *bi*:  $\text{to-nat } b \neq i$  **by** (simp add: *ab*)  
**have**  $A \$ (\text{from-nat } c) \$ (\text{from-nat } c) \text{dvd } (\text{diagonal-to-Smith-i } xs \ A \ i \ \text{bezout}) \$$   
 $\text{from-nat } i \$ \text{from-nat } i$   
**by** (rule *diagonal-to-Smith-i-dvd*, insert *assms*, auto)  
**also have** ...  $\text{dvd } (\text{diagonal-to-Smith-i } xs \ A \ i \ \text{bezout}) \$ a \$ b$   
**by** (rule *diagonal-to-Smith-i-dvd-jj*, insert *assms False ai bi*, auto)  
**finally show** ?thesis .  
**qed**

**lemma** *diagonal-to-Smith-i-dvd2-k*:  
**fixes**  $A::'a::\{\text{bezout-ring}\}^{\wedge}b::\text{mod-type}^{\wedge}c::\text{mod-type}$   
**assumes** *ib*: is-bezout-ext bezout  
**and** *i-min*:  $i < \min(\text{nrows } A) (\text{ncols } A)$   
**and** *elements-xs-range*:  $\forall x. x \in \text{set } xs \longrightarrow x < k$  **and**  $k \leq \min(\text{nrows } A) (\text{ncols } A)$   
**and** *dvd-condition*:  $\forall a b. \text{to-nat } a \in \text{insert } i (\text{set } xs) \wedge \text{to-nat } a = \text{to-nat } b \longrightarrow$   
 $A \$ (\text{from-nat } c) \$ (\text{from-nat } c) \text{dvd } A \$ a \$ b$   
**and** *c-notin*:  $c \notin (\text{set } xs)$   
**and** *c*:  $c < \min(\text{nrows } A) (\text{ncols } A)$   
**and** *distinct*: distinct *xs*  
**and** *ab*:  $\text{to-nat } a = \text{to-nat } b$

**and**  $a\text{-in}$ :  $\text{to-nat } a \in \text{insert } i \text{ (set } xs)$   
**shows**  $A \ \$ \ (\text{from-nat } c) \ \$ \ (\text{from-nat } c) \ \text{dvd} \ (\text{diagonal-to-Smith-}i \ \text{xs } A \ i \ \text{bezout}) \ \$ \ a \ \$ \ b$   
**proof** ( $\text{cases } a = \text{from-nat } i$ )  
  **case**  $\text{True}$   
  **hence**  $b$ :  $b = \text{from-nat } i$   
  **by** ( $\text{metis } ab \ \text{from-nat-to-nat-id } i\text{-min } \text{min-less-iff-conj } \text{nrows-def } \text{to-nat-from-nat-id}$ )  
  **show**  $?thesis$  **by** ( $\text{unfold } \text{True } b, \text{rule } \text{diagonal-to-Smith-}i\text{-dvd}, \text{insert } \text{assms}, \text{auto}$ )  
**next**  
  **case**  $\text{False}$   
  **have**  $ai$ :  $\text{to-nat } a \neq i$  **using**  $\text{False}$  **by**  $\text{auto}$   
  **hence**  $bi$ :  $\text{to-nat } b \neq i$  **by** ( $\text{simp add: } ab$ )  
  **have**  $A \ \$ \ (\text{from-nat } c) \ \$ \ (\text{from-nat } c) \ \text{dvd} \ (\text{diagonal-to-Smith-}i \ \text{xs } A \ i \ \text{bezout}) \ \$ \ \text{from-nat } i \ \$ \ \text{from-nat } i$   
  **by** ( $\text{rule } \text{diagonal-to-Smith-}i\text{-dvd}, \text{insert } \text{assms}, \text{auto}$ )  
  **also have**  $\dots \ \text{dvd} \ (\text{diagonal-to-Smith-}i \ \text{xs } A \ i \ \text{bezout}) \ \$ \ a \ \$ \ b$   
  **by** ( $\text{rule } \text{diagonal-to-Smith-}i\text{-dvd-jj}, \text{insert } \text{assms } \text{False } ai \ bi, \text{auto}$ )  
  **finally show**  $?thesis$  .  
**qed**

**lemma**  $\text{diagonal-to-Smith-row-}i\text{-preserves-previous}$ :  
**fixes**  $A::'a::\{\text{bezout-ring}\}^{\wedge}b::\text{mod-type}^{\wedge}c::\text{mod-type}$   
**assumes**  $ib$ :  $\text{is-bezout-ext } \text{bezout}$   
**and**  $i\text{-min}$ :  $i < \text{min} \ (\text{nrows } A) \ (\text{ncols } A)$   
**and**  $a\text{-not-}b$ :  $\text{to-nat } a \neq \text{to-nat } b$   
**shows**  $\text{Diagonal-to-Smith-row-}i \ A \ i \ \text{bezout} \ \$ \ a \ \$ \ b = A \ \$ \ a \ \$ \ b$   
  **unfolding**  $\text{Diagonal-to-Smith-row-}i\text{-def}$   
  **by** ( $\text{rule } \text{diagonal-to-Smith-}i\text{-preserves-previous}, \text{insert } \text{assms}, \text{auto}$ )

**lemma**  $\text{diagonal-to-Smith-row-}i\text{-preserves-previous-diagonal}$ :  
**fixes**  $A::'a::\{\text{bezout-ring}\}^{\wedge}b::\text{mod-type}^{\wedge}c::\text{mod-type}$   
**assumes**  $ib$ :  $\text{is-bezout-ext } \text{bezout}$   
**and**  $i\text{-min}$ :  $i < \text{min} \ (\text{nrows } A) \ (\text{ncols } A)$   
**and**  $a\text{-notin}$ :  $\text{to-nat } a \notin \text{set } [i + 1..<\text{min} \ (\text{nrows } A) \ (\text{ncols } A)]$   
**and**  $ab$ :  $\text{to-nat } a = \text{to-nat } b$   
**and**  $ai$ :  $\text{to-nat } a \neq i$   
**shows**  $\text{Diagonal-to-Smith-row-}i \ A \ i \ \text{bezout} \ \$ \ a \ \$ \ b = A \ \$ \ a \ \$ \ b$   
**unfolding**  $\text{Diagonal-to-Smith-row-}i\text{-def}$   
**by** ( $\text{rule } \text{diagonal-to-Smith-}i\text{-preserves-previous-diagonal}[\text{OF } ib \ i\text{-min } a\text{-notin } ab \ ai], \text{auto}$ )

**context**  
  **fixes**  $\text{bezout}::'a::\{\text{bezout-ring}\} \Rightarrow 'a \Rightarrow 'a \times 'a \times 'a \times 'a \times 'a$   
  **assumes**  $ib$ :  $\text{is-bezout-ext } \text{bezout}$   
**begin**

**lemma** *diagonal-to-Smith-row-i-dvd-jj*:  
**fixes**  $A::'a::\{\text{bezout-ring}\}^{\wedge}b::\text{mod-type}^{\wedge}c::\text{mod-type}$   
**assumes**  $\text{to-nat } a \in \{i..<\min(\text{nrows } A) (\text{ncols } A)\}$   
**and**  $\text{to-nat } a = \text{to-nat } b$   
**shows**  $(\text{Diagonal-to-Smith-row-}i \ A \ i \ \text{bezout}) \ \$ \ (\text{from-nat } i) \ \$ \ (\text{from-nat } i)$   
 $\text{dvd } (\text{Diagonal-to-Smith-row-}i \ A \ i \ \text{bezout}) \ \$ \ a \ \$ \ b$   
**proof**  $(\text{cases } \text{to-nat } a = i)$   
**case** *True*  
**then show** *?thesis*  
**by**  $(\text{metis } \text{assms}(2) \ \text{dvd-refl } \text{from-nat-to-nat-id})$   
**next**  
**case** *False*  
**show** *?thesis*  
**unfolding** *Diagonal-to-Smith-row-i-def*  
**by**  $(\text{rule } \text{diagonal-to-Smith-i-dvd-jj}, \text{insert } \text{assms } \text{False } \text{ib}, \text{auto})$   
**qed**

**lemma** *diagonal-to-Smith-row-i-dvd-ii*:  
**fixes**  $A::'a::\{\text{bezout-ring}\}^{\wedge}b::\text{mod-type}^{\wedge}c::\text{mod-type}$   
**shows**  $\text{Diagonal-to-Smith-row-}i \ A \ i \ \text{bezout} \ \$ \ \text{from-nat } i \ \$ \ \text{from-nat } i \ \text{dvd } A \ \$$   
 $\text{from-nat } i \ \$ \ \text{from-nat } i$   
**unfolding** *Diagonal-to-Smith-row-i-def*  
**by**  $(\text{rule } \text{diagonal-to-Smith-i-dvd-ii}[\text{OF } \text{ib}])$

**lemma** *diagonal-to-Smith-row-i-dvd-jj'*:  
**fixes**  $A::'a::\{\text{bezout-ring}\}^{\wedge}b::\text{mod-type}^{\wedge}c::\text{mod-type}$   
**assumes**  $a\text{-in: } \text{to-nat } a \in \{i..<\min(\text{nrows } A) (\text{ncols } A)\}$   
**and**  $ab: \text{to-nat } a = \text{to-nat } b$   
**and**  $ci: c \leq i$   
**and**  $\text{dvd-condition: } \forall a \ b. \ \text{to-nat } a \in (\text{set } [i..<\min(\text{nrows } A) (\text{ncols } A)]) \wedge \text{to-nat}$   
 $a = \text{to-nat } b$   
 $\longrightarrow A \ \$ \ \text{from-nat } c \ \$ \ \text{from-nat } c \ \text{dvd } A \ \$ \ a \ \$ \ b$   
**shows**  $(\text{Diagonal-to-Smith-row-}i \ A \ i \ \text{bezout}) \ \$ \ (\text{from-nat } c) \ \$ \ (\text{from-nat } c)$   
 $\text{dvd } (\text{Diagonal-to-Smith-row-}i \ A \ i \ \text{bezout}) \ \$ \ a \ \$ \ b$   
**proof**  $(\text{cases } c = i)$   
**case** *True*  
**then show** *?thesis using assms True diagonal-to-Smith-row-i-dvd-jj*  
**by** *metis*  
**next**  
**case** *False*  
**hence**  $ci2: c < i$  **using**  $ci$  **by** *auto*  
**have**  $1: \text{to-nat } (\text{from-nat } c::'c) \notin (\text{set } [i+1..<\min(\text{nrows } A) (\text{ncols } A)])$   
**by**  $(\text{metis } \text{Suc-eq-plus1 } ci \ \text{atLeastLessThan-iff } \text{from-nat-mono}$   
 $\text{le-imp-less-Suc } \text{less-irrefl } \text{less-le-trans } \text{set-upt } \text{to-nat-le } \text{to-nat-less-card})$   
**have**  $2: \text{to-nat } (\text{from-nat } c) \neq i$   
**using**  $ci2$  *from-nat-mono to-nat-less-card* **by** *fastforce*  
**have**  $3: \text{to-nat } (\text{from-nat } c::'c) = \text{to-nat } (\text{from-nat } c::'b)$

**by** (*metis a-in ab atLeastLessThan-iff ci dual-order.strict-trans2 to-nat-from-nat-id to-nat-less-card*)  
**have** (*Diagonal-to-Smith-row-i A i bezout*) \$ (*from-nat c*) \$ (*from-nat c*)  
= *A* \$ (*from-nat c*) \$ (*from-nat c*)  
**unfolding** *Diagonal-to-Smith-row-i-def*  
**by** (*rule diagonal-to-Smith-i-preserves-previous-diagonal[OF ib - 1 3 2]*, *insert assms, auto*)  
**also have** ... *dvd* (*Diagonal-to-Smith-row-i A i bezout*) \$ *a* \$ *b*  
**unfolding** *Diagonal-to-Smith-row-i-def*  
**by** (*rule diagonal-to-Smith-i-dvd2, insert assms False ci ib, auto*)  
**finally show** *?thesis* .  
**qed**  
**end**

**lemma** *diagonal-to-Smith-aux-append*:  
*diagonal-to-Smith-aux A (xs @ ys) bezout*  
= *diagonal-to-Smith-aux (diagonal-to-Smith-aux A xs bezout) ys bezout*  
**by** (*induct A xs bezout rule: diagonal-to-Smith-aux.induct, auto*)

**lemma** *diagonal-to-Smith-aux-append2[simp]*:  
*diagonal-to-Smith-aux A (xs @ [ys]) bezout*  
= *Diagonal-to-Smith-row-i (diagonal-to-Smith-aux A xs bezout) ys bezout*  
**by** (*induct A xs bezout rule: diagonal-to-Smith-aux.induct, auto*)

**lemma** *isDiagonal-eq-upt-k-min*:  
*isDiagonal A = isDiagonal-upt-k A (min (nrows A) (ncols A))*  
**unfolding** *isDiagonal-def isDiagonal-upt-k-def nrows-def ncols-def*  
**by** (*auto, meson less-trans not-less-iff-gr-or-eq to-nat-less-card*)

**lemma** *isDiagonal-eq-upt-k-max*:  
*isDiagonal A = isDiagonal-upt-k A (max (nrows A) (ncols A))*  
**unfolding** *isDiagonal-def isDiagonal-upt-k-def nrows-def ncols-def*  
**by** (*auto simp add: less-max-iff-disj to-nat-less-card*)

**lemma** *isDiagonal*:  
**assumes** *isDiagonal A*  
**and** *to-nat a ≠ to-nat b* **shows** *A \$ a \$ b = 0*  
**using** *assms* **unfolding** *isDiagonal-def* **by** *auto*

**lemma** *nrows-diagonal-to-Smith-aux[simp]*:  
**shows** *nrows (diagonal-to-Smith-aux A xs bezout) = nrows A* **unfolding** *nrows-def*  
**by** *auto*

**lemma** *ncols-diagonal-to-Smith-aux[simp]*:  
**shows** *ncols (diagonal-to-Smith-aux A xs bezout) = ncols A* **unfolding** *ncols-def*

by auto

context

fixes bezout::'a::{bezout-ring}  $\Rightarrow$  'a  $\Rightarrow$  'a  $\times$  'a  $\times$  'a  $\times$  'a  $\times$  'a

assumes ib: is-bezout-ext bezout

begin

lemma isDiagonal-diagonal-to-Smith-aux:

assumes diag-A: isDiagonal A and k: k < min (nrows A) (ncols A)

shows isDiagonal (diagonal-to-Smith-aux A [0..<k] bezout)

using k

proof (induct k)

case 0

then show ?case using diag-A by auto

next

case (Suc k)

have Diagonal-to-Smith-row-i (diagonal-to-Smith-aux A [0..<k] bezout) k bezout  
\$ a \$ b = 0

if a-not-b: to-nat a  $\neq$  to-nat b for a b

proof -

have Diagonal-to-Smith-row-i (diagonal-to-Smith-aux A [0..<k] bezout) k bezout  
\$ a \$ b

= (diagonal-to-Smith-aux A [0..<k] bezout) \$ a \$ b

by (rule diagonal-to-Smith-row-i-preserves-previous[OF ib - a-not-b], insert  
Suc.prem, auto)

also have ... = 0

by (rule isDiagonal[OF Suc.hyps a-not-b], insert Suc.prem, auto)

finally show ?thesis .

qed

thus ?case unfolding isDiagonal-def by auto

qed

end

lemma to-nat-less-nrows[simp]:

fixes A::'a<sup>^</sup>b::mod-type<sup>^</sup>c::mod-type

and a::'c

shows to-nat a < nrows A

by (simp add: nrows-def to-nat-less-card)

lemma to-nat-less-ncols[simp]:

fixes A::'a<sup>^</sup>b::mod-type<sup>^</sup>c::mod-type

and a::'b

shows to-nat a < ncols A

by (simp add: ncols-def to-nat-less-card)

context

fixes bezout::'a::{bezout-ring}  $\Rightarrow$  'a  $\Rightarrow$  'a  $\times$  'a  $\times$  'a  $\times$  'a  $\times$  'a

assumes ib: is-bezout-ext bezout

**begin**

The variables a and b must be arbitrary in the induction

**lemma** *diagonal-to-Smith-aux-dvd*:

**fixes**  $A::'a::\{\text{bezout-ring}\}^{\wedge}b::\text{mod-type}^{\wedge}c::\text{mod-type}$

**assumes**  $ab: \text{to-nat } a = \text{to-nat } b$

**and**  $c: c < k$  **and**  $ca: c \leq \text{to-nat } a$  **and**  $k: k < \min(\text{nrows } A) (\text{ncols } A)$

**shows** *diagonal-to-Smith-aux*  $A [0..<k]$  *bezout* \$ *from-nat*  $c$  \$ *from-nat*  $c$

*dvd diagonal-to-Smith-aux*  $A [0..<k]$  *bezout* \$  $a$  \$  $b$

**using**  $c$   $ab$   $ca$   $k$

**proof** (*induct*  $k$  *arbitrary*:  $a$   $b$ )

**case**  $0$

**then show** *?case* **by** *auto*

**next**

**case** (*Suc*  $k$ )

**note**  $c = \text{Suc.prem}(1)$

**note**  $ab = \text{Suc.prem}(2)$

**note**  $ca = \text{Suc.prem}(3)$

**note**  $k = \text{Suc.prem}(4)$

**have** *k-min*:  $k < \min(\text{nrows } A) (\text{ncols } A)$  **using**  $k$  **by** *auto*

**have** *a-less-ncols*:  $\text{to-nat } a < \text{ncols } A$  **using**  $ab$  **by** *auto*

**show** *?case*

**proof** (*cases*  $c=k$ )

**case** *True*

**hence**  $k: k \leq \text{to-nat } a$  **using**  $ca$  **by** *auto*

**show** *?thesis* **unfolding** *True*

**by** (*auto*, *rule diagonal-to-Smith-row-i-dvd-jj*[*OF*  $ib - ab$ ], *insert*  $k$  *a-less-ncols*, *auto*)

**next**

**case** *False* **note**  $c\text{-not-}k = \text{False}$

**let** *?Dk*=*diagonal-to-Smith-aux*  $A [0..<k]$  *bezout*

**have**  $ck: c < k$  **using** *Suc.prem* *False* **by** *auto*

**have** *hyp*: *?Dk* \$ *from-nat*  $c$  \$ *from-nat*  $c$  *dvd* *?Dk* \$  $a$  \$  $b$

**by** (*rule* *Suc.hyps*[*OF*  $ck$   $ab$   $ca$   $k\text{-min}$ ])

**have** *Dkk-Daa-bb*: *?Dk* \$ *from-nat*  $c$  \$ *from-nat*  $c$  *dvd* *?Dk* \$  $aa$  \$  $bb$

**if**  $\text{to-nat } aa \in \text{set } [k..<\min(\text{nrows } ?Dk) (\text{ncols } ?Dk)]$  **and**  $\text{to-nat } aa = \text{to-nat } bb$

*bb*

**for**  $aa$   $bb$  **using** *Suc.hyps*  $ck$   $k\text{-min}$  *that*(1) *that*(2) **by** *auto*

**show** *?thesis*

**proof** (*cases*  $k \leq \text{to-nat } a$ )

**case** *True*

**show** *?thesis*

**by** (*auto*, *rule diagonal-to-Smith-row-i-dvd-jj*'[*OF*  $ib - ab$ ])

(*insert* *True* *a-less-ncols*  $ck$  *Dkk-Daa-bb*, *force*+) )

**next**

**case** *False*

**have** *diagonal-to-Smith-aux*  $A [0..<\text{Suc } k]$  *bezout* \$ *from-nat*  $c$  \$ *from-nat*  $c$

= *Diagonal-to-Smith-row-i* *?Dk*  $k$  *bezout* \$ *from-nat*  $c$  \$ *from-nat*  $c$  **by** *auto*

**also have** ... = *?Dk* \$ *from-nat*  $c$  \$ *from-nat*  $c$



```

proof (rule diagonal-to-Smith-row-i-preserves-previous-diagonal[OF ib])
  show  $k < \min (\text{nrows } ?Dk) (\text{ncols } ?Dk)$  using  $k$  by auto
  show  $\text{to-nat } (\text{from-nat } c::'c) = \text{to-nat } (\text{from-nat } c::'b)$ 
    by (metis assms(2) assms(4) less-trans min-less-iff-conj
      ncols-def nrows-def to-nat-from-nat-id)
  show  $\text{to-nat } (\text{from-nat } c::'c) \neq k$ 
    using False ca from-nat-mono' to-nat-less-card to-nat-mono' by fastforce

  show  $\text{to-nat } (\text{from-nat } c::'c) \notin \text{set } [k + 1..<\min (\text{nrows } ?Dk) (\text{ncols } ?Dk)]$ 
    by (metis Suc-eq-plus1 atLeastLessThan-iff c ca from-nat-not-eq
      le-less-trans not-le set-upt to-nat-less-card)
qed
also have ... dvd ?Dk $ a $ b using hyp .
also have ... = Diagonal-to-Smith-row-i ?Dk k bezout $ a $ b
  by (rule diagonal-to-Smith-row-i-preserves-previous-diagonal[symmetric, OF
ib - - ab])
    (insert False k, auto)
  also have ... = diagonal-to-Smith-aux A [0..<Suc k] bezout $ a $ b by auto
finally show ?thesis .
qed
qed
qed

```

```

lemma Smith-normal-form-upt-k-Suc-imp-k:
  fixes  $A::'a::\{\text{bezout-ring}\} \wedge b::\text{mod-type} \wedge c::\text{mod-type}$ 
  assumes  $s$ : Smith-normal-form-upt-k (diagonal-to-Smith-aux A [0..<Suc k] bezout) k
  and  $k$ :  $k < \min (\text{nrows } A) (\text{ncols } A)$ 
  shows Smith-normal-form-upt-k (diagonal-to-Smith-aux A [0..<k] bezout) k
proof (rule Smith-normal-form-upt-k-intro)
  let  $?Dk = \text{diagonal-to-Smith-aux } A [0..<k] \text{ bezout}$ 
  fix  $a::'c$  and  $b::'b$  assume  $\text{to-nat } a = \text{to-nat } b \wedge \text{to-nat } a + 1 < k \wedge \text{to-nat } b + 1 < k$ 
  hence  $ab$ :  $\text{to-nat } a = \text{to-nat } b$  and  $ak$ :  $\text{to-nat } a + 1 < k$  and  $bk$ :  $\text{to-nat } b + 1 < k$  by auto
  have  $a\text{-not-}k$ :  $\text{to-nat } a \neq k$  using  $ak$  by auto
  have  $a1\text{-less-}k1$ :  $\text{to-nat } a + 1 < k + 1$  using  $ak$  by linarith
  have  $?Dk \$ a \$ b = \text{diagonal-to-Smith-aux } A [0..<Suc k] \text{ bezout } \$ a \$ b$ 
  by (auto, rule diagonal-to-Smith-row-i-preserves-previous-diagonal[symmetric, OF ib - - ab a-not-k])
    (insert  $ak k, auto$ )
  also have ... dvd diagonal-to-Smith-aux A [0..<Suc k] bezout $ (a + 1) $ (b + 1)
  using  $ab ak bk s$  unfolding Smith-normal-form-upt-k-def by auto
  also have ... =  $?Dk \$ (a+1) \$ (b+1)$ 
proof (auto, rule diagonal-to-Smith-row-i-preserves-previous-diagonal[OF ib])
  show  $\text{to-nat } (a + 1) \neq k$  using  $ak$ 
  by (metis add-less-same-cancel2 nat-neq-iff not-add-less2 to-nat-0)

```

$to\text{-}nat\text{-}plus\text{-}one\text{-}less\text{-}card'$   $to\text{-}nat\text{-}suc$   
**show**  $to\text{-}nat (a + 1) = to\text{-}nat (b + 1)$   
**by** ( $metis$   $ab$   $ak$   $from\text{-}nat\text{-}suc$   $from\text{-}nat\text{-}to\text{-}nat\text{-}id$   $k$   $less\text{-}asym'$   $min\text{-}less\text{-}iff\text{-}conj$ )

$ncols\text{-}def$   $nrows\text{-}def$   $suc\text{-}not\text{-}zero$   $to\text{-}nat\text{-}from\text{-}nat\text{-}id$   $to\text{-}nat\text{-}plus\text{-}one\text{-}less\text{-}card'$   
**show**  $to\text{-}nat (a + 1) \notin set [k + 1..<min (nrows ?Dk) (ncols ?Dk)]$   
**by** ( $metis$   $a1\text{-}less\text{-}k1$   $add\text{-}to\text{-}nat\text{-}def$   $atLeastLessThan\text{-}iff$   $k$   $less\text{-}asym'$   $min.\text{strict}\text{-}boundedE$ )

$not\text{-}less$   $nrows\text{-}def$   $set\text{-}upt$   $suc\text{-}not\text{-}zero$   $to\text{-}nat\text{-}1$   $to\text{-}nat\text{-}from\text{-}nat\text{-}id$   $to\text{-}nat\text{-}plus\text{-}one\text{-}less\text{-}card'$   
**show**  $k < min (nrows ?Dk) (ncols ?Dk)$  **using**  $k$  **by**  $auto$   
**qed**  
**finally show**  $?Dk \$ a \$ b \text{ dvd } ?Dk \$ (a+1) \$ (b+1)$  .

**next**  
**let**  $?Dk = diagonal\text{-}to\text{-}Smith\text{-}aux A [0..<k]$   $bezout$   
**fix**  $a::'c$  **and**  $b::'b$   
**assume**  $to\text{-}nat a \neq to\text{-}nat b \wedge (to\text{-}nat a < k \vee to\text{-}nat b < k)$   
**hence**  $ab: to\text{-}nat a \neq to\text{-}nat b$  **and**  $ak\text{-}bk: (to\text{-}nat a < k \vee to\text{-}nat b < k)$  **by**  $auto$   
**have**  $?Dk \$ a \$ b = diagonal\text{-}to\text{-}Smith\text{-}aux A [0..<Suc k]$   $bezout \$ a \$ b$   
**by** ( $auto$ ,  $rule$   $diagonal\text{-}to\text{-}Smith\text{-}row\text{-}i\text{-}preserves\text{-}previous[symmetric, OF ib - ab]$ ,  $insert$   $k$ ,  $auto$ )  
**also have**  $\dots = 0$   
**using**  $ab$   $ak\text{-}bk$   $s$  **unfolding**  $Smith\text{-}normal\text{-}form\text{-}upt\text{-}k\text{-}def$   $isDiagonal\text{-}upt\text{-}k\text{-}def$   
**by**  $auto$   
**finally show**  $?Dk \$ a \$ b = 0$  .  
**qed**

**lemma**  $Smith\text{-}normal\text{-}form\text{-}upt\text{-}k\text{-}le:$   
**assumes**  $a \leq k$  **and**  $Smith\text{-}normal\text{-}form\text{-}upt\text{-}k A k$   
**shows**  $Smith\text{-}normal\text{-}form\text{-}upt\text{-}k A a$  **using**  $assms$   
**by** ( $smt$  ( $verit$ )  $Smith\text{-}normal\text{-}form\text{-}upt\text{-}k\text{-}def$   $isDiagonal\text{-}upt\text{-}k\text{-}def$   $less\text{-}le\text{-}trans$ )

**lemma**  $Smith\text{-}normal\text{-}form\text{-}upt\text{-}k\text{-}imp\text{-}Suc\text{-}k:$   
**assumes**  $s: Smith\text{-}normal\text{-}form\text{-}upt\text{-}k (diagonal\text{-}to\text{-}Smith\text{-}aux A [0..<k]) \text{ bezout}$   $k$   
**and**  $k: k < min (nrows A) (ncols A)$   
**shows**  $Smith\text{-}normal\text{-}form\text{-}upt\text{-}k (diagonal\text{-}to\text{-}Smith\text{-}aux A [0..<Suc k]) \text{ bezout}$   $k$   
**proof** ( $rule$   $Smith\text{-}normal\text{-}form\text{-}upt\text{-}k\text{-}intro$ )  
**let**  $?Dk = diagonal\text{-}to\text{-}Smith\text{-}aux A [0..<k]$   $bezout$   
**fix**  $a::'c$  **and**  $b::'b$  **assume**  $to\text{-}nat a = to\text{-}nat b \wedge to\text{-}nat a + 1 < k \wedge to\text{-}nat b + 1 < k$   
**hence**  $ab: to\text{-}nat a = to\text{-}nat b$  **and**  $ak: to\text{-}nat a + 1 < k$  **and**  $bk: to\text{-}nat b + 1 < k$  **by**  $auto$   
**have**  $a\text{-}not\text{-}k: to\text{-}nat a \neq k$  **using**  $ak$  **by**  $auto$   
**have**  $a1\text{-}less\text{-}k1: to\text{-}nat a + 1 < k + 1$  **using**  $ak$  **by**  $linarith$   
**have**  $diagonal\text{-}to\text{-}Smith\text{-}aux A [0..<Suc k]$   $bezout \$ a \$ b = ?Dk \$ a \$ b$   
**by** ( $auto$ ,  $rule$   $diagonal\text{-}to\text{-}Smith\text{-}row\text{-}i\text{-}preserves\text{-}previous\text{-}diagonal[OF ib - - ab a\text{-}not\text{-}k]$ )  
 $(insert$   $ak$   $k$ ,  $auto$ )  
**also have**  $\dots \text{ dvd } ?Dk \$ (a+1) \$ (b+1)$

**using**  $s$   $ak$   $k$   $ab$  **unfolding** *Smith-normal-form-upt-k-def* **by** *auto*  
**also have**  $\dots = \text{diagonal-to-Smith-aux } A [0..<Suc\ k] \text{ bezout } \$ (a + 1) \$ (b + 1)$

**proof** (*auto*, *rule diagonal-to-Smith-row-i-preserves-previous-diagonal[symmetric, OF ib]*)  
**show**  $\text{to-nat } (a + 1) \neq k$  **using**  $ak$   
**by** (*metis add-less-same-cancel2 nat-neq-iff not-add-less2 to-nat-0 to-nat-plus-one-less-card' to-nat-suc*)  
**show**  $\text{to-nat } (a + 1) = \text{to-nat } (b + 1)$   
**by** (*metis ab ak from-nat-suc from-nat-to-nat-id k less-asm' min-less-iff-conj*)

*ncols-def nrows-def suc-not-zero to-nat-from-nat-id to-nat-plus-one-less-card'*  
**show**  $\text{to-nat } (a + 1) \notin \text{set } [k + 1..<\min (nrows\ ?Dk) (ncols\ ?Dk)]$   
**by** (*metis a1-less-k1 add-to-nat-def to-nat-plus-one-less-card' less-asm' min.strict-boundedE*)

*not-less nrows-def set-upt suc-not-zero to-nat-1 to-nat-from-nat-id atLeast-LessThan-iff k*  
**show**  $k < \min (nrows\ ?Dk) (ncols\ ?Dk)$  **using**  $k$  **by** *auto*  
**qed**

**finally show** *diagonal-to-Smith-aux*  $A [0..<Suc\ k] \text{ bezout } \$ a \$ b$   
*dvd diagonal-to-Smith-aux*  $A [0..<Suc\ k] \text{ bezout } \$ (a + 1) \$ (b + 1)$  .

**next**  
**let**  $?Dk = \text{diagonal-to-Smith-aux } A [0..<k] \text{ bezout}$   
**fix**  $a::'c$  **and**  $b::'b$   
**assume**  $\text{to-nat } a \neq \text{to-nat } b \wedge (\text{to-nat } a < k \vee \text{to-nat } b < k)$   
**hence**  $ab$ :  $\text{to-nat } a \neq \text{to-nat } b$  **and**  $ak$ - $bk$ :  $(\text{to-nat } a < k \vee \text{to-nat } b < k)$  **by** *auto*  
**have** *diagonal-to-Smith-aux*  $A [0..<Suc\ k] \text{ bezout } \$ a \$ b = ?Dk\ \$a\ \$b$   
**by** (*auto*, *rule diagonal-to-Smith-row-i-preserves-previous[OF ib - ab]*, *insert k, auto*)  
**also have**  $\dots = 0$   
**using**  $ab$   $ak$ - $bk$   $s$  **unfolding** *Smith-normal-form-upt-k-def isDiagonal-upt-k-def*  
**by** *auto*  
**finally show** *diagonal-to-Smith-aux*  $A [0..<Suc\ k] \text{ bezout } \$ a \$ b = 0$  .

**qed**

**corollary** *Smith-normal-form-upt-k-Suc-eq*:  
**assumes**  $k: k < \min (nrows\ A) (ncols\ A)$   
**shows** *Smith-normal-form-upt-k* (*diagonal-to-Smith-aux*  $A [0..<Suc\ k] \text{ bezout}$ )  $k$   
 $= \text{Smith-normal-form-upt-k } (\text{diagonal-to-Smith-aux } A [0..<k] \text{ bezout})\ k$   
**using** *Smith-normal-form-upt-k-Suc-imp-k Smith-normal-form-upt-k-imp-Suc-k*  $k$

**by** *blast*

**end**

**lemma** *nrows-diagonal-to-Smith-i[simp]*:  $nrows (\text{diagonal-to-Smith-i } xs\ A\ i \text{ bezout}) = nrows\ A$   
**by** (*induct xs A i bezout rule: diagonal-to-Smith-i.induct, auto simp add: nrows-def*)

**lemma** *ncols-diagonal-to-Smith-i[simp]*: *ncols (diagonal-to-Smith-i xs A i bezout) = ncols A*

**by** (*induct xs A i bezout rule: diagonal-to-Smith-i.induct, auto simp add: ncols-def*)

**lemma** *nrows-Diagonal-to-Smith-row-i[simp]*: *nrows (Diagonal-to-Smith-row-i A i bezout) = nrows A*

**unfolding** *Diagonal-to-Smith-row-i-def* **by** *auto*

**lemma** *ncols-Diagonal-to-Smith-row-i[simp]*: *ncols (Diagonal-to-Smith-row-i A i bezout) = ncols A*

**unfolding** *Diagonal-to-Smith-row-i-def* **by** *auto*

**lemma** *isDiagonal-diagonal-step*:

**assumes** *diag-A: isDiagonal A* **and** *i: i < min (nrows A) (ncols A)*

**and** *j: j < min (nrows A) (ncols A)*

**shows** *isDiagonal (diagonal-step A i j d v)*

**proof** –

**have** *i-eq: to-nat (from-nat i::'b) = to-nat (from-nat i::'c)* **using** *i*

**by** (*simp add: ncols-def nrows-def to-nat-from-nat-id*)

**moreover have** *j-eq: to-nat (from-nat j::'b) = to-nat (from-nat j::'c)* **using** *j*

**by** (*simp add: ncols-def nrows-def to-nat-from-nat-id*)

**ultimately show** *?thesis*

**using** *assms*

**unfolding** *isDiagonal-def diagonal-step-def* **by** *auto*

**qed**

**lemma** *isDiagonal-diagonal-to-Smith-i*:

**assumes** *isDiagonal A*

**and** *elements-xs-range:  $\forall x. x \in \text{set } xs \longrightarrow x < \min (\text{nrows } A) (\text{ncols } A)$*

**and** *i < min (nrows A) (ncols A)*

**shows** *isDiagonal (diagonal-to-Smith-i xs A i bezout)*

**using** *assms*

**proof** (*induct xs A i bezout rule: diagonal-to-Smith-i.induct*)

**case** (*1 A i bezout*)

**then show** *?case* **by** *auto*

**next**

**case** (*2 j xs A i bezout*)

**let** *?Aii = A \$ from-nat i \$ from-nat i*

**let** *?Ajj = A \$ from-nat j \$ from-nat j*

**let** *?p=case bezout (A \$ from-nat i \$ from-nat i) (A \$ from-nat j \$ from-nat j)*  
of (*p,q,u,v,d*)  $\Rightarrow$  *p*

**let** *?q=case bezout (A \$ from-nat i \$ from-nat i) (A \$ from-nat j \$ from-nat j)*  
of (*p,q,u,v,d*)  $\Rightarrow$  *q*

**let** *?u=case bezout (A \$ from-nat i \$ from-nat i) (A \$ from-nat j \$ from-nat j)*  
of (*p,q,u,v,d*)  $\Rightarrow$  *u*

**let** *?v=case bezout (A \$ from-nat i \$ from-nat i) (A \$ from-nat j \$ from-nat j)*  
of (*p,q,u,v,d*)  $\Rightarrow$  *v*

**let** *?d=case bezout (A \$ from-nat i \$ from-nat i) (A \$ from-nat j \$ from-nat j)*

```

of (p,q,u,v,d) ⇒ d
  let ?A'=diagonal-step A i j ?d ?v
  have pqvud: (?p, ?q, ?u, ?v,?d) = bezout (A $ from-nat i $ from-nat i) (A $
from-nat j $ from-nat j)
    by (simp add: split-beta)
  show ?case
  proof (cases ?Aii dvd ?Ajj)
    case True
    thus ?thesis
      using 2.hyps 2.prem by auto
  next
  case False
  have diagonal-to-Smith-i (j # xs) A i bezout = diagonal-to-Smith-i xs ?A' i
  bezout
    using False by (simp add: split-beta)
  also have isDiagonal ... thm 2.prem
  proof (rule 2.hyps(2)[OF False])
    show isDiagonal
      (diagonal-step A i j ?d ?v) by (rule isDiagonal-diagonal-step, insert 2.prem,
auto)
    qed (insert pqvud 2.prem, auto)
  finally show ?thesis .
  qed
qed

```

```

lemma isDiagonal-Diagonal-to-Smith-row-i:
  assumes isDiagonal A and i < min (nrows A) (ncols A)
  shows isDiagonal (Diagonal-to-Smith-row-i A i bezout)
  using assms isDiagonal-diagonal-to-Smith-i
  unfolding Diagonal-to-Smith-row-i-def by force

```

```

lemma isDiagonal-diagonal-to-Smith-aux-general:
  assumes elements-xs-range: ∀ x. x ∈ set xs → x < min (nrows A) (ncols A)
  and isDiagonal A
  shows isDiagonal (diagonal-to-Smith-aux A xs bezout)
  using assms
  proof (induct A xs bezout rule: diagonal-to-Smith-aux.induct)
    case (1 A)
    then show ?case by auto
  next
  case (2 A i xs bezout)
  note k = 2.prem(1)
  note elements-xs-range = 2.prem(2)
  have diagonal-to-Smith-aux A (i # xs) bezout
  = diagonal-to-Smith-aux (Diagonal-to-Smith-row-i A i bezout) xs bezout
  by auto
  also have isDiagonal (...)

```

by (rule 2.hyps, insert isDiagonal-Diagonal-to-Smith-row-i 2.prem k, auto)  
 finally show ?case .  
 qed

context

fixes bezout::'a::{bezout-ring}  $\Rightarrow$  'a  $\Rightarrow$  'a  $\times$  'a  $\times$  'a  $\times$  'a  $\times$  'a  
 assumes ib: is-bezout-ext bezout

begin

The algorithm is iterated up to position k (not included). Thus, the matrix is in Smith normal form up to position k (not included).

**lemma** *Smith-normal-form-upt-k-diagonal-to-Smith-aux:*

fixes A::'a::{bezout-ring}  $\wedge$  b::mod-type  $\wedge$  c::mod-type  
 assumes  $k < \min$  (nrows A) (ncols A) and d: isDiagonal A  
 shows *Smith-normal-form-upt-k* (diagonal-to-Smith-aux A [0.. $k$ ] bezout) k  
 using *assms*

**proof** (*induct k*)

case 0

then show ?case by auto

next

case (Suc k)

note *Suc-k* = *Suc.prem*(1)

have *hyp*: *Smith-normal-form-upt-k* (diagonal-to-Smith-aux A [0.. $k$ ] bezout) k

by (rule *Suc.hyps*, insert *Suc.prem*, *simp*)

have  $k < \min$  (nrows A) (ncols A) using *Suc.prem* by auto

let ?A = diagonal-to-Smith-aux A [0.. $k$ ] bezout

let ?D-Suck = diagonal-to-Smith-aux A [0..*Suc k*] bezout

have *set-rw*: [0..*Suc k*] = [0.. $k$ ] @ [ $k$ ] by auto

show ?case

**proof** (rule *Smith-normal-form-upt-k1-intro-diagonal*)

show *Smith-normal-form-upt-k* (?D-Suck) k

by (rule *Smith-normal-form-upt-k-imp-Suc-k*[OF *ib hyp k*])

show ?D-Suck \$ *from-nat* (k - 1) \$ *from-nat* (k - 1) *dvd* ?D-Suck \$ *from-nat* k \$ *from-nat* k

**proof** (rule *diagonal-to-Smith-aux-dvd*[OF *ib - - - Suc-k*])

show *to-nat* (*from-nat* k::'c) = *to-nat* (*from-nat* k::'b)

by (*metis k min-less-iff-conj ncols-def nrows-def to-nat-from-nat-id*)

show  $k - 1 \leq$  *to-nat* (*from-nat* k::'c)

by (*metis diff-le-self k min-less-iff-conj nrows-def to-nat-from-nat-id*)

qed auto

show *isDiagonal* (diagonal-to-Smith-aux A [0..*Suc k*] bezout)

by (rule *isDiagonal-diagonal-to-Smith-aux*[OF *ib d Suc-k*])

qed

qed

end

**lemma** *nrows-diagonal-to-Smith*[*simp*]: *nrows* (diagonal-to-Smith A bezout) = *nrows* A

**unfolding** *diagonal-to-Smith-def* **by** *auto*

**lemma** *ncols-diagonal-to-Smith[simp]*:  $ncols (diagonal-to-Smith A bezout) = ncols A$

**unfolding** *diagonal-to-Smith-def* **by** *auto*

**lemma** *isDiagonal-diagonal-to-Smith*:

**assumes** *d*: *isDiagonal A*

**shows** *isDiagonal (diagonal-to-Smith A bezout)*

**unfolding** *diagonal-to-Smith-def*

**by** (*rule isDiagonal-diagonal-to-Smith-aux-general[OF - d]*, *auto*)

This is the soundness lemma.

**lemma** *Smith-normal-form-diagonal-to-Smith*:

**fixes** *A*::*'a::{bezout-ring} ^b::mod-type ^c::mod-type*

**assumes** *ib*: *is-bezout-ext bezout*

**and** *d*: *isDiagonal A*

**shows** *Smith-normal-form (diagonal-to-Smith A bezout)*

**proof** –

**let** *?k* =  $min (nrows A) (ncols A) - 2$

**let** *?Dk* = (*diagonal-to-Smith-aux A [0..<?k] bezout*)

**have** *min-eq*:  $min (nrows A) (ncols A) - 1 = Suc ?k$

**by** (*metis Suc-1 Suc-diff-Suc min-less-iff-conj ncols-def nrows-def to-nat-1 to-nat-less-card*)

**have** *set-rw*:  $[0..<min (nrows A) (ncols A) - 1] = [0..<?k] @ [?k]$

**unfolding** *min-eq* **by** *auto*

**have** *d2*: *isDiagonal (diagonal-to-Smith A bezout)*

**by** (*rule isDiagonal-diagonal-to-Smith[OF d]*)

**have** *smith-Suc-k*: *Smith-normal-form-upt-k (diagonal-to-Smith A bezout) (Suc ?k)*

**proof** (*rule Smith-normal-form-upt-k1-intro-diagonal[OF - d2]*)

**have** *diagonal-to-Smith A bezout = diagonal-to-Smith-aux A [0..<min (nrows A) (ncols A) - 1] bezout*

**unfolding** *diagonal-to-Smith-def* **by** *auto*

**also have**  $... = Diagonal-to-Smith-row-i ?Dk ?k bezout$

**unfolding** *set-rw*

**unfolding** *diagonal-to-Smith-aux-append2* **by** *auto*

**finally have** *d-rw*: *diagonal-to-Smith A bezout = Diagonal-to-Smith-row-i ?Dk ?k bezout* .

**have** *Smith-normal-form-upt-k ?Dk ?k*

**by** (*rule Smith-normal-form-upt-k-diagonal-to-Smith-aux[OF ib - d]*, *insert min-eq, linarith*)

**thus** *Smith-normal-form-upt-k (diagonal-to-Smith A bezout) ?k* **unfolding** *d-rw*

**by** (*metis Smith-normal-form-upt-k-Suc-eq Suc-1 ib d-rw diagonal-to-Smith-def diff-0-eq-0*

*diff-less min-eq not-gr-zero zero-less-Suc*)

**show** *diagonal-to-Smith A bezout \$ from-nat (?k - 1) \$ from-nat (?k - 1) dvd diagonal-to-Smith A bezout \$ from-nat ?k \$ from-nat ?k*

```

proof (unfold diagonal-to-Smith-def, rule diagonal-to-Smith-aux-dvd[OF ib])
  show  $?k - 1 < \min (\text{nrows } A) (\text{ncols } A) - 1$ 
    using min-eq by linarith
  show  $\min (\text{nrows } A) (\text{ncols } A) - 1 < \min (\text{nrows } A) (\text{ncols } A)$  using min-eq
by linarith
  thus  $\text{to-nat } (\text{from-nat } ?k::'c) = \text{to-nat } (\text{from-nat } ?k::'b)$ 
    by (metis (mono-tags, lifting) Suc-lessD min-eq min-less-iff-conj
      ncols-def nrows-def to-nat-from-nat-id)
  show  $?k - 1 \leq \text{to-nat } (\text{from-nat } ?k::'c)$ 
    by (metis (no-types, lifting) diff-le-self from-nat-not-eq lessI less-le-trans
      min.cobounded1 min-eq nrows-def)
qed
qed
have s-eq: Smith-normal-form (diagonal-to-Smith A bezout)
  = Smith-normal-form-upt-k (diagonal-to-Smith A bezout)
  (Suc (min (nrows (diagonal-to-Smith A bezout)) (ncols (diagonal-to-Smith A
bezout)) - 1))
  unfolding Smith-normal-form-min by (simp add: ncols-def nrows-def)
  let ?min1=(min (nrows (diagonal-to-Smith A bezout)) (ncols (diagonal-to-Smith
A bezout)) - 1)
  show ?thesis unfolding s-eq
proof (rule Smith-normal-form-upt-k1-intro-diagonal[OF - d2])
  show Smith-normal-form-upt-k (diagonal-to-Smith A bezout) ?min1
    using smith-Suc-k min-eq by auto
  have diagonal-to-Smith A bezout $ from-nat ?k $ from-nat ?k
    dvd diagonal-to-Smith A bezout $ from-nat (?k + 1) $ from-nat (?k + 1)
  by (smt (verit) One-nat-def Suc-eq-plus1 ib Suc-pred diagonal-to-Smith-aux-dvd
diagonal-to-Smith-def
    le-add1 lessI min-eq min-less-iff-conj ncols-def nrows-def to-nat-from-nat-id
zero-less-card-finite)
  thus diagonal-to-Smith A bezout $ from-nat (?min1 - 1) $ from-nat (?min1
- 1)
    dvd diagonal-to-Smith A bezout $ from-nat ?min1 $ from-nat ?min1
  using min-eq by auto
qed
qed

```

## 2.5 Implementation and formal proof of the matrices $P$ and $Q$ which transform the input matrix by means of elementary operations.

```

fun diagonal-step-PQ :: 'a::{bezout-ring} ^'cols::mod-type ^'rows::mod-type  $\Rightarrow$  nat
 $\Rightarrow$  nat  $\Rightarrow$  'a bezout  $\Rightarrow$ 
(
  ('a::{bezout-ring} ^'rows::mod-type ^'rows::mod-type)  $\times$ 
  ('a::{bezout-ring} ^'cols::mod-type ^'cols::mod-type)
)
where diagonal-step-PQ A i k bezout =
  (let i-row = from-nat i; k-row = from-nat k; i-col = from-nat i; k-col = from-nat

```



```

k;
  (p, q, u, v, d) = bezout (A $ i-row $ from-nat i) (A $ k-row $ from-nat k);
  P = row-add (interchange-rows (row-add (mat 1) k-row i-row p) i-row k-row
k-row i-row (-v);
  Q = mult-column (column-add (column-add (mat 1) i-col k-col q) k-col i-col
u) k-col (-1)
  in (P,Q)
)

```

### Examples

```

value let A = list-of-list-to-matrix [[12,0,0::int],[0,6,0::int],[0,0,2::int]]::int33;
  i=0; k=1;
  (p, q, u, v, d) = euclid-ext2 (A $ from-nat i $ from-nat i) (A $ from-nat
k $ from-nat k);
  (P,Q) = diagonal-step-PQ A i k euclid-ext2
  in matrix-to-list-of-list (diagonal-step A i k d v)

```

```

value let A = list-of-list-to-matrix [[12,0,0::int],[0,6,0::int],[0,0,2::int]]::int33;
  i=0; k=1;
  (p, q, u, v, d) = euclid-ext2 (A $ from-nat i $ from-nat i) (A $ from-nat
k $ from-nat k);
  (P,Q) = diagonal-step-PQ A i k euclid-ext2
  in matrix-to-list-of-list (P**(A)**Q)

```

```

value let A = list-of-list-to-matrix [[12,0,0::int],[0,6,0::int],[0,0,2::int]]::int33;
  i=0; k=1;
  (p, q, u, v, d) = euclid-ext2 (A $ from-nat i $ from-nat i) (A $ from-nat
k $ from-nat k);
  (P,Q) = diagonal-step-PQ A i k euclid-ext2
  in matrix-to-list-of-list (P**(A)**Q)

```

**lemmas** *diagonal-step-PQ-def = diagonal-step-PQ.simps*

**lemma** *from-nat-neq-rows:*

**fixes** *A::'a<sup>cols</sup>::mod-type<sup>rows</sup>::mod-type*

**assumes** *i: i < (nrows A) and k: k < (nrows A) and ik: i ≠ k*

**shows** *from-nat i ≠ (from-nat k::'rows)*

**proof** (*rule ccontr, auto*)

**let** *?i=from-nat i::'rows*

**let** *?k=from-nat k::'rows*

**assume** *?i = ?k*

**hence** *to-nat ?i = to-nat ?k by auto*

**hence** *i = k*

**unfolding** *to-nat-from-nat-id[OF i[unfolded nrows-def]]*

**unfolding** *to-nat-from-nat-id[OF k[unfolded nrows-def]] .*

**thus** *False using ik by contradiction*

**qed**

**lemma** *from-nat-neq-cols*:  
**fixes**  $A::'a\ \wedge\ cols::mod\text{-}type\ \wedge\ rows::mod\text{-}type$   
**assumes**  $i: i < (ncols\ A)$  **and**  $k: k < (ncols\ A)$  **and**  $ik: i \neq k$   
**shows**  $from\text{-}nat\ i \neq (from\text{-}nat\ k::'cols)$   
**proof** (*rule ccontr, auto*)  
**let**  $?i=from\text{-}nat\ i::'cols$   
**let**  $?k=from\text{-}nat\ k::'cols$   
**assume**  $?i = ?k$   
**hence**  $to\text{-}nat\ ?i = to\text{-}nat\ ?k$  **by** *auto*  
**hence**  $i = k$   
**unfolding**  $to\text{-}nat\text{-}from\text{-}nat\text{-}id[OF\ i[un\text{-}folded\ ncols\text{-}def]]$   
**unfolding**  $to\text{-}nat\text{-}from\text{-}nat\text{-}id[OF\ k[un\text{-}folded\ ncols\text{-}def]]$  .  
**thus** *False* **using**  $ik$  **by** *contradiction*  
**qed**

**lemma** *diagonal-step-PQ-invertible-P*:  
**fixes**  $A::'a::\{bezout\text{-}ring\}\ \wedge\ cols::mod\text{-}type\ \wedge\ rows::mod\text{-}type$   
**assumes**  $PQ: (P,Q) = diagonal\text{-}step\text{-}PQ\ A\ i\ k\ bezout$   
**and**  $pquvd: (p,q,u,v,d) = bezout\ (A\ \$\ from\text{-}nat\ i\ \$\ from\text{-}nat\ i)\ (A\ \$\ from\text{-}nat\ k\ \$\ from\text{-}nat\ k)$   
**and**  $i\text{-}not\text{-}k: i \neq k$   
**and**  $i: i < min\ (nrows\ A)\ (ncols\ A)$  **and**  $k: k < min\ (nrows\ A)\ (ncols\ A)$   
**shows** *invertible P*  
**proof** –  
**let**  $?step1 = (row\text{-}add\ (mat\ 1)\ (from\text{-}nat\ k::'rows)\ (from\text{-}nat\ i)\ p)$   
**let**  $?step2 = interchange\text{-}rows\ ?step1\ (from\text{-}nat\ i)\ (from\text{-}nat\ k)$   
**let**  $?step3 = row\text{-}add\ (?step2)\ (from\text{-}nat\ k)\ (from\text{-}nat\ i)\ (-\ v)$   
**have**  $p: p = fst\ (bezout\ (A\ \$\ from\text{-}nat\ i\ \$\ from\text{-}nat\ i)\ (A\ \$\ from\text{-}nat\ k\ \$\ from\text{-}nat\ k))$   
**using**  $pquvd$  **by** (*metis fst-conv*)  
**have**  $v: -v = (-\ fst\ (snd\ (snd\ (snd\ (bezout\ (A\ \$\ from\text{-}nat\ i\ \$\ from\text{-}nat\ i)\ (A\ \$\ from\text{-}nat\ k\ \$\ from\text{-}nat\ k))))))$   
**using**  $pquvd$  **by** (*metis fst-conv snd-conv*)  
**have**  $i\text{-}not\text{-}k2: from\text{-}nat\ k \neq (from\text{-}nat\ i::'rows)$   
**by** (*rule from-nat-neq-rows, insert i k i-not-k, auto*)  
**have** *invertible ?step3*  
**unfolding**  $row\text{-}add\text{-}mat\text{-}1[of\ -\ -\ -\ ?step2, symmetric]$   
**proof** (*rule invertible-mult*)  
**show** *invertible (row-add (mat 1) (from-nat k::'rows) (from-nat i) (- v))*  
**by** (*rule invertible-row-add[OF i-not-k2]*)  
**show** *invertible ?step2*  
**by** (*metis i-not-k2 interchange-rows-mat-1 invertible-interchange-rows invertible-mult invertible-row-add*)  
**qed**  
**thus** *?thesis*

```

    using PQ p v unfolding diagonal-step-PQ-def Let-def split-beta
    by auto
qed

```

```

lemma diagonal-step-PQ-invertible-Q:
  fixes A::'a::{bezout-ring} ^ cols::mod-type ^ rows::mod-type
  assumes PQ: (P,Q) = diagonal-step-PQ A i k bezout
  and pqvd: (p,q,u,v,d) = bezout (A $ from-nat i $ from-nat i) (A $ from-nat k
  $ from-nat k)
  and i-not-k: i ≠ k
  and i: i < min (nrows A) (ncols A) and k: k < min (nrows A) (ncols A)
shows invertible Q
proof -
  let ?step1 = column-add (mat 1) (from-nat i::'cols) (from-nat k) q
  let ?step2 = column-add ?step1 (from-nat k) (from-nat i) u
  let ?step3 = mult-column ?step2 (from-nat k) (- 1)
  have u: u = (fst (snd (snd (bezout (A $ from-nat i $ from-nat i) (A $ from-nat
  k $ from-nat k))))))
    by (metis fst-conv pqvd snd-conv)
  have q: q = (fst (snd (bezout (A $ from-nat i $ from-nat i) (A $ from-nat k $
  from-nat k))))
    by (metis fst-conv pqvd snd-conv)
  have invertible ?step3
    unfolding column-add-mat-1[of - - - ?step2, symmetric]
    unfolding mult-column-mat-1[of ?step2, symmetric]
  proof (rule invertible-mult)
    show invertible (mult-column (mat 1) (from-nat k::'cols) (- 1::'a))
      by (rule invertible-mult-column[of - - 1], auto)
    show invertible ?step2
      by (metis column-add-mat-1 i i-not-k invertible-column-add invertible-mult k
      min-less-iff-conj ncols-def to-nat-from-nat-id)
  qed
  thus ?thesis
    using PQ pqvd u q unfolding diagonal-step-PQ-def
    by (auto simp add: Let-def split-beta)
qed

```

```

lemma mat-q-1[simp]: mat q $ a $ a = q unfolding mat-def by auto

```

```

lemma mat-q-0[simp]:
  assumes ab: a ≠ b
  shows mat q $ a $ b = 0 using ab unfolding mat-def by auto

```

This is an alternative definition for the matrix P in each step, where entries are given explicitly instead of being computed as a composition of elementary operations.

```

lemma diagonal-step-PQ-P-alt:

```

```

fixes A::'a::{bezout-ring}^'cols::mod-type^'rows::mod-type
  assumes PQ: (P,Q) = diagonal-step-PQ A i k bezout
  and pqvd: (p,q,u,v,d) = bezout (A $ from-nat i $ from-nat i) (A $ from-nat k
$ from-nat k)
  and i: i < min (nrows A) (ncols A) and k: k < min (nrows A) (ncols A) and ik: i
≠ k
shows
  P = (χ a b.
    if a = from-nat i ∧ b = from-nat i then p else
    if a = from-nat i ∧ b = from-nat k then 1 else
    if a = from-nat k ∧ b = from-nat i then -v * p + 1 else
    if a = from-nat k ∧ b = from-nat k then -v else
    if a = b then 1 else 0)
proof -
  have ik1: from-nat i ≠ (from-nat k::'rows)
    using from-nat-neq-rows i ik k by auto
  have P $ a $ b =
    (if a = from-nat i ∧ b = from-nat i then p
     else if a = from-nat i ∧ b = from-nat k then 1
     else if a = from-nat k ∧ b = from-nat i then - v * p + 1
     else if a = from-nat k ∧ b = from-nat k then - v else if a = b
then 1 else 0)
  for a b
    using PQ ik1 pqvd
    unfolding diagonal-step-PQ-def
    unfolding row-add-def interchange-rows-def
    by (auto simp add: Let-def split-beta)
    (metis (mono-tags, opaque-lifting) fst-conv snd-conv)+
  thus ?thesis unfolding vec-eq-iff unfolding vec-lambda-beta by auto
qed

```

This is an alternative definition for the matrix Q in each step, where entries are given explicitly instead of being computed as a composition of elementary operations.

**lemma** *diagonal-step-PQ-Q-alt*:

```

fixes A::'a::{bezout-ring}^'cols::mod-type^'rows::mod-type
  assumes PQ: (P,Q) = diagonal-step-PQ A i k bezout
  and pqvd: (p,q,u,v,d) = bezout (A $ from-nat i $ from-nat i) (A $ from-nat k
$ from-nat k)
  and i: i < min (nrows A) (ncols A) and k: k < min (nrows A) (ncols A) and ik: i
≠ k
shows
  Q = (χ a b.
    if a = from-nat i ∧ b = from-nat i then 1 else
    if a = from-nat i ∧ b = from-nat k then -u else
    if a = from-nat k ∧ b = from-nat i then q else
    if a = from-nat k ∧ b = from-nat k then -q*u-1 else
    if a = b then 1 else 0)
proof -

```

```

have ik1: from-nat i ≠ (from-nat k::'cols)
  using from-nat-neq-cols i ik k by auto
have Q $ a $ b =
  (if a = from-nat i ∧ b = from-nat i then 1 else
  if a = from-nat i ∧ b = from-nat k then -u else
  if a = from-nat k ∧ b = from-nat i then q else
  if a = from-nat k ∧ b = from-nat k then -q*u-1 else
  if a = b then 1 else 0) for a b
using PQ ik1 pqvd unfolding diagonal-step-PQ-def
unfolding column-add-def mult-column-def
by (auto simp add: Let-def split-beta)
  (metis (mono-tags, opaque-lifting) fst-conv snd-conv)+
thus ?thesis unfolding vec-eq-iff unfolding vec-lambda-beta by auto
qed

```

P\*\*A can be rewritten as elementary operations over A.

**lemma** diagonal-step-PQ-PA:

```

fixes A::'a::{bezout-ring}~^cols::mod-type~^rows::mod-type
assumes PQ: (P,Q) = diagonal-step-PQ A i k bezout
  and b: (p,q,u,v,d) = bezout (A $ from-nat i $ from-nat i) (A $ from-nat k $
  from-nat k)
shows P**A = row-add (interchange-rows
  (row-add A (from-nat k) (from-nat i) p) (from-nat i) (from-nat k)) (from-nat k)
  (from-nat i) (- v)
proof -
  let ?i-row = from-nat i::'rows and ?k-row = from-nat k::'rows
  let ?P1 = row-add (mat 1) ?k-row ?i-row p
  let ?P2' = interchange-rows ?P1 ?i-row ?k-row
  let ?P2 = interchange-rows (mat 1) (from-nat i) (from-nat k)
  let ?P3 = row-add (mat 1) (from-nat k) (from-nat i) (- v)
  have P = row-add ?P2' ?k-row ?i-row (- v)
    using PQ b unfolding diagonal-step-PQ-def
    by (auto simp add: Let-def split-beta, metis fstI sndI)
  also have ... = ?P3 ** ?P2'
    unfolding row-add-mat-1[of - - - ?P2', symmetric] by auto
  also have ... = ?P3 ** (?P2 ** ?P1)
    unfolding interchange-rows-mat-1[of - - ?P1, symmetric] by auto
  also have ... ** A = row-add (interchange-rows
  (row-add A (from-nat k) (from-nat i) p) (from-nat i) (from-nat k)) (from-nat k)
  (from-nat i) (- v)
    by (metis interchange-rows-mat-1 matrix-mul-assoc row-add-mat-1)
  finally show ?thesis .
qed

```

**lemma** diagonal-step-PQ-PAQ':

```

fixes A::'a::{bezout-ring}~^cols::mod-type~^rows::mod-type
assumes PQ: (P,Q) = diagonal-step-PQ A i k bezout
  and b: (p,q,u,v,d) = bezout (A $ from-nat i $ from-nat i) (A $ from-nat k $

```

*from-nat k*)  
**shows**  $P**A**Q = (\text{mult-column } (\text{column-add } (\text{column-add } (P**A) (\text{from-nat } i) (\text{from-nat } k) q) (\text{from-nat } k) (\text{from-nat } i) u) (\text{from-nat } k) (- 1))$   
**proof** –  
**let**  $?i\text{-col} = \text{from-nat } i::'\text{cols}$  **and**  $?k\text{-col} = \text{from-nat } k::'\text{cols}$   
**let**  $?Q1 = (\text{column-add } (\text{mat } 1) ?i\text{-col } ?k\text{-col } q)$   
**let**  $?Q2' = (\text{column-add } ?Q1 ?k\text{-col } ?i\text{-col } u)$   
**let**  $?Q2 = \text{column-add } (\text{mat } 1) (\text{from-nat } k) (\text{from-nat } i) u$   
**let**  $?Q3 = \text{mult-column } (\text{mat } 1) (\text{from-nat } k) (- 1)$   
**have**  $Q = \text{mult-column } ?Q2' ?k\text{-col } (-1)$   
**using**  $PQ$  **b unfolding** *diagonal-step-PQ-def*  
**by** (*auto simp add: Let-def split-beta, metis fstI sndI*)  
**also have**  $\dots = ?Q2' ** ?Q3$   
**unfolding** *mult-column-mat-1* [*of ?Q2', symmetric*] **by** *auto*  
**also have**  $\dots = (?Q1**?Q2)**?Q3$   
**unfolding** *column-add-mat-1* [*of ?Q1, symmetric*] **by** *auto*  
**also have**  $(P**A) ** ((?Q1**?Q2)**?Q3) =$   
 $(\text{mult-column } (\text{column-add } (\text{column-add } (P**A) ?i\text{-col } ?k\text{-col } q) ?k\text{-col } ?i\text{-col } u)$   
 $?k\text{-col } (- 1))$   
**by** (*metis (no-types, lifting) column-add-mat-1 matrix-mul-assoc mult-column-mat-1*)  
**finally show** *?thesis* .  
**qed**

**corollary** *diagonal-step-PQ-PAQ*:

**fixes**  $A::'a::\{\text{bezout-ring}\} \wedge \text{cols}::\text{mod-type} \wedge \text{rows}::\text{mod-type}$   
**assumes**  $PQ: (P,Q) = \text{diagonal-step-PQ } A \ i \ k \ \text{bezout}$   
**and**  $b: (p,q,u,v,d) = \text{bezout } (A \ \$ \ \text{from-nat } i \ \$ \ \text{from-nat } i) (A \ \$ \ \text{from-nat } k \ \$ \ \text{from-nat } k)$   
**shows**  $P**A**Q = (\text{mult-column } (\text{column-add } (\text{column-add } (\text{row-add } (\text{interchange-rows } (\text{row-add } A (\text{from-nat } k) (\text{from-nat } i) p) (\text{from-nat } i) (\text{from-nat } k)) (\text{from-nat } k) (\text{from-nat } i) (- v)) (\text{from-nat } i) (\text{from-nat } k) q) (\text{from-nat } k) (\text{from-nat } i) u) (\text{from-nat } k) (- 1))$   
**using** *diagonal-step-PQ-PA diagonal-step-PQ-PAQ' assms* **by** *metis*

**lemma** *isDiagonal-imp-0*:

**assumes** *isDiagonal A*  
**and**  $\text{from-nat } a \neq \text{from-nat } b$   
**and**  $a < \min (\text{nrows } A) (\text{ncols } A)$   
**and**  $b < \min (\text{nrows } A) (\text{ncols } A)$   
**shows**  $A \ \$ \ \text{from-nat } a \ \$ \ \text{from-nat } b = 0$   
**by** (*metis assms isDiagonal min.strict-boundedE ncols-def nrows-def to-nat-from-nat-id*)

**lemma** *diagonal-step-PQ*:

**fixes**  $A::'a::\{\text{bezout-ring}\} \wedge \text{cols}::\text{mod-type} \wedge \text{rows}::\text{mod-type}$

```

assumes  $PQ$ :  $(P, Q) = \text{diagonal-step-PQ } A \ i \ k \ \text{bezout}$ 
and  $b$ :  $(p, q, u, v, d) = \text{bezout } (A \ \$ \ \text{from-nat } i \ \$ \ \text{from-nat } i) \ (A \ \$ \ \text{from-nat } k \ \$ \ \text{from-nat } k)$ 
and  $i$ :  $i < \min (\text{nrows } A) (\text{ncols } A)$  and  $k$ :  $k < \min (\text{nrows } A) (\text{ncols } A)$  and  $ik$ :  $i \neq k$ 
and  $ib$ :  $\text{is-bezout-ext } \text{bezout}$  and  $\text{diag}$ :  $\text{isDiagonal } A$ 
shows  $\text{diagonal-step } A \ i \ k \ d \ v = P ** A ** Q$ 
proof –
  let  $?i\text{-row} = \text{from-nat } i :: \text{'rows}$ 
    and  $?k\text{-row} = \text{from-nat } k :: \text{'rows}$  and  $?i\text{-col} = \text{from-nat } i :: \text{'cols}$  and  $?k\text{-col} = \text{from-nat } k :: \text{'cols}$ 
    let  $?P1 = (\text{row-add } (\text{mat } 1) \ ?k\text{-row} \ ?i\text{-row} \ p)$ 
    let  $?Aii = A \ \$ \ ?i\text{-row} \ \$ \ ?i\text{-col}$ 
    let  $?Akk = A \ \$ \ ?k\text{-row} \ \$ \ ?k\text{-col}$ 
    have  $k1$ :  $k < \text{ncols } A$  and  $k2$ :  $k < \text{nrows } A$  and  $i1$ :  $i < \text{nrows } A$  and  $i2$ :  $i < \text{ncols } A$ 
using  $i \ k$  by  $\text{auto}$ 
    have  $Aik0$ :  $A \ \$ \ ?i\text{-row} \ \$ \ ?k\text{-col} = 0$ 
      by  $(\text{metis } \text{diag } i \ k \ \text{isDiagonal } k \ \text{min.strict-boundedE } \text{ncols-def } \text{nrows-def } \text{to-nat-from-nat-id})$ 
    have  $Aki0$ :  $A \ \$ \ ?k\text{-row} \ \$ \ ?i\text{-col} = 0$ 
      by  $(\text{metis } \text{diag } i \ k \ \text{isDiagonal } k \ \text{min.strict-boundedE } \text{ncols-def } \text{nrows-def } \text{to-nat-from-nat-id})$ 
    have  $du$ :  $d * u = - A \ \$ \ ?k\text{-row} \ \$ \ ?k\text{-col}$ 
      using  $b \ ib$  unfolding  $\text{is-bezout-ext-def}$ 
      by  $(\text{auto } \text{simp } \text{add: split-beta}) \ (\text{metis } \text{fst-conv } \text{snd-conv})$ 
    have  $dv$ :  $d * v = A \ \$ \ ?i\text{-row} \ \$ \ ?i\text{-col}$ 
      using  $b \ ib$  unfolding  $\text{is-bezout-ext-def}$ 
      by  $(\text{auto } \text{simp } \text{add: split-beta}) \ (\text{metis } \text{fst-conv } \text{snd-conv})$ 
    have  $d$ :  $d = p * ?Aii + ?Akk * q$ 
      using  $b \ ib$  unfolding  $\text{is-bezout-ext-def}$ 
      by  $(\text{auto } \text{simp } \text{add: split-beta}) \ (\text{metis } \text{fst-conv } \text{mult.commute } \text{snd-conv})$ 
    have  $(?Aii - v * (p * ?Aii) - v * ?Akk * q) * u = (?Aii - v * ((p * ?Aii) + ?Akk * q)) * u$ 
      by  $(\text{simp } \text{add: diff-diff-add } \text{distrib-left } \text{mult.assoc})$ 
    also have  $\dots = (?Aii * u - d * v * u)$ 
      by  $(\text{simp } \text{add: mult.commute } \text{right-diff-distrib } d)$ 
    also have  $\dots = 0$  by  $(\text{simp } \text{add: dv})$ 
    finally have  $rw$ :  $(?Aii - v * (p * ?Aii) - v * ?Akk * q) * u = 0$  .
    have  $a1$ :  $\text{from-nat } k \neq (\text{from-nat } i :: \text{'rows})$ 
      using  $\text{from-nat-neq-rows } i \ ik \ k$  by  $\text{auto}$ 
    have  $a2$ :  $\text{from-nat } k \neq (\text{from-nat } i :: \text{'cols})$ 
      using  $\text{from-nat-neq-cols } i \ ik \ k$  by  $\text{auto}$ 
    have  $Aab0$ :  $A \ \$ \ a \ \$ \ \text{from-nat } b = 0$  if  $ab$ :  $a \neq \text{from-nat } b$  and  $b\text{-ncols}$ :  $b < \text{ncols } A$  for  $a \ b$ 
      by  $(\text{metis } ab \ b\text{-ncols } \text{diag } \text{from-nat-to-nat-id } \text{isDiagonal } \text{ncols-def } \text{to-nat-from-nat-id})$ 

    have  $Aab0'$ :  $A \ \$ \ \text{from-nat } a \ \$ \ b = 0$  if  $ab$ :  $\text{from-nat } a \neq b$  and  $a\text{-nrows}$ :  $a < \text{nrows } A$  for  $a \ b$ 
      by  $(\text{metis } ab \ a\text{-nrows } \text{diag } \text{from-nat-to-nat-id } \text{isDiagonal } \text{nrows-def } \text{to-nat-from-nat-id})$ 
    show  $?thesis$ 
    proof  $(\text{unfold } \text{diagonal-step-def } \text{vec-eq-iff}, \text{ auto})$ 

```

```

show d = (P ** A ** Q) $ from-nat i $ from-nat i
  and d = (P ** A ** Q) $ from-nat i $ from-nat i
  and d = (P ** A ** Q) $ from-nat i $ from-nat i
unfolding diagonal-step-PQ-PAQ[OF PQ b]
unfolding mult-column-def column-add-def interchange-rows-def row-add-def
  unfolding vec-lambda-beta using a1 a2
  using Aik0 Aki0 d by auto
show v * A $ from-nat k $ from-nat k = (P ** A ** Q) $ from-nat k $ from-nat
k
  and v * A $ from-nat k $ from-nat k = (P ** A ** Q) $ from-nat k $ from-nat
k
  using a1 a2
  unfolding diagonal-step-PQ-PAQ[OF PQ b] mult-column-def column-add-def

  unfolding interchange-rows-def row-add-def
  unfolding vec-lambda-beta unfolding Aik0 Aki0 by (auto simp add: rw)
fix a::'rows and b::'cols
assume ak: a ≠ from-nat k and ai: a ≠ from-nat i
show A $ a $ b = (P ** A ** Q) $ a $ b
  using ai ak a1 a2 Aab0 k1 i2
  unfolding diagonal-step-PQ-PAQ[OF PQ b]
  unfolding mult-column-def column-add-def interchange-rows-def row-add-def
  unfolding vec-lambda-beta by auto
next
fix a::'rows and b::'cols
assume ak: a ≠ from-nat k and ai: b ≠ from-nat i
show A $ a $ b = (P ** A ** Q) $ a $ b
  using ai ak a1 a2 Aab0 Aab0' d du k1 k2 i1 i2
  unfolding diagonal-step-PQ-PAQ[OF PQ b]
  unfolding mult-column-def column-add-def interchange-rows-def row-add-def
  unfolding vec-lambda-beta by auto
next
fix a::'rows and b::'cols
assume ak: b ≠ from-nat k and ai: a ≠ from-nat i
show A $ a $ b = (P ** A ** Q) $ a $ b
  using ai ak a1 a2 Aab0 Aab0' d du k1 k2 i1 i2
  unfolding diagonal-step-PQ-PAQ[OF PQ b]
  unfolding mult-column-def column-add-def interchange-rows-def row-add-def
  unfolding vec-lambda-beta apply auto
proof -
  assume d = p * ?Aii+ ?Akk* q
  then have v * (p * ?Aii) + v * (?Akk* q) = d * v
    by (simp add: ring-class.ring-distrib(1) semiring-normalization-rules(7))
  then have ?Aii- v * (p * ?Aii) - v * (?Akk* q) = 0
    by (simp add: diff-diff-add dv)
  then show ?Aii- v * (p * ?Aii) = v * ?Akk* q
    by force
qed
next

```



```

fix a::'rows and b::'cols
assume ak: b ≠ from-nat k and ai: b ≠ from-nat i
show A $ a $ b = (P ** A ** Q) $ a $ b
  using ai ak a1 a2 Aab0 Aab0' d du k1 k2 i1 i2
  unfolding diagonal-step-PQ-PAQ[OF PQ b]
  unfolding mult-column-def column-add-def interchange-rows-def row-add-def
  unfolding vec-lambda-beta by auto
qed
qed

```

```

fun diagonal-to-Smith-i-PQ ::
  nat list ⇒ nat ⇒ ('a::{bezout-ring} bezout)
  ⇒ (('a ^rows::mod-type ^rows::mod-type) × ('a ^cols::mod-type ^rows::mod-type) ×
  ('a ^cols::mod-type ^cols::mod-type))
  ⇒ (('a ^rows::mod-type ^rows::mod-type) × ('a ^cols::mod-type ^rows::mod-type)
  × ('a ^cols::mod-type ^cols::mod-type))
  where
  diagonal-to-Smith-i-PQ [] i bezout (P,A,Q) = (P,A,Q) |
  diagonal-to-Smith-i-PQ (j#xs) i bezout (P,A,Q) = (
    if A $ (from-nat i) $ (from-nat i) dvd A $ (from-nat j) $ (from-nat j)
    then diagonal-to-Smith-i-PQ xs i bezout (P,A,Q)
    else let (p, q, u, v, d) = bezout (A $ from-nat i $ from-nat i) (A $ from-nat j $
    from-nat j);
      A' = diagonal-step A i j d v;
      (P',Q') = diagonal-step-PQ A i j bezout
    in diagonal-to-Smith-i-PQ xs i bezout (P'**P,A',Q**Q') — Apply the step
  )

```

This is implemented by fun. This way, I can do pattern-matching for  $(P, A, Q)$ .

```

fun Diagonal-to-Smith-row-i-PQ
  where Diagonal-to-Smith-row-i-PQ i bezout (P,A,Q)
  = diagonal-to-Smith-i-PQ [i + 1..<min (nrows A) (ncols A)] i bezout (P,A,Q)

```

Deleted from the simplified and renamed as it would be a definition.

```

declare Diagonal-to-Smith-row-i-PQ.simps[simp del]
lemmas Diagonal-to-Smith-row-i-PQ-def = Diagonal-to-Smith-row-i-PQ.simps

```

```

fun diagonal-to-Smith-aux-PQ
  where
  diagonal-to-Smith-aux-PQ [] bezout (P,A,Q) = (P,A,Q) |
  diagonal-to-Smith-aux-PQ (i#xs) bezout (P,A,Q)
  = diagonal-to-Smith-aux-PQ xs bezout (Diagonal-to-Smith-row-i-PQ i bezout
  (P,A,Q))

```

**lemma** *diagonal-to-Smith-aux-PQ-append*:  
*diagonal-to-Smith-aux-PQ* (*xs* @ *ys*) *bezout* (*P,A,Q*)  
= *diagonal-to-Smith-aux-PQ* *ys* *bezout* (*diagonal-to-Smith-aux-PQ* *xs* *bezout*  
(*P,A,Q*))  
**by** (*induct xs bezout* (*P,A,Q*) *arbitrary: P A Q rule: diagonal-to-Smith-aux-PQ.induct*)  
(*auto, metis prod-cases3*)

**lemma** *diagonal-to-Smith-aux-PQ-append2[simp]*:  
*diagonal-to-Smith-aux-PQ* (*xs* @ [*ys*]) *bezout* (*P,A,Q*)  
= *Diagonal-to-Smith-row-i-PQ* *ys* *bezout* (*diagonal-to-Smith-aux-PQ* *xs* *bezout*  
(*P,A,Q*))  
**proof** (*induct xs bezout* (*P,A,Q*) *arbitrary: P A Q rule: diagonal-to-Smith-aux-PQ.induct*)  
**case** (*1 bezout P A Q*)  
**then show** ?*case*  
**by** (*metis append.simps(1) diagonal-to-Smith-aux-PQ.simps prod.exhaust*)  
**next**  
**case** (*2 i xs bezout P A Q*)  
**then show** ?*case*  
**by** (*metis (no-types, opaque-lifting) append-Cons diagonal-to-Smith-aux-PQ.simps(2)*  
*prod-cases3*)  
**qed**

**context**  
**fixes** *A::'a::{\bezout-ring}^{\wedge}cols::mod-type^{\wedge}rows::mod-type*  
**and** *B::'a::{\bezout-ring}^{\wedge}cols::mod-type^{\wedge}rows::mod-type*  
**and** *P* **and** *Q*  
**and** *bezout::'a bezout*  
**assumes** *PAQ: P\*\*A\*\*Q = B*  
**and** *P: invertible P* **and** *Q: invertible Q*  
**and** *ib: is-bezout-ext bezout*  
**begin**

The output is the same as the one in the version where *P* and *Q* are not computed.

**lemma** *diagonal-to-Smith-i-PQ-eq*:  
**assumes** *P'B'Q': (P',B',Q') = diagonal-to-Smith-i-PQ xs i bezout (P,B,Q)*  
**and** *xs:  $\forall x. x \in \text{set } xs \longrightarrow x < \min(\text{nrows } A) (\text{ncols } A)$*   
**and** *diag: isDiagonal B* **and** *i-notin:  $i \notin \text{set } xs$*  **and** *i:  $i < \min(\text{nrows } A) (\text{ncols } A)$*   
**shows** *B' = diagonal-to-Smith-i xs B i bezout*  
**using** *assms PAQ ib P Q*  
**proof** (*induct xs i bezout (P,B,Q) arbitrary: P B Q rule:diagonal-to-Smith-i-PQ.induct*)  
**case** (*1 i bezout P A Q*)  
**then show** ?*case by auto*  
**next**  
**case** (*2 j xs i bezout P B Q*)

```

let ?Bii = B $ from-nat i $ from-nat i
let ?Bjj = B $ from-nat j $ from-nat j
let ?p=case bezout (B $ from-nat i $ from-nat i) (B $ from-nat j $ from-nat j)
of (p,q,u,v,d) => p
let ?q=case bezout (B $ from-nat i $ from-nat i) (B $ from-nat j $ from-nat j)
of (p,q,u,v,d) => q
let ?u=case bezout (B $ from-nat i $ from-nat i) (B $ from-nat j $ from-nat j)
of (p,q,u,v,d) => u
let ?v=case bezout (B $ from-nat i $ from-nat i) (B $ from-nat j $ from-nat j)
of (p,q,u,v,d) => v
let ?d=case bezout (B $ from-nat i $ from-nat i) (B $ from-nat j $ from-nat j)
of (p,q,u,v,d) => d
let ?B'=diagonal-step B i j ?d ?v
let ?P' = fst (diagonal-step-PQ B i j bezout)
let ?Q' = snd (diagonal-step-PQ B i j bezout)
have pqvud: (?p, ?q, ?u, ?v,?d) = bezout (B $ from-nat i $ from-nat i) (B $
from-nat j $ from-nat j)
by (simp add: split-beta)
note hyp = 2.hyps(2)
note P'B'Q' = 2.prem(1)
note i-min = 2.prem(5)
note PAQ-B = 2.prem(6)
note i-notin = 2.prem(4)
note diagB = 2.prem(3)
note xs-min = 2.prem(2)
note ib = 2.prem(7)
note inv-P = 2.prem(8)
note inv-Q = 2.prem(9)
show ?case
proof (cases ?Bii dvd ?Bjj)
case True
show ?thesis using 2.prem 2.hyps(1) True by auto
next
case False
have aux: diagonal-to-Smith-i-PQ (j # xs) i bezout (P, B, Q)
= diagonal-to-Smith-i-PQ xs i bezout (?P'**P, ?B', Q**?Q')
using False by (auto simp add: split-beta)
have i: i < min (nrows B) (ncols B) using i-min unfolding nrows-def ncols-def
by auto
have j: j < min (nrows B) (ncols B) using xs-min unfolding nrows-def
ncols-def by auto
have aux2: diagonal-to-Smith-i(j # xs) B i bezout = diagonal-to-Smith-i xs ?B'
i bezout
using False by (auto simp add: split-beta)
have res: B' = diagonal-to-Smith-i xs ?B' i bezout
proof (rule hyp[OF False])
show (P', B', Q') = diagonal-to-Smith-i-PQ xs i bezout (?P'**P, ?B', Q**?Q')

using aux P'B'Q' by auto

```

```

have  $B'-P'B'Q'$ :  $?B' = ?P' ** B ** ?Q'$ 
  by (rule diagonal-step-PQ[OF - - i j - ib diagB], insert i-notin pqvd, auto)
show  $?P' ** P ** A ** (Q ** ?Q') = ?B'$ 
  unfolding  $B'-P'B'Q'$  unfolding PAQ-B[symmetric]
  by (simp add: matrix-mul-assoc)
show isDiagonal  $?B'$  by (rule isDiagonal-diagonal-step[OF diagB i j])
show invertible ( $?P' ** P$ )
  by (metis inv-P diagonal-step-PQ-invertible-P i i-notin in-set-member
    invertible-mult j member-rec(1) prod.exhaust-sel)
show invertible ( $Q ** ?Q'$ )
  by (metis diagonal-step-PQ-invertible-Q i i-notin inv-Q
    invertible-mult j list.set-intros(1) prod.collapse)
qed (insert pqvd xs-min i-min i-notin ib, auto)
show ?thesis using aux aux2 res by auto
qed
qed

```

```

lemma diagonal-to-Smith-i-PQ':
  assumes  $P'B'Q'$ :  $(P', B', Q') = \text{diagonal-to-Smith-i-PQ } xs \ i \ \text{bezout } (P, B, Q)$ 
  and  $xs$ :  $\forall x. x \in \text{set } xs \longrightarrow x < \min(\text{nrows } A) (\text{ncols } A)$ 
  and  $diag$ : isDiagonal  $B$  and  $i$ -notin:  $i \notin \text{set } xs$  and  $i$ :  $i < \min(\text{nrows } A) (\text{ncols } A)$ 
shows  $B' = P' ** A ** Q' \wedge \text{invertible } P' \wedge \text{invertible } Q'$ 
  using assms PAQ ib P Q
proof (induct  $xs \ i \ \text{bezout } (P, B, Q)$  arbitrary: P B Q rule:diagonal-to-Smith-i-PQ.induct)
  case (1  $i \ \text{bezout}$ )
    then show ?case using PAQ by auto
  next
    case (2  $j \ xs \ i \ \text{bezout } P \ B \ Q$ )
      let  $?Bii = B \ \$ \ \text{from-nat } i \ \$ \ \text{from-nat } i$ 
      let  $?Bjj = B \ \$ \ \text{from-nat } j \ \$ \ \text{from-nat } j$ 
      let  $?p = \text{case bezout } (B \ \$ \ \text{from-nat } i \ \$ \ \text{from-nat } i) (B \ \$ \ \text{from-nat } j \ \$ \ \text{from-nat } j)$ 
      of  $(p, q, u, v, d) \Rightarrow p$ 
      let  $?q = \text{case bezout } (B \ \$ \ \text{from-nat } i \ \$ \ \text{from-nat } i) (B \ \$ \ \text{from-nat } j \ \$ \ \text{from-nat } j)$ 
      of  $(p, q, u, v, d) \Rightarrow q$ 
      let  $?u = \text{case bezout } (B \ \$ \ \text{from-nat } i \ \$ \ \text{from-nat } i) (B \ \$ \ \text{from-nat } j \ \$ \ \text{from-nat } j)$ 
      of  $(p, q, u, v, d) \Rightarrow u$ 
      let  $?v = \text{case bezout } (B \ \$ \ \text{from-nat } i \ \$ \ \text{from-nat } i) (B \ \$ \ \text{from-nat } j \ \$ \ \text{from-nat } j)$ 
      of  $(p, q, u, v, d) \Rightarrow v$ 
      let  $?d = \text{case bezout } (B \ \$ \ \text{from-nat } i \ \$ \ \text{from-nat } i) (B \ \$ \ \text{from-nat } j \ \$ \ \text{from-nat } j)$ 
      of  $(p, q, u, v, d) \Rightarrow d$ 
      let  $?B' = \text{diagonal-step } B \ i \ j \ ?d \ ?v$ 
      let  $?P' = \text{fst } (\text{diagonal-step-PQ } B \ i \ j \ \text{bezout})$ 
      let  $?Q' = \text{snd } (\text{diagonal-step-PQ } B \ i \ j \ \text{bezout})$ 
      have  $pqvd$ :  $(?p, ?q, ?u, ?v, ?d) = \text{bezout } (B \ \$ \ \text{from-nat } i \ \$ \ \text{from-nat } i) (B \ \$ \ \text{from-nat } j \ \$ \ \text{from-nat } j)$ 
      by (simp add: split-beta)
      show ?case

```

```

proof (cases ?Bii dvd ?Bjj)
  case True
  then show ?thesis using 2.prem.s
    using 2.hyps(1) by auto
next
  case False
  note hyp = 2.hyps(2)
  note P'B'Q' = 2.prem.s(1)
  note i-min = 2.prem.s(5)
  note PAQ-B = 2.prem.s(6)
  note i-notin = 2.prem.s(4)
  note diagB = 2.prem.s(3)
  note xs-min = 2.prem.s(2)
  note ib = 2.prem.s(7)
  note inv-P = 2.prem.s(8)
  note inv-Q = 2.prem.s(9)
  have aux: diagonal-to-Smith-i-PQ (j # xs) i bezout (P, B, Q)
    = diagonal-to-Smith-i-PQ xs i bezout (?P'**P, ?B', Q**?Q')
    using False by (auto simp add: split-beta)
  have i: i < min (nrows B) (ncols B) using i-min unfolding nrows-def ncols-def
by auto
  have j: j < min (nrows B) (ncols B) using xs-min unfolding nrows-def
ncols-def by auto
  show ?thesis
  proof (rule hyp[OF False])
  show (P', B', Q') = diagonal-to-Smith-i-PQ xs i bezout (?P'**P, ?B', Q**?Q')

    using aux P'B'Q' by auto
  have B'-P'B'Q': ?B' = ?P'**B**?Q'
    by (rule diagonal-step-PQ[OF - - i j - ib diagB], insert i-notin pquvd, auto)
  show ?P'**P ** A ** (Q**?Q') = ?B'
    unfolding B'-P'B'Q' unfolding PAQ-B[symmetric]
    by (simp add: matrix-mul-assoc)
  show isDiagonal ?B' by (rule isDiagonal-diagonal-step[OF diagB i j])
  show invertible (?P'** P)
    by (metis inv-P diagonal-step-PQ-invertible-P i i-notin in-set-member
invertible-mult j member-rec(1) prod.exhaust-sel)
  show invertible (Q ** ?Q')
    by (metis diagonal-step-PQ-invertible-Q i i-notin inv-Q
invertible-mult j list.set-intros(1) prod.collapse)
  qed (insert pquvd xs-min i-min i-notin ib, auto)
qed
qed

```

**corollary** diagonal-to-Smith-i-PQ:

```

assumes P'B'Q': (P', B', Q') = diagonal-to-Smith-i-PQ xs i bezout (P, B, Q)
and xs:  $\forall x. x \in \text{set } xs \longrightarrow x < \min (\text{nrows } A) (\text{ncols } A)$ 
and diag: isDiagonal B and i-notin:  $i \notin \text{set } xs$  and i:  $i < \min (\text{nrows } A) (\text{ncols } A)$ 

```

A)

**shows**  $B' = P' ** A ** Q' \wedge \text{invertible } P' \wedge \text{invertible } Q' \wedge B' = \text{diagonal-to-Smith-}i$   
*xs B i bezout*

**using** *assms diagonal-to-Smith-i-PQ' diagonal-to-Smith-i-PQ-eq* **by** *metis*

**lemma** *Diagonal-to-Smith-row-i-PQ-eq:*

**assumes**  $P'B'Q': (P', B', Q') = \text{Diagonal-to-Smith-row-i-PQ } i \text{ bezout } (P, B, Q)$   
**and** *diag: isDiagonal B and i: i < min (nrows A) (ncols A)*

**shows**  $B' = \text{Diagonal-to-Smith-row-i } B \text{ i bezout}$

**using** *assms unfolding Diagonal-to-Smith-row-i-def Diagonal-to-Smith-row-i-PQ-def*  
**using** *diagonal-to-Smith-i-PQ* **by** *(auto simp add: nrows-def ncols-def)*

**lemma** *Diagonal-to-Smith-row-i-PQ':*

**assumes**  $P'B'Q': (P', B', Q') = \text{Diagonal-to-Smith-row-i-PQ } i \text{ bezout } (P, B, Q)$   
**and** *diag: isDiagonal B and i: i < min (nrows A) (ncols A)*

**shows**  $B' = P' ** A ** Q' \wedge \text{invertible } P' \wedge \text{invertible } Q'$

**by** *(rule diagonal-to-Smith-i-PQ'[OF P'B'Q'[unfolded Diagonal-to-Smith-row-i-PQ-def]*  
*- diag - i],*  
*auto simp add: nrows-def ncols-def)*

**lemma** *Diagonal-to-Smith-row-i-PQ:*

**assumes**  $P'B'Q': (P', B', Q') = \text{Diagonal-to-Smith-row-i-PQ } i \text{ bezout } (P, B, Q)$   
**and** *diag: isDiagonal B and i: i < min (nrows A) (ncols A)*

**shows**  $B' = P' ** A ** Q' \wedge \text{invertible } P' \wedge \text{invertible } Q' \wedge B' = \text{Diagonal-to-Smith-row-}i$   
*B i bezout*

**using** *assms Diagonal-to-Smith-row-i-PQ' Diagonal-to-Smith-row-i-PQ-eq* **by** *pres-*  
*burger*

**end**

**context**

**fixes**  $A::'a::\{\text{bezout-ring}\}^{\wedge} \text{cols}::\text{mod-type}^{\wedge} \text{rows}::\text{mod-type}$   
**and**  $B::'a::\{\text{bezout-ring}\}^{\wedge} \text{cols}::\text{mod-type}^{\wedge} \text{rows}::\text{mod-type}$   
**and**  $P$  **and**  $Q$   
**and** *bezout::'a bezout*

**assumes**  $PAQ: P ** A ** Q = B$   
**and**  $P: \text{invertible } P$  **and**  $Q: \text{invertible } Q$   
**and** *ib: is-bezout-ext bezout*

**begin**

**lemma** *diagonal-to-Smith-aux-PQ:*

**assumes**  $P'B'Q': (P', B', Q') = \text{diagonal-to-Smith-aux-PQ } [0..<k] \text{ bezout } (P, B, Q)$   
**and** *diag: isDiagonal B and k:k < min (nrows A) (ncols A)*

**shows**  $B' = P' ** A ** Q' \wedge \text{invertible } P' \wedge \text{invertible } Q' \wedge B' = \text{diagonal-to-Smith-aux}$   
 $B [0..<k] \text{ bezout}$

**using**  $k$   $P'B'Q'$   $P$   $Q$   $PAQ$  *diag*

**proof** *(induct k arbitrary: P B Q P' Q' B')*  
**case**  $0$

```

then show ?case using P Q PAQ by auto
next
case (Suc k P B Q P' Q' B')
note Suc-k = Suc.prem1
note PBQ = Suc.prem2
note P = Suc.prem3
note Q = Suc.prem4
note PAQ-B = Suc.prem5
note diag-B = Suc.prem6
let ?Dk = (diagonal-to-Smith-aux-PQ [0..<k] bezout (P, P ** A ** Q, Q))
let ?P' = fst ?Dk
let ?B' = fst (snd ?Dk)
let ?Q' = snd (snd ?Dk)
have k: k < min (nrows A) (ncols A) using Suc-k by auto
have hyp: ?B' = ?P' ** A ** ?Q' ∧ invertible ?P' ∧ invertible ?Q'
  ∧ ?B' = diagonal-to-Smith-aux B [0..<k] bezout
  by (rule Suc.hyps[OF k - P Q PAQ-B diag-B], auto simp add: PAQ-B)
have diag-B': isDiagonal ?B'
  by (metis diag-B hyp ib isDiagonal-diagonal-to-Smith-aux k ncols-def nrows-def)
have B' = diagonal-to-Smith-aux B [0..<Suc k] bezout
  by (auto, metis Diagonal-to-Smith-row-i-PQ-eq PAQ-B Suc(3) diag-B'
    diagonal-to-Smith-aux-PQ-append2 eq-fst-iff hyp ib k sndI upt.simps(2)
    zero-order(1))
moreover have B' = P' ** A ** Q' ∧ invertible P' ∧ invertible Q'
proof (rule Diagonal-to-Smith-row-i-PQ')
show (P', B', Q') = Diagonal-to-Smith-row-i-PQ k bezout (?P', ?B', ?Q') using
Suc.prem1 by auto
show invertible ?P' using hyp by auto
show ?P' ** A ** ?Q' = ?B' using hyp by auto
show invertible ?Q' using hyp by auto
show is-bezout-ext bezout using ib by auto
show k < min (nrows A) (ncols A) using k by auto
show diag-B': isDiagonal ?B' using diag-B' by auto
qed
ultimately show ?case by auto
qed
end

fun diagonal-to-Smith-PQ
  where diagonal-to-Smith-PQ A bezout
    = diagonal-to-Smith-aux-PQ [0..<min (nrows A) (ncols A) - 1] bezout (mat 1,
A , mat 1)

declare diagonal-to-Smith-PQ.simps[simp del]
lemmas diagonal-to-Smith-PQ-def = diagonal-to-Smith-PQ.simps

lemma diagonal-to-Smith-PQ:
  fixes A::'a::{bezout-ring} ^ cols::'mod-type ^ rows::'mod-type

```

**assumes**  $A$ : *isDiagonal*  $A$  **and**  $ib$ : *is-bezout-ext* *bezout*  
**assumes**  $PBQ$ :  $(P, B, Q) = \text{diagonal-to-Smith-PQ } A$  *bezout*  
**shows**  $B = P**A**Q \wedge \text{invertible } P \wedge \text{invertible } Q \wedge B = \text{diagonal-to-Smith } A$  *bezout*  
**proof** (*unfold diagonal-to-Smith-def, rule diagonal-to-Smith-aux-PQ[OF - - - ib - A]*)  
**let**  $?P = \text{mat } 1::'a \wedge \text{rows}::\text{mod-type} \wedge \text{rows}::\text{mod-type}$   
**let**  $?Q = \text{mat } 1::'a \wedge \text{cols}::\text{mod-type} \wedge \text{cols}::\text{mod-type}$   
**show**  $(P, B, Q) = \text{diagonal-to-Smith-aux-PQ } [0..<\text{min } (nrows A) (ncols A) - 1]$  *bezout*  $(?P, A, ?Q)$   
**using**  $PBQ$  **unfolding** *diagonal-to-Smith-PQ-def* .  
**show**  $?P ** A ** ?Q = A$  **by** *simp*  
**show**  $\text{min } (nrows A) (ncols A) - 1 < \text{min } (nrows A) (ncols A)$   
**by** (*metis (no-types, lifting) One-nat-def diff-less dual-order.strict-iff-order le-less-trans*  
*min-def mod-type-class.to-nat-less-card ncols-def not-less-eq nrows-not-0 zero-order(1)*)  
**qed** (*auto simp add: invertible-mat-1*)

**lemma** *diagonal-to-Smith-PQ-exists*:

**fixes**  $A::'a::\{\text{bezout-ring}\} \wedge \text{cols}::\{\text{mod-type}\} \wedge \text{rows}::\{\text{mod-type}\}$   
**assumes**  $A$ : *isDiagonal*  $A$   
**shows**  $\exists P Q.$   
 $\text{invertible } (P::'a \wedge \text{rows}::\{\text{mod-type}\} \wedge \text{rows}::\{\text{mod-type}\})$   
 $\wedge \text{invertible } (Q::'a \wedge \text{cols}::\{\text{mod-type}\} \wedge \text{cols}::\{\text{mod-type}\})$   
 $\wedge \text{Smith-normal-form } (P**A**Q)$   
**proof** –  
**obtain**  $\text{bezout}::'a$  *bezout* **where**  $ib$ : *is-bezout-ext* *bezout*  
**using** *exists-bezout-ext* **by** *blast*  
**obtain**  $P B Q$  **where**  $PBQ$ :  $(P, B, Q) = \text{diagonal-to-Smith-PQ } A$  *bezout*  
**by** (*metis prod-cases3*)  
**have**  $B = P**A**Q \wedge \text{invertible } P \wedge \text{invertible } Q \wedge B = \text{diagonal-to-Smith } A$  *bezout*  
**by** (*rule diagonal-to-Smith-PQ[OF A ib PBQ]*)  
**moreover** **have** *Smith-normal-form*  $(P**A**Q)$   
**using** *Smith-normal-form-diagonal-to-Smith* *assms calculation ib* **by** *fastforce*  
**ultimately show**  $?thesis$  **by** *auto*  
**qed**

## 2.6 The final soundness theorem

**lemma** *diagonal-to-Smith-PQ'*:

**fixes**  $A::'a::\{\text{bezout-ring}\} \wedge \text{cols}::\{\text{mod-type}\} \wedge \text{rows}::\{\text{mod-type}\}$   
**assumes**  $A$ : *isDiagonal*  $A$  **and**  $ib$ : *is-bezout-ext* *bezout*  
**assumes**  $PBQ$ :  $(P, S, Q) = \text{diagonal-to-Smith-PQ } A$  *bezout*  
**shows**  $S = P**A**Q \wedge \text{invertible } P \wedge \text{invertible } Q \wedge \text{Smith-normal-form } S$   
**using**  $A PBQ$  *Smith-normal-form-diagonal-to-Smith diagonal-to-Smith-PQ ib* **by** *fastforce*



end

### 3 A new bridge to convert theorems from JNF to HOL Analysis and vice-versa, based on the *mod-type* class

```
theory Mod-Type-Connect
  imports
    Perron-Frobenius.HMA-Connect
    Rank-Nullity-Theorem.Mod-Type
    Gauss-Jordan.Elementary-Operations
```

begin

Some lemmas on *Mod-Type.to-nat* and *Mod-Type.from-nat* are added to have them with the same names as the analogous ones for *Bij-Nat.to-nat* and *Bij-Nat.from-nat*.

```
lemma inj-to-nat: inj to-nat by (simp add: inj-on-def)
lemmas from-nat-inj = from-nat-eq-imp-eq
lemma range-to-nat: range (to-nat :: 'a :: mod-type  $\Rightarrow$  nat) = {0 ..< CARD('a)}
  by (simp add: bij-betw-imp-surj-on mod-type-class.bij-to-nat)
```

This theory is an adaptation of the one presented in *Perron-Frobenius.HMA-Connect*, but for matrices and vectors where indexes have the *mod-type* class restriction.

It is worth noting that some definitions still use the old abbreviation for HOL Analysis (HMA, from HOL Multivariate Analysis) instead of HA. This is done to be consistent with the existing names in the Perron-Frobenius development

```
context includes vec.lifting
begin
end
```

```
definition from-hmav :: 'a ^ 'n :: mod-type  $\Rightarrow$  'a Matrix.vec where
  from-hmav v = Matrix.vec CARD('n) ( $\lambda$  i. v $h from-nat i)
```

```
definition from-hmam :: 'a ^ 'nc :: mod-type ^ 'nr :: mod-type  $\Rightarrow$  'a Matrix.mat
where
  from-hmam a = Matrix.mat CARD('nr) CARD('nc) ( $\lambda$  (i,j). a $h from-nat i $h
  from-nat j)
```

```
definition to-hmav :: 'a Matrix.vec  $\Rightarrow$  'a ^ 'n :: mod-type where
  to-hmav v = ( $\chi$  i. v $v to-nat i)
```

```
definition to-hmam :: 'a Matrix.mat  $\Rightarrow$  'a ^ 'nc :: mod-type ^ 'nr :: mod-type
where
```

$to-hma_m a = (\chi i j. a \text{ \&\& } (to-nat i, to-nat j))$

**lemma**  $to-hma-from-hma_v[simp]$ :  $to-hma_v (from-hma_v v) = v$   
**by** (*auto simp: to-hma\_v-def from-hma\_v-def to-nat-less-card*)

**lemma**  $to-hma-from-hma_m[simp]$ :  $to-hma_m (from-hma_m v) = v$   
**by** (*auto simp: to-hma\_m-def from-hma\_m-def to-nat-less-card*)

**lemma**  $from-hma-to-hma_v[simp]$ :  
 $v \in carrier-vec (CARD('n)) \implies from-hma_v (to-hma_v v :: 'a ^ 'n :: mod-type) = v$   
**by** (*auto simp: to-hma\_v-def from-hma\_v-def to-nat-from-nat-id*)

**lemma**  $from-hma-to-hma_m[simp]$ :  
 $A \in carrier-mat (CARD('nr)) (CARD('nc)) \implies from-hma_m (to-hma_m A :: 'a ^ 'nc :: mod-type ^ 'nr :: mod-type) = A$   
**by** (*auto simp: to-hma\_m-def from-hma\_m-def to-nat-from-nat-id*)

**lemma**  $from-hma_v-inj[simp]$ :  $from-hma_v x = from-hma_v y \longleftrightarrow x = y$   
**by** (*intro iffI, insert to-hma-from-hma\_v[of x], auto*)

**lemma**  $from-hma_m-inj[simp]$ :  $from-hma_m x = from-hma_m y \longleftrightarrow x = y$   
**by** (*intro iffI, insert to-hma-from-hma\_m[of x], auto*)

**definition**  $HMA-V :: 'a Matrix.vec \Rightarrow 'a ^ 'n :: mod-type \Rightarrow bool$  **where**  
 $HMA-V = (\lambda v w. v = from-hma_v w)$

**definition**  $HMA-M :: 'a Matrix.mat \Rightarrow 'a ^ 'nc :: mod-type ^ 'nr :: mod-type \Rightarrow bool$  **where**  
 $HMA-M = (\lambda a b. a = from-hma_m b)$

**definition**  $HMA-I :: nat \Rightarrow 'n :: mod-type \Rightarrow bool$  **where**  
 $HMA-I = (\lambda i a. i = to-nat a)$

**context includes** *lifting-syntax*  
**begin**

**lemma**  $Domainp-HMA-V$  [*transfer-domain-rule*]:  
 $Domainp (HMA-V :: 'a Matrix.vec \Rightarrow 'a ^ 'n :: mod-type \Rightarrow bool) = (\lambda v. v \in carrier-vec (CARD('n)))$   
**by** (*intro ext iffI, insert from-hma-to-hma\_v[symmetric], auto simp: from-hma\_v-def HMA-V-def*)

**lemma**  $Domainp-HMA-M$  [*transfer-domain-rule*]:  
 $Domainp (HMA-M :: 'a Matrix.mat \Rightarrow 'a ^ 'nc :: mod-type ^ 'nr :: mod-type \Rightarrow bool) = (\lambda A. A \in carrier-mat CARD('nr) CARD('nc))$

**by** (*intro ext iffI*, *insert from-hma-to-hma<sub>m</sub>[symmetric]*, *auto simp: from-hma<sub>m</sub>-def HMA-M-def*)

**lemma** *Domainp-HMA-I* [*transfer-domain-rule*]:

*Domainp* (*HMA-I* :: *nat* ⇒ *'n* :: *mod-type* ⇒ *bool*) = (λ *i*. *i* < *CARD('n)*) (**is** ?*l* = ?*r*)

**proof** (*intro ext*)

**fix** *i* :: *nat*

**show** ?*l* *i* = ?*r* *i*

**unfolding** *HMA-I-def Domainp-iff*

**by** (*auto intro: exI[of - from-nat i]* *simp: to-nat-from-nat-id to-nat-less-card*)

**qed**

**lemma** *bi-unique-HMA-V* [*transfer-rule*]: *bi-unique HMA-V left-unique HMA-V right-unique HMA-V*

**unfolding** *HMA-V-def bi-unique-def left-unique-def right-unique-def* **by** *auto*

**lemma** *bi-unique-HMA-M* [*transfer-rule*]: *bi-unique HMA-M left-unique HMA-M right-unique HMA-M*

**unfolding** *HMA-M-def bi-unique-def left-unique-def right-unique-def* **by** *auto*

**lemma** *bi-unique-HMA-I* [*transfer-rule*]: *bi-unique HMA-I left-unique HMA-I right-unique HMA-I*

**unfolding** *HMA-I-def bi-unique-def left-unique-def right-unique-def* **by** *auto*

**lemma** *right-total-HMA-V* [*transfer-rule*]: *right-total HMA-V*

**unfolding** *HMA-V-def right-total-def* **by** *simp*

**lemma** *right-total-HMA-M* [*transfer-rule*]: *right-total HMA-M*

**unfolding** *HMA-M-def right-total-def* **by** *simp*

**lemma** *right-total-HMA-I* [*transfer-rule*]: *right-total HMA-I*

**unfolding** *HMA-I-def right-total-def* **by** *simp*

**lemma** *HMA-V-index* [*transfer-rule*]: (*HMA-V* ==> *HMA-I* ==> (=)) (*\$v*) (*\$h*)

**unfolding** *rel-fun-def HMA-V-def HMA-I-def from-hma<sub>v</sub>-def*

**by** (*auto simp: to-nat-less-card*)

**lemma** *HMA-M-index* [*transfer-rule*]:

(*HMA-M* ==> *HMA-I* ==> *HMA-I* ==> (=)) (λ *A i j*. *A* \$\$ (*i,j*)) *index-hma*

**by** (*intro rel-funI*, *simp add: index-hma-def to-nat-less-card HMA-M-def HMA-I-def from-hma<sub>m</sub>-def*)

**lemma** *HMA-V-0* [*transfer-rule*]: *HMA-V* (*0<sub>v</sub>* *CARD('n)*) (*0* :: *'a* :: *zero* ^ *'n*:: *mod-type*)

**unfolding** *HMA-V-def from-hma<sub>v</sub>-def* **by** *auto*

**lemma** *HMA-M-0* [*transfer-rule*]:

*HMA-M* (*0<sub>m</sub>* *CARD('nr)* *CARD('nc)*) (*0* :: '*a* :: *zero*  $\wedge$  '*nc*:: *mod-type*  $\wedge$  '*nr* :: *mod-type*)

**unfolding** *HMA-M-def from-hma<sub>m</sub>-def* **by** *auto*

**lemma** *HMA-M-1* [*transfer-rule*]:

*HMA-M* (*1<sub>m</sub>* (*CARD('n)*)) (*mat 1* :: '*a*::{*zero,one*}  $\wedge$  '*n*:: *mod-type*  $\wedge$  '*n*:: *mod-type*)

**unfolding** *HMA-M-def*

**by** (*auto simp add: mat-def from-hma<sub>m</sub>-def from-nat-inj*)

**lemma** *from-hma<sub>v</sub>-add*: *from-hma<sub>v</sub>* *v* + *from-hma<sub>v</sub>* *w* = *from-hma<sub>v</sub>* (*v* + *w*)

**unfolding** *from-hma<sub>v</sub>-def* **by** *auto*

**lemma** *HMA-V-add* [*transfer-rule*]: (*HMA-V*  $\implies$  *HMA-V*  $\implies$  *HMA-V*)

(+) (+)

**unfolding** *rel-fun-def HMA-V-def*

**by** (*auto simp: from-hma<sub>v</sub>-add*)

**lemma** *from-hma<sub>v</sub>-diff*: *from-hma<sub>v</sub>* *v* - *from-hma<sub>v</sub>* *w* = *from-hma<sub>v</sub>* (*v* - *w*)

**unfolding** *from-hma<sub>v</sub>-def* **by** *auto*

**lemma** *HMA-V-diff* [*transfer-rule*]: (*HMA-V*  $\implies$  *HMA-V*  $\implies$  *HMA-V*)

(-) (-)

**unfolding** *rel-fun-def HMA-V-def*

**by** (*auto simp: from-hma<sub>v</sub>-diff*)

**lemma** *from-hma<sub>m</sub>-add*: *from-hma<sub>m</sub>* *a* + *from-hma<sub>m</sub>* *b* = *from-hma<sub>m</sub>* (*a* + *b*)

**unfolding** *from-hma<sub>m</sub>-def* **by** *auto*

**lemma** *HMA-M-add* [*transfer-rule*]: (*HMA-M*  $\implies$  *HMA-M*  $\implies$  *HMA-M*)

(+) (+)

**unfolding** *rel-fun-def HMA-M-def*

**by** (*auto simp: from-hma<sub>m</sub>-add*)

**lemma** *from-hma<sub>m</sub>-diff*: *from-hma<sub>m</sub>* *a* - *from-hma<sub>m</sub>* *b* = *from-hma<sub>m</sub>* (*a* - *b*)

**unfolding** *from-hma<sub>m</sub>-def* **by** *auto*

**lemma** *HMA-M-diff* [*transfer-rule*]: (*HMA-M*  $\implies$  *HMA-M*  $\implies$  *HMA-M*)

(-) (-)

**unfolding** *rel-fun-def HMA-M-def*

**by** (*auto simp: from-hma<sub>m</sub>-diff*)

**lemma** *scalar-product*: **fixes** *v* :: '*a* :: *semiring-1*  $\wedge$  '*n* :: *mod-type*

**shows** *scalar-prod* (*from-hma<sub>v</sub>* *v*) (*from-hma<sub>v</sub>* *w*) = *scalar-product v w*

**unfolding** *scalar-product-def scalar-prod-def from-hma<sub>v</sub>-def dim-vec*

**by** (*simp add: sum.reindex[OF inj-to-nat, unfolded range-to-nat]*)

**lemma** [simp]:  
 $from-hma_m (y :: 'a \wedge 'nc :: mod-type \wedge 'nr :: mod-type) \in carrier-mat (CARD('nr))$   
 $(CARD('nc))$   
 $dim-row (from-hma_m (y :: 'a \wedge 'nc :: mod-type \wedge 'nr :: mod-type)) = CARD('nr)$   
 $dim-col (from-hma_m (y :: 'a \wedge 'nc :: mod-type \wedge 'nr :: mod-type)) = CARD('nc)$   
**unfolding** *from-hma\_m-def* **by** *simp-all*

**lemma** [simp]:  
 $from-hma_v (y :: 'a \wedge 'n :: mod-type) \in carrier-vec (CARD('n))$   
 $dim-vec (from-hma_v (y :: 'a \wedge 'n :: mod-type)) = CARD('n)$   
**unfolding** *from-hma\_v-def* **by** *simp-all*

**lemma** *HMA-scalar-prod* [transfer-rule]:  
 $(HMA-V \implies HMA-V \implies (=)) \text{ scalar-prod scalar-product}$   
**by** (*auto simp: HMA-V-def scalar-product*)

**lemma** *HMA-row* [transfer-rule]:  $(HMA-I \implies HMA-M \implies HMA-V) (\lambda i$   
*a. Matrix.row a i) row  
**unfolding** *HMA-M-def HMA-I-def HMA-V-def*  
**by** (*auto simp: from-hma\_m-def from-hma\_v-def to-nat-less-card row-def*)*

**lemma** *HMA-col* [transfer-rule]:  $(HMA-I \implies HMA-M \implies HMA-V) (\lambda i$   
*a. col a i) column  
**unfolding** *HMA-M-def HMA-I-def HMA-V-def*  
**by** (*auto simp: from-hma\_m-def from-hma\_v-def to-nat-less-card column-def*)*

**lemma** *HMA-M-mk-mat*[transfer-rule]:  $((HMA-I \implies HMA-I \implies (=)) \implies$   
 $HMA-M)$   
 $(\lambda f. Matrix.mat (CARD('nr)) (CARD('nc)) (\lambda (i,j). f i j))$   
 $(mk-mat :: (('nr \Rightarrow 'nc \Rightarrow 'a) \Rightarrow 'a \wedge 'nc :: mod-type \wedge 'nr :: mod-type))$

**proof** –  
 $\{$   
 $\text{fix } x y i j$   
 $\text{assume } id: \forall (ya :: 'nr) (yb :: 'nc). (x (to-nat ya) (to-nat yb) :: 'a) = y ya yb$   
 $\text{and } i: i < CARD('nr) \text{ and } j: j < CARD('nc)$   
 $\text{from } to-nat-from-nat-id[OF i] \text{ to-nat-from-nat-id[OF j] } id[\text{rule-format, of from-nat}$   
 $i \text{ from-nat } j]$   
 $\text{have } x i j = y (from-nat i) (from-nat j) \text{ by auto}$   
 $\}$   
 $\text{thus ?thesis}$   
**unfolding** *rel-fun-def mk-mat-def HMA-M-def HMA-I-def from-hma\_m-def* **by**  
*auto*  
**qed**

**lemma** *HMA-M-mk-vec*[transfer-rule]:  $((HMA-I \implies (=)) \implies HMA-V)$   
 $(\lambda f. Matrix.vec (CARD('n)) (\lambda i. f i))$   
 $(mk-vec :: (('n \Rightarrow 'a) \Rightarrow 'a \wedge 'n :: mod-type))$

```

proof –
{
  fix  $x\ y\ i$ 
  assume  $id: \forall (ya :: 'n). (x\ (to\ nat\ ya) :: 'a) = y\ ya$ 
  and  $i < CARD('n)$ 
  from  $to\ nat\ from\ nat\ id[OF\ i]\ id[rule\ format, of\ from\ nat\ i]$ 
  have  $x\ i = y\ (from\ nat\ i)$  by auto
}
thus ?thesis
  unfolding  $rel\ fun\ def\ mk\ vec\ def\ HMA\ V\ def\ HMA\ I\ def\ from\ hma_v\ def$  by
auto
qed

```

```

lemma  $mat\ mult\ scalar: A ** B = mk\ mat\ (\lambda\ i\ j. scalar\ product\ (row\ i\ A)\ (column\ j\ B))$ 
unfolding  $vec\ eq\ iff\ matrix\ matrix\ mult\ def\ scalar\ product\ def\ mk\ mat\ def$ 
by (auto simp: row\ def\ column\ def)

```

```

lemma  $mult\ mat\ vec\ scalar: A *v\ v = mk\ vec\ (\lambda\ i. scalar\ product\ (row\ i\ A)\ v)$ 
unfolding  $vec\ eq\ iff\ matrix\ vector\ mult\ def\ scalar\ product\ def\ mk\ mat\ def\ mk\ vec\ def$ 
by (auto simp: row\ def\ column\ def)

```

```

lemma  $dim\ row\ transfer\ rule:$ 
   $HMA\ M\ A\ (A' :: 'a\ ^\ 'nc :: mod\ type\ ^\ 'nr :: mod\ type) \implies (=)\ (dim\ row\ A)$ 
  ( $CARD('nr)$ )
unfolding  $HMA\ M\ def$  by auto

```

```

lemma  $dim\ col\ transfer\ rule:$ 
   $HMA\ M\ A\ (A' :: 'a\ ^\ 'nc :: mod\ type\ ^\ 'nr :: mod\ type) \implies (=)\ (dim\ col\ A)$ 
  ( $CARD('nc)$ )
unfolding  $HMA\ M\ def$  by auto

```

```

lemma  $HMA\ M\ mult\ [transfer\ rule]: (HMA\ M\ ==> HMA\ M\ ==> HMA\ M)$ 
  (*) (**)

```

```

proof –
{
  fix  $A\ B :: 'a :: semiring\ 1\ mat$  and  $A' :: 'a\ ^\ 'n :: mod\ type\ ^\ 'nr :: mod\ type$ 
  and  $B' :: 'a\ ^\ 'nc :: mod\ type\ ^\ 'n :: mod\ type$ 
  assume  $1[transfer\ rule]: HMA\ M\ A\ A'\ HMA\ M\ B\ B'$ 
  note  $[transfer\ rule] = dim\ row\ transfer\ rule[OF\ 1(1)]\ dim\ col\ transfer\ rule[OF\ 1(2)]$ 
  have  $HMA\ M\ (A * B)\ (A' ** B')$ 
  unfolding  $times\ mat\ def\ mat\ mult\ scalar$ 
  by (transfer\ prover\ start, transfer\ step+, transfer, auto)
}
thus ?thesis by blast
qed

```

**lemma** *HMA-V-smult* [*transfer-rule*]: ((=) ==> *HMA-V* ==> *HMA-V*) ( $\cdot v$ )  
 (\*s)  
 unfolding *smult-vec-def*  
 unfolding *rel-fun-def HMA-V-def from-hma\_v-def*  
 by *auto*

**lemma** *HMA-M-mult-vec* [*transfer-rule*]: (*HMA-M* ==> *HMA-V* ==> *HMA-V*)  
 (\*v) (\*v)  
**proof** –  
 {  
 fix *A* :: 'a :: *semiring-1 mat* and *v* :: 'a *Matrix.vec*  
 and *A'* :: 'a ^ 'nc :: *mod-type* ^ 'nr :: *mod-type* and *v'* :: 'a ^ 'nc :: *mod-type*  
 assume 1 [*transfer-rule*]: *HMA-M A A' HMA-V v v'*  
 note [*transfer-rule*] = *dim-row-transfer-rule*  
 have *HMA-V (A \*<sub>v</sub> v) (A' \*<sub>v</sub> v')*  
 unfolding *mult-mat-vec-def mult-mat-vec-scalar*  
 by (*transfer-prover-start, transfer-step+, transfer, auto*)  
 }  
 thus *?thesis* by *blast*  
**qed**

**lemma** *HMA-det* [*transfer-rule*]: (*HMA-M* ==> (=)) *Determinant.det*  
 (*det* :: 'a :: *comm-ring-1* ^ 'n :: *mod-type* ^ 'n :: *mod-type* => 'a)  
**proof** –  
 {  
 fix *a* :: 'a ^ 'n :: *mod-type* ^ 'n :: *mod-type*  
 let *?tn* = *to-nat* :: 'n :: *mod-type* => *nat*  
 let *?fn* = *from-nat* :: *nat* => 'n  
 let *?zn* = {0..< *CARD('n)*}  
 let *?U* = *UNIV* :: 'n *set*  
 let *?p1* = {*p. p permutes ?zn*}  
 let *?p2* = {*p. p permutes ?U*}  
 let *?f* =  $\lambda p i. \text{if } i \in ?U \text{ then } ?fn (p (?tn i)) \text{ else } i$   
 let *?g* =  $\lambda p i. ?fn (p (?tn i))$   
 have *fg*:  $\bigwedge a b c. (\text{if } a \in ?U \text{ then } b \text{ else } c) = b$  by *auto*  
 have *?p2* = *?f* ' *?p1*  
 by (*rule permutes-bij', auto simp: to-nat-less-card to-nat-from-nat-id*)  
 hence *id*: *?p2* = *?g* ' *?p1* by *simp*  
 have *inj-g*: *inj-on ?g ?p1*  
 unfolding *inj-on-def*  
**proof** (*intro ballI impI ext, auto*)  
 fix *p q i*  
 assume *p*: *p permutes ?zn* and *q*: *q permutes ?zn*  
 and *id*:  $(\lambda i. ?fn (p (?tn i))) = (\lambda i. ?fn (q (?tn i)))$   
 {  
 fix *i*

```

    from permutes-in-image[OF p] have pi: p (?tn i) < CARD('n) by (simp
add: to-nat-less-card)
    from permutes-in-image[OF q] have qi: q (?tn i) < CARD('n) by (simp
add: to-nat-less-card)
    from fun-cong[OF id] have ?fn (p (?tn i)) = from-nat (q (?tn i)) .
    from arg-cong[OF this, of ?tn] have p (?tn i) = q (?tn i)
    by (simp add: to-nat-from-nat-id pi qi)
  } note id = this
show p i = q i
proof (cases i < CARD('n))
  case True
  hence ?tn (?fn i) = i by (simp add: to-nat-from-nat-id)
  from id[of ?fn i, unfolded this] show ?thesis .
next
  case False
  thus ?thesis using p q unfolding permutes-def by simp
qed
qed
have mult-cong:  $\bigwedge a b c d. a = b \implies c = d \implies a * c = b * d$  by simp
have sum (λ p.
  signof p * (∏ i∈?zn. a $h ?fn i $h ?fn (p i))) ?p1
= sum (λ p. of-int (sign p) * (∏ i∈UNIV. a $h i $h p i)) ?p2
  unfolding id sum.reindex[OF inj-g]
proof (rule sum.cong[OF refl], unfold mem-Collect-eq o-def, rule mult-cong)
  fix p
  assume p: p permutes ?zn
  let ?q = λ i. ?fn (p (?tn i))
  from id p have q: ?q permutes ?U by auto
  from p have pp: permutation p unfolding permutation-permutes by auto
  let ?ft = λ p i. ?fn (p (?tn i))
  have fin: finite ?zn by simp
  have sign p = sign ?q ∧ p permutes ?zn
  using p fin proof (induction rule: permutes-induct)
    case id
    show ?case by (auto simp: sign-id[unfolded id-def] permutes-id[unfolded
id-def])
  next
  case (swap a b p)
  then have ⟨permutation p⟩
    using permutes-imp-permutation by blast
  let ?sab = Transposition.transpose a b
  let ?sfab = Transposition.transpose (?fn a) (?fn b)
  have p-sab: permutation ?sab by (rule permutation-swap-id)
  have p-sfab: permutation ?sfab by (rule permutation-swap-id)
  from swap(4) have IH1: p permutes ?zn and IH2: sign p = sign (?ft p)
by auto
  have sab-perm: ?sab permutes ?zn using swap(1-2) by (rule permutes-swap-id)
  from permutes-compose[OF IH1 this] have perm1: ?sab o p permutes ?zn .
  from IH1 have p-p1: p ∈ ?p1 by simp

```



```

hence ?ft p ∈ ?ft ‘ ?p1 by (rule imageI)
from this[folded id] have ?ft p permutes ?U by simp
hence p-ftp: permutation (?ft p) unfolding permutation-permutes by auto
{
  fix a b
  assume a: a ∈ ?zn and b: b ∈ ?zn
  hence (?fn a = ?fn b) = (a = b) using swap(1-2)
  by (auto simp add: from-nat-eq-imp-eq)
} note inj = this
from inj[OF swap(1-2)] have id2: sign ?sfab = sign ?sab unfolding
sign-swap-id by simp
have id: ?ft (Transposition.transpose a b ∘ p) = Transposition.transpose
(?fn a) (?fn b) ∘ ?ft p
proof
fix c
show ?ft (Transposition.transpose a b ∘ p) c = (Transposition.transpose
(?fn a) (?fn b) ∘ ?ft p) c
proof (cases p (?tn c) = a ∨ p (?tn c) = b)
  case True
  thus ?thesis by (cases, auto simp add: o-def swap-id-eq)
next
  case False
  hence neq: p (?tn c) ≠ a p (?tn c) ≠ b by auto
  have pc: p (?tn c) ∈ ?zn unfolding permutes-in-image[OF IH1]
  by (simp add: to-nat-less-card)
  from neq[folded inj[OF pc swap(1)] inj[OF pc swap(2)]]
  have ?fn (p (?tn c)) ≠ ?fn a ?fn (p (?tn c)) ≠ ?fn b .
  with neq show ?thesis by (auto simp: o-def swap-id-eq)
qed
qed
show ?case unfolding IH2 id sign-compose[OF p-sab ⟨permutation p⟩]
sign-compose[OF p-sfab p-ftp] id2
by (rule conjI[OF refl perm1])
qed
thus signof p = of-int (sign ?q) by simp
show (∏ i = 0..<CARD('n). a $h ?fn i $h ?fn (p i)) =
(∏ i ∈ UNIV. a $h i $h ?q i) unfolding
range-to-nat[symmetric] prod.reindex[OF inj-to-nat]
by (rule prod.cong[OF refl], unfold o-def, simp)
qed
}
thus ?thesis unfolding HMA-M-def
by (auto simp: from-hmam-def Determinant.det-def det-def)
qed

```

**lemma** HMA-mat[transfer-rule]: ((=) ==> HMA-M) (λ k. k ·<sub>m</sub> 1<sub>m</sub> CARD('n))

(Finite-Cartesian-Product.mat :: 'a::semiring-1 => 'a<sup>~</sup>'n :: mod-type<sup>~</sup>'n :: mod-type)
**unfolding** Finite-Cartesian-Product.mat-def[abs-def] rel-fun-def HMA-M-def

by (auto simp: from-hma<sub>m</sub>-def from-nat-inj)

**lemma** HMA-mat-minus[transfer-rule]: (HMA-M ==> HMA-M ==> HMA-M)

(λ A B. A + map-mat uminus B) ((-) :: 'a :: group-add ^'nc:: mod-type ^'nr::  
 mod-type  
 ⇒ 'a ^'nc:: mod-type ^'nr:: mod-type ⇒ 'a ^'nc:: mod-type ^'nr:: mod-type)  
**unfolding** rel-fun-def HMA-M-def from-hma<sub>m</sub>-def **by** auto

**lemma** HMA-transpose-matrix [transfer-rule]:

(HMA-M ==> HMA-M) transpose-mat transpose  
**unfolding** transpose-mat-def transpose-def HMA-M-def from-hma<sub>m</sub>-def **by** auto

**lemma** HMA-invertible-matrix-mod-type[transfer-rule]:

((Mod-Type-Connect.HMA-M :: - ⇒ 'a :: comm-ring-1 ^'n :: mod-type ^'n ::  
 mod-type  
 ⇒ -) ==> (=)) invertible-mat invertible

**proof** (intro rel-funI, goal-cases)

**case** (1 x y)

**note** rel-xy[transfer-rule] = 1

**have** eq-dim: dim-col x = dim-row x

**using** Mod-Type-Connect.dim-col-transfer-rule Mod-Type-Connect.dim-row-transfer-rule  
 rel-xy

**by** fastforce

**moreover** have ∃ A'. y \*\* A' = mat 1 ∧ A' \*\* y = mat 1

**if** xB: x \* B = 1<sub>m</sub> (dim-row x) **and** Bx: B \* x = 1<sub>m</sub> (dim-row B) **for** B

**proof** -

**let** ?A' = Mod-Type-Connect.to-hma<sub>m</sub> B:: 'a :: comm-ring-1 ^'n :: mod-type ^'  
 'n :: mod-type

**have** rel-BA[transfer-rule]: Mod-Type-Connect.HMA-M B ?A'

**by** (metis (no-types, lifting) Bx Mod-Type-Connect.HMA-M-def eq-dim car-  
 rier-mat-triv dim-col-mat(1)

Mod-Type-Connect.from-hma<sub>m</sub>-def Mod-Type-Connect.from-hma-to-hma<sub>m</sub>  
 index-mult-mat(3)

index-one-mat(3) rel-xy xB)

**have** [simp]: dim-row B = CARD('n) **using** Mod-Type-Connect.dim-row-transfer-rule  
 rel-BA **by** blast

**have** [simp]: dim-row x = CARD('n) **using** Mod-Type-Connect.dim-row-transfer-rule  
 rel-xy **by** blast

**have** y \*\* ?A' = mat 1 **using** xB **by** (transfer, simp)

**moreover** have ?A' \*\* y = mat 1 **using** Bx **by** (transfer, simp)

**ultimately** show ?thesis **by** blast

**qed**

**moreover** have ∃ B. x \* B = 1<sub>m</sub> (dim-row x) ∧ B \* x = 1<sub>m</sub> (dim-row B)

**if** yA: y \*\* A' = mat 1 **and** Ay: A' \*\* y = mat 1 **for** A'

**proof** -

**let** ?B = (Mod-Type-Connect.from-hma<sub>m</sub> A')

```

have [simp]: dim-row  $x = \text{CARD}(n)$  using rel-xy Mod-Type-Connect.dim-row-transfer-rule
by blast
have [transfer-rule]: Mod-Type-Connect.HMA-M  $?B$   $A'$  by (simp add: Mod-Type-Connect.HMA-M-def)
hence [simp]: dim-row  $?B = \text{CARD}(n)$  using dim-row-transfer-rule by auto
have  $x * ?B = 1_m$  (dim-row  $x$ ) using  $yA$  by (transfer', auto)
moreover have  $?B * x = 1_m$  (dim-row  $?B$ ) using  $Ay$  by (transfer', auto)
ultimately show  $?thesis$  by auto
qed
ultimately show  $?case$  unfolding invertible-mat-def invertible-def inverts-mat-def
by auto
qed

```

**end**

Some transfer rules for relating the elementary operations are also proved.

**context**

**includes** *lifting-syntax*

**begin**

**lemma** *HMA-swaprows*[transfer-rule]:

((*Mod-Type-Connect.HMA-M* ::  $- \Rightarrow 'a :: \text{comm-ring-1 } \hat{\ } 'nc :: \text{mod-type } \hat{\ } 'nr :: \text{mod-type } \Rightarrow -$ )

====> (*Mod-Type-Connect.HMA-I* ::  $- \Rightarrow 'nr :: \text{mod-type } \Rightarrow -$ )

====> (*Mod-Type-Connect.HMA-I* ::  $- \Rightarrow 'nr :: \text{mod-type } \Rightarrow -$ )

====> *Mod-Type-Connect.HMA-M*)

( $\lambda A$   $a$   $b$ . *swaprows*  $a$   $b$   $A$ ) *interchange-rows*

**by** (*intro rel-funI, goal-cases, auto simp add: Mod-Type-Connect.HMA-M-def interchange-rows-def*)

(*rule eq-matI, auto simp add: Mod-Type-Connect.from-hma<sub>m</sub>-def Mod-Type-Connect.HMA-I-def*

*to-nat-less-card to-nat-from-nat-id*)

**lemma** *HMA-swapcols*[transfer-rule]:

((*Mod-Type-Connect.HMA-M* ::  $- \Rightarrow 'a :: \text{comm-ring-1 } \hat{\ } 'nc :: \text{mod-type } \hat{\ } 'nr :: \text{mod-type } \Rightarrow -$ )

====> (*Mod-Type-Connect.HMA-I* ::  $- \Rightarrow 'nc :: \text{mod-type } \Rightarrow -$ )

====> (*Mod-Type-Connect.HMA-I* ::  $- \Rightarrow 'nc :: \text{mod-type } \Rightarrow -$ )

====> *Mod-Type-Connect.HMA-M*)

( $\lambda A$   $a$   $b$ . *swapcols*  $a$   $b$   $A$ ) *interchange-columns*

**by** (*intro rel-funI, goal-cases, auto simp add: Mod-Type-Connect.HMA-M-def interchange-columns-def*)

(*rule eq-matI, auto simp add: Mod-Type-Connect.from-hma<sub>m</sub>-def Mod-Type-Connect.HMA-I-def*

*to-nat-less-card to-nat-from-nat-id*)

**lemma** *HMA-addrow*[transfer-rule]:

((*Mod-Type-Connect.HMA-M* ::  $- \Rightarrow 'a :: \text{comm-ring-1 } \hat{\ } 'nc :: \text{mod-type } \hat{\ } 'nr :: \text{mod-type } \Rightarrow -$ )

$\implies (Mod\text{-}Type\text{-}Connect.HMA\text{-}I :: - \Rightarrow 'nr :: mod\text{-}type \Rightarrow -)$   
 $\implies (Mod\text{-}Type\text{-}Connect.HMA\text{-}I :: - \Rightarrow 'nr :: mod\text{-}type \Rightarrow -)$   
 $\implies (=)$   
 $\implies Mod\text{-}Type\text{-}Connect.HMA\text{-}M$   
 $(\lambda A a b q. addrow q a b A) row\text{-}add$   
**by** (*intro rel-funI, goal-cases, auto simp add: Mod-Type-Connect.HMA-M-def*  
*row-add-def*)  
*(rule eq-matI, auto simp add: Mod-Type-Connect.from-hma<sub>m</sub>-def Mod-Type-Connect.HMA-I-def*  
*to-nat-less-card to-nat-from-nat-id)*

**lemma** *HMA-addcol[transfer-rule]:*  
 $((Mod\text{-}Type\text{-}Connect.HMA\text{-}M :: - \Rightarrow 'a :: comm\text{-}ring\text{-}1 \wedge 'nc :: mod\text{-}type \wedge 'nr ::$   
 $mod\text{-}type \Rightarrow -)$   
 $\implies (Mod\text{-}Type\text{-}Connect.HMA\text{-}I :: - \Rightarrow 'nc :: mod\text{-}type \Rightarrow -)$   
 $\implies (Mod\text{-}Type\text{-}Connect.HMA\text{-}I :: - \Rightarrow 'nc :: mod\text{-}type \Rightarrow -)$   
 $\implies (=)$   
 $\implies Mod\text{-}Type\text{-}Connect.HMA\text{-}M$   
 $(\lambda A a b q. addcol q a b A) column\text{-}add$   
**by** (*intro rel-funI, goal-cases, auto simp add: Mod-Type-Connect.HMA-M-def*  
*column-add-def*)  
*(rule eq-matI, auto simp add: Mod-Type-Connect.from-hma<sub>m</sub>-def Mod-Type-Connect.HMA-I-def*  
*to-nat-less-card to-nat-from-nat-id)*

**lemma** *HMA-multrow[transfer-rule]:*  
 $((Mod\text{-}Type\text{-}Connect.HMA\text{-}M :: - \Rightarrow 'a :: comm\text{-}ring\text{-}1 \wedge 'nc :: mod\text{-}type \wedge 'nr ::$   
 $mod\text{-}type \Rightarrow -)$   
 $\implies (Mod\text{-}Type\text{-}Connect.HMA\text{-}I :: - \Rightarrow 'nr :: mod\text{-}type \Rightarrow -)$   
 $\implies (=)$   
 $\implies Mod\text{-}Type\text{-}Connect.HMA\text{-}M$   
 $(\lambda A i q. multrow i q A) mult\text{-}row$   
**by** (*intro rel-funI, goal-cases, auto simp add: Mod-Type-Connect.HMA-M-def*  
*mult-row-def*)  
*(rule eq-matI, auto simp add: Mod-Type-Connect.from-hma<sub>m</sub>-def Mod-Type-Connect.HMA-I-def*  
*to-nat-less-card to-nat-from-nat-id)*

**lemma** *HMA-multcol[transfer-rule]:*  
 $((Mod\text{-}Type\text{-}Connect.HMA\text{-}M :: - \Rightarrow 'a :: comm\text{-}ring\text{-}1 \wedge 'nc :: mod\text{-}type \wedge 'nr ::$   
 $mod\text{-}type \Rightarrow -)$   
 $\implies (Mod\text{-}Type\text{-}Connect.HMA\text{-}I :: - \Rightarrow 'nc :: mod\text{-}type \Rightarrow -)$   
 $\implies (=)$   
 $\implies Mod\text{-}Type\text{-}Connect.HMA\text{-}M$   
 $(\lambda A i q. multcol i q A) mult\text{-}column$   
**by** (*intro rel-funI, goal-cases, auto simp add: Mod-Type-Connect.HMA-M-def*  
*mult-column-def*)  
*(rule eq-matI, auto simp add: Mod-Type-Connect.from-hma<sub>m</sub>-def Mod-Type-Connect.HMA-I-def*

*to-nat-less-card to-nat-from-nat-id*)

**end**

**fun** *HMA-M3* **where**

*HMA-M3* (*P,A,Q*)  
 (*P'* :: '*a* :: *comm-ring-1* ^ '*nr* :: *mod-type* ^ '*nr* :: *mod-type*,  
*A'* :: '*a* ^ '*nc* :: *mod-type* ^ '*nr* :: *mod-type*,  
*Q'* :: '*a* ^ '*nc* :: *mod-type* ^ '*nc* :: *mod-type*) =  
 (*Mod-Type-Connect.HMA-M P P'* ∧ *Mod-Type-Connect.HMA-M A A'* ∧ *Mod-Type-Connect.HMA-M Q Q'*)

**lemma** *HMA-M3-def*:

*HMA-M3 A B* = (*Mod-Type-Connect.HMA-M (fst A) (fst B)*  
 ∧ *Mod-Type-Connect.HMA-M (fst (snd A)) (fst (snd B))*  
 ∧ *Mod-Type-Connect.HMA-M (snd (snd A)) (snd (snd B))*)  
**by** (*smt (verit, ccfv-SIG) HMA-M3.simps prod.collapse*)

**context**

**includes** *lifting-syntax*

**begin**

**lemma** *Domainp-HMA-M3* [*transfer-domain-rule*]:

*Domainp (HMA-M3 :: ->(-×('a::comm-ring-1^'nc::mod-type^'nr::mod-type)×-)>-)*

= ( $\lambda(P,A,Q). P \in \text{carrier-mat CARD('nr) CARD('nr)} \wedge A \in \text{carrier-mat CARD('nr) CARD('nc)}$   
 ∧  $Q \in \text{carrier-mat CARD('nc) CARD('nc)}$ )

**proof** –

**let** *?HMA-M3* = *HMA-M3* :: ->(-×('a::comm-ring-1^'nc::mod-type^'nr::mod-type)×-)>-

**have** 1:  $P \in \text{carrier-mat CARD('nr) CARD('nr)} \wedge$

$A \in \text{carrier-mat CARD('nr) CARD('nc)} \wedge Q \in \text{carrier-mat CARD('nc)}$

*CARD('nc)*

**if** *Domainp ?HMA-M3 (P,A,Q)* **for** *P A Q*

**using** *that unfolding Domainp-iff by (auto simp add: Mod-Type-Connect.HMA-M-def)*

**have** 2: *Domainp ?HMA-M3 (P,A,Q)* **if** *PAQ*:  $P \in \text{carrier-mat CARD('nr) CARD('nr)}$

$\wedge A \in \text{carrier-mat CARD('nr) CARD('nc)} \wedge Q \in \text{carrier-mat CARD('nc) CARD('nc)}$  **for** *P A Q*

**proof** –

**let** *?P* = *Mod-Type-Connect.to-hma<sub>m</sub>* *P* :: '*a* ^ '*nr* :: *mod-type* ^ '*nr* :: *mod-type*

**let** *?A* = *Mod-Type-Connect.to-hma<sub>m</sub>* *A* :: '*a* ^ '*nc* :: *mod-type* ^ '*nr* :: *mod-type*

**let** *?Q* = *Mod-Type-Connect.to-hma<sub>m</sub>* *Q* :: '*a* ^ '*nc* :: *mod-type* ^ '*nc* :: *mod-type*

**have** *HMA-M3 (P,A,Q) (?P,?A,?Q)*

**by** (*auto simp add: Mod-Type-Connect.HMA-M-def PAQ*)

**thus** *?thesis unfolding Domainp-iff by auto*

**qed**

```

have fst x ∈ carrier-mat CARD('nr) CARD('nr) ∧ fst (snd x) ∈ carrier-mat
CARD('nr) CARD('nc)
  ∧ (snd (snd x)) ∈ carrier-mat CARD('nc) CARD('nc)
if Domainp ?HMA-M3 x for x using 1
by (metis (full-types) surjective-pairing that)
moreover have Domainp ?HMA-M3 x
  if fst x ∈ carrier-mat CARD('nr) CARD('nr) ∧ fst (snd x) ∈ carrier-mat
CARD('nr) CARD('nc)
  ∧ (snd (snd x)) ∈ carrier-mat CARD('nc) CARD('nc) for x
using 2
by (metis (full-types) surjective-pairing that)
ultimately show ?thesis by (intro ext iffI, unfold split-beta, metis+)
qed

```

```

lemma bi-unique-HMA-M3 [transfer-rule]: bi-unique HMA-M3 left-unique HMA-M3
right-unique HMA-M3
unfolding HMA-M3-def bi-unique-def left-unique-def right-unique-def
by (auto simp add: Mod-Type-Connect.HMA-M-def)

```

```

lemma right-total-HMA-M3 [transfer-rule]: right-total HMA-M3
unfolding HMA-M-def right-total-def
by (simp add: Mod-Type-Connect.HMA-M-def)

```

end

end

## 4 Missing results

```

theory SNF-Missing-Lemmas
imports
  Hermite.Hermite
  Mod-Type-Connect
  Jordan-Normal-Form.DL-Rank-Submatrix
  List-Index.List-Index
begin

```

This theory presents some missing lemmas that are required for the Smith normal form development. Some of them could be added to different AFP entries, such as the Jordan Normal Form AFP entry by René Thiemann and Akihisa Yamada.

However, not all the lemmas can be added directly, since some imports are required.

```

hide-const (open) C
hide-const (open) measure

```

## 4.1 Miscellaneous lemmas

**lemma** *sum-two-rw*:  $(\sum i = 0..<2. f i) = (\sum i \in \{0,1::nat\}. f i)$   
**by** (*rule sum.cong, auto*)

**lemma** *sum-common-left*:  
**fixes**  $f::'a \Rightarrow 'b::comm-ring-1$   
**assumes** *finite A*  
**shows**  $sum (\lambda i. c * f i) A = c * sum f A$   
**by** (*simp add: mult-hom.hom-sum*)

**lemma** *prod3-intro*:  
**assumes**  $fst A = a$  **and**  $fst (snd A) = b$  **and**  $snd (snd A) = c$   
**shows**  $A = (a,b,c)$  **using** *assms* **by** *auto*

## 4.2 Transfer rules for the HMA\_Connect file of the Perron-Frobenius development

**hide-const** (**open**) *HMA-M HMA-I to-hma<sub>m</sub> from-hma<sub>m</sub>*  
**hide-fact** (**open**) *from-hma<sub>m</sub>-def from-hma-to-hma<sub>m</sub> HMA-M-def HMA-I-def dim-row-transfer-rule dim-col-transfer-rule*

**context**  
**includes** *lifting-syntax*  
**begin**

**lemma** *HMA-invertible-matrix*[*transfer-rule*]:  
 $((HMA-Connect.HMA-M :: - \Rightarrow 'a :: comm-ring-1 \wedge 'n \wedge 'n \Rightarrow -) ==> (=))$   
*invertible-mat invertible*  
**proof** (*intro rel-funI, goal-cases*)  
**case**  $(1 x y)$   
**note**  $rel-xy[transfer-rule] = 1$   
**have** *eq-dim*:  $dim-col x = dim-row x$   
**using** *HMA-Connect.dim-col-transfer-rule HMA-Connect.dim-row-transfer-rule rel-xy*  
**by** *fastforce*  
**moreover** **have**  $\exists A'. y ** A' = Finite-Cartesian-Product.mat 1 \wedge A' ** y = Finite-Cartesian-Product.mat 1$   
**if** *xB*:  $x * B = 1_m (dim-row x)$  **and** *Bx*:  $B * x = 1_m (dim-row B)$  **for** *B*  
**proof** –  
**let**  $?A' = HMA-Connect.to-hma_m B:: 'a :: comm-ring-1 \wedge 'n \wedge 'n$   
**have** *rel-BA*[*transfer-rule*]:  $HMA-M B ?A'$   
**by** (*metis (no-types, lifting) Bx HMA-M-def eq-dim carrier-mat-triv dim-col-mat(1) from-hma<sub>m</sub>-def from-hma-to-hma<sub>m</sub> index-mult-mat(3) index-one-mat(3) rel-xy xB*)  
**have** [*simp*]:  $dim-row B = CARD('n)$  **using** *dim-row-transfer-rule rel-BA* **by** *blast*  
**have** [*simp*]:  $dim-row x = CARD('n)$  **using** *dim-row-transfer-rule rel-xy* **by** *blast*  
**have**  $y ** ?A' = Finite-Cartesian-Product.mat 1$  **using** *xB* **by** (*transfer, simp*)

```

moreover have  $?A' ** y = \text{Finite-Cartesian-Product.mat } 1$  using  $Bx$  by
(transfer, simp)
ultimately show ?thesis by blast
qed
moreover have  $\exists B. x * B = 1_m (\text{dim-row } x) \wedge B * x = 1_m (\text{dim-row } B)$ 
if  $yA: y ** A' = \text{Finite-Cartesian-Product.mat } 1$  and  $Ay: A' ** y = \text{Fi-}$ 
 $\text{nite-Cartesian-Product.mat } 1$  for  $A'$ 
proof –
let  $?B = (\text{from-hma}_m A')$ 
have [simp]:  $\text{dim-row } x = \text{CARD}('n)$  using dim-row-transfer-rule rel-xy by
blast
have [transfer-rule]:  $\text{HMA-M } ?B A'$  by (simp add: HMA-M-def)
hence [simp]:  $\text{dim-row } ?B = \text{CARD}('n)$  using dim-row-transfer-rule by auto
have  $x * ?B = 1_m (\text{dim-row } x)$  using  $yA$  by (transfer', auto)
moreover have  $?B * x = 1_m (\text{dim-row } ?B)$  using  $Ay$  by (transfer', auto)
ultimately show ?thesis by auto
qed
ultimately show ?case unfolding invertible-mat-def invertible-def inverts-mat-def
by auto
qed
end

```

### 4.3 Lemmas obtained from HOL Analysis using local type definitions

```

thm Cartesian-Space.invertible-mult
thm invertible-iff-is-unit
thm det-non-zero-imp-unit
thm mat-mult-left-right-inverse

```

```

lemma invertible-mat-zero:
assumes  $A: A \in \text{carrier-mat } 0 \ 0$ 
shows invertible-mat A
using  $A$  unfolding invertible-mat-def inverts-mat-def one-mat-def times-mat-def
scalar-prod-def
Matrix.row-def col-def carrier-mat-def
by (auto, metis (no-types, lifting) cong-mat not-less-zero)

```

```

lemma invertible-mult-JNF:
fixes  $A::'a::\text{comm-ring-1 mat}$ 
assumes  $A: A \in \text{carrier-mat } n \ n$  and  $B: B \in \text{carrier-mat } n \ n$ 
and  $\text{inv-A}: \text{invertible-mat } A$  and  $\text{inv-B}: \text{invertible-mat } B$ 
shows invertible-mat (A*B)
proof (cases n = 0)
case True
then show ?thesis using assms
by (simp add: invertible-mat-zero)
next
case False

```



**then show** *?thesis* **using**  
*invertible-mult*[**where** *?'a='a::comm-ring-1*, **where** *?'b='n::finite*, **where**  
*?'c='n::finite*,  
**where** *?'d='n::finite*, *untransferred*, *cancel-card-constraint*, *OF assms*] **by**  
*auto*  
**qed**

**lemma** *invertible-iff-is-unit-JNF*:  
**assumes** *A*:  $A \in \text{carrier-mat } n \ n$   
**shows** *invertible-mat*  $A \longleftrightarrow (\text{Determinant.det } A) \text{ dvd } 1$   
**proof** (*cases n=0*)  
**case** *True*  
**then show** *?thesis* **using** *det-dim-zero invertible-mat-zero* *A* **by** *auto*  
**next**  
**case** *False*  
**then show** *?thesis* **using** *invertible-iff-is-unit*[*untransferred*, *cancel-card-constraint*]  
*A* **by** *auto*  
**qed**

#### 4.4 Lemmas about matrices, submatrices and determinants

**thm** *mat-mult-left-right-inverse*  
**lemma** *mat-mult-left-right-inverse*:  
**fixes** *A* :: *'a::comm-ring-1 mat*  
**assumes** *A*:  $A \in \text{carrier-mat } n \ n$   
**and** *B*:  $B \in \text{carrier-mat } n \ n$  **and** *AB*:  $A * B = 1_m \ n$   
**shows**  $B * A = 1_m \ n$   
**proof** –  
**have** *Determinant.det*  $(A * B) = \text{Determinant.det } (1_m \ n)$  **using** *AB* **by** *auto*  
**hence** *Determinant.det*  $A * \text{Determinant.det } B = 1$   
**using** *Determinant.det-mult*[*OF A B*] *det-one* **by** *auto*  
**hence** *det-A*:  $(\text{Determinant.det } A) \text{ dvd } 1$  **and** *det-B*:  $(\text{Determinant.det } B) \text{ dvd } 1$   
**using** *dvd-triv-left dvd-triv-right* **by** *metis+*  
**hence** *inv-A*: *invertible-mat* *A* **and** *inv-B*: *invertible-mat* *B*  
**using** *A B invertible-iff-is-unit-JNF* **by** *blast+*  
**obtain** *B'* **where** *inv-BB'*: *inverts-mat* *B B'* **and** *inv-B'B*: *inverts-mat* *B' B*  
**using** *inv-B unfolding invertible-mat-def* **by** *auto*  
**have** *B'-carrier*:  $B' \in \text{carrier-mat } n \ n$   
**by** (*metis B inv-B'B inv-BB' carrier-matD(1) carrier-matD(2) carrier-mat-triv*  
*index-mult-mat(3) index-one-mat(3) inverts-mat-def*)  
**have**  $B * A * B = B$  **using** *A AB B* **by** *auto*  
**hence**  $B * A * (B * B') = B * B'$   
**by** (*smt (verit, best) A B B'-carrier assoc-mult-mat mult-carrier-mat*)  
**thus** *?thesis*  
**by** (*metis A B carrier-matD(1) carrier-matD(2) index-mult-mat(3) inv-BB'*  
*inverts-mat-def right-mult-one-mat'*)  
**qed**

**context** *comm-ring-1*

**begin**

**lemma** *col-submatrix-UNIV*:

**assumes**  $j < \text{card } \{i. i < \text{dim-col } A \wedge i \in J\}$

**shows**  $\text{col } (\text{submatrix } A \text{ UNIV } J) j = \text{col } A (\text{pick } J j)$

**proof** (*rule eq-vecI*)

**show**  $\text{dim-eq:dim-vec } (\text{col } (\text{submatrix } A \text{ UNIV } J) j) = \text{dim-vec } (\text{col } A (\text{pick } J j))$

**by** (*simp add: dim-submatrix(1)*)

**fix**  $i$  **assume**  $i < \text{dim-vec } (\text{col } A (\text{pick } J j))$

**show**  $\text{col } (\text{submatrix } A \text{ UNIV } J) j \$v i = \text{col } A (\text{pick } J j) \$v i$

**by** (*smt (verit) Collect-cong assms col-def dim-col dim-eq dim-submatrix(1) eq-vecI index-vec pick-UNIV submatrix-index*)

**qed**

**lemma** *submatrix-split2*:  $\text{submatrix } A I J = \text{submatrix } (\text{submatrix } A I \text{ UNIV}) \text{ UNIV } J$  (**is**  $?lhs = ?rhs$ )

**proof** (*rule eq-matI*)

**show**  $dr: \text{dim-row } ?lhs = \text{dim-row } ?rhs$

**by** (*simp add: dim-submatrix(1)*)

**show**  $dc: \text{dim-col } ?lhs = \text{dim-col } ?rhs$

**by** (*simp add: dim-submatrix(2)*)

**fix**  $i j$  **assume**  $i: i < \text{dim-row } ?rhs$

**and**  $j: j < \text{dim-col } ?rhs$

**have**  $?rhs \$\$ (i, j) = (\text{submatrix } A I \text{ UNIV}) \$\$ (\text{pick } \text{UNIV } i, \text{pick } J j)$

**proof** (*rule submatrix-index*)

**show**  $i < \text{card } \{i. i < \text{dim-row } (\text{submatrix } A I \text{ UNIV}) \wedge i \in \text{UNIV}\}$

**by** (*metis (full-types) dim-submatrix(1) i*)

**show**  $j < \text{card } \{j. j < \text{dim-col } (\text{submatrix } A I \text{ UNIV}) \wedge j \in J\}$

**by** (*metis (full-types) dim-submatrix(2) j*)

**qed**

**also have**  $\dots = A \$\$ (\text{pick } I (\text{pick } \text{UNIV } i), \text{pick } \text{UNIV } (\text{pick } J j))$

**proof** (*rule submatrix-index*)

**show**  $\text{pick } \text{UNIV } i < \text{card } \{i. i < \text{dim-row } A \wedge i \in I\}$

**by** (*metis (full-types) dr dim-submatrix(1) i pick-UNIV*)

**show**  $\text{pick } J j < \text{card } \{j. j < \text{dim-col } A \wedge j \in \text{UNIV}\}$

**by** (*metis (full-types) dim-submatrix(2) j pick-le*)

**qed**

**also have**  $\dots = ?lhs \$\$ (i, j)$

**proof** (*unfold pick-UNIV, rule submatrix-index[symmetric]*)

**show**  $i < \text{card } \{i. i < \text{dim-row } A \wedge i \in I\}$

**by** (*metis (full-types) dim-submatrix(1) dr i*)

**show**  $j < \text{card } \{j. j < \text{dim-col } A \wedge j \in J\}$

**by** (*metis (full-types) dim-submatrix(2) dc j*)

**qed**

**finally show**  $?lhs \$\$ (i, j) = ?rhs \$\$ (i, j) ..$

**qed**

**lemma** *submatrix-mult*:

$\text{submatrix } (A * B) I J = \text{submatrix } A I \text{ UNIV} * \text{submatrix } B \text{ UNIV } J$  (**is**  $?lhs =$

$?rhs$   
**proof** (rule eq-matI)  
 show  $\dim\text{-row } ?lhs = \dim\text{-row } ?rhs$  **unfolding** submatrix-def **by** auto  
 show  $\dim\text{-col } ?lhs = \dim\text{-col } ?rhs$  **unfolding** submatrix-def **by** auto  
**fix**  $i\ j$  **assume**  $i: i < \dim\text{-row } ?rhs$  **and**  $j: j < \dim\text{-col } ?rhs$   
**have**  $i1: i < \text{card } \{i. i < \dim\text{-row } (A * B) \wedge i \in I\}$   
   **by** (metis (full-types) dim-submatrix(1) i index-mult-mat(2))  
**have**  $j1: j < \text{card } \{j. j < \dim\text{-col } (A * B) \wedge j \in J\}$   
   **by** (metis dim-submatrix(2) index-mult-mat(3) j)  
**have**  $pi: \text{pick } I\ i < \dim\text{-row } A$  **using**  $i1$  pick-le **by** auto  
**have**  $pj: \text{pick } J\ j < \dim\text{-col } B$  **using**  $j1$  pick-le **by** auto  
**have** row-rw:  $\text{Matrix.row } (\text{submatrix } A\ I\ UNIV)\ i = \text{Matrix.row } A\ (\text{pick } I\ i)$   
   **using**  $i1$  row-submatrix-UNIV **by** auto  
**have** col-rw:  $\text{col } (\text{submatrix } B\ UNIV\ J)\ j = \text{col } B\ (\text{pick } J\ j)$  **using**  $j1$  col-submatrix-UNIV  
**by** auto  
**have**  $?lhs\ \$\$ (i,j) = (A*B)\ \$\$ (\text{pick } I\ i, \text{pick } J\ j)$  **by** (rule submatrix-index[OF  
 $i1\ j1$ ])  
**also** **have**  $\dots = \text{Matrix.row } A\ (\text{pick } I\ i) \cdot \text{col } B\ (\text{pick } J\ j)$  **by** (rule index-mult-mat(1)[OF  
 $pi\ pj$ ])  
**also** **have**  $\dots = \text{Matrix.row } (\text{submatrix } A\ I\ UNIV)\ i \cdot \text{col } (\text{submatrix } B\ UNIV\ J)\ j$   
   **using** row-rw col-rw **by** simp  
**also** **have**  $\dots = (?rhs)\ \$\$ (i,j)$  **by** (rule index-mult-mat[symmetric], insert  $i\ j$ ,  
 auto)  
**finally** **show**  $?lhs\ \$\$ (i, j) = ?rhs\ \$\$ (i, j)$  .  
**qed**

**lemma** det-singleton:  
**assumes**  $A: A \in \text{carrier-mat } 1\ 1$   
**shows**  $\det A = A\ \$\$ (0,0)$   
**using**  $A$  **unfolding** carrier-mat-def Determinant.det-def **by** auto

**lemma** submatrix-singleton-index:  
**assumes**  $A: A \in \text{carrier-mat } n\ m$   
**and**  $an: a < n$  **and**  $bm: b < m$   
**shows**  $\text{submatrix } A\ \{a\}\ \{b\}\ \$\$ (0,0) = A\ \$\$ (a,b)$   
**proof** –  
**have**  $a: \{i. i = a \wedge i < \dim\text{-row } A\} = \{a\}$  **using**  $an$   $A$  **unfolding** carrier-mat-def  
**by** auto  
**have**  $b: \{i. i = b \wedge i < \dim\text{-col } A\} = \{b\}$  **using**  $bm$   $A$  **unfolding** carrier-mat-def  
**by** auto  
**have**  $\text{submatrix } A\ \{a\}\ \{b\}\ \$\$ (0,0) = A\ \$\$ (\text{pick } \{a\}\ 0, \text{pick } \{b\}\ 0)$   
   **by** (rule submatrix-index, insert  $a\ b$ , auto)  
**moreover** **have**  $\text{pick } \{a\}\ 0 = a$  **by** (auto, metis (full-types) LeastI)  
**moreover** **have**  $\text{pick } \{b\}\ 0 = b$  **by** (auto, metis (full-types) LeastI)  
**ultimately** **show**  $?thesis$  **by** simp  
**qed**  
**end**

**lemma** *det-not-inj-on*:

**assumes** *not-inj-on*:  $\neg \text{inj-on } f \{0..<n\}$

**shows**  $\det (\text{mat}_r \ n \ n \ (\lambda i. \text{Matrix.row } B \ (f \ i))) = 0$

**proof** –

**obtain**  $i \ j$  **where**  $i: i < n$  **and**  $j: j < n$  **and**  $f_i = f_j$  **and**  $i \neq j$

**using** *not-inj-on* **unfolding** *inj-on-def* **by** *auto*

**show** *?thesis*

**proof** (*rule det-identical-rows*[*OF - ij i j*])

**let**  $?B = (\text{mat}_r \ n \ n \ (\lambda i. \text{row } B \ (f \ i)))$

**show**  $\text{row } ?B \ i = \text{row } ?B \ j$

**proof** (*rule eq-vecI*, *auto*)

**fix**  $ia$  **assume**  $ia: ia < n$

**have**  $\text{row } ?B \ i \ \$ \ ia = ?B \ \$\$ \ (i, ia)$  **by** (*rule index-row*(1), *insert i ia*, *auto*)

**also have**  $\dots = ?B \ \$\$ \ (j, ia)$  **by** (*simp add: f\_i = f\_j*  $i \ ia \ j$ )

**also have**  $\dots = \text{row } ?B \ j \ \$ \ ia$  **by** (*rule index-row*(1)[*symmetric*], *insert j ia*, *auto*)

**finally show**  $\text{row } ?B \ i \ \$ \ ia = \text{row } (\text{mat}_r \ n \ n \ (\lambda i. \text{row } B \ (f \ i))) \ j \ \$ \ ia$  **by** *simp*

**qed**

**show**  $\text{mat}_r \ n \ n \ (\lambda i. \text{Matrix.row } B \ (f \ i)) \in \text{carrier-mat } n \ n$  **by** *auto*

**qed**

**qed**

**lemma** *mat-row-transpose*:  $(\text{mat}_r \ nr \ nc \ f)^T = \text{mat } nc \ nr \ (\lambda(i,j). \text{vec-index } (f \ j) \ i)$

**by** (*rule eq-matI*, *auto*)

**lemma** *obtain-inverse-matrix*:

**assumes**  $A: A \in \text{carrier-mat } n \ n$  **and**  $i: \text{invertible-mat } A$

**obtains**  $B$  **where**  $\text{inverts-mat } A \ B$  **and**  $\text{inverts-mat } B \ A$  **and**  $B \in \text{carrier-mat } n \ n$

**proof** –

**have**  $(\exists B. \text{inverts-mat } A \ B \ \wedge \ \text{inverts-mat } B \ A)$  **using**  $i$  **unfolding** *invertible-mat-def* **by** *auto*

**from this obtain**  $B$  **where**  $AB: \text{inverts-mat } A \ B$  **and**  $BA: \text{inverts-mat } B \ A$  **by** *auto*

**moreover have**  $B \in \text{carrier-mat } n \ n$  **using**  $A \ AB \ BA$  **unfolding** *carrier-mat-def* *inverts-mat-def*

**by** (*auto*, *metis index-mult-mat*(3) *index-one-mat*(3))+

**ultimately show** *?thesis* **using** *that* **by** *blast*

**qed**

**lemma** *invertible-mat-smult-mat*:

**fixes**  $A :: 'a::\text{comm-ring-1} \ \text{mat}$

**assumes**  $\text{inv-A}: \text{invertible-mat } A$  **and**  $k: k \ \text{dvd} \ 1$

**shows**  $\text{invertible-mat } (k \cdot_m \ A)$

**proof** –  
**obtain**  $n$  **where**  $A: A \in \text{carrier-mat } n \ n$  **using**  $\text{inv-}A$  **unfolding**  $\text{invertible-mat-def}$   
**by**  $\text{auto}$   
**have**  $\text{det-dvd-1}: \text{Determinant.det } A \ \text{dvd } 1$  **using**  $\text{inv-}A$   $\text{invertible-iff-is-unit-JNF}[OF$   
 $A]$  **by**  $\text{auto}$   
**have**  $\text{Determinant.det } (k \cdot_m A) = k \wedge \text{dim-col } A * \text{Determinant.det } A$  **by**  $\text{simp}$   
**also have** ...  $\text{dvd } 1$  **by**  $(\text{rule unit-prod}, \text{insert } k \ \text{det-dvd-1} \ \text{dvd-power-same}, \text{force+})$   
**finally show**  $?thesis$  **using**  $\text{invertible-iff-is-unit-JNF}$  **by**  $(\text{metis } A \ \text{smult-carrier-mat})$   
**qed**

**lemma**  $\text{invertible-mat-one}[\text{simp}]: \text{invertible-mat } (1_m \ n)$   
**unfolding**  $\text{invertible-mat-def}$  **using**  $\text{inverts-mat-def}$  **by**  $\text{fastforce}$

**lemma**  $\text{four-block-mat-dim0}$ :  
**assumes**  $A: A \in \text{carrier-mat } n \ n$   
**and**  $B: B \in \text{carrier-mat } n \ 0$   
**and**  $C: C \in \text{carrier-mat } 0 \ n$   
**and**  $D: D \in \text{carrier-mat } 0 \ 0$   
**shows**  $\text{four-block-mat } A \ B \ C \ D = A$   
**unfolding**  $\text{four-block-mat-def}$  **using**  $\text{assms}$  **by**  $\text{auto}$

**lemma**  $\text{det-four-block-mat-lower-right-id}$ :  
**assumes**  $A: A \in \text{carrier-mat } m \ m$   
**and**  $B: B = 0_m \ m \ (n-m)$   
**and**  $C: C = 0_m \ (n-m) \ m$   
**and**  $D: D = 1_m \ (n-m)$   
**and**  $n > m$   
**shows**  $\text{Determinant.det } (\text{four-block-mat } A \ B \ C \ D) = \text{Determinant.det } A$   
**using**  $\text{assms}$   
**proof**  $(\text{induct } n \ \text{arbitrary}: A \ B \ C \ D)$   
**case**  $0$   
**then show**  $?case$  **by**  $\text{auto}$   
**next**  
**case**  $(\text{Suc } n)$   
**let**  $?block = (\text{four-block-mat } A \ B \ C \ D)$   
**let**  $?B = \text{Matrix.mat } m \ (n-m) \ (\lambda(i,j). \ 0)$   
**let**  $?C = \text{Matrix.mat } (n-m) \ m \ (\lambda(i,j). \ 0)$   
**let**  $?D = 1_m \ (n-m)$   
**have**  $\text{mat-eq}: (\text{mat-delete } ?block \ n \ n) = \text{four-block-mat } A \ ?B \ ?C \ ?D$  (**is**  $?lhs =$   
 $?rhs$ )  
**proof**  $(\text{rule eq-matI})$   
**fix**  $i \ j$  **assume**  $i: i < \text{dim-row } (\text{four-block-mat } A \ ?B \ ?C \ ?D)$   
**and**  $j: j < \text{dim-col } (\text{four-block-mat } A \ ?B \ ?C \ ?D)$   
**let**  $?f = (\text{if } i < \text{dim-row } A \ \text{then if } j < \text{dim-col } A \ \text{then } A \ \$\$ (i, j) \ \text{else } B \ \$\$ (i,$   
 $j - \text{dim-col } A)$   
**else if**  $j < \text{dim-col } A \ \text{then } C \ \$\$ (i - \text{dim-row } A, j) \ \text{else } D \ \$\$ (i - \text{dim-row } A,$   
 $j - \text{dim-col } A))$   
**let**  $?g = (\text{if } i < \text{dim-row } A \ \text{then if } j < \text{dim-col } A \ \text{then } A \ \$\$ (i, j) \ \text{else } ?B \ \$\$$

```

(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)
  have (mat-delete ?block n n) $$ (i,j) = ?block $$ (i,j)
  using i j Suc.prems unfolding mat-delete-def by auto
  also have ... = ?f
  by (rule index-mat-four-block, insert Suc.prems i j, auto)
  also have ... = ?g using i j Suc.prems by auto
  also have ... = four-block-mat A ?B ?C ?D $$ (i,j)
  by (rule index-mat-four-block[symmetric], insert Suc.prems i j, auto)
  finally show ?lhs $$ (i,j) = ?rhs $$ (i,j) .
qed (insert Suc.prems, auto)
have nn-1: ?block $$ (n, n) = 1 using Suc.prems by auto
have rw0: ( $\sum i < n.$  ?block $$ (i,n) * Determinant.cofactor ?block i n) = 0
proof (rule sum.neutral, rule)
  fix x assume x: x ∈ {..n}
  have block-index: ?block $$ (x,n) = (if x < dim-row A then if n < dim-col A
then A $$ (x, n)
  else B $$ (x, n - dim-col A) else if n < dim-col A then C $$ (x - dim-row
A, n)
  else D $$ (x - dim-row A, n - dim-col A))
  by (rule index-mat-four-block, insert Suc.prems x, auto)
  have four-block-mat A B C D $$ (x,n) = 0 using x Suc.prems by auto
  thus four-block-mat A B C D $$ (x, n) * Determinant.cofactor (four-block-mat
A B C D) x n = 0
  by simp
qed
have Determinant.det ?block = ( $\sum i < \text{Suc } n.$  ?block $$ (i, n) * Determinant.cofactor
?block i n)
  by (rule laplace-expansion-column, insert Suc.prems, auto)
  also have ... = ?block $$ (n, n) * Determinant.cofactor ?block n n
  + ( $\sum i < n.$  ?block $$ (i,n) * Determinant.cofactor ?block i n)
  by simp
  also have ... = ?block $$ (n, n) * Determinant.cofactor ?block n n using rw0
by auto
  also have ... = Determinant.cofactor ?block n n using nn-1 by simp
  also have ... = Determinant.det (mat-delete ?block n n) unfolding cofactor-def
by auto
  also have ... = Determinant.det (four-block-mat A ?B ?C ?D) using mat-eq by
simp
  also have ... = Determinant.det A (is Determinant.det ?lhs = Determinant.det
?rhs)
proof (cases n = m)
  case True
  have ?lhs = ?rhs by (rule four-block-mat-dim0, insert Suc.prems True, auto)
  then show ?thesis by simp
next
  case False
  show ?thesis by (rule Suc.hyps, insert Suc.prems False, auto)

```

qed  
 finally show ?case .  
 qed

**lemma** *mult-eq-first-row*:

assumes  $A: A \in \text{carrier-mat } 1 \ n$   
 and  $B: B \in \text{carrier-mat } m \ n$   
 and  $m0: m \neq 0$   
 and  $r: \text{Matrix.row } A \ 0 = \text{Matrix.row } B \ 0$   
**shows**  $\text{Matrix.row } (A * V) \ 0 = \text{Matrix.row } (B * V) \ 0$   
**proof** (rule *eq-vecI*)  
 show  $\text{dim-vec } (\text{Matrix.row } (A * V) \ 0) = \text{dim-vec } (\text{Matrix.row } (B * V) \ 0)$  **using**  
 $A \ B \ r$  **by** *auto*  
 fix  $i$  **assume**  $i: i < \text{dim-vec } (\text{Matrix.row } (B * V) \ 0)$   
 have  $\text{Matrix.row } (A * V) \ 0 \ \$v \ i = (A * V) \ \$\$ \ (0, i)$  **by** (rule *index-row*, *insert*  
 $i \ A$ , *auto*)  
 also have  $\dots = \text{Matrix.row } A \ 0 \cdot \text{col } V \ i$  **by** (rule *index-mult-mat*, *insert*  $A \ i$ ,  
*auto*)  
 also have  $\dots = \text{Matrix.row } B \ 0 \cdot \text{col } V \ i$  **using**  $r$  **by** *auto*  
 also have  $\dots = (B * V) \ \$\$ \ (0, i)$  **by** (rule *index-mult-mat[symmetric]*, *insert*  $m0$   
 $B \ i$ , *auto*)  
 also have  $\dots = \text{Matrix.row } (B * V) \ 0 \ \$v \ i$  **by** (rule *index-row[symmetric]*, *insert*  
 $i \ B \ m0$ , *auto*)  
 finally show  $\text{Matrix.row } (A * V) \ 0 \ \$v \ i = \text{Matrix.row } (B * V) \ 0 \ \$v \ i$  .  
 qed

**lemma** *smult-mat-mat-one-element*:

assumes  $A: A \in \text{carrier-mat } 1 \ 1$  and  $B: B \in \text{carrier-mat } 1 \ n$   
**shows**  $A * B = A \ \$\$ \ (0, 0) \cdot_m \ B$   
**proof** (rule *eq-matI*)  
 fix  $i \ j$  **assume**  $i: i < \text{dim-row } (A \ \$\$ \ (0, 0) \cdot_m \ B)$  and  $j: j < \text{dim-col } (A \ \$\$ \ (0,$   
 $0) \cdot_m \ B)$   
 have  $i0: i = 0$  **using**  $A \ B \ i$  **by** *auto*  
 have  $(A * B) \ \$\$ \ (i, j) = \text{Matrix.row } A \ i \cdot \text{col } B \ j$   
 by (rule *index-mult-mat*, *insert*  $i \ j \ A \ B$ , *auto*)  
 also have  $\dots = \text{Matrix.row } A \ i \ \$v \ 0 * \text{col } B \ j \ \$v \ 0$  **unfolding** *scalar-prod-def*  
**using**  $B$  **by** *auto*  
 also have  $\dots = A \ \$\$ \ (i, i) * B \ \$\$ \ (i, j)$  **using**  $A \ i \ i0 \ j$  **by** *auto*  
 also have  $\dots = (A \ \$\$ \ (i, i) \cdot_m \ B) \ \$\$ \ (i, j)$   
**unfolding**  $i$  **by** (rule *index-smult-mat[symmetric]*, *insert*  $i \ j \ B$ , *auto*)  
 finally show  $(A * B) \ \$\$ \ (i, j) = (A \ \$\$ \ (0, 0) \cdot_m \ B) \ \$\$ \ (i, j)$  **using**  $i0$  **by** *simp*  
 qed (*insert*  $A \ B$ , *auto*)

**lemma** *determinant-one-element*:

assumes  $A: A \in \text{carrier-mat } 1 \ 1$  **shows**  $\text{Determinant.det } A = A \ \$\$ \ (0, 0)$   
**proof** –  
 have  $\text{Determinant.det } A = \text{prod-list } (\text{diag-mat } A)$

by (rule det-upper-triangular[OF - A], insert A, unfold upper-triangular-def, auto)  
 also have ... = A \$\$ (0,0) using A unfolding diag-mat-def by auto  
 finally show ?thesis .  
 qed

**lemma** *invertible-mat-transpose*:

assumes *inv-A*: *invertible-mat* (A::'a::comm-ring-1 mat)  
 shows *invertible-mat*  $A^T$

**proof** –

obtain *n* where A:  $A \in \text{carrier-mat } n \ n$   
 using *inv-A* unfolding *invertible-mat-def* *square-mat.simps* by auto  
 hence At:  $A^T \in \text{carrier-mat } n \ n$  by *simp*  
 have *Determinant.det*  $A^T = \text{Determinant.det } A$   
 by (metis *Determinant.det-def* *Determinant.det-transpose carrier-matI*  
*index-transpose-mat(2)* *index-transpose-mat(3)*)  
 also have ... *dvd 1* using *invertible-iff-is-unit-JNF*[OF A] *inv-A* by *simp*  
 finally show ?thesis using *invertible-iff-is-unit-JNF*[OF At] by auto

qed

**lemma** *dvd-elements-mult-matrix-left*:

assumes A: (A::'a::comm-ring-1 mat)  $\in \text{carrier-mat } m \ n$   
 and P:  $P \in \text{carrier-mat } m \ m$   
 and x:  $(\forall i \ j. \ i < m \wedge j < n \longrightarrow x \ \text{dvd} \ A \ \$\$ (i,j))$   
 shows  $(\forall i \ j. \ i < m \wedge j < n \longrightarrow x \ \text{dvd} \ (P * A) \ \$\$ (i,j))$

**proof** –

have  $x \ \text{dvd} \ (P * A) \ \$\$ (i, j)$  if *i*:  $i < m$  and *j*:  $j < n$  for *i j*

**proof** –

have  $(P * A) \ \$\$ (i, j) = (\sum ia = 0..<m. \ \text{Matrix.row } P \ i \ \$v \ ia * \ \text{col } A \ j \ \$v \ ia)$

unfolding *times-mat-def* *scalar-prod-def* using A P *j i* by auto

also have ... =  $(\sum ia = 0..<m. \ \text{Matrix.row } P \ i \ \$v \ ia * \ A \ \$\$ (ia,j))$

by (rule *sum.cong*, insert A *j*, auto)

also have  $x \ \text{dvd} \ ...$  using *x* by (meson *atLeastLessThan-iff* *dvd-mult* *dvd-sum*

*j*)

finally show ?thesis .

qed

thus ?thesis by auto

qed

**lemma** *dvd-elements-mult-matrix-right*:

assumes A: (A::'a::comm-ring-1 mat)  $\in \text{carrier-mat } m \ n$   
 and Q:  $Q \in \text{carrier-mat } n \ n$   
 and x:  $(\forall i \ j. \ i < m \wedge j < n \longrightarrow x \ \text{dvd} \ A \ \$\$ (i,j))$   
 shows  $(\forall i \ j. \ i < m \wedge j < n \longrightarrow x \ \text{dvd} \ (A * Q) \ \$\$ (i,j))$

**proof** –



```

have x dvd (A*Q) $$ (i, j) if i: i < m and j: j < n for i j
proof -
  have (A*Q) $$ (i, j) = ( $\sum ia = 0..<n. Matrix.row A i $v ia * col Q j $v ia$ )
    unfolding times-mat-def scalar-prod-def using A Q j i by auto
  also have ... = ( $\sum ia = 0..<n. A $$ (i, ia) * col Q j $v ia$ )
    by (rule sum.cong, insert A Q i, auto)
  also have x dvd ... using x
    by (meson atLeastLessThan-iff dvd-mult2 dvd-sum i)
  finally show ?thesis .
qed
thus ?thesis by auto
qed

```

```

lemma dvd-elements-mult-matrix-left-right:
  assumes A: (A::'a::comm-ring-1 mat) ∈ carrier-mat m n
  and P: P ∈ carrier-mat m m
  and Q: Q ∈ carrier-mat n n
  and x: (∀ i j. i < m ∧ j < n → x dvd A $$ (i, j))
shows (∀ i j. i < m ∧ j < n → x dvd (P*A*Q) $$ (i, j))
  using dvd-elements-mult-matrix-left[OF A P x]
  by (meson P A Q dvd-elements-mult-matrix-right mult-carrier-mat)

```

**definition** *append-cols* :: 'a :: zero mat ⇒ 'a mat ⇒ 'a mat (**infixr** '@<sub>c</sub>' 65) **where**  
 $A @_c B = \text{four-block-mat } A B (0_m \ 0 \ (\text{dim-col } A)) (0_m \ 0 \ (\text{dim-col } B))$

```

lemma append-cols-carrier[simp,intro]:
  A ∈ carrier-mat n a ⇒ B ∈ carrier-mat n b ⇒ (A @c B) ∈ carrier-mat n (a+b)
  unfolding append-cols-def by auto

```

```

lemma append-cols-mult-left:
  assumes A: A ∈ carrier-mat n a
  and B: B ∈ carrier-mat n b
  and P: P ∈ carrier-mat n n
shows P * (A @c B) = (P*A) @c (P*B)
proof -
  let ?P = four-block-mat P (0m n 0) (0m 0 n) (0m 0 0)
  have P = ?P by (rule eq-matI, auto)
  hence P * (A @c B) = ?P * (A @c B) by simp
  also have ?P * (A @c B) = four-block-mat (P * A + 0m n 0 * 0m 0 (dim-col A))
    (P * B + 0m n 0 * 0m 0 (dim-col B)) (0m 0 n * A + 0m 0 0 * 0m 0 (dim-col A))
    (0m 0 n * B + 0m 0 0 * 0m 0 (dim-col B)) unfolding append-cols-def
  by (rule mult-four-block-mat, insert A B P, auto)
  also have ... = four-block-mat (P * A) (P * B) (0m 0 (dim-col (P*A))) (0m 0 (dim-col (P*B)))

```

by (rule cong-four-block-mat, insert P, auto)  
also have ... = (P\*A) @<sub>c</sub> (P\*B) **unfolding** append-cols-def **by** auto  
**finally show** ?thesis .  
**qed**

**lemma** append-cols-mult-right-id:

assumes A: (A::'a::semiring-1 mat) ∈ carrier-mat n 1  
and B: B ∈ carrier-mat n (m-1)  
and C: C = four-block-mat (1<sub>m</sub> 1) (0<sub>m</sub> 1 (m-1)) (0<sub>m</sub> (m-1) 1) D  
and D: D ∈ carrier-mat (m-1) (m-1)  
**shows** (A @<sub>c</sub> B) \* C = A @<sub>c</sub> (B \* D)  
**proof** -  
let ?C = four-block-mat (1<sub>m</sub> 1) (0<sub>m</sub> 1 (m-1)) (0<sub>m</sub> (m-1) 1) D  
have (A @<sub>c</sub> B) \* C = (A @<sub>c</sub> B) \* ?C **unfolding** C **by** auto  
also have ... = four-block-mat A B (0<sub>m</sub> 0 (dim-col A)) (0<sub>m</sub> 0 (dim-col B)) \*  
?C  
**unfolding** append-cols-def **by** auto  
also have ... = four-block-mat (A \* 1<sub>m</sub> 1 + B \* 0<sub>m</sub> (m-1) 1) (A \* 0<sub>m</sub> 1 (m-1) + B \* D)  
(0<sub>m</sub> 0 (dim-col A) \* 1<sub>m</sub> 1 + 0<sub>m</sub> 0 (dim-col B) \* 0<sub>m</sub> (m-1) 1)  
(0<sub>m</sub> 0 (dim-col A) \* 0<sub>m</sub> 1 (m-1) + 0<sub>m</sub> 0 (dim-col B) \* D)  
**by** (rule mult-four-block-mat, insert assms, auto)  
also have ... = four-block-mat A (B \* D) (0<sub>m</sub> 0 (dim-col A)) (0<sub>m</sub> 0 (dim-col  
(B\*D)))  
**by** (rule cong-four-block-mat, insert assms, auto)  
also have ... = A @<sub>c</sub> (B \* D) **unfolding** append-cols-def **by** auto  
**finally show** ?thesis .  
**qed**

**lemma** append-cols-mult-right-id2:

assumes A: (A::'a::semiring-1 mat) ∈ carrier-mat n a  
and B: B ∈ carrier-mat n b  
and C: C = four-block-mat D (0<sub>m</sub> a b) (0<sub>m</sub> b a) (1<sub>m</sub> b)  
and D: D ∈ carrier-mat a a  
**shows** (A @<sub>c</sub> B) \* C = (A \* D) @<sub>c</sub> B  
**proof** -  
let ?C = four-block-mat D (0<sub>m</sub> a b) (0<sub>m</sub> b a) (1<sub>m</sub> b)  
have (A @<sub>c</sub> B) \* C = (A @<sub>c</sub> B) \* ?C **unfolding** C **by** auto  
also have ... = four-block-mat A B (0<sub>m</sub> 0 a) (0<sub>m</sub> 0 b) \* ?C  
**unfolding** append-cols-def **using** A B **by** auto  
also have ... = four-block-mat (A \* D + B \* 0<sub>m</sub> b a) (A \* 0<sub>m</sub> a b + B \* 1<sub>m</sub> b)  
(0<sub>m</sub> 0 a \* D + 0<sub>m</sub> 0 b \* 0<sub>m</sub> b a) (0<sub>m</sub> 0 a \* 0<sub>m</sub> a b + 0<sub>m</sub> 0 b \* 1<sub>m</sub> b)  
**by** (rule mult-four-block-mat, insert A B C D, auto)  
also have ... = four-block-mat (A \* D) B (0<sub>m</sub> 0 (dim-col (A\*D))) (0<sub>m</sub> 0 (dim-col  
B))  
**by** (rule cong-four-block-mat, insert assms, auto)  
also have ... = (A \* D) @<sub>c</sub> B **unfolding** append-cols-def **by** auto  
**finally show** ?thesis .

qed

**lemma** *append-cols-nth*:

**assumes**  $A: A \in \text{carrier-mat } n \ a$

**and**  $B: B \in \text{carrier-mat } n \ b$

**and**  $i: i < n$  **and**  $j: j < a + b$

**shows**  $(A @_c B) \$\$ (i, j) = (\text{if } j < \text{dim-col } A \text{ then } A \$\$(i, j) \text{ else } B \$\$(i, j - a))$  (**is**  $?lhs = ?rhs$ )

**proof** –

**let**  $?C = (0_m \ 0 \ (\text{dim-col } A))$

**let**  $?D = (0_m \ 0 \ (\text{dim-col } B))$

**have**  $i2: i < \text{dim-row } A + \text{dim-row } ?D$  **using**  $i \ A$  **by** *auto*

**have**  $j2: j < \text{dim-col } A + \text{dim-col } (0_m \ 0 \ (\text{dim-col } B))$  **using**  $j \ B \ A$  **by** *auto*

**have**  $(A @_c B) \$\$ (i, j) = \text{four-block-mat } A \ B \ ?C \ ?D \$\$ (i, j)$

**unfolding** *append-cols-def* **by** *auto*

**also have**  $\dots = (\text{if } i < \text{dim-row } A \text{ then if } j < \text{dim-col } A \text{ then } A \$\$ (i, j)$

$\text{else } B \$\$ (i, j - \text{dim-col } A) \text{ else if } j < \text{dim-col } A \text{ then } ?C \$\$ (i - \text{dim-row } A, j)$

$\text{else } 0_m \ 0 \ (\text{dim-col } B) \$\$ (i - \text{dim-row } A, j - \text{dim-col } A))$

**by** (*rule index-mat-four-block(1)[OF i2 j2]*)

**also have**  $\dots = ?rhs$  **using**  $i \ A$  **by** *auto*

**finally show** *?thesis* .

qed

**lemma** *append-cols-split*:

**assumes**  $d: \text{dim-col } A > 0$

**shows**  $A = \text{mat-of-cols } (\text{dim-row } A) \ [\text{col } A \ 0] @_c$

$\text{mat-of-cols } (\text{dim-row } A) \ (\text{map } (\text{col } A) \ [1..<\text{dim-col } A])$  (**is**  $?lhs = ?A1$

$@_c \ ?A2$ )

**proof** (*rule eq-matI*)

**fix**  $i \ j$  **assume**  $i: i < \text{dim-row } (?A1 @_c ?A2)$  **and**  $j: j < \text{dim-col } (?A1 @_c ?A2)$

**have**  $(?A1 @_c ?A2) \$\$ (i, j) = (\text{if } j < \text{dim-col } ?A1 \text{ then } ?A1 \$\$(i, j) \text{ else } ?A2 \$\$(i, j - (\text{dim-col } ?A1)))$

**by** (*rule append-cols-nth, insert i j, auto simp add: append-cols-def*)

**also have**  $\dots = A \$\$ (i, j)$

**proof** (*cases j < dim-col ?A1*)

**case** *True*

**then show** *?thesis*

**by** (*metis One-nat-def Suc-eq-plus1 add.right-neutral append-cols-def col-def i*

*index-mat-four-block(2) index-vec index-zero-mat(2) less-one list.size(3)*

*list.size(4)*

*mat-of-cols-Cons-index-0 mat-of-cols-carrier(2) mat-of-cols-carrier(3)*)

**next**

**case** *False*

**then show** *?thesis*

**by** (*metis (no-types, lifting) Suc-eq-plus1 Suc-less-eq Suc-pred add-diff-cancel-right'*

*append-cols-def*

*diff-zero i index-col index-mat-four-block(2) index-mat-four-block(3) index-zero-mat(2)*)

$index-zero-mat(3) j$  length-map length-upt linordered-semidom-class.add-diff-inverse  
 $list.size(3)$   
 $list.size(4)$  mat-of-cols-carrier(2) mat-of-cols-carrier(3) mat-of-cols-index  
 $nth-map-upt$   
 $plus-1-eq-Suc$  upt-0)  
**qed**  
**finally show**  $A \text{ $$ } (i, j) = (?A1 @_c ?A2) \text{ $$ } (i, j) ..$   
**qed** (auto simp add: append-cols-def d)

**lemma** *append-rows-nth*:

**assumes**  $A: A \in carrier-mat\ a\ n$   
**and**  $B: B \in carrier-mat\ b\ n$   
**and**  $i: i < a+b$  **and**  $j: j < n$   
**shows**  $(A @_r B) \text{ $$ } (i, j) = (if\ i < dim-row\ A\ then\ A\ \text{ $$ } (i,j)\ else\ B\ \text{ $$ } (i-a,j))$  (is  
 $?lhs = ?rhs$ )  
**proof** –  
**let**  $?C = (0_m\ (dim-row\ A)\ 0)$   
**let**  $?D = (0_m\ (dim-row\ B)\ 0)$   
**have**  $i2: i < dim-row\ A + dim-row\ ?D$  **using**  $i\ j\ A\ B$  **by** auto  
**have**  $j2: j < dim-col\ A + dim-col\ ?D$  **using**  $i\ j\ A\ B$  **by** auto  
**have**  $(A @_r B) \text{ $$ } (i, j) = four-block-mat\ A\ ?C\ B\ ?D \text{ $$ } (i, j)$   
**unfolding** *append-rows-def* **by** auto  
**also have**  $... = (if\ i < dim-row\ A\ then\ if\ j < dim-col\ A\ then\ A \text{ $$ } (i, j)\ else\ ?C$   
 $\text{ $$ } (i, j - dim-col\ A)$   
 $else\ if\ j < dim-col\ A\ then\ B \text{ $$ } (i - dim-row\ A, j)\ else\ ?D \text{ $$ } (i - dim-row\ A,$   
 $j - dim-col\ A))$   
**by** (rule *index-mat-four-block(1)*[OF  $i2\ j2$ ])  
**also have**  $... = ?rhs$  **using**  $i\ A\ j\ B$  **by** auto  
**finally show** *thesis* .  
**qed**

**lemma** *append-rows-split*:

**assumes**  $k: k \leq dim-row\ A$   
**shows**  $A = (mat-of-rows\ (dim-col\ A)\ [Matrix.row\ A\ i.\ i \leftarrow [0..<k]]) @_r$   
 $(mat-of-rows\ (dim-col\ A)\ [Matrix.row\ A\ i.\ i \leftarrow [k..<dim-row\ A]])$  (is  
 $?lhs = ?A1 @_r ?A2$ )  
**proof** (rule *eq-matI*)  
**have**  $(?A1 @_r ?A2) \in carrier-mat\ (k + (dim-row\ A - k))\ (dim-col\ A)$   
**by** (rule *carrier-append-rows, insert k, auto*)  
**hence**  $A1-A2: (?A1 @_r ?A2) \in carrier-mat\ (dim-row\ A)\ (dim-col\ A)$  **using**  $k$   
**by** *simp*  
**thus**  $dim-row\ A = dim-row\ (?A1 @_r ?A2)$  **and**  $dim-col\ A = dim-col\ (?A1 @_r$   
 $?A2)$  **by** auto  
**fix**  $i\ j$  **assume**  $i: i < dim-row\ (?A1 @_r ?A2)$  **and**  $j: j < dim-col\ (?A1 @_r ?A2)$   
**have**  $(?A1 @_r ?A2) \text{ $$ } (i, j) = (if\ i < dim-row\ ?A1\ then\ ?A1 \text{ $$ } (i,j)\ else$   
 $?A2 \text{ $$ } (i - (dim-row\ ?A1), j))$   
**by** (rule *append-rows-nth, insert k i j, auto simp add: append-rows-def*)  
**also have**  $... = A \text{ $$ } (i, j)$

```

proof (cases i < dim-row ?A1)
  case True
  then show ?thesis
    by (metis (no-types, lifting) Matrix.row-def add.left-neutral add.right-neutral
append-rows-def
      index-mat(1) index-mat-four-block(3) index-vec index-zero-mat(3) j
length-map length-upt
      mat-of-rows-carrier(2,3) mat-of-rows-def nth-map-upt prod.simps(2))
  next
  case False
  let ?xs = (map (Matrix.row A) [k..<dim-row A])
  have dim-row-A1: dim-row ?A1 = k by auto
  have ?A2 $$ (i-k,j) = ?xs ! (i-k) $v j
    by (rule mat-of-rows-index, insert i k False A1-A2 j, auto)
  also have ... = A $$ (i,j) using A1-A2 False i j by auto
  finally show ?thesis using A1-A2 False i j by auto
qed
finally show A $$ (i, j) = (?A1 @r ?A2) $$ (i,j) by simp
qed

```

**lemma** transpose-mat-append-rows:

```

assumes A: A ∈ carrier-mat a n and B: B ∈ carrier-mat b n
shows (A @r B)T = AT @c BT
proof -
  have (four-block-mat A (0m a n) B (0m b n))T = four-block-mat AT BT (0m a
n)T (0m b n)T for n
    by (meson assms(1) assms(2) transpose-four-block-mat zero-carrier-mat)
  then show ?thesis
    by (metis Matrix.transpose-transpose append-cols-def append-rows-def assms(1)
assms(2)
      carrier-matD(2) index-transpose-mat(2) transpose-carrier-mat zero-transpose-mat)
qed

```

**lemma** transpose-mat-append-cols:

```

assumes A: A ∈ carrier-mat n a and B: B ∈ carrier-mat n b
shows (A @c B)T = AT @r BT
by (smt (verit, ccfv-threshold) Matrix.transpose-transpose assms(1) assms(2)
transpose-carrier-mat transpose-mat-append-rows)

```

**lemma** append-rows-mult-right:

```

assumes A: (A::'a::comm-semiring-1 mat) ∈ carrier-mat a n and B: B ∈ car-
rier-mat b n
  and Q: Q ∈ carrier-mat n n
shows (A @r B) * Q = (A * Q) @r (B * Q)
proof -
  have transpose-mat ((A @r B) * Q) = QT * (A @r B)T
    by (rule transpose-mult, insert A B Q, auto)

```

**also have**  $\dots = Q^T * (A^T @_c B^T)$  **using** *transpose-mat-append-rows* **assms** **by** *metis*  
**also have**  $\dots = Q^T * A^T @_c Q^T * B^T$   
**using** *append-cols-mult-left* **assms** **by** (*metis* *transpose-carrier-mat*)  
**also have** *transpose-mat*  $\dots = (A * Q) @_r (B * Q)$   
**by** (*smt* (*verit*, *ccfv-threshold*) *A B Matrix.transpose-mult Matrix.transpose-transpose*  
*append-cols-def* *append-rows-def* *assms*(3) *carrier-matD*(1) *index-mult-mat*(2)  
*index-transpose-mat*(3) *mult-carrier-mat* *transpose-four-block-mat* *zero-carrier-mat*  
*zero-transpose-mat*)  
**finally show** *?thesis* **by** *simp*  
**qed**

**lemma** *append-rows-mult-left-id*:

**assumes** *A*: (*A*::'a::comm-semiring-1 mat)  $\in$  *carrier-mat* 1 *n*  
**and** *B*: *B*  $\in$  *carrier-mat* (*m*-1) *n*  
**and** *C*: *C* = *four-block-mat* (*1*<sub>*m*</sub> 1) (*0*<sub>*m*</sub> 1 (*m* - 1)) (*0*<sub>*m*</sub> (*m* - 1) 1) *D*  
**and** *D*: *D*  $\in$  *carrier-mat* (*m*-1) (*m*-1)  
**shows** *C* \* (*A* @<sub>*r*</sub> *B*) = *A* @<sub>*r*</sub> (*D* \* *B*)  
**proof** –  
**have** *transpose-mat* (*C* \* (*A* @<sub>*r*</sub> *B*)) = (*A* @<sub>*r*</sub> *B*)<sup>*T*</sup> \* *C*<sup>*T*</sup>  
**by** (*metis* (*no-types*, *lifting*) *B C D Matrix.transpose-mult* *append-rows-def* *A*  
*carrier-matD*  
*carrier-mat-triv* *index-mat-four-block*(2,3) *index-zero-mat*(2) *one-carrier-mat*)  
**also have**  $\dots = (A^T @_c B^T) * C^T$  **using** *transpose-mat-append-rows*[*OF* *A B*]  
**by** *auto*  
**also have**  $\dots = A^T @_c (B^T * D^T)$  **by** (*rule* *append-cols-mult-right-id*) (*use* *A B*  
*C D* **in** *auto*)  
**also have** *transpose-mat*  $\dots = A @_r (D * B)$  **using** *A*  
**by** (*metis* (*no-types*, *opaque-lifting*)  
*Matrix.transpose-mult* *Matrix.transpose-transpose* *assms*(2) *assms*(4)  
*mult-carrier-mat* *transpose-mat-append-rows*)  
**finally show** *?thesis* **by** *auto*  
**qed**

**lemma** *append-rows-mult-left-id2*:

**assumes** *A*: (*A*::'a::comm-semiring-1 mat)  $\in$  *carrier-mat* *a* *n*  
**and** *B*: *B*  $\in$  *carrier-mat* *b* *n*  
**and** *C*: *C* = *four-block-mat* *D* (*0*<sub>*m*</sub> *a* *b*) (*0*<sub>*m*</sub> *b* *a*) (*1*<sub>*m*</sub> *b*)  
**and** *D*: *D*  $\in$  *carrier-mat* *a* *a*  
**shows** *C* \* (*A* @<sub>*r*</sub> *B*) = (*D* \* *A*) @<sub>*r*</sub> *B*  
**proof** –  
**have** (*C* \* (*A* @<sub>*r*</sub> *B*))<sup>*T*</sup> = (*A* @<sub>*r*</sub> *B*)<sup>*T*</sup> \* *C*<sup>*T*</sup> **by** (*rule* *transpose-mult*, *insert* *assms*,  
*auto*)  
**also have**  $\dots = (A^T @_c B^T) * C^T$  **by** (*metis* *A B* *transpose-mat-append-rows*)  
**also have**  $\dots = (A^T * D^T @_c B^T)$  **by** (*rule* *append-cols-mult-right-id2*, *insert*  
*assms*, *auto*)  
**also have**  $\dots^T = (D * A) @_r B$   
**by** (*metis* *A B D* *transpose-mult* *transpose-transpose* *mult-carrier-mat* *transpose-mat-append-rows*)

finally show *?thesis* by *simp*  
qed

**lemma** *four-block-mat-preserves-column*:

assumes  $A: (A::'a::\text{semiring-1 mat}) \in \text{carrier-mat } n \ m$   
and  $B: B = \text{four-block-mat } (1_m \ 1) \ (0_m \ 1 \ (m - 1)) \ (0_m \ (m - 1) \ 1) \ C$   
and  $C: C \in \text{carrier-mat } (m-1) \ (m-1)$   
and  $i: i < n$  and  $m: 0 < m$   
shows  $(A*B) \ \$\$ \ (i,0) = A \ \$\$ \ (i,0)$   
**proof** –  
let  $?A1 = \text{mat-of-cols } n \ [\text{col } A \ 0]$   
let  $?A2 = \text{mat-of-cols } n \ (\text{map } (\text{col } A) \ [1..<\text{dim-col } A])$   
have  $n2: \text{dim-row } A = n$  using  $A$  by *auto*  
have  $A = ?A1 \ @_c \ ?A2$  by (rule *append-cols-split*[of  $A$ , *unfolded*  $n2$ ], *insert*  $m$   
 $A$ , *auto*)  
hence  $A * B = (?A1 \ @_c \ ?A2) * B$  by *simp*  
also have  $\dots = ?A1 \ @_c \ (?A2 * C)$  by (rule *append-cols-mult-right-id*[OF - -  $B$   
 $C$ ], *insert*  $A$ , *auto*)  
also have  $\dots \ \$\$ \ (i,0) = ?A1 \ \$\$ \ (i,0)$  using *append-cols-nth* by (*simp* *add*:  
*append-cols-def*  $i$ )  
also have  $\dots = A \ \$\$ \ (i,0)$   
by (*metis*  $A \ i$  *carrier-matD*(1) *col-def* *index-vec* *mat-of-cols-Cons-index-0*)  
finally show *?thesis* .  
qed

**definition** *lower-triangular*  $A = (\forall i \ j. \ i < j \wedge i < \text{dim-row } A \wedge j < \text{dim-col } A \rightarrow A \ \$\$ \ (i,j) = 0)$

**lemma** *lower-triangular-index*:

assumes *lower-triangular*  $A \ i < j \ i < \text{dim-row } A \ j < \text{dim-col } A$   
shows  $A \ \$\$ \ (i,j) = 0$   
using *assms* **unfolding** *lower-triangular-def* by *auto*

**lemma** *commute-multiples-identity*:

assumes  $A: (A::'a::\text{comm-ring-1 mat}) \in \text{carrier-mat } n \ n$   
shows  $A * (k \cdot_m \ (1_m \ n)) = (k \cdot_m \ (1_m \ n)) * A$   
**proof** –  
have  $(\sum ia = 0..<n. A \ \$\$ \ (i, ia) * (k * (\text{if } ia = j \ \text{then } 1 \ \text{else } 0)))$   
 $= (\sum ia = 0..<n. k * (\text{if } i = ia \ \text{then } 1 \ \text{else } 0) * A \ \$\$ \ (ia, j))$  (is *?lhs=?rhs*)  
if  $i: i < n$  and  $j: j < n$  for  $i \ j$   
**proof** –  
let  $?f = \lambda ia. A \ \$\$ \ (i, ia) * (k * (\text{if } ia = j \ \text{then } 1 \ \text{else } 0))$   
let  $?g = \lambda ia. k * (\text{if } i = ia \ \text{then } 1 \ \text{else } 0) * A \ \$\$ \ (ia, j)$   
have  $rw0: (\sum ia \in \{\{0..<n\} - \{j\}\}. ?f \ ia) = 0$  by (rule *sum.neutral*, *auto*)  
have  $rw0': (\sum ia \in \{\{0..<n\} - \{i\}\}. ?g \ ia) = 0$  by (rule *sum.neutral*, *auto*)  
have  $?lhs = ?f \ j + (\sum ia \in \{\{0..<n\} - \{j\}\}. ?f \ ia)$   
by (*smt* (*verit*, *del-insts*) *atLeast0LessThan* *finite-atLeastLessThan*  
*lessThan-iff* *sum.remove* *that*(2))

**also have** ... =  $A \text{ \$(\$ (i, j) * k)}$  **using** *rw0* **by** *auto*  
**also have** ... =  $?g \ i + (\sum ia \in (\{0..<n\} - \{i\}). ?g \ ia)$  **using** *rw0'* **by** *auto*  
**also have** ... = *?rhs*  
**by** (*smt (verit, ccfv-SIG) atLeast0LessThan finite-lessThan lessThan-iff*  
*sum.remove that(1)*)  
**finally show** *?thesis* .  
**qed**  
**thus** *?thesis* **using** *A*  
**by** (*metis (no-types, lifting) left-mult-one-mat mult-smult-assoc-mat mult-smult-distrib*  
*one-carrier-mat right-mult-one-mat*)  
**qed**

**lemma** *det-2*:

**assumes** *A*:  $A \in \text{carrier-mat } 2 \ 2$   
**shows** *Determinant.det*  $A = A \text{ \$(\$ (0,0) * A \$(\$ (1,1) - A \$(\$ (0,1) * A \$(\$ (1,0))}$   
**proof** –  
**let** *?A* = (*Mod-Type-Connect.to-hma<sub>m</sub>* *A*)::'*a*<sup>2</sup><sup>2</sup>  
**have** [*transfer-rule*]: *Mod-Type-Connect.HMA-M* *A* *?A*  
**unfolding** *Mod-Type-Connect.HMA-M-def* **using** *from-hma-to-hma<sub>m</sub>* *A* **by**  
*auto*  
**have** [*transfer-rule*]: *Mod-Type-Connect.HMA-I* *0* *0*  
**unfolding** *Mod-Type-Connect.HMA-I-def* **by** (*simp add: to-nat-0*)  
**have** [*transfer-rule*]: *Mod-Type-Connect.HMA-I* *1* *1*  
**unfolding** *Mod-Type-Connect.HMA-I-def* **by** (*simp add: to-nat-1*)  
**have** *Determinant.det*  $A = \text{Determinants.det } ?A$  **by** (*transfer, simp*)  
**also have** ... =  $?A \ \$h \ 1 \ \$h \ 1 * ?A \ \$h \ 2 \ \$h \ 2 - ?A \ \$h \ 1 \ \$h \ 2 * ?A \ \$h \ 2 \ \$h \ 1$   
**unfolding** *det-2* **by** *simp*  
**also have** ... =  $?A \ \$h \ 0 \ \$h \ 0 * ?A \ \$h \ 1 \ \$h \ 1 - ?A \ \$h \ 0 \ \$h \ 1 * ?A \ \$h \ 1 \ \$h \ 0$   
**by** (*smt (verit, ccfv-SIG) Groups.mult-ac(2) exhaust-2 semiring-norm(160)*)  
**also have** ... =  $A \text{ \$(\$ (0,0) * A \$(\$ (1,1) - A \$(\$ (0,1) * A \$(\$ (1,0))}$   
**unfolding** *index-hma-def[symmetric]* **by** (*transfer, auto*)  
**finally show** *?thesis* .  
**qed**

**lemma** *mat-diag-smult*:  $\text{mat-diag } n \ (\lambda x. (k::'a::\text{comm-ring-1})) = (k \cdot_m \ 1_m \ n)$

**proof** –  
**have**  $\text{mat-diag } n \ (\lambda x. k) = \text{mat-diag } n \ (\lambda x. k * 1)$  **by** *auto*  
**also have** ... =  $\text{mat-diag } n \ (\lambda x. k) * \text{mat-diag } n \ (\lambda x. 1)$  **using** *mat-diag-diag*  
**by** (*simp add: mat-diag-def*)  
**also have** ... =  $\text{mat-diag } n \ (\lambda x. k) * (1_m \ n)$  **by** *auto* **thm** *mat-diag-mult-left*  
**also have** ... =  $\text{Matrix.mat } n \ n \ (\lambda(i, j). k * (1_m \ n) \ \$(i, j))$  **by** (*rule mat-diag-mult-left, auto*)  
**also have** ... =  $(k \cdot_m \ 1_m \ n)$  **unfolding** *smult-mat-def* **by** *auto*  
**finally show** *?thesis* .  
**qed**

**lemma** *invertible-mat-four-block-mat-lower-right*:

**assumes** *A*: (*A*::'*a*::*comm-ring-1* *mat*)  $\in \text{carrier-mat } n \ n$  **and** *inv-A*: *invert-*



*ible-mat A*  
**shows** *invertible-mat (four-block-mat (1<sub>m</sub> 1) (0<sub>m</sub> 1 n) (0<sub>m</sub> n 1) A)*  
**proof** –  
**let** *?I = (four-block-mat (1<sub>m</sub> 1) (0<sub>m</sub> 1 n) (0<sub>m</sub> n 1) A)*  
**have** *Determinant.det ?I = Determinant.det (1<sub>m</sub> 1) \* Determinant.det A*  
**by** (*rule det-four-block-mat-lower-left-zero-col, insert assms, auto*)  
**also have** *... = Determinant.det A* **by** *auto*  
**finally have** *Determinant.det ?I = Determinant.det A .*  
**thus** *?thesis*  
**by** (*metis (no-types, lifting) assms carrier-matD(1) carrier-matD(2) carrier-mat-triv*  
*index-mat-four-block(2) index-mat-four-block(3) index-one-mat(2) index-one-mat(3)*  
*invertible-iff-is-unit-JNF*)  
**qed**

**lemma** *invertible-mat-four-block-mat-lower-right-id:*  
**assumes** *A: (A::'a::comm-ring-1 mat) ∈ carrier-mat m m* **and** *B: B = 0<sub>m</sub> m*  
*(n-m)* **and** *C: C = 0<sub>m</sub> (n-m) m*  
**and** *D: D = 1<sub>m</sub> (n-m)* **and** *n > m* **and** *inv-A: invertible-mat A*  
**shows** *invertible-mat (four-block-mat A B C D)*  
**proof** –  
**have** *Determinant.det (four-block-mat A B C D) = Determinant.det A*  
**by** (*rule det-four-block-mat-lower-right-id, insert assms, auto*)  
**thus** *?thesis using inv-A*  
**by** (*metis (no-types, lifting) assms(1) assms(4) carrier-matD(1) carrier-matD(2)*  
*carrier-mat-triv*  
*index-mat-four-block(2) index-mat-four-block(3) index-one-mat(2) index-one-mat(3)*  
*invertible-iff-is-unit-JNF*)  
**qed**

**lemma** *split-block4-decreases-dim-row:*  
**assumes** *E: (A,B,C,D) = split-block E 1 1*  
**and** *E1: dim-row E > 1* **and** *E2: dim-col E > 1*  
**shows** *dim-row D < dim-row E*  
**proof** –  
**have** *D ∈ carrier-mat (1 + (dim-row E - 2)) (1 + (dim-col E - 2))*  
**by** (*rule split-block(4)[OF E[symmetric]], insert E1 E2, auto*)  
**hence** *D ∈ carrier-mat (dim-row E - 1) (dim-col E - 1)* **using** *E1 E2* **by** *auto*  
**thus** *?thesis using E1* **by** *auto*  
**qed**

**lemma** *inv-P'PAQQ':*  
**assumes** *A: A ∈ carrier-mat n n*  
**and** *P: P ∈ carrier-mat n n*  
**and** *inv-P: inverts-mat P' P*  
**and** *inv-Q: inverts-mat Q Q'*  
**and** *Q: Q ∈ carrier-mat n n*  
**and** *P': P' ∈ carrier-mat n n*

**and**  $Q': Q' \in \text{carrier-mat } n \ n$   
**shows**  $(P'*(P*A*Q)*Q') = A$   
**proof** –  
**have**  $(P'*(P*A*Q)*Q') = (P'*(P*A*Q*Q'))$   
**by** (*meson*  $P \ P' \ Q \ Q'$  *assms*(1) *assoc-mult-mat mult-carrier-mat*)  
**also have**  $\dots = ((P'*P)*A*(Q*Q'))$   
**by** (*smt* (*verit*, *ccfv-SIG*)  $P' \ Q'$  *assms*(1) *assms*(2) *assms*(5) *assoc-mult-mat mult-carrier-mat*)  
**finally show** *?thesis*  
**by** (*metis*  $P' \ Q \ A \ \text{inv-}P \ \text{inv-}Q \ \text{carrier-mat}D(1) \ \text{inverts-mat-def} \ \text{left-mult-one-mat right-mult-one-mat}$ )  
**qed**

**lemma**

**assumes**  $U \in \text{carrier-mat } 2 \ 2$  **and**  $V \in \text{carrier-mat } 2 \ 2$  **and**  $A = U * V$   
**shows** *mat-mult2-00*:  $A \ \$\$ (0,0) = U \ \$\$ (0,0)*V \ \$\$ (0,0) + U \ \$\$ (0,1)*V \ \$\$ (1,0)$   
**and** *mat-mult2-01*:  $A \ \$\$ (0,1) = U \ \$\$ (0,0)*V \ \$\$ (0,1) + U \ \$\$ (0,1)*V \ \$\$ (1,1)$   
**and** *mat-mult2-10*:  $A \ \$\$ (1,0) = U \ \$\$ (1,0)*V \ \$\$ (0,0) + U \ \$\$ (1,1)*V \ \$\$ (1,0)$   
**and** *mat-mult2-11*:  $A \ \$\$ (1,1) = U \ \$\$ (1,0)*V \ \$\$ (0,1) + U \ \$\$ (1,1)*V \ \$\$ (1,1)$   
**using** *assms unfolding times-mat-def Matrix.row-def col-def scalar-prod-def*  
**using** *sum-two-rw by auto*

#### 4.5 Lemmas about sorted lists, insert and pick

**lemma** *sorted-distinct-imp-sorted-wrt*:

**assumes** *sorted xs and distinct xs*  
**shows** *sorted-wrt (<) xs*  
**using** *assms*  
**by** (*induct xs, insert le-neq-trans, auto*)

**lemma** *sorted-map-strict*:

**assumes** *strict-mono-on {0..<n} g*  
**shows** *sorted (map g [0..<n])*  
**using** *assms*  
**by** (*induct n, auto simp add: sorted-append strict-mono-on-def less-imp-le*)

**lemma** *sorted-list-of-set-map-strict*:

**assumes** *strict-mono-on {0..<n} g*  
**shows** *sorted-list-of-set (g ‘ {0..<n}) = map g [0..<n]*  
**using** *assms*  
**proof** (*induct n*)  
**case**  $0$   
**then show** *?case by auto*

```

next
  case (Suc n)
  note sg = Suc.prem
  have sg-n: strict-mono-on {0..<n} g using sg unfolding strict-mono-on-def by
  auto
  have g-image-rw: g ` {0..<Suc n} = insert (g n) (g ` {0..<n})
    by (simp add: set-upt-Suc)
  have sorted-list-of-set (g ` {0..<Suc n}) = sorted-list-of-set (insert (g n) (g `
  {0..<n}))
    using g-image-rw by simp
  also have ... = insort (g n) (sorted-list-of-set (g ` {0..<n}))
  proof (rule sorted-list-of-set-insert)
    have inj-on g {0..<Suc n} using sg strict-mono-on-imp-inj-on by blast
    thus g n ∉ g ` {0..<n} unfolding inj-on-def by fastforce
  qed (simp)
  also have ... = insort (g n) (map g [0..<n])
    using Suc.hyps sg unfolding strict-mono-on-def by auto
  also have ... = map g [0..<Suc n]
  proof (simp, rule sorted-insort-is-snoc)
    show sorted (map g [0..<n]) by (rule sorted-map-strict[OF sg-n])
    show ∀x∈set (map g [0..<n]). x ≤ g n using sg unfolding strict-mono-on-def
    by (simp add: less-imp-le)
  qed
  finally show ?case .
qed

```

**lemma** *sorted-nth-strict-mono*:

```

sorted xs ⇒ distinct xs ⇒ i < j ⇒ j < length xs ⇒ xs!i < xs!j
by (simp add: less-le nth-eq-iff-index-eq sorted-iff-nth-mono-less)

```

**lemma** *sorted-list-of-set-0-LEAST*:

```

assumes finI: finite I and I: I ≠ {}
shows sorted-list-of-set I ! 0 = (LEAST n. n∈I)
proof (rule Least-equality[symmetric])
  show sorted-list-of-set I ! 0 ∈ I
    by (metis I Max-in finI gr-zeroI in-set-conv-nth not-less-zero set-sorted-list-of-set)
  fix y assume y ∈ I
  thus sorted-list-of-set I ! 0 ≤ y
    by (metis eq-iff finI in-set-conv-nth neq0-conv sorted-iff-nth-mono-less
    sorted-list-of-set(1) sorted-sorted-list-of-set)
qed

```

**lemma** *sorted-list-of-set-eq-pick*:

```

assumes i: i < length (sorted-list-of-set I)
shows sorted-list-of-set I ! i = pick I i
proof -
  have finI: finite I

```

```

proof (rule ccontr)
  assume infinite I
  hence length (sorted-list-of-set I) = 0 by auto
  thus False using i by simp
qed
show ?thesis
using i
proof (induct i)
  case 0
  have I: I ≠ {} using 0.prem sorted-list-of-set-empty by blast
  show ?case unfolding pick.simps by (rule sorted-list-of-set-0-LEAST[OF finI
I])
next
  case (Suc i)
  note x-less = Suc.prem
  show ?case
  proof (unfold pick.simps, rule Least-equality[symmetric], rule conjI)
    show 1: pick I i < sorted-list-of-set I ! Suc i
    by (metis Suc.hyps Suc.prem Suc-lessD distinct-sorted-list-of-set find-first-unique
lessI
  nat-less-le sorted-sorted-list-of-set sorted-wrt-nth-less)
  show sorted-list-of-set I ! Suc i ∈ I
    using Suc.prem finI nth-mem set-sorted-list-of-set by blast
  have rw: sorted-list-of-set I ! i = pick I i
    using Suc.hyps Suc-lessD x-less by blast
  have sorted-less: sorted-list-of-set I ! i < sorted-list-of-set I ! Suc i
    by (simp add: 1 rw)
  fix y assume y: y ∈ I ∧ pick I i < y
  show sorted-list-of-set I ! Suc i ≤ y
    by (smt (verit) antisym-conv finI in-set-conv-nth less-Suc-eq less-Suc-eq-le
nat-neq-iff rw
  sorted-iff-nth-mono-less sorted-list-of-set(1) sorted-sorted-list-of-set x-less
y)
qed
qed
qed

```

$b$  is the position where we add,  $a$  the element to be added and  $i$  the position that is checked

```

lemma insort-nth':
  assumes  $\forall j < b. xs ! j < a$  and sorted xs and  $a \notin set\ xs$ 
    and  $i < length\ xs + 1$  and  $i < b$ 
    and  $xs \neq []$  and  $b < length\ xs$ 
  shows insort a xs ! i = xs ! i
  using assms
proof (induct xs arbitrary: a b i)
  case Nil
  then show ?case by auto
next

```

```

case (Cons x xs)
note less = Cons.prems(1)
note sorted = Cons.prems(2)
note a-notin = Cons.prems(3)
note i-length = Cons.prems(4)
note i-b = Cons.prems(5)
note b-length = Cons.prems(7)
show ?case
proof (cases a ≤ x)
  case True
    have insort a (x # xs) ! i = (a # x # xs) ! i using True by simp
    also have ... = (x # xs) ! i
      using Cons.prems(1) Cons.prems(5) True by force
    finally show ?thesis .
  next
    case False note x-less-a = False
    have insort a (x # xs) ! i = (x # insort a xs) ! i using False by simp
    also have ... = (x # xs) ! i
    proof (cases i = 0)
      case True
        then show ?thesis by auto
      next
        case False
        have (x # insort a xs) ! i = (insort a xs) ! (i-1)
          by (simp add: False nth-Cons')
        also have ... = xs ! (i-1)
        proof (rule Cons.hyps)
          show sorted xs using sorted by simp
          show a ∉ set xs using a-notin by simp
          show i - 1 < length xs + 1 using i-length False by auto
          show xs ≠ [] using i-b b-length by force
          show i - 1 < b - 1 by (simp add: False diff-less-mono i-b leI)
          show b - 1 < length xs using b-length i-b by auto
          show  $\forall j < b - 1. xs ! j < a$  using less less-diff-conv by auto
        qed
        also have ... = (x # xs) ! i by (simp add: False nth-Cons')
        finally show ?thesis .
      qed
    finally show ?thesis .
  qed
qed

```

```

lemma insort-nth:
assumes sorted xs and a ∉ set xs
  and i < index (insort a xs) a
  and xs ≠ []
shows insort a xs ! i = xs ! i
using assms

```

```

proof (induct xs arbitrary: a i)
case Nil
  then show ?case by auto
next
  case (Cons x xs)
  note sorted = Cons.prem1
  note a-notin = Cons.prem2
  note i-index = Cons.prem3
  show ?case
  proof (cases a ≤ x)
    case True
    have insort a (x # xs) ! i = (a # x # xs) ! i using True by simp
    also have ... = (x # xs) ! i
      using Cons.prem1 Cons.prem3 True by force
    finally show ?thesis .
  next
  case False note x-less-a = False
  show ?thesis
  proof (cases xs = [])
    case True
    have x ≠ a using False by auto
    then show ?thesis using True i-index False by auto
  next
  case False note xs-not-empty = False
  have insort a (x # xs) ! i = (x # insort a xs) ! i using x-less-a by simp
  also have ... = (x # xs) ! i
  proof (cases i = 0)
    case True
    then show ?thesis by auto
  next
  case False note i0 = False
  have (x # insort a xs) ! i = (insort a xs) ! (i-1)
    by (simp add: False nth-Cons')
  also have ... = xs ! (i-1)
  proof (rule Cons.hyps[OF - - - xs-not-empty])
    show sorted xs using sorted by simp
    show a ∉ set xs using a-notin by simp
    have index (insort a (x # xs)) a = index ((x # insort a xs)) a
      using x-less-a by auto
    also have ... = index (insort a xs) a + 1
      unfolding index-Cons using x-less-a by simp
    finally show i - 1 < index (insort a xs) a using False i-index by linarith
  qed
  also have ... = (x # xs) ! i by (simp add: False nth-Cons')
  finally show ?thesis .
qed
finally show ?thesis .
qed
qed

```

qed

lemma *insort-nth2*:

assumes *sorted xs* and  $a \notin \text{set } xs$   
and  $i < \text{length } xs$  and  $i \geq \text{index } (\text{insort } a \text{ } xs) \ a$   
and  $xs \neq []$   
shows  $\text{insort } a \text{ } xs ! (\text{Suc } i) = xs ! i$   
using *assms*  
proof (induct *xs* arbitrary:  $a \ i$ )  
case *Nil*  
then show ?*case* by *auto*  
next  
case (Cons  $x \ xs$ )  
note  $\text{sorted} = \text{Cons.prems}(1)$   
note  $a\text{-notin} = \text{Cons.prems}(2)$   
note  $i\text{-length} = \text{Cons.prems}(3)$   
note  $\text{index-}i = \text{Cons.prems}(4)$   
show ?*case*  
proof (cases  $a \leq x$ )  
case *True*  
have  $\text{insort } a \ (x \# \ xs) ! (\text{Suc } i) = (a \# \ x \# \ xs) ! (\text{Suc } i)$  using *True* by *simp*  
also have  $\dots = (x \# \ xs) ! i$   
using  $\text{Cons.prems}(1) \ \text{Cons.prems}(5) \ \text{True}$  by *force*  
finally show ?*thesis* .  
next  
case *False* note  $x\text{-less-}a = \text{False}$   
have  $\text{insort } a \ (x \# \ xs) ! (\text{Suc } i) = (x \# \ \text{insort } a \ \ xs) ! (\text{Suc } i)$  using *False* by  
*simp*  
also have  $\dots = (x \# \ xs) ! i$   
proof (cases  $i = 0$ )  
case *True*  
then show ?*thesis* using  $\text{index-}i \ \text{linear } x\text{-less-}a$  by *fastforce*  
next  
case *False* note  $i0 = \text{False}$   
show ?*thesis*  
proof –  
have  $\text{Suc-}i: \text{Suc } (i - 1) = i$   
using  $i0$  by *auto*  
have  $(x \# \ \text{insort } a \ \ xs) ! (\text{Suc } i) = (\text{insort } a \ \ xs) ! i$   
by (*simp* add:  $\text{nth-Cons}'$ )  
also have  $\dots = (\text{insort } a \ \ xs) ! \text{Suc } (i - 1)$  using  $\text{Suc-}i$  by *simp*  
also have  $\dots = xs ! (i - 1)$   
proof (rule  $\text{Cons.hyps}$ )  
show *sorted xs* using *sorted* by *simp*  
show  $a \notin \text{set } xs$  using  $a\text{-notin}$  by *simp*  
show  $i - 1 < \text{length } xs$  using  $i\text{-length}$  using  $\text{Suc-}i$  by *auto*  
thus  $xs \neq []$  by *auto*  
have  $\text{index } (\text{insort } a \ (x \# \ xs)) \ a = \text{index } ((x \# \ \text{insort } a \ \ xs)) \ a$  using  
 $x\text{-less-}a$  by *simp*

**also have**  $\dots = \text{index} (\text{insort } a \text{ } xs) \ a + 1$  **unfolding** *index-Cons* **using**  
*x-less-a* **by** *simp*  
**finally show**  $\text{index} (\text{insort } a \text{ } xs) \ a \leq i - 1$  **using** *index-i i0* **by** *auto*  
**qed**  
**also have**  $\dots = (x \# xs) ! i$  **using** *Suc-i* **by** *auto*  
**finally show** *?thesis* .  
**qed**  
**qed**  
**finally show** *?thesis* .  
**qed**  
**qed**

**lemma** *pick-index*:

**assumes**  $a: a \in I$  **and**  $a'\text{-card}: a' < \text{card } I$   
**shows**  $(\text{pick } I \ a' = a) = (\text{index} (\text{sorted-list-of-set } I) \ a = a')$   
**proof** –  
**have**  $\text{finI}: \text{finite } I$  **using**  $a'\text{-card}$  *card.infinite* **by** *force*  
**have**  $\text{length-I}: \text{length} (\text{sorted-list-of-set } I) = \text{card } I$   
**by** (*metis a'-card card.infinite distinct-card distinct-sorted-list-of-set not-less-zero set-sorted-list-of-set*)  
**let**  $?i = \text{index} (\text{sorted-list-of-set } I) \ a$   
**have**  $(\text{sorted-list-of-set } I) ! a' = \text{pick } I \ a'$   
**by** (*rule sorted-list-of-set-eq-pick, auto simp add: finI a'-card length-I*)  
**moreover have**  $(\text{sorted-list-of-set } I) ! ?i = a$   
**by** (*rule nth-index, simp add: a finI*)  
**ultimately show** *?thesis*  
**by** (*metis a'-card distinct-sorted-list-of-set index-nth-id length-I*)  
**qed**

**end**

## 5 The Cauchy–Binet formula

**theory** *Cauchy-Binet*

**imports**

*Diagonal-To-Smith*

*SNF-Missing-Lemmas*

**begin**

### 5.1 Previous missing results about *pick* and *insert*

**lemma** *pick-insert*:

**assumes**  $a\text{-notin-}I: a \notin I$  **and**  $i2: i < \text{card } I$   
**and**  $a\text{-def}: \text{pick} (\text{insert } a \text{ } I) \ a' = a$   
**and**  $ia': i < a'$   
**and**  $a'\text{-card}: a' < \text{card } I + 1$   
**shows**  $\text{pick} (\text{insert } a \text{ } I) \ i = \text{pick } I \ i$   
**proof** –  
**have**  $\text{finI}: \text{finite } I$



```

using i2
using card.infinite by force
have pick (insert a I) i = sorted-list-of-set (insert a I) ! i
proof (rule sorted-list-of-set-eq-pick[symmetric])
  have finite (insert a I)
    using card.infinite i2 by force
  thus i < length (sorted-list-of-set (insert a I))
    by (metis a-notin-I card-insert-disjoint distinct-card finite-insert
      i2 less-Suc-eq sorted-list-of-set(1) sorted-list-of-set(3))
qed
also have ... = insert a (sorted-list-of-set I) ! i
  by (simp add: a-notin-I finI)
also have ... = (sorted-list-of-set I) ! i
proof (rule insert-nth[OF])
  show sorted (sorted-list-of-set I) by auto
  show a ∉ set (sorted-list-of-set I) using a-notin-I
    by (metis card.infinite i2 not-less-zero set-sorted-list-of-set)
  have index (sorted-list-of-set (insert a I)) a = a'
    using pick-index a-def
    using a'-card a-notin-I finI by auto
  hence index (insert a (sorted-list-of-set I)) a = a'
    by (simp add: a-notin-I finI)
  thus i < index (insert a (sorted-list-of-set I)) a using ia' by auto
  show sorted-list-of-set I ≠ [] using finI i2 by fastforce
qed
also have ... = pick I i
proof (rule sorted-list-of-set-eq-pick)
  have finite I using card.infinite i2 by fastforce
  thus i < length (sorted-list-of-set I)
    by (metis distinct-card distinct-sorted-list-of-set i2 set-sorted-list-of-set)
qed
finally show ?thesis .
qed

```

**lemma** *pick-insert2*:

```

assumes a-notin-I: a ∉ I and i2: i < card I
  and a-def: pick (insert a I) a' = a
  and ia': i ≥ a'
  and a'-card: a' < card I + 1
shows pick (insert a I) i < pick I i
proof (cases i = 0)
  case True
  then show ?thesis
    by (auto, metis (mono-tags, lifting) DL-Missing-Sublist.pick.simps(1) Least-le
      a-def a-notin-I
      dual-order.order-iff-strict i2 ia' insertCI le-zero-eq not-less-Least pick-in-set-le)
  next
  case False

```

**hence**  $i0: i = \text{Suc } (i - 1)$  **using**  $a'\text{-card } ia'$  **by** *auto*  
**have**  $finI: \text{finite } I$   
**using**  $i2 \text{ card.infinite}$  **by** *force*  
**have**  $index\text{-}a'1: \text{index } (\text{sorted-list-of-set } (\text{insert } a \ I)) \ a = a'$   
**using**  $\text{pick-index}$   
**using**  $a'\text{-card } a\text{-def } a\text{-notin-}I \ finI$  **by** *auto*  
**hence**  $index\text{-}a': \text{index } (\text{insort } a \ (\text{sorted-list-of-set } I)) \ a = a'$   
**by**  $(\text{simp add: } a\text{-notin-}I \ finI)$   
**have**  $i1\text{-length}: i - 1 < \text{length } (\text{sorted-list-of-set } I)$  **using**  $i2$   
**by**  $(\text{metis } \text{distinct-card } \text{distinct-sorted-list-of-set } \text{sorted-list-of-set } \text{finI}$   
 $\text{less-imp-diff-less } \text{set-sorted-list-of-set})$   
**have**  $1: \text{pick } (\text{insert } a \ I) \ i = \text{sorted-list-of-set } (\text{insert } a \ I) \ ! \ i$   
**proof**  $(\text{rule } \text{sorted-list-of-set-eq-pick}[\text{symmetric}])$   
**have**  $\text{finite } (\text{insert } a \ I)$   
**using**  $\text{card.infinite } i2$  **by** *force*  
**thus**  $i < \text{length } (\text{sorted-list-of-set } (\text{insert } a \ I))$   
**by**  $(\text{metis } a\text{-notin-}I \ \text{card-insert-disjoint } \text{distinct-card } \text{finite-insert}$   
 $i2 \ \text{less-Suc-eq } \text{sorted-list-of-set}(1) \ \text{sorted-list-of-set}(3))$   
**qed**  
**also have**  $2: \dots = \text{insort } a \ (\text{sorted-list-of-set } I) \ ! \ i$   
**by**  $(\text{simp add: } a\text{-notin-}I \ finI)$   
**also have**  $\dots = \text{insort } a \ (\text{sorted-list-of-set } I) \ ! \ \text{Suc } (i-1)$  **using**  $i0$  **by** *auto*  
**also have**  $\dots < \text{pick } I \ i$   
**proof**  $(\text{cases } i = a^{\wedge})$   
**case** *True*  
**have**  $(\text{sorted-list-of-set } I) \ ! \ i > a$   
**by**  $(\text{smt } 1 \ \text{Suc-less-eq } \text{True } a\text{-def } a\text{-notin-}I \ \text{distinct-card } \text{distinct-sorted-list-of-set}$   
 $\text{finI } i2$   
 $ia' \ \text{index-a}' \ \text{insort-nth2 } \text{length-insort } \text{lessI } \text{list.size}(3) \ \text{nat-less-le } \text{not-less-zero}$   
 $\text{pick-in-set-le } \text{set-sorted-list-of-set } \text{sorted-list-of-set}(2)$   
 $\text{sorted-list-of-set-insert}$   
 $\text{sorted-list-of-set-eq-pick } \text{sorted-wrt-nth-less})$   
**moreover have**  $a = \text{insort } a \ (\text{sorted-list-of-set } I) \ ! \ i$  **using**  $\text{True } 1 \ 2 \ a\text{-def}$  **by**  
*auto*  
**ultimately show**  $?thesis$  **using**  $1 \ 2$   
**by**  $(\text{metis } \text{distinct-card } \text{finI } i0 \ i2 \ \text{set-sorted-list-of-set}$   
 $\text{sorted-list-of-set}(3) \ \text{sorted-list-of-set-eq-pick})$   
**next**  
**case** *False*  
**have**  $\text{insort } a \ (\text{sorted-list-of-set } I) \ ! \ \text{Suc } (i-1) = (\text{sorted-list-of-set } I) \ ! \ (i-1)$   
**by**  $(\text{rule } \text{insort-nth2}, \ \text{insert } i1\text{-length } \text{False } ia' \ \text{index-a}', \ \text{auto } \text{simp add: } a\text{-notin-}I$   
 $\text{finI})$   
**also have**  $\dots = \text{pick } I \ (i-1)$   
**by**  $(\text{rule } \text{sorted-list-of-set-eq-pick}[\text{OF } i1\text{-length}])$   
**also have**  $\dots < \text{pick } I \ i$  **using**  $i0 \ i2 \ \text{pick-mono-le}$  **by** *auto*  
**finally show**  $?thesis$  .  
**qed**  
**finally show**  $?thesis$  .  
**qed**

```

lemma pick-insert3:
  assumes a-notin-I:  $a \notin I$  and i2:  $i < \text{card } I$ 
    and a-def:  $\text{pick } (\text{insert } a \ I) \ a' = a$ 
    and ia':  $i \geq a'$ 
    and a'-card:  $a' < \text{card } I + 1$ 
  shows  $\text{pick } (\text{insert } a \ I) \ (\text{Suc } i) = \text{pick } I \ i$ 
proof (cases  $i = 0$ )
  case True
  have a-LEAST:  $a < (\text{LEAST } aa. aa \in I)$ 
    using True a-def a-notin-I i2 ia' pick-insert2 by fastforce
  have Least-rw:  $(\text{LEAST } aa. aa = a \vee aa \in I) = a$ 
    by (rule Least-equality, insert a-notin-I, auto,
        metis a-LEAST le-less-trans nat-le-linear not-less-Least)
  let ?P =  $\lambda aa. (aa = a \vee aa \in I) \wedge (\text{LEAST } aa. aa = a \vee aa \in I) < aa$ 
  let ?Q =  $\lambda aa. aa \in I$ 
  have ?P = ?Q unfolding Least-rw fun-eq-iff
    by (auto, metis a-LEAST le-less-trans not-le not-less-Least)
  thus ?thesis using True by auto
next
  case False
  have finI: finite I
    using i2 card.infinite by force
  have index-a'1:  $\text{index } (\text{sorted-list-of-set } (\text{insert } a \ I)) \ a = a'$ 
    using pick-index
    using a'-card a-def a-notin-I finI by auto
  hence index-a':  $\text{index } (\text{insort } a \ (\text{sorted-list-of-set } I)) \ a = a'$ 
    by (simp add: a-notin-I finI)
  have i1-length:  $i < \text{length } (\text{sorted-list-of-set } I)$  using i2
    by (metis distinct-card distinct-sorted-list-of-set finI set-sorted-list-of-set)
  have 1:  $\text{pick } (\text{insert } a \ I) \ (\text{Suc } i) = \text{sorted-list-of-set } (\text{insert } a \ I) \ ! \ (\text{Suc } i)$ 
  proof (rule sorted-list-of-set-eq-pick[symmetric])
    have finite (insert a I)
      using card.infinite i2 by force
    thus  $\text{Suc } i < \text{length } (\text{sorted-list-of-set } (\text{insert } a \ I))$ 
    by (metis Suc-mono a-notin-I card-insert-disjoint distinct-card distinct-sorted-list-of-set
        finI i2 set-sorted-list-of-set)
  qed
  also have 2:  $\dots = \text{insort } a \ (\text{sorted-list-of-set } I) \ ! \ \text{Suc } i$ 
    by (simp add: a-notin-I finI)
  also have  $\dots = \text{pick } I \ i$ 
  proof (cases  $i = a'$ )
    case True
    show ?thesis
      by (metis True a-notin-I finI i1-length index-a' insort-nth2 le-refl list.size(3)
          not-less0
              set-sorted-list-of-set sorted-list-of-set(2) sorted-list-of-set-eq-pick)
  next
  case False

```

```

have insert a (sorted-list-of-set I) ! Suc i = (sorted-list-of-set I) ! i
by (rule insert-nth2, insert i1-length False ia' index-a', auto simp add: a-notin-I
finI)
also have ... = pick I i
by (rule sorted-list-of-set-eq-pick[OF i1-length])
finally show ?thesis .
qed
finally show ?thesis .
qed

```

**lemma** pick-insert-index:

```

assumes Ik: card I = k
and a-notin-I: a  $\notin$  I
and ik: i < k
and a-def: pick (insert a I) a' = a
and a'k: a' < card I + 1
shows pick (insert a I) (insert-index a' i) = pick I i
proof (cases i < a')
case True
have pick (insert a I) i = pick I i
by (rule pick-insert[OF a-notin-I - a-def - a'k], auto simp add: Ik ik True)
thus ?thesis using True unfolding insert-index-def by auto
next
case False note i-ge-a' = False
have fin-aI: finite (insert a I)
using Ik finite-insert ik by fastforce
let ?P =  $\lambda aa. (aa = a \vee aa \in I) \wedge$  pick (insert a I) i < aa
let ?Q =  $\lambda aa. aa \in I \wedge$  pick (insert a I) i < aa
have ?P = ?Q using a-notin-I unfolding fun-eq-iff
by (auto, metis False Ik a-def card.infinite card-insert-disjoint ik less-SucI
linorder-neqE-nat not-less-zero order.asym pick-mono-le)
hence Least ?P = Least ?Q by simp
also have ... = pick I i
proof (rule Least-equality, rule conjI)
show pick I i  $\in$  I
by (simp add: Ik ik pick-in-set-le)
show pick (insert a I) i < pick I i
by (rule pick-insert2[OF a-notin-I - a-def - a'k], insert False, auto simp add:
Ik ik)
fix y assume y: y  $\in$  I  $\wedge$  pick (insert a I) i < y
let ?xs = sorted-list-of-set (insert a I)
have y  $\in$  set ?xs using y by (metis fin-aI insertI2 set-sorted-list-of-set y)
from this obtain j where xs-j-y: ?xs ! j = y and j: j < length ?xs
using in-set-conv-nth by metis
have ij: i < j
by (metis (no-types, lifting) Ik a-notin-I card.infinite card-insert-disjoint ik j
less-SucI
linorder-neqE-nat not-less-zero order.asym pick-mono-le sorted-list-of-set-eq-pick

```

```

xs-j-y y)
  have pick I i = pick (insert a I) (Suc i)
    by (rule pick-insert3[symmetric, OF a-notin-I - a-def - a'k], insert False Ik
    ik, auto)
  also have ... ≤ pick (insert a I) j
    by (metis Ik Suc-lessI card.infinite distinct-card distinct-sorted-list-of-set eq-iff
    finite-insert ij ik j less-imp-le-nat not-less-zero pick-mono-le set-sorted-list-of-set)
  also have ... = ?xs ! j by (rule sorted-list-of-set-eq-pick[symmetric, OF j])
  also have ... = y by (rule xs-j-y)
  finally show pick I i ≤ y .
qed
finally show ?thesis unfolding insert-index-def using False by auto
qed

```

## 5.2 Start of the proof

**definition** *strict-from-inj*  $n f = (\lambda i. \text{if } i \in \{0..<n\} \text{ then } (\text{sorted-list-of-set } (f' \{0..<n\})) ! i \text{ else } i)$

**lemma** *strict-strict-from-inj*:

```

fixes f::nat ⇒ nat
assumes inj-on f {0..<n} shows strict-mono-on {0..<n} (strict-from-inj n f)
proof -
  let ?I=f' {0..<n}
  have strict-from-inj n f x < strict-from-inj n f y
    if xy: x < y and x: x ∈ {0..<n} and y: y ∈ {0..<n} for x y
  proof -
    let ?xs = (sorted-list-of-set ?I)
    have sorted-xs: sorted ?xs by (rule sorted-sorted-list-of-set)
    have strict-from-inj n f x = (sorted-list-of-set ?I) ! x
      unfolding strict-from-inj-def using x by auto
    also have ... < (sorted-list-of-set ?I) ! y
    proof (rule sorted-nth-strict-mono; clarsimp?)
      show y < card (f ' {0..<n})
        by (metis assms atLeastLessThan-iff card-atLeastLessThan card-image
        diff-zero y)
    qed (simp add: xy)
    also have ... = strict-from-inj n f y using y unfolding strict-from-inj-def by
    simp
    finally show ?thesis .
  qed
  thus ?thesis unfolding strict-mono-on-def by simp
qed

```

**lemma** *strict-from-inj-image'*:  
 assumes  $f: \text{inj-on } f \{0..<n\}$

**shows** *strict-from-inj*  $n$   $f \text{ ‘ } \{0..<n\} = f \text{ ‘ } \{0..<n\}$   
**proof** (*auto*)  
**let**  $?I = f \text{ ‘ } \{0..<n\}$   
**fix**  $xa$  **assume**  $xa: xa < n$   
**have** *inj-on*: *inj-on*  $f \text{ ‘ } \{0..<n\}$  **using**  $f$  **by** *auto*  
**have** *length-I*: *length* (*sorted-list-of-set*  $?I$ ) =  $n$   
**by** (*metis card-atLeastLessThan card-image diff-zero distinct-card distinct-sorted-list-of-set finite-atLeastLessThan finite-imageI inj-on sorted-list-of-set(1)*)  
  
**have** *strict-from-inj*  $n$   $f$   $xa = \text{sorted-list-of-set } ?I ! xa$   
**using**  $xa$  **unfolding** *strict-from-inj-def* **by** *auto*  
**also have**  $\dots = \text{pick } ?I xa$   
**by** (*rule sorted-list-of-set-eq-pick, unfold length-I, auto simp add: xa*)  
**also have**  $\dots \in f \text{ ‘ } \{0..<n\}$  **by** (*rule pick-in-set-le, simp add: card-image inj-on xa*)  
**finally show** *strict-from-inj*  $n$   $f$   $xa \in f \text{ ‘ } \{0..<n\}$  .  
**obtain**  $i$  **where** *sorted-list-of-set* ( $f \text{ ‘ } \{0..<n\}$ ) !  $i = f xa$  **and**  $i < n$   
**by** (*metis atLeast0LessThan finite-atLeastLessThan finite-imageI imageI in-set-conv-nth length-I lessThan-iff sorted-list-of-set(1) xa*)  
**thus**  $f xa \in \text{strict-from-inj } n \text{ ‘ } \{0..<n\}$   
**by** (*metis atLeast0LessThan imageI lessThan-iff strict-from-inj-def*)  
**qed**

**definition**  $Z (n::nat) (m::nat) = \{(f,\pi) \mid f \pi. f \in \{0..<n\} \rightarrow \{0..<m\} \wedge (\forall i. i \notin \{0..<n\} \longrightarrow f i = i) \wedge \pi \text{ permutes } \{0..<n\}\}$

**lemma** *Z-alt-def*:  $Z n m = \{f. f \in \{0..<n\} \rightarrow \{0..<m\} \wedge (\forall i. i \notin \{0..<n\} \longrightarrow f i = i)\} \times \{\pi. \pi \text{ permutes } \{0..<n\}\}$   
**unfolding** *Z-def* **by** *auto*

**lemma** *det-mul-finsum-alt*:

**assumes**  $A: A \in \text{carrier-mat } n \ m$   
**and**  $B: B \in \text{carrier-mat } m \ n$   
**shows**  $\det (A*B) = \det (\text{mat}_r \ n \ n (\lambda i. \text{finsum-vec } \text{TYPE}('a::\text{comm-ring-1}) \ n (\lambda k. B \ \$\$ (k, i) \cdot_v \text{Matrix.col } A \ k) \ \{0..<m\}))$   
**proof** –  
**have**  $AT: A^T \in \text{carrier-mat } m \ n$  **using**  $A$  **by** *auto*  
**have**  $BT: B^T \in \text{carrier-mat } n \ m$  **using**  $B$  **by** *auto*  
**let**  $?f = (\lambda i. \text{finsum-vec } \text{TYPE}('a) \ n (\lambda k. B^T \ \$\$ (i, k) \cdot_v \text{Matrix.row } A^T \ k) \ \{0..<m\})$   
**let**  $?g = (\lambda i. \text{finsum-vec } \text{TYPE}('a) \ n (\lambda k. B \ \$\$ (k, i) \cdot_v \text{Matrix.col } A \ k) \ \{0..<m\})$   
**let**  $?lhs = \text{mat}_r \ n \ n \ ?f$   
**let**  $?rhs = \text{mat}_r \ n \ n \ ?g$   
**have** *lhs-rhs*:  $?lhs = ?rhs$   
**proof** (*rule eq-matI*)  
**show** *dim-row*  $?lhs = \text{dim-row } ?rhs$  **by** *auto*

```

show  $\dim\text{-col } ?lhs = \dim\text{-col } ?rhs$  by auto
fix  $i\ j$  assume  $i: i < \dim\text{-row } ?rhs$  and  $j: j < \dim\text{-col } ?rhs$ 
have  $j\text{-}n: j < n$  using  $j$  by auto
have  $?lhs \ \$\$ (i, j) = ?f\ i\ \$v\ j$  by (rule index-mat, insert i j, auto)
also have  $\dots = (\sum k \in \{0..<m\}. (B^T \ \$\$ (i, k) \cdot_v \text{row } A^T\ k) \ \$j)$ 
  by (rule index-finsum-vec[OF - j-n], auto simp add: A)
also have  $\dots = (\sum k \in \{0..<m\}. (B \ \$\$ (k, i) \cdot_v \text{col } A\ k) \ \$j)$ 
proof (rule sum.cong, auto)
  fix  $x$  assume  $x: x < m$ 
  have  $\text{row-rw}: \text{Matrix.row } A^T\ x = \text{col } A\ x$  by (rule row-transpose, insert A x,
auto)
  have  $B\text{-rw}: B^T \ \$\$ (i, x) = B \ \$\$ (x, i)$ 
    by (rule index-transpose-mat, insert x i B, auto)
  have  $(B^T \ \$\$ (i, x) \cdot_v \text{Matrix.row } A^T\ x) \ \$v\ j = B^T \ \$\$ (i, x) * \text{Matrix.row}$ 
 $A^T\ x \ \$v\ j$ 
    by (rule index-smult-vec, insert A j-n, auto)
  also have  $\dots = B \ \$\$ (x, i) * \text{col } A\ x \ \$v\ j$  unfolding  $\text{row-rw } B\text{-rw}$  by simp
  also have  $\dots = (B \ \$\$ (x, i) \cdot_v \text{col } A\ x) \ \$v\ j$ 
    by (rule index-smult-vec[symmetric], insert A j-n, auto)
  finally show  $(B^T \ \$\$ (i, x) \cdot_v \text{Matrix.row } A^T\ x) \ \$v\ j = (B \ \$\$ (x, i) \cdot_v \text{col } A$ 
 $x) \ \$v\ j$  .
  qed
  also have  $\dots = ?g\ i\ \$v\ j$ 
    by (rule index-finsum-vec[symmetric, OF - j-n], auto simp add: A)
  also have  $\dots = ?rhs \ \$\$ (i, j)$  by (rule index-mat[symmetric], insert i j, auto)
  finally show  $?lhs \ \$\$ (i, j) = ?rhs \ \$\$ (i, j)$  .
qed
have  $\det (A*B) = \det (B^T * A^T)$ 
  using det-transpose
  by (metis A B Matrix.transpose-mult mult-carrier-mat)
also have  $\dots = \det (\text{mat}_r\ n\ n\ (\lambda i. \text{finsum-vec } TYPE('a)\ n\ (\lambda k. B^T \ \$\$ (i, k) \cdot_v$ 
 $\text{Matrix.row } A^T\ k) \ \{0..<m\}))$ 
  using mat-mul-finsum-alt[OF BT AT] by auto
also have  $\dots = \det (\text{mat}_r\ n\ n\ (\lambda i. \text{finsum-vec } TYPE('a)\ n\ (\lambda k. B \ \$\$ (k, i) \cdot_v$ 
 $\text{Matrix.col } A\ k) \ \{0..<m\}))$ 
  by (rule arg-cong[of - - det], rule lhs-rhs)
  finally show  $?thesis$  .
qed

```

**lemma** *det-cols-mul*:

```

assumes  $A: A \in \text{carrier-mat } n\ m$ 
and  $B: B \in \text{carrier-mat } m\ n$ 
shows  $\det (A*B) = (\sum f \mid (\forall i \in \{0..<n\}. f\ i \in \{0..<m\}) \wedge (\forall i. i \notin \{0..<n\}$ 
 $\longrightarrow f\ i = i).$ 
   $(\prod i = 0..<n. B \ \$\$ (f\ i, i) * \text{Determinant.det } (\text{mat}_r\ n\ n\ (\lambda i. \text{col } A\ (f\ i))))$ 
proof -
  let  $?V = \{0..<n\}$ 
  let  $?U = \{0..<m\}$ 

```

**let**  $?F = \{f. (\forall i \in \{0..<n\}. f\ i \in ?U) \wedge (\forall i. i \notin \{0..<n\} \longrightarrow f\ i = i)\}$   
**let**  $?g = \lambda f. \det (\text{mat}_r\ n\ n\ (\lambda i. B\ \$\$ (f\ i, i) \cdot_v\ \text{col}\ A\ (f\ i)))$   
**have**  $fn: \text{finite } \{0..<m\}$  **by** *auto*  
**have**  $fn: \text{finite } \{0..<n\}$  **by** *auto*  
**have**  $\text{det-rw}: \det (\text{mat}_r\ n\ n\ (\lambda i. B\ \$\$ (f\ i, i) \cdot_v\ \text{col}\ A\ (f\ i))) =$   
 $(\text{prod } (\lambda i. B\ \$\$ (f\ i, i)) \{0..<n\}) * \det (\text{mat}_r\ n\ n\ (\lambda i. \text{col}\ A\ (f\ i)))$   
**if**  $f: (\forall i \in \{0..<n\}. f\ i \in \{0..<m\}) \wedge (\forall i. i \notin \{0..<n\} \longrightarrow f\ i = i)$  **for**  $f$   
**by** (*rule det-rows-mul, insert A col-dim, auto*)  
**have**  $\det (A*B) = \det (\text{mat}_r\ n\ n\ (\lambda i. \text{finsum-vec } \text{TYPE}('a::\text{comm-ring-1})\ n\ (\lambda k. B\ \$\$ (k, i) \cdot_v\ \text{Matrix.col}\ A\ k) ?U))$   
**by** (*rule det-mul-finsum-alt[OF A B]*)  
**also have**  $\dots = \text{sum } ?g\ ?F$  **by** (*rule det-linear-rows-sum[OF fm], auto simp add: A*)  
**also have**  $\dots = (\sum f \in ?F. \text{prod } (\lambda i. B\ \$\$ (f\ i, i)) \{0..<n\} * \det (\text{mat}_r\ n\ n\ (\lambda i. \text{col}\ A\ (f\ i))))$   
**using** *det-rw* **by** *auto*  
**finally show**  $?thesis$  .  
**qed**

**lemma** *det-cols-mul'*:

**assumes**  $A: A \in \text{carrier-mat } n\ m$   
**and**  $B: B \in \text{carrier-mat } m\ n$   
**shows**  $\det (A*B) = (\sum f \mid (\forall i \in \{0..<n\}. f\ i \in \{0..<m\}) \wedge (\forall i. i \notin \{0..<n\} \longrightarrow f\ i = i) \longrightarrow f\ i = i).$   
 $(\prod i = 0..<n. A\ \$\$ (i, f\ i)) * \det (\text{mat}_r\ n\ n\ (\lambda i. \text{row } B\ (f\ i)))$   
**proof** –  
**let**  $?F = \{f. (\forall i \in \{0..<n\}. f\ i \in \{0..<m\}) \wedge (\forall i. i \notin \{0..<n\} \longrightarrow f\ i = i)\}$   
**have**  $t: A * B = (B^T * A^T)^T$  **using** *transpose-mult[OF A B] transpose-transpose*  
**by** *metis*  
**have**  $\det (B^T * A^T) = (\sum f \in ?F. (\prod i = 0..<n. A^T\ \$\$ (f\ i, i)) * \det (\text{mat}_r\ n\ n\ (\lambda i. \text{col } B^T\ (f\ i))))$   
**by** (*rule det-cols-mul, auto simp add: A B*)  
**also have**  $\dots = (\sum f \in ?F. (\prod i = 0..<n. A\ \$\$ (i, f\ i)) * \det (\text{mat}_r\ n\ n\ (\lambda i. \text{row } B\ (f\ i))))$   
**proof** (*rule sum.cong, rule refl*)  
**fix**  $f$  **assume**  $f: f \in ?F$   
**have**  $(\prod i = 0..<n. A^T\ \$\$ (f\ i, i)) = (\prod i = 0..<n. A\ \$\$ (i, f\ i))$   
**proof** (*rule prod.cong, rule refl*)  
**fix**  $x$  **assume**  $x: x \in \{0..<n\}$   
**show**  $A^T\ \$\$ (f\ x, x) = A\ \$\$ (x, f\ x)$   
**by** (*rule index-transpose-mat(1), insert f A x, auto*)  
**qed**  
**moreover have**  $\det (\text{mat}_r\ n\ n\ (\lambda i. \text{col } B^T\ (f\ i))) = \det (\text{mat}_r\ n\ n\ (\lambda i. \text{row } B\ (f\ i)))$   
**proof** –  
**have** *row-eq-colT*:  $\text{row } B\ (f\ i)\ \$v\ j = \text{col } B^T\ (f\ i)\ \$v\ j$  **if**  $i < n$  **and**  $j: j < n$  **for**  $i\ j$   
**proof** –  
**have** *fi-m*:  $f\ i < m$  **using**  $f\ i$  **by** *auto*



**have**  $\text{col } B^T (f\ i) \$v\ j = B^T \$\$ (j, f\ i)$  **by** (*rule index-col, insert B fi-m j, auto*)  
**also have**  $\dots = B \$\$ (f\ i, j)$  **using** *B fi-m j by auto*  
**also have**  $\dots = \text{row } B (f\ i) \$v\ j$  **by** (*rule index-row[symmetric], insert B fi-m j, auto*)  
**finally show** *?thesis ..*  
**qed**  
**show** *?thesis by (rule arg-cong[of - - det], rule eq-matI, insert row-eq-colT, auto)*  
**qed**  
**ultimately show**  $(\prod i = 0..<n. A^T \$\$ (f\ i, i)) * \text{det} (\text{mat}_r\ n\ n (\lambda i. \text{col } B^T (f\ i))) =$   
 $(\prod i = 0..<n. A \$\$ (i, f\ i)) * \text{det} (\text{mat}_r\ n\ n (\lambda i. \text{row } B (f\ i)))$  **by** *simp*  
**qed**  
**finally show** *?thesis*  
**by** (*metis (no-types, lifting) A B det-transpose transpose-mult mult-carrier-mat*)  
**qed**

**lemma**

**assumes**  $F: F = \{f. f \in \{0..<n\} \rightarrow \{0..<m\} \wedge (\forall i. i \notin \{0..<n\} \longrightarrow f\ i = i)\}$   
**and**  $p: \pi \text{ permutes } \{0..<n\}$   
**shows**  $(\sum f \in F. (\prod i = 0..<n. B \$\$ (f\ i, \pi\ i))) = (\sum f \in F. (\prod i = 0..<n. B \$\$ (f\ i, i)))$   
**proof** –  
**let**  $?h = (\lambda f. f \circ \pi)$   
**have** *inj-on-F: inj-on ?h F*  
**proof** (*rule inj-onI*)  
**fix**  $f\ g$  **assume**  $f \circ \pi = g \circ \pi$   
**have**  $f\ x = g\ x$  **for**  $x$   
**proof** (*cases x \in \{0..<n\}*)  
**case** *True*  
**then show** *?thesis*  
**by** (*metis fop comp-apply p permutes-def*)  
**next**  
**case** *False*  
**then show** *?thesis*  
**by** (*metis fop comp-eq-elim p permutes-def*)  
**qed**  
**thus**  $f = g$  **by** *auto*  
**qed**  
**have**  $hF: ?h' F = F$   
**unfolding** *image-def*  
**proof** *auto*  
**fix**  $xa$  **assume**  $xa \in F$  **show**  $xa \circ \pi \in F$   
**unfolding** *o-def F*  
**using**  $F\ PiE\ p\ xa$   
**by** (*auto, smt F atLeastLessThan-iff mem-Collect-eq p permutes-def xa*)  
**show**  $\exists x \in F. xa = x \circ \pi$

**proof** (rule *bestI*[of -  $xa \circ \text{Hilbert-Choice.inv } \pi$ ])  
**show**  $xa = xa \circ \text{Hilbert-Choice.inv } \pi \circ \pi$   
**using**  $p$  **by** *auto*  
**show**  $xa \circ \text{Hilbert-Choice.inv } \pi \in F$   
**unfolding** *o-def F*  
**using**  $F \text{ PiE } p \text{ xa}$   
**by** (*auto, smt atLeastLessThan-iff permutes-def permutes-less(3)*)  
**qed**  
**qed**  
**have** *prod-rw*:  $(\prod i = 0..<n. B \text{ \$\$ } (f \ i, \ i)) = (\prod i = 0..<n. B \text{ \$\$ } (f \ (\pi \ i), \ \pi \ i))$   
**if**  $f \in F$  **for**  $f$   
**using** *prod.permute[OF p]* **by** *auto*  
**let**  $?g = \lambda f. (\prod i = 0..<n. B \text{ \$\$ } (f \ i, \ \pi \ i))$   
**have**  $(\sum f \in F. (\prod i = 0..<n. B \text{ \$\$ } (f \ i, \ i))) = (\sum f \in F. (\prod i = 0..<n. B \text{ \$\$ } (f \ (\pi \ i), \ \pi \ i)))$   
**using** *prod-rw* **by** *auto*  
**also** **have**  $\dots = (\sum f \in (?h'F). \prod i = 0..<n. B \text{ \$\$ } (f \ i, \ \pi \ i))$   
**using** *sum.reindex[OF inj-on-F, of ?g]* **unfolding** *hF* **by** *auto*  
**also** **have**  $\dots = (\sum f \in F. \prod i = 0..<n. B \text{ \$\$ } (f \ i, \ \pi \ i))$  **unfolding** *hF* **by** *auto*  
**finally** **show** *?thesis ..*  
**qed**

**lemma** *detAB-Znm-aux*:

**assumes**  $F: F = \{f. f \in \{0..<n\} \rightarrow \{0..<m\} \wedge (\forall i. i \notin \{0..<n\} \longrightarrow f \ i = i)\}$   
**shows**  $(\sum \pi \mid \pi \text{ permutes } \{0..<n\}. (\sum f \in F. \text{prod } (\lambda i. B \text{ \$\$ } (f \ i, \ i)) \{0..<n\} \\ * (\text{signof } \pi * (\prod i = 0..<n. A \text{ \$\$ } (\pi \ i, \ f \ i)))))) \\ = (\sum \pi \mid \pi \text{ permutes } \{0..<n\}. \sum f \in F. (\prod i = 0..<n. B \text{ \$\$ } (f \ i, \ \pi \ i)) \\ * (\text{signof } \pi * (\prod i = 0..<n. A \text{ \$\$ } (i, \ f \ i)))))$   
**proof** –  
**have**  $(\sum \pi \mid \pi \text{ permutes } \{0..<n\}. (\sum f \in F. \text{prod } (\lambda i. B \text{ \$\$ } (f \ i, \ i)) \{0..<n\} \\ * (\text{signof } \pi * (\prod i = 0..<n. A \text{ \$\$ } (\pi \ i, \ f \ i)))))) = \\ (\sum \pi \mid \pi \text{ permutes } \{0..<n\}. \sum f \in F. \text{signof } \pi * (\prod i = 0..<n. B \text{ \$\$ } (f \ i, \ i) * \\ A \text{ \$\$ } (\pi \ i, \ f \ i)))$   
**by** (*smt mult.left-commute prod.cong prod.distrib sum.cong*)  
**also** **have**  $\dots = (\sum \pi \mid \pi \text{ permutes } \{0..<n\}. \sum f \in F. \text{signof } (\text{Hilbert-Choice.inv } \pi) \\ * (\prod i = 0..<n. B \text{ \$\$ } (f \ i, \ i) * A \text{ \$\$ } (\text{Hilbert-Choice.inv } \pi \ i, \ f \ i)))$   
**by** (*rule sum-permutations-inverse*)  
**also** **have**  $\dots = (\sum \pi \mid \pi \text{ permutes } \{0..<n\}. \sum f \in F. \text{signof } (\text{Hilbert-Choice.inv } \pi) \\ * (\prod i = 0..<n. B \text{ \$\$ } (f \ (\pi \ i), \ (\pi \ i)) * A \text{ \$\$ } (\text{Hilbert-Choice.inv } \pi \ (\pi \ i), \ f \ (\pi \ i))))$   
**proof** (*rule sum.cong*)  
**fix**  $x$  **assume**  $x: x \in \{\pi. \pi \text{ permutes } \{0..<n\}\}$   
**let**  $?inv-x = \text{Hilbert-Choice.inv } x$   
**have**  $p: x \text{ permutes } \{0..<n\}$  **using**  $x$  **by** *simp*  
**have** *prod-rw*:  $(\prod i = 0..<n. B \text{ \$\$ } (f \ i, \ i) * A \text{ \$\$ } (?inv-x \ i, \ f \ i)) \\ = (\prod i = 0..<n. B \text{ \$\$ } (f \ (x \ i), \ x \ i) * A \text{ \$\$ } (?inv-x \ (x \ i), \ f \ (x \ i)))$  **if**  $f \in F$

```

for  $f$ 
  using  $\text{prod.permute}[OF\ p]$  by  $\text{auto}$ 
  then show  $(\sum f \in F. \text{signof } ?\text{inv-}x * (\prod i = 0..<n. B\ \$\$ (f\ i,\ i) * A\ \$\$ (?inv-x\ i,\ f\ i))) =$ 
     $(\sum f \in F. \text{signof } ?\text{inv-}x * (\prod i = 0..<n. B\ \$\$ (f\ (x\ i),\ x\ i) * A\ \$\$ (?inv-x\ (x\ i),\ f\ (x\ i))))$ 
    by  $\text{auto}$ 
  qed ( $\text{simp}$ )
  also have  $\dots = (\sum \pi \mid \pi\ \text{permutes}\ \{0..<n\}. \sum f \in F. \text{signof } \pi$ 
     $* (\prod i = 0..<n. B\ \$\$ (f\ (\pi\ i),\ (\pi\ i)) * A\ \$\$ (i,\ f\ (\pi\ i))))$ 
    by ( $\text{rule sum.cong, auto, rule sum.cong, auto}$ )
    ( $\text{metis (no-types, lifting) finite-atLeastLessThan signof-inv}$ )
  also have  $\dots = (\sum \pi \mid \pi\ \text{permutes}\ \{0..<n\}. \sum f \in F. \text{signof } \pi$ 
     $* (\prod i = 0..<n. B\ \$\$ (f\ i,\ (\pi\ i)) * A\ \$\$ (i,\ f\ i)))$ 
  proof ( $\text{rule sum.cong}$ )
    fix  $\pi$  assume  $p: \pi \in \{\pi. \pi\ \text{permutes}\ \{0..<n\}\}$ 
    hence  $p: \pi\ \text{permutes}\ \{0..<n\}$  by  $\text{auto}$ 
    let  $?inv\text{-}\pi = (\text{Hilbert-Choice.inv } \pi)$ 
    let  $?h = (\lambda f. f \circ (\text{Hilbert-Choice.inv } \pi))$ 
  have  $\text{inj-on-}F: \text{inj-on } ?h\ F$ 
  proof ( $\text{rule inj-onI}$ )
    fix  $f\ g$  assume  $f \circ ?inv\text{-}\pi = g \circ ?inv\text{-}\pi$ 
    have  $f\ x = g\ x$  for  $x$ 
    proof ( $\text{cases } x \in \{0..<n\}$ )
      case  $\text{True}$ 
      then show  $?thesis$ 
      by ( $\text{metis fop o-inv-o-cancel p permutes-inj}$ )
    next
      case  $\text{False}$ 
      then show  $?thesis$ 
      by ( $\text{metis fop o-inv-o-cancel p permutes-inj}$ )
    qed
  thus  $f = g$  by  $\text{auto}$ 
  qed
  have  $hF: ?h' F = F$ 
  unfolding  $\text{image-def}$ 
  proof  $\text{auto}$ 
    fix  $xa$  assume  $xa: xa \in F$  show  $xa \circ ?inv\text{-}\pi \in F$ 
    unfolding  $\text{o-def } F$ 
    using  $F\ \text{PiE } p\ xa$ 
    by ( $\text{auto, smt atLeastLessThan-iff permutes-def permutes-less(3)}$ )
  show  $\exists x \in F. xa = x \circ ?inv\text{-}\pi$ 
  proof ( $\text{rule bexI[of - xa } \circ \pi]$ )
    show  $xa = xa \circ \pi \circ \text{Hilbert-Choice.inv } \pi$ 
    using  $p$  by  $\text{auto}$ 
    show  $xa \circ \pi \in F$ 
    unfolding  $\text{o-def } F$ 
    using  $F\ \text{PiE } p\ xa$ 
    by ( $\text{auto, smt atLeastLessThan-iff permutes-def permutes-less(3)}$ )

```

```

qed
qed
let ?g = λf. signof π * (∏ i = 0..<n. B $$ (f (π i), π i) * A $$ (i, f (π i)))
  show (∑ f ∈ F. signof π * (∏ i = 0..<n. B $$ (f (π i), π i) * A $$ (i, f (π i)))) =
    (∑ f ∈ F. signof π * (∏ i = 0..<n. B $$ (f i, π i) * A $$ (i, f i)))
  using sum.reindex[OF inj-on-F, of ?g] p unfolding hF unfolding o-def by
  auto
  qed (simp)
  also have ... = (∑ π | π permutes {0..<n}. ∑ f ∈ F. (∏ i = 0..<n. B $$ (f i, π i)))
    * (signof π * (∏ i = 0..<n. A $$ (i, f i)))
    by (smt mult.left-commute prod.cong prod.distrib sum.cong)
  finally show ?thesis .
qed

```

**lemma** *detAB-Znm*:

```

assumes A: A ∈ carrier-mat n m
  and B: B ∈ carrier-mat m n
  shows det (A*B) = (∑ (f, π) ∈ Z n m. signof π * (∏ i = 0..<n. A $$ (i, f i) * B $$ (f i, π i)))
proof –
  let ?V = {0..<n}
  let ?U = {0..<m}
  let ?PU = {p. p permutes ?U}
  let ?F = {f. (∀ i ∈ {0..<n}. f i ∈ ?U) ∧ (∀ i. i ∉ {0..<n} → f i = i)}
  let ?f = λf. det (matr n n (λ i. A $$ (i, f i) ·v row B (f i)))
  let ?g = λf. det (matr n n (λ i. B $$ (f i, i) ·v col A (f i)))
  have fm: finite {0..<m} by auto
  have fn: finite {0..<n} by auto
  have F: ?F = {f. f ∈ {0..<n} → {0..<m} ∧ (∀ i. i ∉ {0..<n} → f i = i)} by
  auto
  have det-rw: det (matr n n (λ i. B $$ (f i, i) ·v col A (f i))) =
    (prod (λ i. B $$ (f i, i)) {0..<n}) * det (matr n n (λ i. col A (f i)))
  if f: (∀ i ∈ {0..<n}. f i ∈ {0..<m}) ∧ (∀ i. i ∉ {0..<n} → f i = i) for f
  by (rule det-rows-mul, insert A col-dim, auto)
  have det-rw2: det (matr n n (λ i. col A (f i)))
  = (∑ π | π permutes {0..<n}. signof π * (∏ i = 0..<n. A $$ (π i, f i)))
  if f: f ∈ ?F for f
  proof (unfold Determinant.det-def, auto, rule sum.cong, auto)
  fix x assume x: x permutes {0..<n}
  have (∏ i = 0..<n. col A (f i) $v x i) = (∏ i = 0..<n. A $$ (x i, f i))
  proof (rule prod.cong)
  fix xa assume xa: xa ∈ {0..<n} show col A (f xa) $v x xa = A $$ (x xa, f xa)
  by (metis A atLeastLessThan-iff carrier-matD(1) col-def index-vec permutes-less(1) x xa)
  qed (auto)

```

```

then show  $\text{signof } x * (\prod i = 0..<n. \text{col } A (f i) \$v x i)$ 
  =  $\text{signof } x * (\prod i = 0..<n. A \$\$ (x i, f i))$  by auto
qed
have  $\text{fin-n: finite } \{0..<n\}$  and  $\text{fin-m: finite } \{0..<m\}$  by auto
have  $\det (A*B) = \det (\text{mat}_r n n (\lambda i. \text{finsum-vec TYPE('a::comm-ring-1) } n$ 
   $(\lambda k. B \$\$ (k, i) \cdot_v \text{Matrix.col } A k) \{0..<m\}))$ 
  by (rule det-mul-finsum-alt[OF A B])
also have  $\dots = \text{sum } ?g ?F$  by (rule det-linear-rows-sum[OF fm], auto simp add:
A)
also have  $\dots = (\sum f \in ?F. \text{prod } (\lambda i. B \$\$ (f i, i) \{0..<n\}) * \det (\text{mat}_r n n (\lambda i.$ 
 $\text{col } A (f i)))$ 
  using det-rw by auto
also have  $\dots = (\sum f \in ?F. \text{prod } (\lambda i. B \$\$ (f i, i) \{0..<n\}) *$ 
 $(\sum \pi \mid \pi \text{ permutes } \{0..<n\}. \text{signof } \pi * (\prod i = 0..<n. A \$\$ (\pi i, f (i))))$ 
  by (rule sum.cong, auto simp add: det-rw2)
also have  $\dots =$ 
 $(\sum f \in ?F. \sum \pi \mid \pi \text{ permutes } \{0..<n\}. \text{prod } (\lambda i. B \$\$ (f i, i) \{0..<n\})$ 
 $* (\text{signof } \pi * (\prod i = 0..<n. A \$\$ (\pi i, f (i))))$ 
  by (simp add: mult-hom.hom-sum)
also have  $\dots = (\sum \pi \mid \pi \text{ permutes } \{0..<n\}. \sum f \in ?F. \text{prod } (\lambda i. B \$\$ (f i, i)$ 
 $\{0..<n\})$ 
 $* (\text{signof } \pi * (\prod i = 0..<n. A \$\$ (\pi i, f i)))$ 
  by (rule VS-Connect.class-semiring.finsum-finsum-swap,
   $\text{insert finite-permutations finite-bounded-functions[OF fin-m fin-n], auto)$ 
thm detAB-Znm-aux
also have  $\dots = (\sum \pi \mid \pi \text{ permutes } \{0..<n\}. \sum f \in ?F. (\prod i = 0..<n. B \$\$ (f i,$ 
 $\pi i))$ 
 $* (\text{signof } \pi * (\prod i = 0..<n. A \$\$ (i, f i)))$  by (rule detAB-Znm-aux, auto)
also have  $\dots = (\sum f \in ?F. \sum \pi \mid \pi \text{ permutes } \{0..<n\}. (\prod i = 0..<n. B \$\$ (f i, \pi$ 
 $i))$ 
 $* (\text{signof } \pi * (\prod i = 0..<n. A \$\$ (i, f i)))$ 
  by (rule VS-Connect.class-semiring.finsum-finsum-swap,
   $\text{insert finite-permutations finite-bounded-functions[OF fin-m fin-n], auto)$ 
also have  $\dots = (\sum f \in ?F. \sum \pi \mid \pi \text{ permutes } \{0..<n\}. \text{signof } \pi$ 
 $* (\prod i = 0..<n. A \$\$ (i, f i) * B \$\$ (f i, \pi i)))$ 
  unfolding prod.distrib by (rule sum.cong, auto, rule sum.cong, auto)
also have  $\dots = \text{sum } (\lambda(f,\pi). (\text{signof } \pi$ 
 $* (\text{prod } (\lambda i. A \$\$ (i, f i) * B \$\$ (f i, \pi i) \{0..<n\}))) (Z n m)$ 
  unfolding Z-alt-def unfolding sum.cartesian-product[symmetric] F by auto
finally show ?thesis .
qed

```

**context**

**fixes**  $n m$  **and**  $A B :: 'a::\text{comm-ring-1} \text{ mat}$

**assumes**  $A: A \in \text{carrier-mat } n m$

**and**  $B: B \in \text{carrier-mat } m n$

**begin**

**private definition**  $Z\text{-inj} = (\{f. f \in \{0..<n\} \rightarrow \{0..<m\} \wedge (\forall i. i \notin \{0..<n\} \longrightarrow f\ i = i)$

$\wedge \text{inj-on } f\ \{0..<n\}\} \times \{\pi. \pi \text{ permutes } \{0..<n\}\})$

**private definition**  $Z\text{-not-inj} = (\{f. f \in \{0..<n\} \rightarrow \{0..<m\} \wedge (\forall i. i \notin \{0..<n\} \longrightarrow f\ i = i)$

$\wedge \neg \text{inj-on } f\ \{0..<n\}\} \times \{\pi. \pi \text{ permutes } \{0..<n\}\})$

**private definition**  $Z\text{-strict} = (\{f. f \in \{0..<n\} \rightarrow \{0..<m\} \wedge (\forall i. i \notin \{0..<n\} \longrightarrow f\ i = i)$

$\wedge \text{strict-mono-on } \{0..<n\}\ f\} \times \{\pi. \pi \text{ permutes } \{0..<n\}\})$

**private definition**  $Z\text{-not-strict} = (\{f. f \in \{0..<n\} \rightarrow \{0..<m\} \wedge (\forall i. i \notin \{0..<n\} \longrightarrow f\ i = i)$

$\wedge \neg \text{strict-mono-on } \{0..<n\}\ f\} \times \{\pi. \pi \text{ permutes } \{0..<n\}\})$

**private definition**  $\text{weight } f\ \pi$

$= (\text{signof } \pi) * (\text{prod } (\lambda i. A\ \$\$ (i, f\ i) * B\ \$\$ (f\ i, \pi\ i))\ \{0..<n\})$

**private definition**  $Z\text{-good } g = (\{f. f \in \{0..<n\} \rightarrow \{0..<m\} \wedge (\forall i. i \notin \{0..<n\} \longrightarrow f\ i = i)$

$\wedge \text{inj-on } f\ \{0..<n\} \wedge (f'\{0..<n\} = g'\{0..<n\})\} \times \{\pi. \pi \text{ permutes } \{0..<n\}\})$

**private definition**  $F\text{-strict} = \{f. f \in \{0..<n\} \rightarrow \{0..<m\}$

$\wedge (\forall i. i \notin \{0..<n\} \longrightarrow f\ i = i) \wedge \text{strict-mono-on } \{0..<n\}\ f\}$

**private definition**  $F\text{-inj} = \{f. f \in \{0..<n\} \rightarrow \{0..<m\}$

$\wedge (\forall i. i \notin \{0..<n\} \longrightarrow f\ i = i) \wedge \text{inj-on } f\ \{0..<n\}\}$

**private definition**  $F\text{-not-inj} = \{f. f \in \{0..<n\} \rightarrow \{0..<m\}$

$\wedge (\forall i. i \notin \{0..<n\} \longrightarrow f\ i = i) \wedge \neg \text{inj-on } f\ \{0..<n\}\}$

**private definition**  $F = \{f. f \in \{0..<n\} \rightarrow \{0..<m\} \wedge (\forall i. i \notin \{0..<n\} \longrightarrow f\ i = i)\}$

The Cauchy–Binet formula is proven in <https://core.ac.uk/download/pdf/82475020.pdf> In that work, they define  $\sigma \equiv \text{inv } \varphi \circ \pi$ . I had problems following this proof in Isabelle, since I was demanded to show that such permutations commute, which is false. It is a notation problem of the  $\circ$  operator, the author means  $\sigma \equiv \pi \circ \text{inv } \varphi$  using the Isabelle notation (i.e.,  $\sigma\ x = \pi\ ((\text{inv } \varphi)\ x)$ ).

**lemma**  $\text{step-weight}$ :

**fixes**  $\varphi\ \pi$

**defines**  $\sigma \equiv \pi \circ \text{Hilbert-Choice.inv } \varphi$

**assumes**  $f\text{-inj}: f \in F\text{-inj}$  **and**  $gF: g \in F$  **and**  $pi: \pi \text{ permutes } \{0..<n\}$

**and**  $phi: \varphi \text{ permutes } \{0..<n\}$  **and**  $fg\text{-phi}: \forall x \in \{0..<n\}. f\ x = g\ (\varphi\ x)$

**shows**  $\text{weight } f\ \pi = (\text{signof } \varphi) * (\prod i = 0..<n. A\ \$\$ (i, g\ (\varphi\ i)))$

$* (\text{signof } \sigma) * (\prod i = 0..<n. B\ \$\$ (g\ i, \sigma\ i))$

**proof** –

```

let ?A = (∏ i = 0..<n. A $$ (i, g (φ i)))
let ?B = (∏ i = 0..<n. B $$ (g i, σ i))
have sigma: σ permutes {0..<n} unfolding σ-def
  by (rule permutes-compose[OF permutes-inv[OF phi] pi])
have A-rw: ?A = (∏ i = 0..<n. A $$ (i, f i)) using fg-phi by auto
have ?B = (∏ i = 0..<n. B $$ (g (φ i), σ (φ i)))
  by (rule prod.permute[unfolded o-def, OF phi])
also have ... = (∏ i = 0..<n. B $$ (f i, π i))
  using fg-phi
  unfolding σ-def unfolding o-def unfolding permutes-inverses(2)[OF phi] by
  auto
  finally have B-rw: ?B = (∏ i = 0..<n. B $$ (f i, π i)) .
  have (signof φ) * ?A * (signof σ) * ?B = (signof φ) * (signof σ) * ?A * ?B
  by auto
  also have ... = signof (φ ∘ σ) * ?A * ?B unfolding signof-compose[OF phi
  sigma] by simp
  also have ... = signof π * ?A * ?B
    by (metis (no-types, lifting) σ-def mult.commute o-inv-o-cancel permutes-inj
    phi sigma signof-compose)
  also have ... = signof π * (∏ i = 0..<n. A $$ (i, f i)) * (∏ i = 0..<n. B $$ (f
  i, π i))
    using A-rw B-rw by auto
  also have ... = signof π * (∏ i = 0..<n. A $$ (i, f i) * B $$ (f i, π i)) by auto
  also have ... = weight f π unfolding weight-def by simp
  finally show ?thesis ..
qed

```

lemma Z-good-fun-alt-sum:

```

fixes g
defines Z-good-fun ≡ {f. f ∈ {0..<n} → {0..<m} ∧ (∀ i. i ∉ {0..<n} → f i
= i)
  ∧ inj-on f {0..<n} ∧ (f' {0..<n} = g' {0..<n})}
assumes g: g ∈ F-inj
shows (∑ f ∈ Z-good-fun. P f) = (∑ π ∈ {π. π permutes {0..<n}}. P (g ∘ π))
proof -
let ?f = λ π. g ∘ π
let ?P = {π. π permutes {0..<n}}
have fP: ?f' ?P = Z-good-fun
proof (unfold Z-good-fun-def, auto)
fix xa xb assume xa permutes {0..<n} and xb < n
hence xa xb < n by auto
thus g (xa xb) < m using g unfolding F-inj-def by fastforce
next
fix xa i assume xa permutes {0..<n} and i-ge-n: ¬ i < n
hence xa i = i unfolding permutes-def by auto
thus g (xa i) = i using g i-ge-n unfolding F-inj-def by auto
next
fix xa assume xa permutes {0..<n} thus inj-on (g ∘ xa) {0..<n}

```

```

    by (metis (mono-tags, lifting) F-inj-def atLeast0LessThan comp-inj-on g
        mem-Collect-eq permutes-image permutes-inj-on)
  next
    fix  $\pi$   $xb$  assume  $\pi$  permutes  $\{0..<n\}$  and  $xb < n$  thus  $g\ xb \in (\lambda x. g (\pi\ x))$ 
    ‘  $\{0..<n\}$ 
      by (metis (full-types) atLeast0LessThan imageI image-image lessThan-iff
          permutes-image)
    next
      fix  $x$  assume  $x1: x \in \{0..<n\} \rightarrow \{0..<m\}$  and  $x2: \forall i. \neg i < n \rightarrow x\ i = i$ 
      and  $inj\text{-on}\text{-}x: inj\text{-on}\ x\ \{0..<n\}$  and  $xg: x\ \{0..<n\} = g\ \{0..<n\}$ 
      let  $?t = \lambda i. if\ i < n\ then\ (THE\ j. j < n \wedge x\ i = g\ j)$  else  $i$ 
      show  $x \in (\circ)\ g\ \{ \pi. \pi\ permutes\ \{0..<n\} \}$ 
      proof (unfold image-def, auto, rule exI[of - ?t], rule conjI)
        have  $?t\ i = i$  if  $i: i \notin \{0..<n\}$  for  $i$ 
          using  $i$  by auto
        moreover have  $\exists!j. ?t\ j = i$  for  $i$ 
        proof (cases  $i < n$ )
          case True
            obtain  $a$  where  $xa\text{-}gi: x\ a = g\ i$  and  $a: a < n$  using  $xg\ True$ 
            by (metis (mono-tags, opaque-lifting) atLeast0LessThan imageE imageI
                lessThan-iff)
            show ?thesis
            proof (rule ex1I[of - a])
              have  $the\text{-}ai: (THE\ j. j < n \wedge x\ a = g\ j) = i$ 
              proof (rule theI2)
                show  $i < n \wedge x\ a = g\ i$  using  $xa\text{-}gi\ True$  by auto
                fix  $xa$  assume  $xa < n \wedge x\ a = g\ xa$  thus  $xa = i$ 
                by (metis (mono-tags, lifting) F-inj-def True atLeast0LessThan
                    g inj-onD lessThan-iff mem-Collect-eq xa-gi)
                thus  $xa = i$  .
              qed
            thus  $ta: ?t\ a = i$  using  $a$  by auto
            fix  $j$  assume  $tj: ?t\ j = i$ 
            show  $j = a$ 
            proof (cases  $j < n$ )
              case True
                obtain  $b$  where  $xj\text{-}gb: x\ j = g\ b$  and  $b: b < n$  using  $xg\ True$ 
                by (metis (mono-tags, opaque-lifting) atLeast0LessThan imageE imageI
                    lessThan-iff)
                let  $?P = \lambda ja. ja < n \wedge x\ j = g\ ja$ 
                have  $the\text{-}ji: (THE\ ja. ja < n \wedge x\ j = g\ ja) = i$  using  $tj\ True$  by auto
                have  $?P\ (THE\ ja. ?P\ ja)$ 
                proof (rule theI)
                  show  $b < n \wedge x\ j = g\ b$  using  $xj\text{-}gb\ b$  by auto
                  fix  $xa$  assume  $xa < n \wedge x\ j = g\ xa$  thus  $xa = b$ 
                  by (metis (mono-tags, lifting) F-inj-def b atLeast0LessThan
                      g inj-onD lessThan-iff mem-Collect-eq xj-gb)
                qed
                hence  $x\ j = g\ i$  unfolding  $the\text{-}ji$  by auto

```



```

    hence  $x j = x a$  using  $xa-gi$  by auto
    then show ?thesis using inj-on-x a True unfolding inj-on-def by auto
next
  case False
  then show ?thesis using tj True by auto
qed
qed
next
case False note i-ge-n = False
show ?thesis
proof (rule ex1I[of - i])
  show  $? \tau i = i$  using False by simp
  fix j assume tj:  $? \tau j = i$ 
  show  $j = i$ 
  proof (cases  $j < n$ )
    case True
    obtain a where  $xj-ga: x j = g a$  and  $a: a < n$  using xg True
    by (metis (mono-tags, opaque-lifting) atLeast0LessThan imageE imageI
lessThan-iff)
    have (THE ja.  $ja < n \wedge x j = g ja$ )  $< n$ 
    proof (rule theI2)
      show  $a < n \wedge x j = g a$  using  $xj-ga$  a by auto
      fix xa assume a1:  $xa < n \wedge x j = g xa$  thus  $xa = a$ 
        using F-inj-def a atLeast0LessThan g inj-on-eq-iff  $xj-ga$  by fastforce
      show  $xa < n$  by (simp add: a1)
    qed
    then show ?thesis using tj i-ge-n by auto
  next
  case False
  then show ?thesis using tj by auto
qed
qed
qed
ultimately show  $? \tau$  permutes  $\{0..<n\}$  unfolding permutes-def by auto
show  $x = g \circ ? \tau$ 
proof -
  have  $x xa = g$  (THE j.  $j < n \wedge x xa = g j$ ) if xa:  $xa < n$  for xa
  proof -
    obtain c where  $c: c < n$  and  $xxa-gc: x xa = g c$ 
    by (metis (mono-tags, opaque-lifting) atLeast0LessThan imageE imageI
lessThan-iff xa xg)
    show ?thesis
    proof (rule theI2)
      show  $c1: c < n \wedge x xa = g c$  using c  $xxa-gc$  by auto
      fix xb assume c2:  $xb < n \wedge x xa = g xb$  thus  $xb = c$ 
        by (metis (mono-tags, lifting) F-inj-def c1 atLeast0LessThan
g inj-onD lessThan-iff mem-Collect-eq)
      show  $x xa = g xb$  using c1 c2 by simp
    qed
  qed

```

```

qed
moreover have  $x \ x a = g \ x a$  if  $x a: \neg x a < n$  for  $x a$ 
  using  $g \ x1 \ x2 \ x a$  unfolding  $F$ -inj-def by simp
  ultimately show ?thesis unfolding  $o$ -def  $fun$ -eq-iff by auto
qed
qed
qed
have inj: inj-on ?f ?P
proof (rule inj-onI)
  fix  $x \ y$  assume  $x: x \in ?P$  and  $y: y \in ?P$  and  $gx-gy: g \circ x = g \circ y$ 
  have  $x \ i = y \ i$  for  $i$ 
  proof (cases i < n)
    case True
      hence  $x_i: x \ i \in \{0..<n\}$  and  $y_i: y \ i \in \{0..<n\}$  using  $x \ y$  by auto
      have  $g \ (x \ i) = g \ (y \ i)$  using  $gx-gy$  unfolding  $o$ -def by meson
      thus ?thesis using  $x_i \ y_i$  using  $g$  unfolding  $F$ -inj-def inj-on-def by blast
    next
      case False
      then show ?thesis using  $x \ y$  unfolding permutes-def by auto
  qed
  thus  $x = y$  unfolding  $fun$ -eq-iff by auto
qed
have  $(\sum f \in Z\text{-good-fun. } P \ f) = (\sum f \in ?f' ?P. P \ f)$  using  $fP$  by simp
also have  $\dots = \text{sum } (P \circ (\circ) \ g) \ \{\pi. \ \pi \ \text{permutes } \{0..<n\}\}$ 
  by (rule sum.reindex[OF inj])
also have  $\dots = (\sum \pi \mid \pi \ \text{permutes } \{0..<n\}. P \ (g \circ \pi))$  by auto
finally show ?thesis .
qed

```

**lemma** *F-injI*:

```

assumes  $f \in \{0..<n\} \rightarrow \{0..<m\}$ 
and  $(\forall i. i \notin \{0..<n\} \longrightarrow f \ i = i)$  and inj-on  $f \ \{0..<n\}$ 
shows  $f \in F$ -inj using assms unfolding  $F$ -inj-def by simp

```

**lemma** *F-inj-composition-permutation*:

```

assumes  $\phi: \phi \ \text{permutes } \{0..<n\}$ 
and  $g: g \in F$ -inj
shows  $g \circ \phi \in F$ -inj
proof (rule F-injI)
  show  $g \circ \phi \in \{0..<n\} \rightarrow \{0..<m\}$ 
    using  $g$  unfolding permutes-def  $F$ -inj-def
    by (simp add: Pi-iff phi)
  show  $\forall i. i \notin \{0..<n\} \longrightarrow (g \circ \phi) \ i = i$ 
    using  $g \ \phi$  unfolding permutes-def  $F$ -inj-def by simp
  show inj-on  $(g \circ \phi) \ \{0..<n\}$ 
    by (rule comp-inj-on, insert g permutes-inj-on[OF phi] permutes-image[OF phi])
    (auto simp add: F-inj-def)
qed

```

**lemma** *F-strict-imp-F-inj*:  
**assumes**  $f: f \in F\text{-strict}$   
**shows**  $f \in F\text{-inj}$   
**using**  $f$  *strict-mono-on-imp-inj-on*  
**unfolding** *F-strict-def F-inj-def* **by** *auto*

**lemma** *one-step*:

**assumes**  $g1: g \in F\text{-strict}$   
**shows**  $\det(\text{submatrix } A \text{ UNIV } (g\{0..<n\})) * \det(\text{submatrix } B \text{ } (g\{0..<n\}) \text{ UNIV})$   
 $= (\sum (x, y) \in Z\text{-good } g. \text{weight } x \ y) \text{ (is ?lhs = ?rhs)}$   
**proof** –  
**define** *Z-good-fun* **where**  $Z\text{-good-fun} = \{f. f \in \{0..<n\} \rightarrow \{0..<m\} \wedge (\forall i. i \notin \{0..<n\} \rightarrow f\ i = i) \wedge \text{inj-on } f \ \{0..<n\} \wedge (f\{0..<n\} = g\{0..<n\})\}$   
**let**  $?Perm = \{\pi. \pi \text{ permutes } \{0..<n\}\}$   
**let**  $?P = (\lambda f. \sum \pi \in ?Perm. \text{weight } f \ \pi)$   
**let**  $?inv = \text{Hilbert-Choice.inv}$   
**have**  $g: g \in F\text{-inj}$  **by** (rule *F-strict-imp-F-inj[OF g1]*)  
**have**  $\det A: (\sum \varphi \in \{\pi. \pi \text{ permutes } \{0..<n\}\}. \text{signof } \varphi * (\prod i = 0..<n. A \ \$$ (i, g (\varphi \ i))))$   
 $= \det(\text{submatrix } A \text{ UNIV } (g\{0..<n\}))$   
**proof** –  
**have**  $\{j. j < \text{dim-col } A \wedge j \in g\{0..<n\}\} = \{j. j \in g\{0..<n\}\}$   
**using**  $g$  *A* **unfolding** *F-inj-def* **by** *fastforce*  
**also have**  $\text{card } \dots = n$  **using** *F-inj-def card-image g* **by** *force*  
**finally have**  $\text{card-J}: \text{card } \{j. j < \text{dim-col } A \wedge j \in g\{0..<n\}\} = n$  **by** *simp*  
**have**  $\text{subA-carrier}: \text{submatrix } A \text{ UNIV } (g\{0..<n\}) \in \text{carrier-mat } n \ n$   
**unfolding** *submatrix-def card-J* **using**  $A$  **by** *auto*  
**have**  $\det(\text{submatrix } A \text{ UNIV } (g\{0..<n\})) = (\sum p \mid p \text{ permutes } \{0..<n\}. \text{signof } p * (\prod i = 0..<n. \text{submatrix } A \text{ UNIV } (g\{0..<n\}) \ \$$ (i, p \ i)))$   
**using**  $\text{subA-carrier}$  **unfolding** *Determinant.det-def* **by** *auto*  
**also have**  $\dots = (\sum \varphi \in \{\pi. \pi \text{ permutes } \{0..<n\}\}. \text{signof } \varphi * (\prod i = 0..<n. A \ \$$ (i, g (\varphi \ i))))$   
 $\ \$$ (i, g (\varphi \ i)))$   
**proof** (rule *sum.cong*)  
**fix**  $x$  **assume**  $x: x \in \{\pi. \pi \text{ permutes } \{0..<n\}\}$   
**have**  $(\prod i = 0..<n. \text{submatrix } A \text{ UNIV } (g\{0..<n\}) \ \$$ (i, x \ i))$   
 $= (\prod i = 0..<n. A \ \$$ (i, g (x \ i)))$   
**proof** (rule *prod.cong, rule refl*)  
**fix**  $i$  **assume**  $i: i \in \{0..<n\}$   
**have**  $\text{pick-rw}: \text{pick } (g\{0..<n\}) (x \ i) = g (x \ i)$   
**proof** –  
**have**  $\text{index}(\text{sorted-list-of-set } (g\{0..<n\})) (g (x \ i)) = x \ i$   
**proof** –  
**have**  $\text{rw}: \text{sorted-list-of-set } (g\{0..<n\}) = \text{map } g \ [0..<n]$   
**by** (rule *sorted-list-of-set-map-strict, insert g1, simp add: F-strict-def*)

**have**  $index (sorted-list-of-set (g\{0..<n\})) (g (x i)) = index (map g [0..<n]) (g (x i))$   
**unfolding**  $rw$  **by**  $auto$   
**also have**  $... = index [0..<n] (x i)$   
**by**  $(rule\ index-map-inj-on[of\ -\ \{0..<n\}],\ insert\ x\ i\ g,\ auto\ simp\ add:$   
*F-inj-def)*  
**also have**  $... = x\ i$  **using**  $x\ i$  **by**  $auto$   
**finally show**  $?thesis$  .  
**qed**  
**moreover have**  $(g (x i)) \in (g\ '\{0..<n\})$  **using**  $x\ g\ i$  **unfolding**  $F-inj-def$   
**by**  $auto$   
**moreover have**  $x\ i < card (g\ '\{0..<n\})$  **using**  $x\ i\ g$  **by**  $(simp\ add:$   
*F-inj-def\ card-image)*  
**ultimately show**  $?thesis$  **using**  $pick-index$  **by**  $auto$   
**qed**  
**have**  $submatrix\ A\ UNIV (g\ '\{0..<n\})\ \$\$ (i,\ x\ i) = A\ \$\$ (pick\ UNIV\ i,\ pick$   
 $(g\ '\{0..<n\}) (x\ i))$   
**by**  $(rule\ submatrix-index,\ insert\ i\ A\ card-J\ x,\ auto)$   
**also have**  $... = A\ \$\$ (i,\ g (x\ i))$  **using**  $pick-rw\ pick-UNIV$  **by**  $auto$   
**finally show**  $submatrix\ A\ UNIV (g\ '\{0..<n\})\ \$\$ (i,\ x\ i) = A\ \$\$ (i,\ g (x$   
 $i))$  .  
**qed**  
**thus**  $signof\ x * (\prod i = 0..<n.\ submatrix\ A\ UNIV (g\ '\{0..<n\})\ \$\$ (i,\ x\ i))$   
 $= signof\ x * (\prod i = 0..<n.\ A\ \$\$ (i,\ g (x\ i)))$  **by**  $auto$   
**qed**  $(simp)$   
**finally show**  $?thesis$  **by**  $simp$   
**qed**  
**have**  $detB-rw: (\sum \pi \in ?Perm.\ signof (\pi \circ ?inv\ \varphi) * (\prod i = 0..<n.\ B\ \$\$ (g\ i,$   
 $(\pi \circ ?inv\ \varphi)\ i)))$   
 $= (\sum \pi \in ?Perm.\ signof (\pi) * (\prod i = 0..<n.\ B\ \$\$ (g\ i,\ \pi\ i)))$   
**if**  $\phi: \varphi$  **permutes**  $\{0..<n\}$  **for**  $\varphi$   
**proof** –  
**let**  $?h = \lambda \pi.\ \pi \circ ?inv\ \varphi$   
**let**  $?g = \lambda \pi.\ signof (\pi) * (\prod i = 0..<n.\ B\ \$\$ (g\ i,\ \pi\ i))$   
**have**  $?h\ ?Perm = ?Perm$   
**proof** –  
**have**  $\pi \circ ?inv\ \varphi$  **permutes**  $\{0..<n\}$  **if**  $\pi$  **permutes**  $\{0..<n\}$  **for**  $\pi$   
**using**  $permutes-compose\ permutes-inv\ \phi$  **that** **by**  $blast$   
**moreover have**  $x \in (\lambda \pi.\ \pi \circ ?inv\ \varphi)\ '\ ?Perm$  **if**  $x$  **permutes**  $\{0..<n\}$  **for**  $x$   
**proof** –  
**have**  $x \circ \varphi$  **permutes**  $\{0..<n\}$   
**using**  $permutes-compose\ \phi$  **that** **by**  $blast$   
**moreover have**  $x = x \circ \varphi \circ ?inv\ \varphi$  **using**  $\phi$  **by**  $auto$   
**ultimately show**  $?thesis$  **unfolding**  $image-def$  **by**  $auto$   
**qed**  
**ultimately show**  $?thesis$  **by**  $auto$   
**qed**  
**hence**  $(\sum \pi \in ?Perm.\ ?g\ \pi) = (\sum \pi \in ?h\ ?Perm.\ ?g\ \pi)$  **by**  $simp$   
**also have**  $... = sum (?g \circ ?h)\ ?Perm$

```

proof (rule sum.reindex)
  show inj-on ( $\lambda\pi. \pi \circ ?inv \varphi$ ) { $\pi. \pi$  permutes { $0..<n$ }}
    by (metis (no-types, lifting) inj-onI o-inv-o-cancel permutes-inj phi)
qed
also have ... = ( $\sum \pi \in ?Perm. \text{signof } (\pi \circ ?inv \varphi) * (\prod i = 0..<n. B \text{ \textasciitilde} \$\$ (g$ 
i, ( $\pi \circ ?inv \varphi$ ) i))))
  unfolding o-def by auto
  finally show ?thesis by simp
qed

have detB:  $\det (\text{submatrix } B (g \text{ \textasciitilde} \{0..<n\}) UNIV)$ 
  = ( $\sum \pi \in ?Perm. \text{signof } \pi * (\prod i = 0..<n. B \text{ \textasciitilde} \$\$ (g \text{ \textasciitilde} i, \pi \text{ \textasciitilde} i))$ )
proof -
  have { $i. i < \text{dim-row } B \wedge i \in g \text{ \textasciitilde} \{0..<n\}$ } = { $i. i \in g \text{ \textasciitilde} \{0..<n\}$ }
    using g B unfolding F-inj-def by fastforce
  also have card ... = n using F-inj-def card-image g by force
  finally have card-I:  $\text{card } \{j. j < \text{dim-row } B \wedge j \in g \text{ \textasciitilde} \{0..<n\}\} = n$  by simp
  have subB-carrier:  $\text{submatrix } B (g \text{ \textasciitilde} \{0..<n\}) UNIV \in \text{carrier-mat } n \text{ \textasciitilde} n$ 
    unfolding submatrix-def using card-I B by auto
  have  $\det (\text{submatrix } B (g \text{ \textasciitilde} \{0..<n\}) UNIV) = (\sum p \in ?Perm. \text{signof } p$ 
    * ( $\prod i=0..<n. \text{submatrix } B (g \text{ \textasciitilde} \{0..<n\}) UNIV \text{ \textasciitilde} \$\$ (i, p \text{ \textasciitilde} i))$ )
    unfolding Determinant.det-def using subB-carrier by auto
  also have ... = ( $\sum \pi \in ?Perm. \text{signof } \pi * (\prod i = 0..<n. B \text{ \textasciitilde} \$\$ (g \text{ \textasciitilde} i, \pi \text{ \textasciitilde} i))$ )
proof (rule sum.cong, rule refl)
  fix x assume x:  $x \in \{\pi. \pi \text{ permutes } \{0..<n\}\}$ 
  have ( $\prod i=0..<n. \text{submatrix } B (g \text{ \textasciitilde} \{0..<n\}) UNIV \text{ \textasciitilde} \$\$ (i, x \text{ \textasciitilde} i) = (\prod i=0..<n. B \text{ \textasciitilde} \$\$ (g \text{ \textasciitilde} i, x \text{ \textasciitilde} i))$ )
proof (rule prod.cong, rule refl)
  fix i assume i:  $i \in \{0..<n\}$ 
  have pick-rw:  $\text{pick } (g \text{ \textasciitilde} \{0..<n\}) i = g \text{ \textasciitilde} i$ 
proof -
  have index (sorted-list-of-set ( $g \text{ \textasciitilde} \{0..<n\}$ )) ( $g \text{ \textasciitilde} i$ ) = i
proof -
  have rw:  $\text{sorted-list-of-set } (g \text{ \textasciitilde} \{0..<n\}) = \text{map } g [0..<n]$ 
    by (rule sorted-list-of-set-map-strict, insert g1, simp add: F-strict-def)
  have index (sorted-list-of-set ( $g \text{ \textasciitilde} \{0..<n\}$ )) ( $g \text{ \textasciitilde} i$ ) = index (map g [0..<n])
(g i)
    unfolding rw by auto
  also have ... = index [0..<n] (i)
    by (rule index-map-inj-on[of - {0..<n}], insert x i g, auto simp add:
F-inj-def)
  also have ... = i using i by auto
  finally show ?thesis .
qed
moreover have ( $g \text{ \textasciitilde} i$ )  $\in (g \text{ \textasciitilde} \{0..<n\})$  using x g i unfolding F-inj-def
by auto
  moreover have  $i < \text{card } (g \text{ \textasciitilde} \{0..<n\})$  using x i g by (simp add: F-inj-def
card-image)
  ultimately show ?thesis using pick-index by auto

```

**qed**  
**have** *submatrix*  $B (g \{0..<n\}) UNIV \ \$\$ (i, x i) = B \ \$\$ (pick (g \{0..<n\})$   
*i, pick UNIV (x i))*  
**by** (*rule submatrix-index, insert i B card-I x, auto*)  
**also have**  $\dots = B \ \$\$ (g i, x i)$  **using** *pick-rw pick-UNIV by auto*  
**finally show** *submatrix*  $B (g \{0..<n\}) UNIV \ \$\$ (i, x i) = B \ \$\$ (g i, x i)$

**qed**  
**thus**  $signof\ x * (\prod i = 0..<n. submatrix\ B (g \{0..<n\}) UNIV \ \$\$ (i, x i))$   
 $= signof\ x * (\prod i = 0..<n. B \ \$\$ (g i, x i))$  **by** *simp*

**qed**  
**finally show** *?thesis .*

**qed**

**have**  $?rhs = (\sum f \in Z\text{-good-fun.} \sum \pi \in ?Perm. weight\ f\ \pi)$   
**unfolding** *Z-good-def sum.cartesian-product Z-good-fun-def by blast*  
**also have**  $\dots = (\sum \varphi \in \{\pi. \pi\ permutes\ \{0..<n\}\}. ?P (g \circ \varphi))$  **unfolding** *Z-good-fun-def*  
**by** (*rule Z-good-fun-alt-sum[OF g]*)

**also have**  $\dots = (\sum \varphi \in \{\pi. \pi\ permutes\ \{0..<n\}\}. \sum \pi \in \{\pi. \pi\ permutes\ \{0..<n\}\}.$   
 $signof\ \varphi * (\prod i = 0..<n. A \ \$\$ (i, g (\varphi i))) * signof (\pi \circ ?inv\ \varphi)$   
 $* (\prod i = 0..<n. B \ \$\$ (g i, (\pi \circ ?inv\ \varphi) i)))$

**proof** (*rule sum.cong, simp, rule sum.cong, simp*)

**fix**  $\varphi\ \pi$  **assume**  $phi: \varphi \in ?Perm$  **and**  $pi: \pi \in ?Perm$

**show**  $weight (g \circ \varphi)\ \pi = signof\ \varphi * (\prod i = 0..<n. A \ \$\$ (i, g (\varphi i))) *$   
 $signof (\pi \circ ?inv\ \varphi) * (\prod i = 0..<n. B \ \$\$ (g i, (\pi \circ ?inv\ \varphi) i))$

**proof** (*rule step-weight*)

**show**  $g \circ \varphi \in F\text{-inj}$  **by** (*rule F-inj-composition-permutation[OF - g], insert*  
*phi, auto*)

**show**  $g \in F$  **using** *g unfolding F-def F-inj-def by simp*

**qed** (*insert phi pi, auto*)

**qed**

**also have**  $\dots = (\sum \varphi \in \{\pi. \pi\ permutes\ \{0..<n\}\}. signof\ \varphi * (\prod i = 0..<n. A \ \$\$$   
 $(i, g (\varphi i))) *$   
 $(\sum \pi \mid \pi\ permutes\ \{0..<n\}. signof (\pi \circ ?inv\ \varphi) * (\prod i = 0..<n. B \ \$\$ (g i,$   
 $(\pi \circ ?inv\ \varphi) i))))$

**by** (*metis (mono-tags, lifting) Groups.mult-ac(1) semiring-0-class.sum-distrib-left*  
*sum.cong*)

**also have**  $\dots = (\sum \varphi \in ?Perm. signof\ \varphi * (\prod i = 0..<n. A \ \$\$ (i, g (\varphi i))) *$   
 $(\sum \pi \in ?Perm. signof\ \pi * (\prod i = 0..<n. B \ \$\$ (g i, \pi i))))$  **using** *detB-rw by*  
*auto*

**also have**  $\dots = (\sum \varphi \in ?Perm. signof\ \varphi * (\prod i = 0..<n. A \ \$\$ (i, g (\varphi i)))) *$   
 $(\sum \pi \in ?Perm. signof\ \pi * (\prod i = 0..<n. B \ \$\$ (g i, \pi i)))$   
**by** (*simp add: semiring-0-class.sum-distrib-right*)

**also have**  $\dots = ?lhs$  **unfolding** *detA detB ..*  
**finally show** *?thesis ..*

**qed**

**lemma** *gather-by-strictness:*

$sum (\lambda g. sum (\lambda (f, \pi). weight f \pi) (Z\text{-good } g)) F\text{-strict}$   
 $= sum (\lambda g. det (submatrix A UNIV (g' \{0..<n\})) * det (submatrix B (g' \{0..<n\}) UNIV)) F\text{-strict}$   
**proof** (rule *sum.cong*)  
**fix**  $f$  **assume**  $f: f \in F\text{-strict}$   
**show**  $(\sum_{(x, y) \in Z\text{-good } f} weight x y)$   
 $= det (submatrix A UNIV (f' \{0..<n\})) * det (submatrix B (f' \{0..<n\}) UNIV)$   
**by** (rule *one-step[symmetric]*, rule  $f$ )  
**qed** (*simp*)

**lemma** *finite-Z-strict[simp]: finite Z-strict*  
**proof** (*unfold Z-strict-def*, rule *finite-cartesian-product*)  
**have**  $finN: finite \{0..<n\}$  **and**  $finM: finite \{0..<m\}$  **by** *auto*  
**let**  $?A = \{f \in \{0..<n\} \rightarrow \{0..<m\}. (\forall i. i \notin \{0..<n\} \longrightarrow f i = i) \wedge strict\text{-mono-on } \{0..<n\} f\}$   
**let**  $?B = \{f \in \{0..<n\} \rightarrow \{0..<m\}. (\forall i. i \notin \{0..<n\} \longrightarrow f i = i)\}$   
**have**  $B: \{f. (\forall i \in \{0..<n\}. f i \in \{0..<m\}) \wedge (\forall i. i \notin \{0..<n\} \longrightarrow f i = i)\} = ?B$  **by** *auto*  
**have**  $?A \subseteq ?B$  **by** *auto*  
**moreover** **have**  $finite ?B$  **using**  $B$  *finite-bounded-functions[OF finM finN]* **by** *auto*  
**ultimately** **show**  $finite ?A$  **using** *rev-finite-subset* **by** *blast*  
**show**  $finite \{\pi. \pi \text{ permutes } \{0..<n\}\}$  **using** *finite-permutations* **by** *blast*  
**qed**

**lemma** *finite-Z-not-strict[simp]: finite Z-not-strict*  
**proof** (*unfold Z-not-strict-def*, rule *finite-cartesian-product*)  
**have**  $finN: finite \{0..<n\}$  **and**  $finM: finite \{0..<m\}$  **by** *auto*  
**let**  $?A = \{f \in \{0..<n\} \rightarrow \{0..<m\}. (\forall i. i \notin \{0..<n\} \longrightarrow f i = i) \wedge \neg strict\text{-mono-on } \{0..<n\} f\}$   
**let**  $?B = \{f \in \{0..<n\} \rightarrow \{0..<m\}. (\forall i. i \notin \{0..<n\} \longrightarrow f i = i)\}$   
**have**  $B: \{f. (\forall i \in \{0..<n\}. f i \in \{0..<m\}) \wedge (\forall i. i \notin \{0..<n\} \longrightarrow f i = i)\} = ?B$  **by** *auto*  
**have**  $?A \subseteq ?B$  **by** *auto*  
**moreover** **have**  $finite ?B$  **using**  $B$  *finite-bounded-functions[OF finM finN]* **by** *auto*  
**ultimately** **show**  $finite ?A$  **using** *rev-finite-subset* **by** *blast*  
**show**  $finite \{\pi. \pi \text{ permutes } \{0..<n\}\}$  **using** *finite-permutations* **by** *blast*  
**qed**

**lemma** *finite-Znm[simp]: finite (Z n m)*  
**proof** (*unfold Z-alt-def*, rule *finite-cartesian-product*)  
**have**  $finN: finite \{0..<n\}$  **and**  $finM: finite \{0..<m\}$  **by** *auto*  
**let**  $?A = \{f \in \{0..<n\} \rightarrow \{0..<m\}. (\forall i. i \notin \{0..<n\} \longrightarrow f i = i)\}$   
**let**  $?B = \{f \in \{0..<n\} \rightarrow \{0..<m\}. (\forall i. i \notin \{0..<n\} \longrightarrow f i = i)\}$   
**have**  $B: \{f. (\forall i \in \{0..<n\}. f i \in \{0..<m\}) \wedge (\forall i. i \notin \{0..<n\} \longrightarrow f i = i)\} = ?B$  **by** *auto*  
**have**  $?A \subseteq ?B$  **by** *auto*

**moreover have** *finite* ?B **using** B *finite-bounded-functions*[OF *finM finN*] **by**  
*auto*  
**ultimately show** *finite* ?A **using** *rev-finite-subset* **by** *blast*  
**show** *finite* { $\pi$ .  $\pi$  *permutes* {0..*n*}} **using** *finite-permutations* **by** *blast*  
**qed**

**lemma** *finite-F-inj[simp]*: *finite F-inj*

**proof** –

**have** *finN*: *finite* {0..*n*} **and** *finM*: *finite* {0..*m*} **by** *auto*  
**let** ?A={ $f \in \{0..\langle n \rangle\} \rightarrow \{0..\langle m \rangle\}$ . ( $\forall i. i \notin \{0..\langle n \rangle\} \longrightarrow f\ i = i$ )  $\wedge$  *inj-on*  $f$   
{0..*n*}}  
**let** ?B={ $f \in \{0..\langle n \rangle\} \rightarrow \{0..\langle m \rangle\}$ . ( $\forall i. i \notin \{0..\langle n \rangle\} \longrightarrow f\ i = i$ )}  
**have** B: { $f$ . ( $\forall i \in \{0..\langle n \rangle\}$ .  $f\ i \in \{0..\langle m \rangle\}$ )  $\wedge$  ( $\forall i. i \notin \{0..\langle n \rangle\} \longrightarrow f\ i = i$ )} =  
?B **by** *auto*  
**have** ?A $\subseteq$ ?B **by** *auto*  
**moreover have** *finite* ?B **using** B *finite-bounded-functions*[OF *finM finN*] **by**  
*auto*  
**ultimately show** *finite F-inj* **unfolding** *F-inj-def* **using** *rev-finite-subset* **by**  
*blast*  
**qed**

**lemma** *finite-F-strict[simp]*: *finite F-strict*

**proof** –

**have** *finN*: *finite* {0..*n*} **and** *finM*: *finite* {0..*m*} **by** *auto*  
**let** ?A={ $f \in \{0..\langle n \rangle\} \rightarrow \{0..\langle m \rangle\}$ . ( $\forall i. i \notin \{0..\langle n \rangle\} \longrightarrow f\ i = i$ )  $\wedge$  *strict-mono-on*  
{0..*n*}  $f$ }  
**let** ?B={ $f \in \{0..\langle n \rangle\} \rightarrow \{0..\langle m \rangle\}$ . ( $\forall i. i \notin \{0..\langle n \rangle\} \longrightarrow f\ i = i$ )}  
**have** B: { $f$ . ( $\forall i \in \{0..\langle n \rangle\}$ .  $f\ i \in \{0..\langle m \rangle\}$ )  $\wedge$  ( $\forall i. i \notin \{0..\langle n \rangle\} \longrightarrow f\ i = i$ )} =  
?B **by** *auto*  
**have** ?A $\subseteq$ ?B **by** *auto*  
**moreover have** *finite* ?B **using** B *finite-bounded-functions*[OF *finM finN*] **by**  
*auto*  
**ultimately show** *finite F-strict* **unfolding** *F-strict-def* **using** *rev-finite-subset*  
**by** *blast*  
**qed**

**lemma** *nth-strict-mono*:

**fixes**  $f::nat \Rightarrow nat$   
**assumes** *strictf*: *strict-mono*  $f$  **and**  $i: i < n$   
**shows**  $f\ i = (\text{sorted-list-of-set } (f\ \{0..\langle n \rangle\}))\ !\ i$   
**proof** –

**let** ?I =  $f\ \{0..\langle n \rangle\}$   
**have** *length* (*sorted-list-of-set* ( $f\ \{0..\langle n \rangle\}$ )) = *card* ?I  
**by** (*metis* *distinct-card* *finite-atLeastLessThan* *finite-imageI*  
*sorted-list-of-set*(1) *sorted-list-of-set*(3))  
**also have** ... =  $n$   
**by** (*simp* *add*: *card-image* *strict-mono-imp-inj-on* *strictf*)  
**finally have** *length-I*: *length* (*sorted-list-of-set* ?I) =  $n$  .  
**have** *card-eq*: *card* { $a \in ?I$ .  $a < f\ i$ } =  $i$



```

    using i
  proof (induct i)
    case 0
    then show ?case
      by (auto simp add: strict-mono-less strictf)
  next
    case (Suc i)
    have i: i < n using Suc.prem1 by auto
    let ?J' = {a ∈ f ` {0..<n}. a < f i}
    let ?J = {a ∈ f ` {0..<n}. a < f (Suc i)}
    have cardJ': card ?J' = i by (rule Suc.hyps[OF i])
    have J: ?J = insert (f i) ?J'
    proof (auto)
      fix xa assume 1: f xa ≠ f i and 2: f xa < f (Suc i)
      show f xa < f i
        using 1 2 not-less-less-Suc-eq strict-mono-less strictf by fastforce
    next
      fix xa assume f xa < f i thus f xa < f (Suc i)
        using less-SucI strict-mono-less strictf by blast
    next
      show f i ∈ f ` {0..<n} using i by auto
      show f i < f (Suc i) using strictf strict-mono-less by auto
    qed
    have card ?J = Suc (card ?J') by (unfold J, rule card-insert-disjoint, auto)
    then show ?case using cardJ' by auto
  qed

  have sorted-list-of-set ?I ! i = pick ?I i
    by (rule sorted-list-of-set-eq-pick, simp add: ⟨card (f ` {0..<n}) = n⟩ i)
  also have ... = pick ?I (card {a ∈ ?I. a < f i}) unfolding card-eq by simp
  also have ... = f i by (rule pick-card-in-set, simp add: i)
  finally show ?thesis ..
qed

lemma nth-strict-mono-on:
  fixes f::nat ⇒ nat
  assumes strictf: strict-mono-on {0..<n} f and i: i < n
  shows f i = (sorted-list-of-set (f ` {0..<n})) ! i
  proof -
    let ?I = f ` {0..<n}
    have length (sorted-list-of-set (f ` {0..<n})) = card ?I
      by (metis distinct-card finite-atLeastLessThan finite-imageI
        sorted-list-of-set(1) sorted-list-of-set(3))
    also have ... = n
      by (metis (mono-tags, lifting) card-atLeastLessThan card-image diff-zero
        inj-on-def strict-mono-on-eqD strictf)
    finally have length-I: length (sorted-list-of-set ?I) = n .
    have card-eq: card {a ∈ ?I. a < f i} = i
      using i
    proof (induct i)

```

```

case 0
then show ?case
  by (auto, metis (no-types, lifting) atLeast0LessThan lessThan-iff less-Suc-eq
    not-less0 not-less-eq strict-mono-on-def strictf)
next
  case (Suc i)
  have i: i < n using Suc.premis by auto
  let ?J' = {a ∈ f ' {0..<n}. a < f i}
  let ?J = {a ∈ f ' {0..<n}. a < f (Suc i)}
  have cardJ': card ?J' = i by (rule Suc.hyps[OF i])
  have J: ?J = insert (f i) ?J'
  proof (auto)
    fix xa assume 1: f xa ≠ f i and 2: f xa < f (Suc i) and 3: xa < n
    show f xa < f i
    by (metis (full-types) 1 2 3 antisym-conv3 atLeast0LessThan i lessThan-iff
      less-SucE order.asym strict-mono-onD strictf)
  next
  fix xa assume f xa < f i and xa < n thus f xa < f (Suc i)
  using less-SucI strictf
  by (metis (no-types, lifting) Suc.premis atLeast0LessThan
    lessI lessThan-iff less-trans strict-mono-onD)
  next
  show f i ∈ f ' {0..<n} using i by auto
  show f i < f (Suc i)
  using Suc.premis strict-mono-onD strictf by fastforce
  qed
  have card ?J = Suc (card ?J') by (unfold J, rule card-insert-disjoint, auto)
  then show ?case using cardJ' by auto
  qed
  have sorted-list-of-set ?I ! i = pick ?I i
  by (rule sorted-list-of-set-eq-pick, simp add: ⟨card (f ' {0..<n}) = n⟩ i)
  also have ... = pick ?I (card {a ∈ ?I. a < f i}) unfolding card-eq by simp
  also have ... = f i by (rule pick-card-in-set, simp add: i)
  finally show ?thesis ..
qed

lemma strict-fun-eq:
  assumes f: f ∈ F-strict and g: g ∈ F-strict and fg: f'{0..<n} = g'{0..<n}
  shows f = g
  proof (unfold fun-eq-iff, auto)
    fix x
    show f x = g x
    proof (cases x<n)
      case True
      have strictf: strict-mono-on {0..<n} f and strictg: strict-mono-on {0..<n} g
      using f g unfolding F-strict-def by auto
      have f x = (sorted-list-of-set (f'{0..<n})) ! x by (rule nth-strict-mono-on[OF
        strictf True])
      also have ... = (sorted-list-of-set (g'{0..<n})) ! x unfolding fg by simp

```

```

    also have ... = g x by (rule nth-strict-mono-on[symmetric, OF strictg True])
    finally show ?thesis .
  next
    case False
    then show ?thesis using f g unfolding F-strict-def by auto
  qed
qed

```

**lemma** *strict-from-inj-preserves-F*:

```

  assumes f: f ∈ F-inj
  shows strict-from-inj n f ∈ F
  proof -
    {
      fix x assume x: x < n
      have inj-on: inj-on f {0..<n} using f unfolding F-inj-def by auto
      have {a. a < m ∧ a ∈ f ' {0..<n}} = f'{0..<n} using f unfolding F-inj-def
    by auto
      hence card-eq: card {a. a < m ∧ a ∈ f ' {0..<n}} = n
      by (simp add: card-image inj-on)
      let ?I = f'{0..<n}
      have length (sorted-list-of-set (f ' {0..<n})) = card ?I
      by (metis distinct-card finite-atLeastLessThan finite-imageI
        sorted-list-of-set(1) sorted-list-of-set(3))
      also have ... = n
      by (simp add: card-image strict-mono-imp-inj-on inj-on)
      finally have length-I: length (sorted-list-of-set ?I) = n .
      have sorted-list-of-set (f ' {0..<n}) ! x = pick (f ' {0..<n}) x
      by (rule sorted-list-of-set-eq-pick, unfold length-I, auto simp add: x)
      also have ... < m by (rule pick-le, unfold card-eq, rule x)
      finally have sorted-list-of-set (f ' {0..<n}) ! x < m .
    }
    thus ?thesis unfolding strict-from-inj-def F-def by auto
  qed

```

**lemma** *strict-from-inj-F-strict*: *strict-from-inj n xa ∈ F-strict*

```

  if xa: xa ∈ F-inj for xa
  proof -
    have strict-mono-on {0..<n} (strict-from-inj n xa)
    by (rule strict-strict-from-inj, insert xa, simp add: F-inj-def)
    thus ?thesis using strict-from-inj-preserves-F[OF xa] unfolding F-def F-strict-def
  by auto
  qed

```

**lemma** *strict-from-inj-image*:

```

  assumes f: f ∈ F-inj
  shows strict-from-inj n f ' {0..<n} = f'{0..<n}
  proof (auto)
    let ?I = f ' {0..<n}

```

```

fix xa assume xa: xa < n
have inj-on: inj-on f {0..n} using f unfolding F-inj-def by auto
  have {a. a < m ∧ a ∈ f ‘ {0..n}} = f ‘ {0..n} using f unfolding F-inj-def
by auto
  hence card-eq: card {a. a < m ∧ a ∈ f ‘ {0..n}} = n
    by (simp add: card-image inj-on)
  let ?I = f ‘ {0..n}
  have length (sorted-list-of-set (f ‘ {0..n})) = card ?I
    by (metis distinct-card finite-atLeastLessThan finite-imageI
      sorted-list-of-set(1) sorted-list-of-set(3))
  also have ... = n
    by (simp add: card-image strict-mono-imp-inj-on inj-on)
  finally have length-I: length (sorted-list-of-set ?I) = n .
have strict-from-inj n f xa = sorted-list-of-set ?I ! xa
  using xa unfolding strict-from-inj-def by auto
also have ... = pick ?I xa
  by (rule sorted-list-of-set-eq-pick, unfold length-I, auto simp add: xa)
also have ... ∈ f ‘ {0..n} by (rule pick-in-set-le, simp add: <card (f ‘ {0..n})
= n> xa)
finally show strict-from-inj n f xa ∈ f ‘ {0..n} .
obtain i where sorted-list-of-set (f ‘ {0..n}) ! i = f xa and i < n
  by (metis atLeast0LessThan finite-atLeastLessThan finite-imageI imageI
in-set-conv-nth length-I lessThan-iff sorted-list-of-set(1) xa)
thus f xa ∈ strict-from-inj n f ‘ {0..n}
  by (metis atLeast0LessThan imageI lessThan-iff strict-from-inj-def)
qed

```

**lemma** *Z-good-alt*:

```

assumes g: g ∈ F-strict
shows Z-good g = {x ∈ F-inj. strict-from-inj n x = g} × {π. π permutes {0..n}}
proof –
  define Z-good-fun where Z-good-fun = {f. f ∈ {0..n} → {0..m} ∧ (∀ i. i ∉
{0..n} → f i = i)
  ∧ inj-on f {0..n} ∧ (f ‘ {0..n} = g ‘ {0..n})}
  have Z-good-fun = {x ∈ F-inj. strict-from-inj n x = g}
  proof (auto)
    fix f assume f: f ∈ Z-good-fun thus f-inj: f ∈ F-inj unfolding F-inj-def
Z-good-fun-def by auto
    show strict-from-inj n f = g
    proof (rule strict-fun-eq[OF - g])
      show strict-from-inj n f ‘ {0..n} = g ‘ {0..n}
        using f-inj f strict-from-inj-image
        unfolding Z-good-fun-def F-inj-def by auto
      show strict-from-inj n f ∈ F-strict
        using F-strict-def f-inj strict-from-inj-F-strict by blast
    qed
  next
    fix f assume f-inj: f ∈ F-inj and g-strict-f: g = strict-from-inj n f

```

**have**  $f \text{ xa} \in g \text{ ' } \{0..<n\}$  **if**  $\text{xa} < n$  **for**  $\text{xa}$   
**using**  $f\text{-inj } g\text{-strict-}f \text{ strict-from-inj-image that by auto}$   
**moreover have**  $g \text{ xa} \in f \text{ ' } \{0..<n\}$  **if**  $\text{xa} < n$  **for**  $\text{xa}$   
**by** ( $\text{metis } f\text{-inj } g\text{-strict-}f \text{ imageI lessThan-atLeast0 lessThan-iff strict-from-inj-image}$   
*that*)  
**ultimately show**  $f \in Z\text{-good-fun}$   
**using**  $f\text{-inj } g\text{-strict-}f \text{ unfolding } Z\text{-good-fun-def } F\text{-inj-def}$   
**by auto**  
**qed**  
**thus**  $?thesis \text{ unfolding } Z\text{-good-fun-def } Z\text{-good-def by simp}$   
**qed**

**lemma**  $\text{weight-0: } (\sum (f, \pi) \in Z\text{-not-inj. weight } f \ \pi) = 0$

**proof** –

**let**  $?F = \{f. (\forall i \in \{0..<n\}. f \ i \in \{0..<m\}) \wedge (\forall i. i \notin \{0..<n\} \longrightarrow f \ i = i)\}$

**let**  $?Perm = \{\pi. \pi \text{ permutes } \{0..<n\}\}$

**have**  $(\sum (f, \pi) \in Z\text{-not-inj. weight } f \ \pi)$

$= (\sum f \in F\text{-not-inj. } (\prod i = 0..<n. A \ \$\$ (i, f \ i)) * \det (\text{mat}_r \ n \ n \ (\lambda i. \text{row } B \ (f \ i))))$

**proof** –

**have**  $\text{dim-row-rw: } \text{dim-row } (\text{mat}_r \ n \ n \ (\lambda i. \text{col } A \ (f \ i))) = n$  **for**  $f$  **by auto**

**have**  $\text{dim-row-rw2: } \text{dim-row } (\text{mat}_r \ n \ n \ (\lambda i. \text{Matrix.row } B \ (f \ i))) = n$  **for**  $f$  **by auto**

**have**  $\text{prod-rw: } (\prod i = 0..<n. B \ \$\$ (f \ i, \pi \ i)) = (\prod i = 0..<n. \text{row } B \ (f \ i) \ \$v \ \pi \ i)$

**if**  $f: f \in F\text{-not-inj}$  **and**  $\text{pi: } \pi \in ?Perm$  **for**  $f \ \pi$

**proof** ( $\text{rule prod.cong, rule refl}$ )

**fix**  $x$  **assume**  $x: x \in \{0..<n\}$

**have**  $f \ x < \text{dim-row } B$  **using**  $f \ B \ x \text{ unfolding } F\text{-not-inj-def by fastforce}$

**moreover have**  $\pi \ x < \text{dim-col } B$  **using**  $x \ \text{pi } B \text{ by auto}$

**ultimately show**  $B \ \$\$ (f \ x, \pi \ x) = \text{Matrix.row } B \ (f \ x) \ \$v \ \pi \ x$  **by** ( $\text{rule index-row[symmetric]}$ )

**qed**

**have**  $\text{sum-rw: } (\sum \pi \mid \pi \text{ permutes } \{0..<n\}. \text{signof } \pi * (\prod i = 0..<n. B \ \$\$ (f \ i, \pi \ i)))$

$= \det (\text{mat}_r \ n \ n \ (\lambda i. \text{row } B \ (f \ i)))$  **if**  $f: f \in F\text{-not-inj}$  **for**  $f$

**unfolding**  $\text{Determinant.det-def}$  **using**  $\text{dim-row-rw2 prod-rw } f \text{ by auto}$

**have**  $(\sum (f, \pi) \in Z\text{-not-inj. weight } f \ \pi) = (\sum f \in F\text{-not-inj. } \sum \pi \in ?Perm. \text{weight } f \ \pi)$

**unfolding**  $Z\text{-not-inj-def}$  **unfolding**  $\text{sum.cartesian-product}$

**unfolding**  $F\text{-not-inj-def by simp}$

**also have**  $\dots = (\sum f \in F\text{-not-inj. } \sum \pi \mid \pi \text{ permutes } \{0..<n\}. \text{signof } \pi$

$* (\prod i = 0..<n. A \ \$\$ (i, f \ i) * B \ \$\$ (f \ i, \pi \ i)))$

**unfolding**  $\text{weight-def by simp}$

**also have**  $\dots = (\sum f \in F\text{-not-inj. } (\prod i = 0..<n. A \ \$\$ (i, f \ i))$

$* (\sum \pi \mid \pi \text{ permutes } \{0..<n\}. \text{signof } \pi * (\prod i = 0..<n. B \ \$\$ (f \ i, \pi \ i))))$

**by** ( $\text{rule sum.cong, rule refl, auto}$ )

( $\text{metis (no-types, lifting) mult.left-commute mult-hom.hom-sum sum.cong}$ )

**also have** ... =  $(\sum f \in F\text{-not-inj. } (\prod i = 0..<n. A \text{ } (i, f i)))$   
 $* \det (\text{mat}_r \ n \ n \ (\lambda i. \text{row } B \ (f i)))$  **using** *sum-rw* **by** *auto*  
**finally show** *?thesis* **by** *auto*  
**qed**  
**also have** ... = 0  
**by** (*rule sum.neutral, insert det-not-inj-on[of - n B], auto simp add: F-not-inj-def*)  
**finally show** *?thesis* .  
**qed**

### 5.3 Final theorem

**lemma** *Cauchy-Binet1:*

**shows**  $\det (A*B) =$   
 $\text{sum } (\lambda f. \det (\text{submatrix } A \ \text{UNIV } (f\{0..<n\})) * \det (\text{submatrix } B \ (f\{0..<n\}))$   
 $\text{UNIV})) \ F\text{-strict}$   
**(is** *?lhs = ?rhs*)

**proof** –

**have** *sum0*:  $(\sum (f, \pi) \in Z\text{-not-inj. } \text{weight } f \ \pi) = 0$  **by** (*rule weight-0*)  
**let** *?f = strict-from-inj n*  
**have** *sum-rw*:  $\text{sum } g \ F\text{-inj} = (\sum y \in F\text{-strict. } \text{sum } g \ \{x \in F\text{-inj. } ?f \ x = y\})$  **for**  
*g*  
**by** (*rule sum.group[symmetric], insert strict-from-inj-F-strict, auto*)  
**have** *Z-Union*:  $Z\text{-inj} \cup Z\text{-not-inj} = Z \ n \ m$   
**unfolding** *Z-def Z-not-inj-def Z-inj-def* **by** *auto*  
**have** *Z-Inter*:  $Z\text{-inj} \cap Z\text{-not-inj} = \{\}$   
**unfolding** *Z-def Z-not-inj-def Z-inj-def* **by** *auto*  
**have**  $\det (A*B) = (\sum (f, \pi) \in Z \ n \ m. \ \text{weight } f \ \pi)$   
**using** *detAB-Znm[OF A B]* **unfolding** *weight-def* **by** *auto*  
**also have** ... =  $(\sum (f, \pi) \in Z\text{-inj. } \text{weight } f \ \pi) + (\sum (f, \pi) \in Z\text{-not-inj. } \text{weight } f \ \pi)$   
**by** (*metis Z-Inter Z-Union finite-Un finite-Znm sum.union-disjoint*)  
**also have** ... =  $(\sum (f, \pi) \in Z\text{-inj. } \text{weight } f \ \pi)$  **using** *sum0* **by** *force*  
**also have** ... =  $(\sum f \in F\text{-inj. } \sum \pi \in \{\pi. \ \pi \text{ permutes } \{0..<n\}\}. \ \text{weight } f \ \pi)$   
**unfolding** *Z-inj-def* **unfolding** *F-inj-def sum.cartesian-product ..*  
**also have** ... =  $(\sum y \in F\text{-strict. } \sum f \in \{x \in F\text{-inj. } \text{strict-from-inj } n \ x = y\}. \ \text{sum } (\text{weight } f) \ \{\pi. \ \pi \text{ permutes } \{0..<n\}\})$   
**unfolding** *sum-rw ..*  
**also have** ... =  $(\sum y \in F\text{-strict. } \sum (f, \pi) \in (\{x \in F\text{-inj. } \text{strict-from-inj } n \ x = y\} \times \{\pi. \ \pi \text{ permutes } \{0..<n\}\}). \ \text{weight } f \ \pi)$   
**unfolding** *F-inj-def sum.cartesian-product ..*  
**also have** ... =  $\text{sum } (\lambda g. \ \text{sum } (\lambda (f, \pi). \ \text{weight } f \ \pi) \ (Z\text{-good } g)) \ F\text{-strict}$   
**using** *Z-good-alt* **by** *auto*  
**also have** ... = *?rhs* **unfolding** *gather-by-strictness* **by** *simp*  
**finally show** *?thesis* .  
**qed**

**lemma** *Cauchy-Binet:*

$\det (A*B) = (\sum I \in \{I. \ I \subseteq \{0..<m\} \wedge \text{card } I = n\}. \ \det (\text{submatrix } A \ \text{UNIV } I) * \det (\text{submatrix } B \ I \ \text{UNIV}))$

**proof** –

```

let ?f=(λI. (λi. if i<n then sorted-list-of-set I ! i else i))
let ?setI = {I. I ⊆ {0..<m} ∧ card I = n}
have inj-on: inj-on ?f ?setI
proof (rule inj-onI)
  fix I J assume I: I ∈ ?setI and J: J ∈ ?setI and fI-fJ: ?f I = ?f J
  have x ∈ J if x: x ∈ I for x
    by (metis (mono-tags) fI-fJ I J distinct-card in-set-conv-nth mem-Collect-eq
        sorted-list-of-set(1) sorted-list-of-set(3) subset-eq-atLeast0-lessThan-finite
x)
  moreover have x ∈ I if x: x ∈ J for x
    by (metis (mono-tags) fI-fJ I J distinct-card in-set-conv-nth mem-Collect-eq
        sorted-list-of-set(1) sorted-list-of-set(3) subset-eq-atLeast0-lessThan-finite
x)
  ultimately show I = J by auto
qed
have rw: ?f I ‘ {0..<n} = I if I: I ∈ ?setI for I
proof -
  have sorted-list-of-set I ! xa ∈ I if xa < n for xa
    by (metis (mono-tags, lifting) I distinct-card distinct-sorted-list-of-set mem-Collect-eq
        nth-mem set-sorted-list-of-set subset-eq-atLeast0-lessThan-finite that)
  moreover have ∃ xa∈{0..<n}. x = sorted-list-of-set I ! xa if x: x∈I for x
    by (metis (full-types) x I atLeast0LessThan distinct-card in-set-conv-nth
mem-Collect-eq
        lessThan-iff sorted-list-of-set(1) sorted-list-of-set(3) subset-eq-atLeast0-lessThan-finite)
  ultimately show ?thesis unfolding image-def by auto
qed
have f-setI: ?f ‘ ?setI = F-strict
proof -
  have sorted-list-of-set I ! xa < m if I: I ⊆ {0..<m} and n = card I and xa
< card I
    for I xa
    by (metis I ⟨xa < card I⟩ atLeast0LessThan distinct-card finite-atLeastLessThan
lessThan-iff
        pick-in-set-le rev-finite-subset sorted-list-of-set(1)
        sorted-list-of-set(3) sorted-list-of-set-eq-pick subsetCE)
  moreover have strict-mono-on {0..<card I} (λi. if i < card I then sorted-list-of-set
I ! i else i)
    if I ⊆ {0..<m} and n = card I for I
    by (smt ⟨I ⊆ {0..<m}⟩ atLeastLessThan-iff distinct-card finite-atLeastLessThan
pick-mono-le
        rev-finite-subset sorted-list-of-set(1) sorted-list-of-set(3)
        sorted-list-of-set-eq-pick strict-mono-on-def)
  moreover have x ∈ ?f ‘ {I. I ⊆ {0..<m} ∧ card I = n}
    if x1: x ∈ {0..<n} → {0..<m} and x2: ∀ i. ¬ i < n → x i = i
    and s: strict-mono-on {0..<n} x for x
  proof -
    have inj-x: inj-on x {0..<n}
      using s strict-mono-on-imp-inj-on by blast
    hence card-xn: card (x ‘ {0..<n}) = n by (simp add: card-image)

```

```

have x-eq: x = ( $\lambda i$ . if  $i < n$  then sorted-list-of-set (x ‘ {0.. $n$ }) ! i else i)
  unfolding fun-eq-iff
  using nth-strict-mono-on s using x2 by auto
show ?thesis
  unfolding image-def by (auto, rule exI[of -x‘{0.. $n$ }], insert card-xn x1
x-eq, auto)
qed
ultimately show ?thesis unfolding F-strict-def by auto
qed
let ?g = ( $\lambda f$ . det (submatrix A UNIV (f‘{0.. $n$ })) * det(submatrix B (f‘{0.. $n$ })
UNIV))
have det (A*B) = sum (( $\lambda f$ . det (submatrix A UNIV (f ‘ {0.. $n$ }))
* det (submatrix B (f ‘ {0.. $n$ }) UNIV))  $\circ$  ?f) {I. I  $\subseteq$  {0.. $m$ }  $\wedge$  card I =
n}
  unfolding Cauchy-Binet1 f-setI[symmetric] by (rule sum.reindex[OF inj-on])
  also have ... = ( $\sum I \in \{I. I \subseteq \{0..< m\} \wedge \text{card } I = n\}. \text{det}(\text{submatrix } A \text{ UNIV } I) * \text{det}(\text{submatrix } B \text{ I UNIV})$ )
  by (rule sum.cong, insert rw, auto)
finally show ?thesis .
qed
end

end

```

## 6 Definition of Smith normal form in JNF

**theory** *Smith-Normal-Form-JNF*

**imports**

*SNF-Missing-Lemmas*

**begin**

Now, we define diagonal matrices and Smith normal form in JNF

**definition** *isDiagonal-mat* A = ( $\forall i j. i \neq j \wedge i < \text{dim-row } A \wedge j < \text{dim-col } A \longrightarrow A \text{ \text{\$} } (i, j) = 0$ )

**definition** *Smith-normal-form-mat* A =

(  
 $(\forall a. a + 1 < \min (\text{dim-row } A) (\text{dim-col } A) \longrightarrow A \text{ \text{\$} } (a, a) \text{ dvd } A \text{ \text{\$} } (a+1, a+1))$   
 $\wedge \text{isDiagonal-mat } A$   
 )

**lemma** *SNF-first-divides*:

**assumes** *SNF-A*: *Smith-normal-form-mat* A **and** (A::('a::comm-ring-1) mat)  $\in$  *carrier-mat* n m

**and** i:  $i < \min (\text{dim-row } A) (\text{dim-col } A)$

**shows** A  $\text{\text{\$} } (0, 0)$  dvd A  $\text{\text{\$} } (i, i)$

**using** i

**proof** (*induct* i)

**case** 0



```

then show ?case by auto
next
case (Suc i)
show ?case
  by (metis (full-types) Smith-normal-form-mat-def Suc.hyps Suc.prem1
      Suc-eq-plus1 Suc-lessD SNF-A dvd-trans)
qed

lemma Smith-normal-form-mat-intro:
  assumes ( $\forall a. a + 1 < \min (\dim\text{-row } A) (\dim\text{-col } A) \longrightarrow A \text{ $$$ } (a,a) \text{ dvd } A \text{ $$$ } (a+1,a+1)$ )
  and isDiagonal-mat A
  shows Smith-normal-form-mat A
  unfolding Smith-normal-form-mat-def using assms by auto

lemma Smith-normal-form-mat-m0[simp]:
  assumes A: A ∈ carrier-mat m 0
  shows Smith-normal-form-mat A
  using A unfolding Smith-normal-form-mat-def isDiagonal-mat-def by auto

lemma Smith-normal-form-mat-0m[simp]:
  assumes A: A ∈ carrier-mat 0 m
  shows Smith-normal-form-mat A
  using A unfolding Smith-normal-form-mat-def isDiagonal-mat-def by auto

lemma S00-dvd-all-A:
  assumes A: (A::'a::comm-ring-1 mat) ∈ carrier-mat m n
  and P: P ∈ carrier-mat m m
  and Q: Q ∈ carrier-mat n n
  and inv-P: invertible-mat P
  and inv-Q: invertible-mat Q
  and S-PAQ: S = P*A*Q
  and SNF-S: Smith-normal-form-mat S
  and i: i < m and j: j < n
  shows S$$$ (0,0) dvd A $$$ (i,j)
  proof -
  have S00: ( $\forall i j. i < m \wedge j < n \longrightarrow S$$$ (0,0) \text{ dvd } S$$$ (i,j)$ )
  using SNF-S unfolding Smith-normal-form-mat-def isDiagonal-mat-def
  by (smt (verit) P Q SNF-first-divides A S-PAQ SNF-S carrier-matD
      dvd-0-right min-less-iff-conj mult-carrier-mat)
  obtain P' where PP': inverts-mat P P' and P'P: inverts-mat P' P
  using inv-P unfolding invertible-mat-def by auto
  obtain Q' where QQ': inverts-mat Q Q' and Q'Q: inverts-mat Q' Q
  using inv-Q unfolding invertible-mat-def by auto
  have A-P'SQ': P'*S*Q' = A
  proof -
  have P'*S*Q' = P'*(P*A*Q)*Q' unfolding S-PAQ by auto
  also have ... = (P'*P)*A*(Q*Q')
  by (smt (verit, ccfv-threshold) A P'P PP' Q'Q assms(2) assms(3) as-

```

*soc-mult-mat*  
*carrier-matD(2) carrier-matI index-mult-mat(2) index-mult-mat(3)*  
*inverts-mat-def one-carrier-mat)*  
**also have** ... = A  
**by** (*metis A P'P QQ' A Q P carrier-matD(1) index-mult-mat(3) index-one-mat(3) inverts-mat-def*  
*left-mult-one-mat right-mult-one-mat)*  
**finally show** ?thesis .  
**qed**  
**have** ( $\forall i j. i < m \wedge j < n \longrightarrow S \text{ $$$ } (0,0) \text{ dvd } (P' * S * Q') \text{ $$$ } (i,j)$ )  
**proof** (*rule dvd-elements-mult-matrix-left-right[OF - - - S00]*)  
**show**  $S \in \text{carrier-mat } m \ n$  **using** P A Q S-PAQ **by** auto  
**show**  $P' \in \text{carrier-mat } m \ m$   
**by** (*metis (mono-tags, lifting) A-P'SQ' PP' P A carrier-matD carrier-matI index-mult-mat(2)*  
*index-mult-mat(3) inverts-mat-def one-carrier-mat)*  
**show**  $Q' \in \text{carrier-mat } n \ n$   
**by** (*metis (mono-tags, lifting) A-P'SQ' Q'Q Q A carrier-matD(2) carrier-matI*  
*index-mult-mat(3) inverts-mat-def one-carrier-mat)*  
**qed**  
**thus** ?thesis **using** A-P'SQ' i j **by** auto  
**qed**

**lemma** *SNF-first-divides-all*:  
**assumes** *SNF-A: Smith-normal-form-mat A* **and**  $A: (A::('a::\text{comm-ring-1}) \text{ mat}) \in \text{carrier-mat } m \ n$   
**and**  $i: i < m$  **and**  $j: j < n$   
**shows**  $A \text{ $$$ } (0,0) \text{ dvd } A \text{ $$$ } (i,j)$   
**proof** (*cases i=j*)  
**case** True  
**then show** ?thesis **using** *assms SNF-first-divides* **by** (*metis carrier-matD min-less-iff-conj*)  
**next**  
**case** False  
**hence**  $A \text{ $$$ } (i,j) = 0$  **using** *SNF-A i j A unfolding Smith-normal-form-mat-def isDiagonal-mat-def* **by** auto  
**then show** ?thesis **by** auto  
**qed**

**lemma** *SNF-divides-diagonal*:  
**fixes**  $A::('a::\text{comm-ring-1}) \text{ mat}$   
**assumes**  $A: A \in \text{carrier-mat } n \ m$   
**and** *SNF-A: Smith-normal-form-mat A*  
**and**  $j: j < \min n \ m$   
**and**  $ij: i \leq j$   
**shows**  $A \text{ $$$ } (i,i) \text{ dvd } A \text{ $$$ } (j,j)$   
**using**  $ij \ j$

```

proof (induct j)
  case 0
  then show ?case by auto
next
  case (Suc j)
  show ?case
  proof (cases i ≤ j)
    case True
    have A $$ (i, i) dvd A $$ (j, j) using Suc.hyps Suc.prem1 True by simp
    also have ... dvd A $$ (Suc j, Suc j)
      using SNF-A Suc.prem1 A
    unfolding Smith-normal-form-mat-def by auto
    finally show ?thesis by auto
  next
  case False
  hence i = Suc j using Suc.prem1 by auto
  then show ?thesis by auto
qed
qed

```

**lemma** *Smith-zero-imp-zero*:

```

fixes A: 'a::comm-ring-1 mat
assumes A: A ∈ carrier-mat m n
  and SNF: Smith-normal-form-mat A
  and Aii: A $$ (i, i) = 0
  and j: j < min m n
  and ij: i ≤ j
shows A $$ (j, j) = 0

```

**proof** –

```

  have A $$ (i, i) dvd A $$ (j, j) by (rule SNF-divides-diagonal[OF A SNF j ij])
  thus ?thesis using Aii by auto

```

**qed**

**lemma** *SNF-preserved-multiples-identity*:

```

assumes S: S ∈ carrier-mat m n and SNF: Smith-normal-form-mat (S: 'a::comm-ring-1
mat)

```

```

  shows Smith-normal-form-mat (S * (k ·m 1m n))

```

**proof** (rule Smith-normal-form-mat-intro)

```

  have rw: S * (k ·m 1m n) = Matrix.mat m n (λ(i, j). S $$ (i, j) * k)

```

```

  unfolding mat-diag-smult[symmetric] by (rule mat-diag-mult-right[OF S])

```

```

  show isDiagonal-mat (S * (k ·m 1m n))

```

```

  using SNF S unfolding Smith-normal-form-mat-def isDiagonal-mat-def rw
  by auto

```

```

  show ∀ a. a + 1 < min (dim-row (S * (k ·m 1m n))) (dim-col (S * (k ·m 1m
n))) →

```

```

    (S * (k ·m 1m n)) $$ (a, a) dvd (S * (k ·m 1m n)) $$ (a + 1, a + 1)

```

```

  using SNF S unfolding Smith-normal-form-mat-def isDiagonal-mat-def rw

```

```

  by (auto simp add: mult-dvd-mono)

```

**qed**

end

## 7 Some theorems about rings and ideals

```
theory Rings2-Extended
  imports
    Echelon-Form.Rings2
    HOL-Types-To-Sets.Types-To-Sets
begin
```

### 7.1 Missing properties on ideals

```
lemma ideal-generated-subset2:
  assumes  $\forall b \in B. b \in \text{ideal-generated } A$ 
  shows  $\text{ideal-generated } B \subseteq \text{ideal-generated } A$ 
  by (metis (mono-tags, lifting) InterE assms ideal-generated-def
  ideal-ideal-generated mem-Collect-eq subsetI)
```

```
context comm-ring-1
begin
```

```
lemma ideal-explicit: ideal-generated  $S$ 
  =  $\{y. \exists f U. \text{finite } U \wedge U \subseteq S \wedge (\sum i \in U. f i * i) = y\}$ 
  by (simp add: ideal-generated-eq-left-ideal left-ideal-explicit)
end
```

```
lemma ideal-generated-minus:
  assumes  $a: a \in \text{ideal-generated } (S - \{a\})$ 
  shows  $\text{ideal-generated } S = \text{ideal-generated } (S - \{a\})$ 
proof (cases  $a \in S$ )
  case True note a-in-S = True
  show ?thesis
  proof
    show  $\text{ideal-generated } S \subseteq \text{ideal-generated } (S - \{a\})$ 
  proof (rule ideal-generated-subset2, auto)
    fix  $b$  assume  $b: b \in S$  show  $b \in \text{ideal-generated } (S - \{a\})$ 
  proof (cases  $b = a$ )
    case True
    then show ?thesis using  $a$  by auto
  next
    case False
    then show ?thesis using  $b$ 
    by (simp add: ideal-generated-in)
  qed
  qed
  show  $\text{ideal-generated } (S - \{a\}) \subseteq \text{ideal-generated } S$ 
  by (rule ideal-generated-subset, auto)
qed
```

```

next
  case False
  then show ?thesis by simp
qed

lemma ideal-generated-dvd-eq:
  assumes a-dvd-b:  $a \text{ dvd } b$ 
  and a:  $a \in S$ 
  and a-not-b:  $a \neq b$ 
  shows  $\text{ideal-generated } S = \text{ideal-generated } (S - \{b\})$ 
proof
  show  $\text{ideal-generated } S \subseteq \text{ideal-generated } (S - \{b\})$ 
  proof (rule ideal-generated-subset2, auto)
    fix x assume x:  $x \in S$ 
    show  $x \in \text{ideal-generated } (S - \{b\})$ 
    proof (cases  $x = b$ )
      case True
      obtain k where b-ak:  $b = a * k$  using a-dvd-b unfolding dvd-def by blast
      let ?f =  $\lambda c. k$ 
      have  $(\sum_{i \in \{a\}} i * ?f i) = x$  using True b-ak by auto
      moreover have  $\{a\} \subseteq S - \{b\}$  using a-not-b a by auto
      moreover have finite  $\{a\}$  by auto
      ultimately show ?thesis
      unfolding ideal-def
      by (metis True b-ak ideal-def ideal-generated-in ideal-ideal-generated insert-subset right-ideal-def)
    next
      case False
      then show ?thesis by (simp add: ideal-generated-in x)
    qed
  qed
  show  $\text{ideal-generated } (S - \{b\}) \subseteq \text{ideal-generated } S$  by (rule ideal-generated-subset, auto)
qed

lemma ideal-generated-dvd-eq-diff-set:
  assumes i-in-I:  $i \in I$  and i-in-J:  $i \notin J$  and i-dvd-j:  $\forall j \in J. i \text{ dvd } j$ 
  and f: finite J
  shows  $\text{ideal-generated } I = \text{ideal-generated } (I - J)$ 
  using f i-in-J i-dvd-j i-in-I
  proof (induct J arbitrary: I)
  case empty
  then show ?case by auto
  next
  case (insert x J)
  have  $\text{ideal-generated } I = \text{ideal-generated } (I - \{x\})$ 
  by (rule ideal-generated-dvd-eq[of i], insert insert.prem1 , auto)
  also have  $\dots = \text{ideal-generated } ((I - \{x\}) - J)$ 
  by (rule insert.hyps, insert insert.prem1 insert.hyps, auto)

```

**also have** ... = *ideal-generated* ( $I - \text{insert } x J$ )  
**using** *Diff-insert2*[*of I x J*] **by** *auto*  
**finally show** *?case* .  
**qed**

**context** *comm-ring-1*  
**begin**

**lemma** *ideal-generated-singleton-subset*:  
**assumes**  $d: d \in \text{ideal-generated } S$  **and**  $\text{fin-}S: \text{finite } S$   
**shows**  $\text{ideal-generated } \{d\} \subseteq \text{ideal-generated } S$   
**proof**  
**fix**  $x$  **assume**  $x: x \in \text{ideal-generated } \{d\}$   
**obtain**  $k$  **where**  $x\text{-kd}: x = k*d$  **using**  $x$  **using** *obtain-sum-ideal-generated*[*OF*  
 $x$ ]  
**by** (*metis finite.emptyI finite.insertI sum-singleton*)  
**show**  $x \in \text{ideal-generated } S$   
**using**  $d$  *ideal-eq-right-ideal ideal-ideal-generated right-ideal-def mult-commute*  
 $x\text{-kd}$  **by** *auto*  
**qed**

**lemma** *ideal-generated-singleton-dvd*:  
**assumes**  $i: \text{ideal-generated } S = \text{ideal-generated } \{d\}$  **and**  $x: x \in S$   
**shows**  $d \text{ dvd } x$   
**by** (*metis i x finite.intros dvd-ideal-generated-singleton*  
*ideal-generated-in ideal-generated-singleton-subset*)

**lemma** *ideal-generated-UNIV-insert*:  
**assumes**  $\text{ideal-generated } S = \text{UNIV}$   
**shows**  $\text{ideal-generated } (\text{insert } a S) = \text{UNIV}$  **using** *assms*  
**using** *local.ideal-generated-subset* **by** *blast*

**lemma** *ideal-generated-UNIV-union*:  
**assumes**  $\text{ideal-generated } S = \text{UNIV}$   
**shows**  $\text{ideal-generated } (A \cup S) = \text{UNIV}$   
**using** *assms local.ideal-generated-subset*  
**by** (*metis UNIV-I Un-subset-iff equalityI subsetI*)

**lemma** *ideal-explicit2*:  
**assumes**  $\text{finite } S$   
**shows**  $\text{ideal-generated } S = \{y. \exists f. (\sum_{i \in S} f i * i) = y\}$   
**by** (*smt (verit) Collect-cong assms ideal-explicit obtain-sum-ideal-generated mem-Collect-eq*  
*subsetI*)

**lemma** *ideal-generated-unit*:  
**assumes**  $u: u \text{ dvd } 1$   
**shows**  $\text{ideal-generated } \{u\} = \text{UNIV}$   
**proof** –

```

have  $x \in \text{ideal-generated } \{u\}$  for  $x$ 
proof -
  obtain  $inv-u$  where  $inv-u: inv-u * u = 1$  using  $u$  unfolding  $dvd-def$ 
  using  $local.mult-ac(2)$  by  $blast$ 
  have  $x = x * inv-u * u$  using  $inv-u$  by ( $simp$  add:  $local.mult-ac(1)$ )
  also have  $\dots \in \{k * u \mid k. k \in UNIV\}$  by  $auto$ 
  also have  $\dots = \text{ideal-generated } \{u\}$  unfolding  $\text{ideal-generated-singleton}$  by
 $simp$ 
  finally show  $?thesis$  .
qed
thus  $?thesis$  by  $auto$ 
qed

```

```

lemma  $\text{ideal-generated-dvd-subset}$ :
  assumes  $x: \forall x \in S. d \text{ dvd } x$  and  $S: \text{finite } S$ 
  shows  $\text{ideal-generated } S \subseteq \text{ideal-generated } \{d\}$ 
proof
  fix  $x$  assume  $x \in \text{ideal-generated } S$ 
  from  $this$  obtain  $f$  where  $f: (\sum i \in S. f i * i) = x$  using  $\text{ideal-explicit2}[OF S]$ 
  by  $auto$ 
  have  $d \text{ dvd } (\sum i \in S. f i * i)$  by ( $rule \text{dvd-sum}, \text{insert } x, \text{auto}$ )
  thus  $x \in \text{ideal-generated } \{d\}$ 
  using  $f \text{ dvd-ideal-generated-singleton}' \text{ ideal-generated-in singletonI}$  by  $blast$ 
qed

```

```

lemma  $\text{ideal-generated-mult-unit}$ :
  assumes  $f: \text{finite } S$  and  $u: u \text{ dvd } 1$ 
  shows  $\text{ideal-generated } ((\lambda x. u*x)' S) = \text{ideal-generated } S$ 
  using  $f$ 
proof ( $induct S$ )
  case  $empty$ 
  then show  $?case$  by  $auto$ 
next
  case ( $insert x S$ )
  obtain  $inv-u$  where  $inv-u: inv-u * u = 1$  using  $u$  unfolding  $dvd-def$ 
  using  $mult-ac$  by  $blast$ 
  have  $f: \text{finite } (insert (u*x) ((\lambda x. u*x)' S))$  using  $\text{insert.hyps}$  by  $auto$ 
  have  $f2: \text{finite } (insert x S)$  by ( $simp$  add:  $\text{insert}(1)$ )
  have  $f3: \text{finite } S$  by ( $simp$  add:  $\text{insert}$ )
  have  $f4: \text{finite } ((* u)' S)$  by ( $simp$  add:  $\text{insert}$ )
  have  $inj-ux: \text{inj-on } (\lambda x. u*x) S$  unfolding  $\text{inj-on-def}$ 
  by ( $auto,metis \text{inv-u local.mult-1-left local.semiring-normalization-rules}(18)$ )
  have  $\text{ideal-generated } ((\lambda x. u*x)' (insert x S)) = \text{ideal-generated } (insert (u*x) ((\lambda x. u*x)' S))$ 
  by  $auto$ 
  also have  $\dots = \{y. \exists f. (\sum i \in \text{insert } (u*x) ((\lambda x. u*x)' S). f i * i) = y\}$ 
  using  $\text{ideal-explicit2}[OF f]$  by  $auto$ 

```

**also have** ... =  $\{y. \exists f. (\sum i \in (\text{insert } x \ S). f \ i * i) = y\}$  **(is ?L = ?R)**  
**proof** –  
**have**  $a \in ?L$  **if**  $a \in ?R$  **for**  $a$   
**proof** –  
**obtain**  $f$  **where**  $\text{sum-rw}: (\sum i \in (\text{insert } x \ S). f \ i * i) = a$  **using**  $a$  **by** *auto*  
**define**  $b$  **where**  $b = (\sum i \in S. f \ i * i)$   
**have**  $b \in \text{ideal-generated } S$  **unfolding**  $b\text{-def}$  *ideal-explicit2[OF f3]* **by** *auto*  
**hence**  $b \in \text{ideal-generated } ((* \ u \ ' \ S)$  **using** *insert.hyps(3)* **by** *auto*  
**from this obtain**  $g$  **where**  $(\sum i \in ((* \ u \ ' \ S). g \ i * i) = b$   
**unfolding** *ideal-explicit2[OF f4]* **by** *auto*  
**hence**  $\text{sum-rw2}: (\sum i \in S. f \ i * i) = (\sum i \in ((* \ u \ ' \ S). g \ i * i)$  **unfolding**  $b\text{-def}$   
**by** *auto*  
**let**  $?g = \lambda i. \text{if } i = u * x \text{ then } f \ x * \text{inv-}u \text{ else } g \ i$   
**have**  $\text{sum-rw3}: \text{sum } ((\lambda i. g \ i * i) \circ (\lambda x. u * x)) \ S = \text{sum } ((\lambda i. ?g \ i * i) \circ$   
 $(\lambda x. u * x)) \ S$   
**by** (*rule sum.cong, auto, metis inv-u local.insert(2) local.mult-1-right*  
*local.mult-ac(2) local.semiring-normalization-rules(18)*)  
**have**  $\text{sum-rw4}: (\sum i \in (\lambda x. u * x) \ ' \ S. g \ i * i) = \text{sum } ((\lambda i. g \ i * i) \circ (\lambda x. u * x))$   
 $S$   
**by** (*rule sum.reindex[OF inj-ux]*)  
**have**  $a = f \ x * x + (\sum i \in S. f \ i * i)$   
**using**  $\text{sum-rw}$  *local.insert(1) local.insert(2)* **by** *auto*  
**also have** ... =  $f \ x * x + (\sum i \in (\lambda x. u * x) \ ' \ S. g \ i * i)$  **using**  $\text{sum-rw2}$  **by** *auto*  
**also have** ... =  $?g \ (u * x) * (u * x) + (\sum i \in (\lambda x. u * x) \ ' \ S. g \ i * i)$   
**using**  $\text{inv-}u$  **by** (*smt (verit) local.mult-1-right local.mult-ac(1)*)  
**also have** ... =  $?g \ (u * x) * (u * x) + \text{sum } ((\lambda i. g \ i * i) \circ (\lambda x. u * x)) \ S$   
**using**  $\text{sum-rw4}$  **by** *auto*  
**also have** ... =  $((\lambda i. ?g \ i * i) \circ (\lambda x. u * x)) \ x + \text{sum } ((\lambda i. g \ i * i) \circ (\lambda x.$   
 $u * x)) \ S$  **by** *auto*  
**also have** ... =  $((\lambda i. ?g \ i * i) \circ (\lambda x. u * x)) \ x + \text{sum } ((\lambda i. ?g \ i * i) \circ (\lambda x.$   
 $u * x)) \ S$   
**using**  $\text{sum-rw3}$  **by** *auto*  
**also have** ... =  $\text{sum } ((\lambda i. ?g \ i * i) \circ (\lambda x. u * x)) \ (\text{insert } x \ S)$   
**by** (*rule sum.insert[symmetric], auto simp add: insert*)  
**also have** ... =  $(\sum i \in \text{insert } (u * x) \ ((\lambda x. u * x) \ ' \ S). ?g \ i * i)$   
**by** (*smt (verit) abel-semigroup commute f2 image-insert inv-u mult.abel-semigroup-axioms*  
*mult-1-right*  
*semiring-normalization-rules(18) sum.reindex-nontrivial*)  
**also have** ... =  $(\sum i \in (\lambda x. u * x) \ ' \ (\text{insert } x \ S). ?g \ i * i)$  **by** *auto*  
**finally show**  $?thesis$  **by** *auto*  
**qed**  
**moreover have**  $a \in ?R$  **if**  $a \in ?L$  **for**  $a$   
**proof** –  
**obtain**  $f$  **where**  $\text{sum-rw}: (\sum i \in (\text{insert } (u * x) \ ((* \ u \ ' \ S)). f \ i * i) = a$  **using**  
 $a$  **by** *auto*  
**have**  $u * x \notin ((* \ u \ ' \ S)$   
**by** (*metis UNIV-I inj-on-image-mem-iff inj-on-inverseI inv-u local.insert(2)*  
*local.mult-1-left*  
*local.semiring-normalization-rules(18) subsetI*)



```

let ?f = (λx. f x * x)
have sum ?f ((*) u ' S) ∈ ideal-generated ((*) u ' S)
  unfolding ideal-explicit2[OF f4] by auto
from this obtain g where sum-rw1: sum (λi. g i * i) S = sum ?f ((*) u '
S))
  using insert.hyps(3) unfolding ideal-explicit2[OF f3] by blast
let ?g = (λi. if i = x then (f (u*x) * u) * x else g i * i)
let ?g' = λi. if i = x then f (u*x) * u else g i
  have sum-rw2: sum (λi. g i * i) S = sum ?g S by (rule sum.cong, insert
inj-ux ux-notin, auto)
  have a = (∑ i∈(insert (u * x) ((*) u ' S)). f i * i) using sum-rw by simp
  also have ... = ?f (u*x) + sum ?f ((*) u ' S)
  by (rule sum.insert[OF f4], insert inj-ux) (metis UNIV-I inj-on-image-mem-iff
inj-on-inverseI
  inv-u local.insert(2) local.mult-1-left local.semiring-normalization-rules(18)
subsetI)
  also have ... = ?f (u*x) + sum (λi. g i * i) S unfolding sum-rw1 by auto
  also have ... = ?g x + sum ?g S unfolding sum-rw2 using mult.assoc by
auto
  also have ... = sum ?g (insert x S) by (rule sum.insert[symmetric, OF f3
insert.hyps(2)])
  also have ... = sum (λi. ?g' i * i) (insert x S) by (rule sum.cong, auto)
  finally show ?thesis by fast
qed
ultimately show ?thesis by blast
qed
also have ... = ideal-generated (insert x S) using ideal-explicit2[OF f2] by auto
finally show ?case by auto
qed

```

**corollary** *ideal-generated-mult-unit2*:

```

assumes u: u dvd 1
shows ideal-generated {u*a,u*b} = ideal-generated {a,b}
proof -
let ?S = {a,b}
have ideal-generated {u*a,u*b} = ideal-generated ((λx. u*x)' {a,b}) by auto
also have ... = ideal-generated {a,b} by (rule ideal-generated-mult-unit[OF - u],
simp)
finally show ?thesis .
qed

```

**lemma** *ideal-generated-1[simp]*: *ideal-generated {1} = UNIV*

by (metis ideal-generated-unit dvd-ideal-generated-singleton order-refl)

**lemma** *ideal-generated-pair*: *ideal-generated {a,b} = {p\*a+q\*b | p q. True}*

**proof** -

```

have i: ideal-generated {a,b} = {y. ∃f. (∑ i∈{a,b}. f i * i) = y} using ideal-explicit2
by auto
show ?thesis

```

```

proof (cases a=b)
  case True
    show ?thesis using True i
      by (auto, metis mult-ac(2) semiring-normalization-rules)
      (metis (no-types, opaque-lifting) add-minus-cancel mult-ac ring-distrib semir-
ing-normalization-rules)
    next
      case False
        have 1:  $\exists p q. (\sum_{i \in \{a, b\}} f i * i) = p * a + q * b$  for f
          by (rule exI[of - f a], rule exI[of - f b], rule sum-two-elements[OF False])
        moreover have  $\exists f. (\sum_{i \in \{a, b\}} f i * i) = p * a + q * b$  for p q
          by (rule exI[of -  $\lambda i. \text{if } i=a \text{ then } p \text{ else } q$ ],
            unfold sum-two-elements[OF False], insert False, auto)
        ultimately show ?thesis using i by auto
      qed
    qed

```

```

lemma ideal-generated-pair-exists-pq1:
  assumes i: ideal-generated {a,b} = (UNIV::'a set)
  shows  $\exists p q. p*a + q*b = 1$ 
  using i unfolding ideal-generated-pair
  by (smt (verit) iso-tuple-UNIV-I mem-Collect-eq)

```

```

lemma ideal-generated-pair-UNIV:
  assumes sa-tb-u:  $s*a + t*b = u$  and u: u dvd 1
  shows ideal-generated {a,b} = UNIV
proof -
  have f: finite {a,b} by simp
  obtain inv-u where inv-u:  $\text{inv-}u * u = 1$  using u unfolding dvd-def
    by (metis mult.commute)
  have x  $\in$  ideal-generated {a,b} for x
  proof (cases a = b)
    case True
      then show ?thesis
        by (metis UNIV-I dvd-def dvd-ideal-generated-singleton' ideal-generated-unit
insert-absorb2
          mult.commute sa-tb-u semiring-normalization-rules(34) subsetI sub-
set-antisym u)
    next
      case False note a-not-b = False
      let ?f =  $\lambda y. \text{if } y = a \text{ then } \text{inv-}u * x * s \text{ else } \text{inv-}u * x * t$ 
      have  $(\sum_{i \in \{a, b\}} ?f i * i) = ?f a * a + ?f b * b$  by (rule sum-two-elements[OF
a-not-b])
      also have ... = x using a-not-b sa-tb-u inv-u
      by (auto, metis mult-ac(1) mult-ac(2) ring-distrib(1) semiring-normalization-rules(12))
      finally show ?thesis unfolding ideal-explicit2[OF f] by auto
    qed
  thus ?thesis by auto
qed

```

**lemma** *ideal-generated-pair-exists*:  
**assumes**  $l$ : (*ideal-generated*  $\{a,b\} = \text{ideal-generated } \{d\}$ )  
**shows**  $(\exists p q. p*a+q*b = d)$   
**proof** –  
**have**  $d$ :  $d \in \text{ideal-generated } \{d\}$  **by** (*simp add: ideal-generated-in*)  
**hence**  $d \in \text{ideal-generated } \{a,b\}$  **using**  $l$  **by** *auto*  
**from** *this* **obtain**  $p q$  **where**  $d = p*a+q*b$  **using** *ideal-generated-pair[of a b]* **by**  
*auto*  
**thus** *?thesis* **by** *auto*  
**qed**

**lemma** *obtain-ideal-generated-pair*:  
**assumes**  $c \in \text{ideal-generated } \{a,b\}$   
**obtains**  $p q$  **where**  $p*a+q*b=c$   
**proof** –  
**have**  $c \in \{p * a + q * b \mid p q. \text{True}\}$  **using** *assms ideal-generated-pair* **by** *auto*  
**thus** *?thesis* **using** *that* **by** *auto*  
**qed**

**lemma** *ideal-generated-pair-exists-UNIV*:  
**shows** (*ideal-generated*  $\{a,b\} = \text{ideal-generated } \{1\}$ ) =  $(\exists p q. p*a+q*b = 1)$  (**is**  
*?lhs = ?rhs*)  
**proof**  
**assume**  $r$ : *?rhs*  
**have**  $x \in \text{ideal-generated } \{a,b\}$  **for**  $x$   
**proof** (*cases a=b*)  
**case** *True*  
**then show** *?thesis*  
**by** (*metis UNIV-I r dvd-ideal-generated-singleton finite.intros ideal-generated-1*  
*ideal-generated-pair-UNIV ideal-generated-singleton-subset*)  
**next**  
**case** *False*  
**have**  $f$ : *finite*  $\{a,b\}$  **by** *simp*  
**have**  $1$ :  $1 \in \text{ideal-generated } \{a,b\}$   
**using** *ideal-generated-pair-UNIV local.one-dvd r* **by** *blast*  
**hence**  $i$ : *ideal-generated*  $\{a,b\} = \{y. \exists f. (\sum i \in \{a,b\}. f i * i) = y\}$   
**using** *ideal-explicit2[of {a,b}]* **by** *auto*  
**from** *this* **obtain**  $f$  **where**  $f: f a * a + f b * b = 1$  **using** *sum-two-elements 1*  
*False* **by** *auto*  
**let**  $?f = \lambda y. \text{if } y = a \text{ then } x * f a \text{ else } x * f b$   
**have**  $(\sum i \in \{a,b\}. ?f i * i) = x$  **unfolding** *sum-two-elements[OF False]* **using**  
*f False*  
**using** *mult-ac(1) ring-distrib(1) semiring-normalization-rules(12)* **by** *force*  
**thus** *?thesis* **unfolding**  $i$  **by** *auto*  
**qed**  
**thus** *?lhs* **by** *auto*

**next**  
 assume *?lhs thus ?rhs using ideal-generated-pair-exists[of a b 1] by auto*  
**qed**  
  
**corollary** *ideal-generated-UNIV-obtain-pair:*  
 assumes *ideal-generated {a,b} = ideal-generated {1}*  
 shows  $(\exists p q. p*a+q*b = d)$   
**proof** –  
 obtain *x y where x\*a+y\*b = 1 using ideal-generated-pair-exists-UNIV assms*  
**by auto**  
 hence  $d*x*a+d*y*b=d$   
 using *local.mult-ac(1) local.ring-distrib(1) local.semiring-normalization-rules(12)*  
**by force**  
 thus *?thesis by auto*  
**qed**

**lemma** *sum-three-elements:*  
 shows  $\exists x y z. 'a. (\sum i \in \{a,b,c\}. f i * i) = x * a + y * b + z * c$   
**proof** (*cases a ≠ b ∧ b ≠ c ∧ a ≠ c*)  
 case *True*  
 then show *?thesis by (auto, metis add.assoc)*  
**next**  
 case *False*  
 have 1:  $\exists x y z. f c * c = x * c + y * c + z * c$   
 by (*rule exI[of - 0], rule exI[of - 0], rule exI[of - f c], auto*)  
 have 2:  $\exists x y z. f b * b + f c * c = x * b + y * b + z * c$   
 by (*rule exI[of - 0], rule exI[of - f b], rule exI[of - f c], auto*)  
 have 3:  $\exists x y z. f a * a + f c * c = x * a + y * c + z * c$   
 by (*rule exI[of - f a], rule exI[of - 0], rule exI[of - f c], auto*)  
 have 4:  $\exists x y z. (\sum i \in \{c, b, c\}. f i * i) = x * c + y * b + z * c$  **if** *a: a = c and*  
*b: b ≠ c*  
 by (*rule exI[of - 0], rule exI[of - f b], rule exI[of - f c], insert a b,*  
*auto simp add: insert-commute*)  
 show *?thesis using False*  
 by (*cases b=c, cases a=c, auto simp add: 1 2 3 4*)  
**qed**

**lemma** *sum-three-elements':*  
 shows  $\exists f. 'a \Rightarrow 'a. (\sum i \in \{a,b,c\}. f i * i) = x * a + y * b + z * c$   
**proof** (*cases a ≠ b ∧ b ≠ c ∧ a ≠ c*)  
 case *True*  
 let *?f = λi. if i = a then x else if i = b then y else if i = c then z else 0*  
 show *?thesis by (rule exI[of - ?f], insert True mult.assoc, auto simp add: local.add-ac)*  
**next**  
 case *False*  
 have 1:  $\exists f. f c * c = x * c + y * c + z * c$

**by** (rule *exI*[of -  $\lambda i$ . if  $i = c$  then  $x+y+z$  else  $0$ ], auto simp add: local.ring-distrib)
  
**have** 2:  $\exists f. f a * a + f c * c = x * a + y * c + z * c$  **if**  $bc: b = c$  **and**  $ac: a \neq c$ 
  
**by** (rule *exI*[of -  $\lambda i$ . if  $i = a$  then  $x$  else  $y+z$ ], insert *ac bc add-ac ring-distrib*, auto)
  
**have** 3:  $\exists f. f b * b + f c * c = x * b + y * b + z * c$  **if**  $bc: b \neq c$  **and**  $ac: a = b$ 
  
**by** (rule *exI*[of -  $\lambda i$ . if  $i = a$  then  $x+y$  else  $z$ ], insert *ac bc add-ac ring-distrib*, auto)
  
**have** 4:  $\exists f. (\sum i \in \{c, b, c\}. f i * i) = x * c + y * b + z * c$  **if**  $a: a = c$  **and**  $b: b \neq c$ 
  
**by** (rule *exI*[of -  $\lambda i$ . if  $i = c$  then  $x+z$  else  $y$ ], insert *a b add-ac ring-distrib*, auto simp add: *insert-commute*)
  
**show** ?thesis **using** False
  
**by** (cases  $b=c$ , cases  $a=c$ , auto simp add: 1 2 3 4)
  
**qed**

**lemma** *ideal-generated-triple-pair-rewrite*:

**assumes**  $i1: \text{ideal-generated } \{a, b, c\} = \text{ideal-generated } \{d\}$ 
  
**and**  $i2: \text{ideal-generated } \{a, b\} = \text{ideal-generated } \{d'\}$ 
  
**shows**  $\text{ideal-generated } \{d', c\} = \text{ideal-generated } \{d\}$ 
  
**proof**
  
**have**  $d': d' \in \text{ideal-generated } \{a, b\}$  **using**  $i2$  **by** (simp add: *ideal-generated-in*)
  
**show**  $\text{ideal-generated } \{d', c\} \subseteq \text{ideal-generated } \{d\}$ 
  
**proof**
  
**fix**  $x$  **assume**  $x: x \in \text{ideal-generated } \{d', c\}$ 
  
**obtain**  $f1 f2$  **where**  $f: f1*d' + f2*c = x$  **using** *obtain-ideal-generated-pair*[OF  $x$ ] **by** auto
  
**obtain**  $g1 g2$  **where**  $g: g1*a + g2*b = d'$  **using** *obtain-ideal-generated-pair*[OF  $d'$ ] **by** blast
  
**have** 1:  $f1*g1*a + f1*g2*b + f2*c = x$ 
  
**using**  $f g$  *local.ring-distrib(1) local.semiring-normalization-rules(18)* **by** auto
  
**have**  $x \in \text{ideal-generated } \{a, b, c\}$ 
  
**proof** -
  
**obtain**  $f$  **where**  $(\sum i \in \{a, b, c\}. f i * i) = f1*g1*a + f1*g2*b + f2*c$ 
  
**using** *sum-three-elements' 1* **by** blast
  
**moreover** **have**  $\text{ideal-generated } \{a, b, c\} = \{y. \exists f. (\sum i \in \{a, b, c\}. f i * i) = y\}$ 
  
**using** *ideal-explicit2*[of  $\{a, b, c\}$ ] **by** simp
  
**ultimately show** ?thesis **using** 1 **by** auto
  
**qed**
  
**thus**  $x \in \text{ideal-generated } \{d\}$  **using**  $i1$  **by** auto
  
**qed**
  
**show**  $\text{ideal-generated } \{d\} \subseteq \text{ideal-generated } \{d', c\}$ 
  
**proof** (rule *ideal-generated-singleton-subset*)
  
**obtain**  $f1 f2 f3$  **where**  $f: f1*a + f2*b + f3*c = d$ 
  
**proof** -

```

have  $d \in \text{ideal-generated } \{a,b,c\}$  using i1 by (simp add: ideal-generated-in)
from this obtain f where  $d: (\sum_{i \in \{a,b,c\}} f i * i) = d$ 
  using ideal-explicit2[of  $\{a,b,c\}$ ] by auto
obtain x y z where  $(\sum_{i \in \{a,b,c\}} f i * i) = x * a + y * b + z * c$ 
  using sum-three-elements by blast
  thus ?thesis using d that by auto
qed
obtain k where  $k: f1*a + f2*b = k*d'$ 
proof –
  have  $f1*a + f2*b \in \text{ideal-generated}\{a,b\}$  using ideal-generated-pair by blast
  also have  $\dots = \text{ideal-generated } \{d'\}$  using i2 by simp
  also have  $\dots = \{k*d' \mid k. k \in UNIV\}$  using ideal-generated-singleton by auto
  finally show ?thesis using that by auto
qed
have  $k*d' + f3*c = d$  using f k by auto
thus  $d \in \text{ideal-generated } \{d', c\}$ 
  using ideal-generated-pair by blast
qed (simp)
qed

```

```

lemma ideal-generated-dvd:
  assumes i:  $\text{ideal-generated } \{a,b::'a\} = \text{ideal-generated}\{d\}$ 
  and a:  $d' \text{ dvd } a$  and b:  $d' \text{ dvd } b$ 
shows  $d' \text{ dvd } d$ 
proof –
  obtain p q where  $p*a + q*b = d$ 
  using i ideal-generated-pair-exists by blast
  thus ?thesis using a b by auto
qed

```

```

lemma ideal-generated-dvd2:
  assumes i:  $\text{ideal-generated } S = \text{ideal-generated}\{d::'a\}$ 
  and finite S
  and x:  $\forall x \in S. d' \text{ dvd } x$ 
shows  $d' \text{ dvd } d$ 
  by (metis assms dvd-ideal-generated-singleton ideal-generated-dvd-subset)

```

**end**

## 7.2 An equivalent characterization of Bézout rings

The goal of this subsection is to prove that a ring is Bézout ring if and only if every finitely generated ideal is principal.

**definition** *finitely-generated-ideal*  $I = (\text{ideal } I \wedge (\exists S. \text{finite } S \wedge \text{ideal-generated } S = I))$

**context**

**assumes** *SORT-CONSTRAINT*('a::comm-ring-1)

**begin**

```

lemma sum-two-elements':
  fixes  $d::'a$ 
  assumes  $s: (\sum i \in \{a,b\}. f i * i) = d$ 
  obtains  $p$  and  $q$  where  $d = p * a + q * b$ 
proof (cases a=b)
  case True
  then show ?thesis
    by (metis (no-types, lifting) add-diff-cancel-left' emptyE finite.emptyI insert-absorb2
      left-diff-distrib' s sum.insert sum-singleton that)
next
  case False
  show ?thesis using  $s$  unfolding sum-two-elements[OF False]
    using that by auto
qed

```

This proof follows Theorem 6-3 in "First Course in Rings and Ideals" by Burton

```

lemma all-fin-gen-ideals-are-principal-imp-bezout:
  assumes  $all: \forall I::'a \text{ set. finitely-generated-ideal } I \longrightarrow \text{principal-ideal } I$ 
  shows OFCLASS ('a, bezout-ring-class)
proof (intro-classes)
  fix  $a b::'a$ 
  obtain  $d$  where ideal-d: ideal-generated {a,b} = ideal-generated {d}
    using  $all$  unfolding finitely-generated-ideal-def
    by (metis finite.emptyI finite-insert ideal-ideal-generated principal-ideal-def)
  have a-in-d: a ∈ ideal-generated {d}
    using ideal-d ideal-generated-subset-generator by blast
  have b-in-d: b ∈ ideal-generated {d}
    using ideal-d ideal-generated-subset-generator by blast
  have d-in-ab: d ∈ ideal-generated {a,b}
    using ideal-d ideal-generated-subset-generator by auto
  obtain  $f$  where  $(\sum i \in \{a,b\}. f i * i) = d$  using obtain-sum-ideal-generated[OF
d-in-ab] by auto
  from this obtain  $p q$  where d-eq: d = p*a + q*b using sum-two-elements' by
blast
  moreover have d-dvd-a: d dvd a
    by (metis dvd-ideal-generated-singleton ideal-d ideal-generated-subset insert-commute
      subset-insertI)
  moreover have d dvd b
    by (metis dvd-ideal-generated-singleton ideal-d ideal-generated-subset subset-insertI)
  moreover have  $d' \text{ dvd } d$  if d'-dvd: d' dvd a ∧ d' dvd b for  $d'$ 
proof –
  obtain  $s1 s2$  where s1-dvd: a = s1*d' and s2-dvd: b = s2*d'
    using mult.commute d'-dvd unfolding dvd-def by auto
  have  $d = p*a + q*b$  using d-eq .
  also have  $\dots = p * s1 * d' + q * s2 * d'$  unfolding s1-dvd s2-dvd by auto
  also have  $\dots = (p * s1 + q * s2) * d'$  by (simp add: ring-class.ring-distrib(2))

```

```

    finally show  $d' \text{ dvd } d$  using mult.commute unfolding dvd-def by auto
qed
ultimately show  $\exists p q d. p * a + q * b = d \wedge d \text{ dvd } a \wedge d \text{ dvd } b$ 
 $\wedge (\forall d'. d' \text{ dvd } a \wedge d' \text{ dvd } b \longrightarrow d' \text{ dvd } d)$  by auto
qed
end

context bezout-ring
begin

lemma exists-bezout-extended:
  assumes  $S$ : finite S and  $ne$ :  $S \neq \{\}$ 
  shows  $\exists f d. (\sum a \in S. f a * a) = d \wedge (\forall a \in S. d \text{ dvd } a) \wedge (\forall d'. (\forall a \in S. d' \text{ dvd } a) \longrightarrow d' \text{ dvd } d)$ 
  using  $S$   $ne$ 
proof (induct  $S$ )
  case empty
  then show ?case by auto
next
  case (insert x S)
  show ?case
  proof (cases  $S = \{\}$ )
    case True
    let  $?f = \lambda x. 1$ 
    show ?thesis by (rule exI[of - ?f], insert True, auto)
  next
    case False note  $ne = \text{False}$ 
    note  $x \text{ notin } S = \text{insert.hyps}(2)$ 
    obtain  $f d$  where sum-eq-d:  $(\sum a \in S. f a * a) = d$ 
      and d-dvd-each-a:  $(\forall a \in S. d \text{ dvd } a)$ 
      and d-is-gcd:  $(\forall d'. (\forall a \in S. d' \text{ dvd } a) \longrightarrow d' \text{ dvd } d)$ 
      using insert.hyps(3)[OF ne] by auto
    have  $\exists p q d'. p * d + q * x = d' \wedge d' \text{ dvd } d \wedge d' \text{ dvd } x \wedge (\forall c. c \text{ dvd } d \wedge c \text{ dvd } x \longrightarrow c \text{ dvd } d')$ 
      using exists-bezout by auto
    from this obtain  $p q d'$  where pd-qx-d':  $p*d + q*x = d'$ 
      and d'-dvd-d:  $d' \text{ dvd } d$  and d'-dvd-x:  $d' \text{ dvd } x$ 
      and d'-dvd:  $\forall c. (c \text{ dvd } d \wedge c \text{ dvd } x) \longrightarrow c \text{ dvd } d'$  by blast
    let  $?f = \lambda a. \text{if } a = x \text{ then } q \text{ else } p * f a$ 
    have  $(\sum a \in \text{insert } x S. ?f a * a) = d'$ 
    proof -
      have  $(\sum a \in \text{insert } x S. ?f a * a) = (\sum a \in S. ?f a * a) + ?f x * x$ 
        by (simp add: add-commute insert.hyps(1) insert.hyps(2))
      also have  $\dots = p * (\sum a \in S. f a * a) + q * x$ 
        unfolding sum-distrib-left
        by (auto, rule sum.cong, insert  $x \text{ notin } S$ ,
          auto simp add: mult.semigroup-axioms semigroup.assoc)
      finally show ?thesis using pd-qx-d' sum-eq-d by auto
    end
  end
end

```



```

qed
moreover have (∀ a ∈ insert x S. d' dvd a)
by (metis d'-dvd-d d'-dvd-x d-dvd-each-a insert-iff local.dvdE local.dvd-mult-left)
moreover have (∀ c. (∀ a ∈ insert x S. c dvd a) → c dvd d')
by (simp add: d'-dvd d-is-gcd)
ultimately show ?thesis by auto
qed
qed

end

lemma ideal-generated-empty: ideal-generated {} = {0}
unfolding ideal-generated-def using ideal-generated-0
by (metis empty-subsetI ideal-generated-def ideal-generated-subset ideal-ideal-generated
ideal-not-empty subset-singletonD)

lemma bezout-imp-all-fin-gen-ideals-are-principal:
fixes I::'a :: bezout-ring set
assumes fin: finitely-generated-ideal I
shows principal-ideal I
proof -
obtain S where fin-S: finite S and ideal-gen-S: ideal-generated S = I
using fin unfolding finitely-generated-ideal-def by auto
show ?thesis
proof (cases S = {})
case True
then show ?thesis
using ideal-gen-S unfolding True
using ideal-generated-empty ideal-generated-0 principal-ideal-def by fastforce
next
case False note ne = False
obtain d f where sum-S-d: (∑ i ∈ S. f i * i) = d
and d-dvd-a: (∀ a ∈ S. d dvd a) and d-is-gcd: (∀ d'. (∀ a ∈ S. d' dvd a) → d' dvd
d)
using exists-bezout-extended[OF fin-S ne] by auto
have d-in-S: d ∈ ideal-generated S
by (metis fin-S ideal-def ideal-generated-subset-generator
ideal-ideal-generated sum-S-d sum-left-ideal)
have ideal-generated {d} ⊆ ideal-generated S
by (rule ideal-generated-singleton-subset[OF d-in-S fin-S])
moreover have ideal-generated S ⊆ ideal-generated {d}
proof
fix x assume x-in-S: x ∈ ideal-generated S
obtain f where sum-S-x: (∑ a ∈ S. f a * a) = x
using fin-S obtain-sum-ideal-generated x-in-S by blast
have d-dvd-each-a: ∃ k. a = k * d if a ∈ S for a
by (metis d-dvd-a dvdE mult.commute that)
let ?g = λa. SOME k. a = k*d

```

```

have  $x = (\sum a \in S. f a * a)$  using sum-S-x by simp
also have  $\dots = (\sum a \in S. f a * (?g a * d))$ 
proof (rule sum.cong)
  fix  $a$  assume  $a \text{-in-} S: a \in S$ 
  obtain  $k$  where  $a \text{-kd}: a = k * d$  using d-dvd-each-a a-in-S by auto
  have  $a = ((SOME k. a = k * d) * d)$  by (rule someI-ex, auto simp add:
a-kd)
  thus  $f a * a = f a * ((SOME k. a = k * d) * d)$  by auto
qed (simp)
also have  $\dots = (\sum a \in S. f a * ?g a * d)$  by (rule sum.cong, auto)
also have  $\dots = (\sum a \in S. f a * ?g a) * d$  using sum-distrib-right[of - S d] by
auto
finally show  $x \in \text{ideal-generated } \{d\}$ 
  by (meson contra-subsetD dvd-ideal-generated-singleton' dvd-triv-right
ideal-generated-in singletonI)
qed
ultimately show ?thesis unfolding principal-ideal-def using ideal-gen-S by
auto
qed
qed

```

Now we have the required lemmas to prove the theorem that states that a ring is Bézout ring if and only if every finitely generated ideal is principal. They are the following ones.

- *all-fin-gen-ideals-are-principal-imp-bezout*
- *bezout-imp-all-fin-gen-ideals-are-principal*

However, in order to prove the final lemma, we need the lemmas with no type restrictions. For instance, we need a version of theorem *bezout-imp-all-fin-gen-ideals-are-principal* as

*OFCLASS('a,bezout-ring)  $\implies$  the theorem with generic types (i.e., 'a with no type restrictions)*

or as

*class.bezout-ring - - -  $\implies$  the theorem with generic types (i.e., 'a with no type restrictions)*

Thanks to local type definitions, we can obtain it automatically by means of *internalize-sort*.

**lemma** *bezout-imp-all-fin-gen-ideals-are-principal-unsatisfactory:*

**assumes**  $a1: \text{class.bezout-ring } (*) (1::'b::\text{comm-ring-1}) (+) 0 (-) \text{uminus}$

**shows**  $\forall I::'b \text{ set. finitely-generated-ideal } I \longrightarrow \text{principal-ideal } I$

**using** *bezout-imp-all-fin-gen-ideals-are-principal[internalize-sort 'a::bezout-ring]*

**using**  $a1$  **by** *auto*

The standard library does not connect *OFCLASS* and *class.bezout-ring* in both directions. Here we show that *OFCLASS  $\implies$  class.bezout-ring*.

**lemma** *OFCLASS-bezout-ring-imp-class-bezout-ring*:  
**assumes** *OFCLASS('a::comm-ring-1,bezout-ring-class)*  
**shows** *class.bezout-ring ((\*)::'a⇒'a⇒'a) 1 (+) 0 (-) uminus*  
**using** *assms*  
**unfolding** *bezout-ring-class-def class.bezout-ring-def*  
**using** *conjunctionD2[of OFCLASS('a, comm-ring-1-class)*  
*class.bezout-ring-axioms ((\*)::'a⇒'a⇒'a) (+)]*  
**by** (*auto, intro-locales*)

The other implication can be obtained by `thm bezout-ring.intro-of-class`

**thm** *bezout-ring.intro-of-class*

Final theorem (with OFCLASS)

**lemma** *bezout-ring-iff-fin-gen-principal-ideal*:  
 $(\bigwedge I::'a::comm-ring-1\ set.\ finitely-generated-ideal\ I \implies principal-ideal\ I)$   
 $\equiv OFCLASS('a, bezout-ring-class)$

**proof**

**show**  $(\bigwedge I::'a::comm-ring-1\ set.\ finitely-generated-ideal\ I \implies principal-ideal\ I)$   
 $\implies OFCLASS('a, bezout-ring-class)$

**using** *all-fin-gen-ideals-are-principal-imp-bezout* [**where**  $?'a='a$ ] **by** *auto*

**show**  $\bigwedge I::'a::comm-ring-1\ set.\ OFCLASS('a, bezout-ring-class)$   
 $\implies finitely-generated-ideal\ I \implies principal-ideal\ I$

**using** *bezout-imp-all-fin-gen-ideals-are-principal-unsatisfactory* [**where**  $?'b='a$ ]

**using** *OFCLASS-bezout-ring-imp-class-bezout-ring* [**where**  $?'a='a$ ] **by** *auto*

**qed**

Final theorem (with *class.bezout-ring*)

**lemma** *bezout-ring-iff-fin-gen-principal-ideal2*:  
 $(\forall I::'a::comm-ring-1\ set.\ finitely-generated-ideal\ I \longrightarrow principal-ideal\ I)$   
 $= (class.bezout-ring ((*)::'a⇒'a⇒'a) 1 (+) 0 (-) uminus)$

**proof**

**show**  $\forall I::'a::comm-ring-1\ set.\ finitely-generated-ideal\ I \longrightarrow principal-ideal\ I$   
 $\implies class.bezout-ring\ (*)\ 1\ (+)\ (0::'a)\ (-)\ uminus$

**using** *all-fin-gen-ideals-are-principal-imp-bezout* [**where**  $?'a='a$ ]

**using** *OFCLASS-bezout-ring-imp-class-bezout-ring* [**where**  $?'a='a$ ]

**by** *auto*

**show** *class.bezout-ring*  $(*)\ 1\ (+)\ (0::'a)\ (-)\ uminus \implies \forall I::'a\ set.$   
*finitely-generated-ideal*  $I \longrightarrow principal-ideal\ I$

**using** *bezout-imp-all-fin-gen-ideals-are-principal-unsatisfactory* **by** *auto*

**qed**

**end**

## 8 Connection between *mod-ring* and *mod-type*

This file shows that the type *mod-ring*, which is defined in the Berlekamp–Zassenhaus development, is an instantiation of the type class *mod-type*.

```

theory Finite-Field-Mod-Type-Connection
  imports
    Berlekamp-Zassenhaus.Finite-Field
    Rank-Nullity-Theorem.Mod-Type
begin

instantiation mod-ring :: (finite) ord
begin
definition less-eq-mod-ring :: 'a mod-ring  $\Rightarrow$  'a mod-ring  $\Rightarrow$  bool
  where less-eq-mod-ring x y = (to-int-mod-ring x  $\leq$  to-int-mod-ring y)

definition less-mod-ring :: 'a mod-ring  $\Rightarrow$  'a mod-ring  $\Rightarrow$  bool
  where less-mod-ring x y = (to-int-mod-ring x  $<$  to-int-mod-ring y)

instance proof qed
end

instantiation mod-ring :: (finite) linorder
begin
instance by (intro-classes, unfold less-eq-mod-ring-def less-mod-ring-def) (transfer,
auto)
end

instance mod-ring :: (finite) wellorder
proof –
have wf {(x :: 'a mod-ring, y). x  $<$  y}
  by (auto simp add: trancl-def tranclp-less intro!: finite-acyclic-wf acyclicI)
  thus OFCLASS('a mod-ring, wellorder-class)
  by(rule wf-wellorderI) intro-classes
qed

lemma strict-mono-to-int-mod-ring: strict-mono to-int-mod-ring
  unfolding strict-mono-def unfolding less-mod-ring-def by auto

instantiation mod-ring :: (nontriv) mod-type
begin
definition Rep-mod-ring :: 'a mod-ring  $\Rightarrow$  int
  where Rep-mod-ring x = to-int-mod-ring x

definition Abs-mod-ring :: int  $\Rightarrow$  'a mod-ring
  where Abs-mod-ring x = of-int-mod-ring x

instance
proof (intro-classes)
  show type-definition (Rep::'a mod-ring  $\Rightarrow$  int) Abs {0..<int CARD('a mod-ring)}
  unfolding Rep-mod-ring-def Abs-mod-ring-def type-definition-def by (transfer,

```

```

auto)
show 1 < int CARD('a mod-ring) using less-imp-of-nat-less nontriv by fastforce
show 0 = (Abs::int ⇒ 'a mod-ring) 0
  by (simp add: Abs-mod-ring-def)
show 1 = (Abs::int ⇒ 'a mod-ring) 1
  by (metis (mono-tags, opaque-lifting) Abs-mod-ring-def of-int-hom.hom-one
of-int-of-int-mod-ring)
fix x y::'a mod-ring
show x + y = Abs ((Rep x + Rep y) mod int CARD('a mod-ring))
  unfolding Abs-mod-ring-def Rep-mod-ring-def by (transfer, auto)
show - x = Abs (- Rep x mod int CARD('a mod-ring))
  unfolding Abs-mod-ring-def Rep-mod-ring-def by (transfer, auto simp add:
zmod-zminus1-eq-if)
show x * y = Abs (Rep x * Rep y mod int CARD('a mod-ring))
  unfolding Abs-mod-ring-def Rep-mod-ring-def by (transfer, auto)
show x - y = Abs ((Rep x - Rep y) mod int CARD('a mod-ring))
  unfolding Abs-mod-ring-def Rep-mod-ring-def by (transfer, auto)
show strict-mono (Rep::'a mod-ring ⇒ int) unfolding Rep-mod-ring-def
  by (rule strict-mono-to-int-mod-ring)
qed
end
end

```

## 9 Generality of the Algorithm to transform from diagonal to Smith normal form

```

theory Admits-SNF-From-Diagonal-Iff-Bezout-Ring
  imports
    Diagonal-To-Smith
    Rings2-Extended
    Smith-Normal-Form-JNF
    Finite-Field-Mod-Type-Connection
begin

```

```

hide-const (open) mat

```

This section provides a formal proof on the generality of the algorithm that transforms a diagonal matrix into its Smith normal form. More concretely, we prove that all diagonal matrices with coefficients in a ring  $R$  admit Smith normal form if and only if  $R$  is a Bézout ring.

Since our algorithm is defined for Bézout rings and for any matrices (including non-square and singular ones), this means that it does not exist another algorithm that performs the transformation in a more abstract structure.

Firstly, we hide some definitions and facts, since we are interested in the ones developed for the *mod-type* class.

```

hide-const (open) Bij-Nat.to-nat Bij-Nat.from-nat Countable.to-nat Countable.from-nat

```

**hide-fact (open)** *Bij-Nat.to-nat-from-nat-id Bij-Nat.to-nat-less-card*

**definition** *admits-SNF-HA*  $(A::'a::\text{comm-ring-1}^{\wedge n}::\{\text{mod-type}\}^{\wedge n}::\{\text{mod-type}\}) =$   
 $(\text{isDiagonal } A$   
 $\longrightarrow (\exists P Q. \text{invertible } ((P::'a::\text{comm-ring-1}^{\wedge n}::\{\text{mod-type}\}^{\wedge n}::\{\text{mod-type}\}))$   
 $\wedge \text{invertible } (Q::'a::\text{comm-ring-1}^{\wedge n}::\{\text{mod-type}\}^{\wedge n}::\{\text{mod-type}\}) \wedge \text{Smith-normal-form}$   
 $(P**A**Q)))$

**definition** *admits-SNF-JNF*  $A = (\text{square-mat } (A::'a::\text{comm-ring-1 mat}) \wedge \text{isDiagonal-mat } A$   
 $\longrightarrow (\exists P Q. P \in \text{carrier-mat } (\text{dim-row } A) (\text{dim-row } A) \wedge Q \in \text{carrier-mat}$   
 $(\text{dim-row } A) (\text{dim-row } A)$   
 $\wedge \text{invertible-mat } P \wedge \text{invertible-mat } Q \wedge \text{Smith-normal-form-mat } (P*A*Q)))$

## 9.1 Proof of the $\Leftarrow$ implication in HA.

**lemma** *exists-f-PAQ-Aii'*:

**fixes**  $A::'a::\{\text{comm-ring-1}\}^{\wedge n}::\{\text{mod-type}\}^{\wedge n}::\{\text{mod-type}\}$   
**assumes** *diag-A: isDiagonal A*  
**shows**  $\exists f. (P**A**Q) \$h i \$h i = (\sum_{i \in (\text{UNIV}::'n \text{ set})}. f i * A \$h i \$h i)$   
**proof** –  
**have**  $rw: (\sum_{ka \in \text{UNIV}}. P \$h i \$h ka * A \$h ka \$h k) = P \$h i \$h k * A \$h k$   
 $\$h k$  **for**  $k$   
**proof** –  
**have**  $(\sum_{ka \in \text{UNIV}}. P \$h i \$h ka * A \$h ka \$h k) = (\sum_{ka \in \{k\}}. P \$h i \$h ka$   
 $* A \$h ka \$h k)$   
**proof** (*rule sum.mono-neutral-right, auto*)  
**fix**  $ia$  **assume**  $P \$h i \$h ia * A \$h ia \$h k \neq 0$   
**hence**  $A \$h ia \$h k \neq 0$  **by** *auto*  
**thus**  $ia = k$  **using** *diag-A unfolding isDiagonal-def* **by** *auto*  
**qed**  
**also have**  $\dots = P \$h i \$h k * A \$h k \$h k$  **by** *auto*  
**finally show** *?thesis* .  
**qed**  
**let**  $?f = \lambda k. (\sum_{ka \in \text{UNIV}}. P \$h i \$h ka) * Q \$h k \$h i$   
**have**  $(P**A**Q) \$h i \$h i = (\sum_{k \in \text{UNIV}}. (\sum_{ka \in \text{UNIV}}. P \$h i \$h ka * A \$h$   
 $ka \$h k) * Q \$h k \$h i)$   
**unfolding** *matrix-matrix-mult-def* **by** *auto*  
**also have**  $\dots = (\sum_{k \in \text{UNIV}}. P \$h i \$h k * Q \$h k \$h i * A \$h k \$h k)$   
**unfolding** *rw*  
**by** (*meson semiring-normalization-rules(16)*)  
**finally show** *?thesis* **by** *auto*  
**qed**

We apply *internalize-sort* to the lemma that we need

**lemmas** *diagonal-to-Smith-PQ-exists-internalize-sort*  
 $= \text{diagonal-to-Smith-PQ-exists}[\text{internalize-sort } 'a :: \text{bezout-ring}]$

We get the  $\Leftarrow$  implication in HA.

**lemma** *bezout-ring-imp-diagonal-admits-SNF*:  
**assumes** of: *OFCLASS('a::comm-ring-1, bezout-ring-class)*  
**shows**  $\forall A::'a \wedge n::\{\text{mod-type}\} \wedge n::\{\text{mod-type}\}. \text{isDiagonal } A$   
 $\longrightarrow (\exists P Q.$   
 $\quad \text{invertible } (P::'a \wedge n::\{\text{mod-type}\} \wedge n::\{\text{mod-type}\}) \wedge$   
 $\quad \text{invertible } (Q::'a \wedge n::\{\text{mod-type}\} \wedge n::\{\text{mod-type}\}) \wedge$   
 $\quad \text{Smith-normal-form } (P**A**Q))$   
**proof** (*rule allI, rule impI*)  
**fix**  $A::'a \wedge n::\{\text{mod-type}\} \wedge n::\{\text{mod-type}\}$   
**assume**  $A: \text{isDiagonal } A$   
**have**  $br: \text{class.bezout-ring } (*) (1::'a) (+) 0 (-) \text{uminus}$   
**by** (*rule OFCLASS-bezout-ring-imp-class-bezout-ring[OF of]*)  
**show**  $\exists P Q.$   
 $\quad \text{invertible } (P::'a \wedge n::\{\text{mod-type}\} \wedge n::\{\text{mod-type}\}) \wedge$   
 $\quad \text{invertible } (Q::'a \wedge n::\{\text{mod-type}\} \wedge n::\{\text{mod-type}\}) \wedge$   
 $\quad \text{Smith-normal-form } (P**A**Q)$  **by** (*rule diagonal-to-Smith-PQ-exists-internalize-sort[OF*  
*br A]*)  
**qed**

## 9.2 Trying to prove the $\implies$ implication in HA.

There is a problem: we need to define a matrix with a concrete dimension, which is not possible in HA (the dimension depends on the number of elements on a set, and Isabelle/HOL does not feature dependent types)

**lemma**  
**assumes**  $\forall A::'a::\text{comm-ring-1} \wedge n::\{\text{mod-type}\} \wedge n::\{\text{mod-type}\}. \text{admits-SNF-HA } A$   
**shows** *OFCLASS('a::comm-ring-1, bezout-ring-class)* **oops**

## 9.3 Proof of the $\implies$ implication in JNF.

**lemma** *exists-f-PAQ-Aii*:  
**assumes**  $\text{diag-A}: \text{isDiagonal-mat } (A::'a::\text{comm-ring-1 mat})$   
**and**  $P: P \in \text{carrier-mat } n \ n$   
**and**  $A: A \in \text{carrier-mat } n \ n$   
**and**  $Q: Q \in \text{carrier-mat } n \ n$   
**and**  $i: i < n$   
**shows**  $\exists f. (P*A*Q) \ \S\S (i, i) = (\sum_{i \in \text{set } (\text{diag-mat } A)}. f \ i * \ i)$   
**proof** –  
**let**  $?xs = \text{diag-mat } A$   
**let**  $?n = \text{length } ?xs$   
**have**  $\text{length-n}: \text{length } (\text{diag-mat } A) = n$   
**by** (*metis A carrier-matD(1) diag-mat-def diff-zero length-map length-upt*)  
**have**  $xs\text{-index}: ?xs \ ! \ i = A \ \S\S (i, i)$  **if**  $i < n$  **for**  $i$   
**by** (*metis (no-types, lifting) add.left-neutral diag-mat-def length-map length-n length-upt nth-map-upt that*)  
**have**  $i\text{-length}: i < \text{length } ?xs$  **using**  $i \ \text{length-n}$  **by** *auto*

**have**  $rw: (\sum ka = 0..<?n. P \ \$\$ (i, ka) * A \ \$\$ (ka, k)) = P \ \$\$ (i, k) * A \ \$\$ (k, k)$   
**if**  $k: k < \text{length } ?xs$  **for**  $k$   
**proof** –  
**have**  $(\sum ka = 0..<?n. P \ \$\$ (i, ka) * A \ \$\$ (ka, k)) = (\sum ka \in \{k\}. P \ \$\$ (i, ka) * A \ \$\$ (ka, k))$   
**by**  $(rule \ \text{sum.mono-neutral-right}, \ \text{auto} \ \text{simp} \ \text{add}: \ k, \ \text{insert} \ \text{diag-A} \ A \ \text{length-n} \ \text{that}, \ \text{unfold} \ \text{isDiagonal-mat-def}, \ \text{fastforce})$   
**also have**  $\dots = P \ \$\$ (i, k) * A \ \$\$ (k, k)$  **by**  $\text{auto}$   
**finally show**  $?thesis$  .  
**qed**  
**let**  $?positions\text{-of} = \lambda x. \{i. A \ \$\$ (i, i) = x \wedge i < \text{length } ?xs\}$   
**let**  $?T = \text{set } ?xs$   
**let**  $?S = \{0..<?n\}$   
**let**  $?f = \lambda x. (\sum k \in \{i. A \ \$\$ (i, i) = x \wedge i < \text{length } (\text{diag-mat } A)\}. P \ \$\$ (i, k) * Q \ \$\$ (k, i))$   
**let**  $?g = (\lambda k. P \ \$\$ (i, k) * Q \ \$\$ (k, i) * A \ \$\$ (k, k))$   
**have**  $UNION\text{-positions-of}: \bigcup (?positions\text{-of } ' ?T) = ?S$  **unfolding**  $\text{diag-mat-def}$   
**by**  $\text{auto}$   
**have**  $(P * A * Q) \ \$\$ (i, i) = (\sum ia = 0..<?n. \ \text{Matrix.row } (\text{Matrix.mat } ?n \ ?n \ (\lambda(i, j). \sum ia = 0..<?n. \ \text{Matrix.row } P \ i \ \$v \ ia * \ \text{col } A \ j \ \$v \ ia)) \ i \ \$v \ ia * \ \text{col } Q \ i \ \$v \ ia)$   
**unfolding**  $\text{times-mat-def} \ \text{scalar-prod-def}$   
**using**  $P \ Q \ i\text{-length} \ \text{length-n} \ A$  **by**  $\text{auto}$   
**also have**  $\dots = (\sum k = 0..<?n. (\sum ka = 0..<?n. P \ \$\$ (i, ka) * A \ \$\$ (ka, k)) * Q \ \$\$ (k, i))$   
**proof**  $(rule \ \text{sum.cong}, \ \text{auto})$   
**fix**  $x$  **assume**  $x: x < \text{length } ?xs$   
**have**  $rw\text{-col}Q: \text{col } Q \ i \ \$v \ x = Q \ \$\$ (x, i)$   
**using**  $Q \ i\text{-length} \ x \ \text{length-n} \ A$  **by**  $\text{auto}$   
**have**  $rw2: \text{Matrix.row } (\text{Matrix.mat } ?n \ ?n \ (\lambda(i, j). \sum ia = 0..<\text{length } ?xs. \ \text{Matrix.row } P \ i \ \$v \ ia * \ \text{col } A \ j \ \$v \ ia)) \ i \ \$v \ x$   
 $= (\sum ia = 0..<\text{length } ?xs. \ \text{Matrix.row } P \ i \ \$v \ ia * \ \text{col } A \ x \ \$v \ ia)$   
**unfolding**  $\text{row-mat}[OF \ i\text{-length}]$  **unfolding**  $\text{index-vec}[OF \ x]$  **by**  $\text{auto}$   
**also have**  $\dots = (\sum ia = 0..<\text{length } ?xs. P \ \$\$ (i, ia) * A \ \$\$ (ia, x))$   
**by**  $(rule \ \text{sum.cong}, \ \text{insert } P \ i\text{-length} \ x \ \text{length-n} \ A, \ \text{auto})$   
**finally show**  $\text{Matrix.row } (\text{Matrix.mat } ?n \ ?n \ (\lambda(i, j). \sum ia = 0..<?n. \ \text{Matrix.row } P \ i \ \$v \ ia * \ \text{col } A \ j \ \$v \ ia)) \ i \ \$v \ x * \ \text{col } Q \ i \ \$v \ x$   
 $= (\sum ka = 0..<?n. P \ \$\$ (i, ka) * A \ \$\$ (ka, x)) * Q \ \$\$ (x, i)$  **unfolding**  
 $rw\text{-col}Q$  **by**  $\text{auto}$   
**qed**  
**also have**  $\dots = (\sum k = 0..<?n. P \ \$\$ (i, k) * Q \ \$\$ (k, i) * A \ \$\$ (k, k))$   
**by**  $(\text{smt} \ (\text{verit}) \ rw \ \text{semiring-normalization-rules}(16) \ \text{sum.ivl-cong})$   
**also have**  $\dots = \text{sum } ?g \ (\bigcup (?positions\text{-of } ' ?T))$   
**using**  $UNION\text{-positions-of}$  **by**  $\text{auto}$   
**also have**  $\dots = (\sum x \in ?T. \text{sum } ?g \ (?positions\text{-of } x))$   
**by**  $(rule \ \text{sum.UNION-disjoint}, \ \text{auto})$



**also have** ... =  $(\sum x \in \text{set } (\text{diag-mat } A)). (\sum k \in \{i. A \text{ \textit{\$} \$ (i, i) = x} \wedge i < \text{length } (\text{diag-mat } A)\}.$   
 $P \text{ \textit{\$} \$ (i, k) * } Q \text{ \textit{\$} \$ (k, i) * } x$   
**by** (*rule sum.cong, auto simp add: Groups-Big.sum-distrib-right*)  
**finally show** ?thesis **by** *auto*  
**qed**

Proof of the  $\implies$  implication in JNF.

**lemma** *diagonal-admits-SNF-imp-bezout-ring-JNF:*

**assumes** *admits-SNF:  $\forall A n. (A::'a \text{ mat}) \in \text{carrier-mat } n \ n \wedge \text{isDiagonal-mat } A \longrightarrow (\exists P Q. P \in \text{carrier-mat } n \ n \wedge Q \in \text{carrier-mat } n \ n \wedge \text{invertible-mat } P \wedge \text{invertible-mat } Q$*

$\wedge \text{Smith-normal-form-mat } (P * A * Q)$

**shows** *OFCLASS('a::comm-ring-1, bezout-ring-class)*

**proof** (*rule all-fin-gen-ideals-are-principal-imp-bezout, rule allI, rule impI*)

**fix** *I::'a set*

**assume** *fin: finitely-generated-ideal I*

**obtain** *S where ig-S: ideal-generated S = I and fin-S: finite S*

**using** *fin unfolding finitely-generated-ideal-def* **by** *auto*

**show** *principal-ideal I*

**proof** (*cases S = {}*)

**case** *True*

**then show** ?thesis

**by** (*metis ideal-generated-0 ideal-generated-empty ig-S principal-ideal-def*)

**next**

**case** *False*

**obtain** *xs where set-xs: set xs = S and d: distinct xs*

**using** *finite-distinct-list[OF fin-S]* **by** *blast*

**hence** *length-eq-card: length xs = card S* **using** *distinct-card* **by** *force*

**let** *?n = length xs*

**let** *?A = Matrix.mat ?n ?n ( $\lambda(a,b). \text{if } a = b \text{ then } xs!a \text{ else } 0$ )*

**have** *A-carrier: ?A  $\in$  carrier-mat ?n ?n* **by** *auto*

**have** *diag-A: isDiagonal-mat ?A* **unfolding** *isDiagonal-mat-def* **by** *auto*

**have** *set-xs-eq: set xs = {?A \textit{\\$} \\$ (i, i) | i. i < dim-row ?A}*

**by** (*auto, smt (verit) case-prod-conv d distinct-Ex1 index-mat(1)*)

**have** *set-xs-diag-mat: set xs = set (diag-mat ?A)*

**using** *set-xs-eq unfolding diag-mat-def* **by** *auto*

**obtain** *P Q where P: P  $\in$  carrier-mat ?n ?n*

**and** *Q: Q  $\in$  carrier-mat ?n ?n and inv-P: invertible-mat P and inv-Q: invertible-mat Q*

**and** *SNF-PAQ: Smith-normal-form-mat (P \* ?A \* Q)*

**using** *admits-SNF A-carrier diag-A* **by** *blast*

**define** *ys where ys-def: ys = diag-mat (P \* ?A \* Q)*

**have** *ys:  $\forall i < ?n. ys ! i = (P * ?A * Q) \text{ \textit{\$} \$ (i, i)}$*  **using** *P* **by** (*auto simp add: ys-def diag-mat-def*)

**have** *length-ys: length ys = ?n* **unfolding** *ys-def*

**by** (*metis (no-types, lifting) P carrier-matD(1) diag-mat-def*

*index-mult-mat(2) length-map map-nth*)

**have** *n0: ?n > 0* **using** *False set-xs* **by** *blast*

```

have set-ys-diag-mat: set ys = set (diag-mat (P*?A*Q)) using ys-def by auto
let ?i = ys ! 0
have dvd-all:  $\forall a \in \text{set } ys. ?i \text{ dvd } a$ 
proof
  fix a assume a:  $a \in \text{set } ys$ 
  obtain j where ys-j-a:  $ys ! j = a$  and jn:  $j < ?n$  by (metis a in-set-conv-nth
length-ys)
  have jP:  $j < \text{dim-row } P$  using jn P by auto
  have jQ:  $j < \text{dim-col } Q$  using jn Q by auto
  have (P*?A*Q)$(0,0) dvd (P*?A*Q)$(j,j)
    by (rule SNF-first-divides[OF SNF-PAQ], auto simp add: jP jQ)
  thus  $ys ! 0 \text{ dvd } a$  using ys length-ys ys-j-a jn n0 by auto
qed
have ideal-generated S = ideal-generated (set xs) using set-xs by simp
also have ... = ideal-generated (set ys)
proof
  show ideal-generated (set xs)  $\subseteq$  ideal-generated (set ys)
  proof (rule ideal-generated-subset2, rule ballI)
    fix b assume b:  $b \in \text{set } xs$ 
    obtain i where b-A-ii:  $b = ?A \ \$\$ (i,i)$  and i-length:  $i < \text{length } xs$ 
    using b set-xs-eq by auto
    obtain P' where inverts-mat-P':  $\text{inverts-mat } P \ P' \wedge \text{inverts-mat } P' \ P$ 
    using inv-P unfolding invertible-mat-def by auto
    have P':  $P' \in \text{carrier-mat } ?n \ ?n$ 
    using inverts-mat-P'
    unfolding carrier-mat-def inverts-mat-def
    by (auto,metis P carrier-matD index-mult-mat(3) one-carrier-mat)+
    obtain Q' where inverts-mat-Q':  $\text{inverts-mat } Q \ Q' \wedge \text{inverts-mat } Q' \ Q$ 
    using inv-Q unfolding invertible-mat-def by auto
    have Q':  $Q' \in \text{carrier-mat } ?n \ ?n$ 
    using inverts-mat-Q'
    unfolding carrier-mat-def inverts-mat-def
    by (auto,metis Q carrier-matD index-mult-mat(3) one-carrier-mat)+
    have rw-PAQ:  $(P'*(P*?A*Q)*Q') \ \$\$ (i, i) = ?A \ \$\$ (i,i)$ 
    using inv-P'PAQQ'[OF A-carrier P - - Q P' Q'] inverts-mat-P' in-
verts-mat-Q' by auto
    have diag-PAQ:  $\text{isDiagonal-mat } (P*?A*Q)$ 
    using SNF-PAQ unfolding Smith-normal-form-mat-def by auto
    have PAQ-carrier:  $(P*?A*Q) \in \text{carrier-mat } ?n \ ?n$  using P Q by auto
    obtain f where f:  $(P'*(P*?A*Q)*Q') \ \$\$ (i, i) = (\sum_{i \in \text{set } (diag-mat}
(P*?A*Q)). f i * i)$ 
    using exists-f-PAQ-Aii[OF diag-PAQ P' PAQ-carrier Q' i-length] by auto
    hence  $?A \ \$\$ (i,i) = (\sum_{i \in \text{set } (diag-mat } (P*?A*Q)). f i * i)$  unfolding
rw-PAQ .
    thus  $b \in \text{ideal-generated } (\text{set } ys)$ 
    unfolding ideal-explicit using set-ys-diag-mat b-A-ii by auto
  qed
  show ideal-generated (set ys)  $\subseteq$  ideal-generated (set xs)
  proof (rule ideal-generated-subset2, rule ballI)

```

```

fix b assume b: b ∈ set ys
have d: distinct (diag-mat ?A)
  by (metis (no-types, lifting) A-carrier card-distinct carrier-matD(1)
diag-mat-def
length-eq-card length-map map-nth set-xs set-xs-diag-mat)
obtain i where b-PAQ-ii: (P*?A*Q) $$ (i, i) = b and i-length: i < length
xs using b ys
  by (metis (no-types, lifting) in-set-conv-nth length-ys)
obtain f where (P * ?A * Q) $$ (i, i) = (∑ i ∈ set (diag-mat ?A). f i * i)
  using exists-f-PAQ-Aii[OF diag-A P - Q i-length] by auto
  thus b ∈ ideal-generated (set xs)
  using b-PAQ-ii unfolding set-xs-diag-mat ideal-explicit by auto
qed
qed
also have ... = ideal-generated (set ys - (set ys - {ys!0}))
proof (rule ideal-generated-dvd-eq-diff-set)
  show ?i ∈ set ys using n0
  by (simp add: length-ys)
  show ?i ∉ set ys - {?i} by auto
  show ∀ j ∈ set ys - {?i}. ?i dvd j using dvd-all by auto
  show finite (set ys - {?i}) by auto
qed
also have ... = ideal-generated {?i}
by (metis Diff-cancel Diff-not-in insert-Diff insert-Diff-if length-ys n0 nth-mem)
  finally show principal-ideal I unfolding principal-ideal-def using ig-S by
auto
  qed
qed

```

**corollary** diagonal-admits-SNF-imp-bezout-ring-JNF-alt:

```

assumes admits-SNF: ∀ A. square-mat (A::'a mat) ∧ isDiagonal-mat A
→ (∃ P Q. P ∈ carrier-mat (dim-row A) (dim-row A)
  ∧ Q ∈ carrier-mat (dim-row A) (dim-row A) ∧ invertible-mat P ∧ invertible-mat
Q
  ∧ Smith-normal-form-mat (P*A*Q))
shows OFCLASS('a::comm-ring-1, bezout-ring-class)
proof (rule diagonal-admits-SNF-imp-bezout-ring-JNF, rule allI, rule allI, rule
impI)
  fix A::'a mat and n assume A: A ∈ carrier-mat n n ∧ isDiagonal-mat A
  have square-mat A using A by auto
  thus ∃ P Q. P ∈ carrier-mat n n ∧ Q ∈ carrier-mat n n
  ∧ invertible-mat P ∧ invertible-mat Q ∧ Smith-normal-form-mat (P * A * Q)
  using A admits-SNF by blast
qed

```

## 9.4 Trying to transfer the $\implies$ implication to HA.

We first hide some constants defined in *Mod-Type-Connect* in order to use the ones presented in *Perron-Frobenius.HMA-Connect* by default.

**context**

**includes** *lifting-syntax*

**begin**

**lemma** *to-nat-mod-type-Bij-Nat*:

**fixes** *a::'n::mod-type*

**obtains** *b::'n* **where** *mod-type-class.to-nat a = Bij-Nat.to-nat b*

**using** *Bij-Nat.to-nat-from-nat-id mod-type-class.to-nat-less-card* **by** *metis*

**lemma** *inj-on-Bij-nat-from-nat*: *inj-on (Bij-Nat.from-nat::nat  $\Rightarrow$  'a) {0..*CARD('a::finite)}**

**by** (*auto simp add: inj-on-def Bij-Nat.from-nat-def length-univ-list-card nth-eq-iff-index-eq univ-list(1)*)

This lemma only holds if *a* and *b* have the same type. Otherwise, it is possible that *Bij-Nat.to-nat a = Bij-Nat.to-nat b*

**lemma** *Bij-Nat-to-nat-neq*:

**fixes** *a b ::'n::mod-type*

**assumes** *to-nat a  $\neq$  to-nat b*

**shows** *Bij-Nat.to-nat a  $\neq$  Bij-Nat.to-nat b*

**using** *assms to-nat-inj* **by** *blast*

The following proof (a transfer rule for diagonal matrices) is weird, since it does not hold *Bij-Nat.to-nat a = mod-type-class.to-nat a*.

At first, it seems possible to obtain the element *a'* that satisfies *Bij-Nat.to-nat a' = mod-type-class.to-nat a* and then continue with the proof, but then we cannot prove *HMA-I (Bij-Nat.to-nat a') a*.

This means that we must use the previous lemma *Bij-Nat-to-nat-neq*, but this imposes the matrix to be square.

**lemma** *HMA-isDiagonal[transfer-rule]*: (*HMA-M  $\implies$  (=)*)

*isDiagonal-mat (isDiagonal::('a::{zero})<sup>'n::{mod-type}</sup><sup>'n::{mod-type}</sup>  $\Rightarrow$  bool))*

**proof** (*intro rel-funI, goal-cases*)

**case** (*1 x y*)

**note** *rel-xy [transfer-rule] = 1*

**have** *y \$h a \$h b = 0*

**if** *all0:  $\forall i j. i \neq j \wedge i < \dim\text{-row } x \wedge j < \dim\text{-col } x \longrightarrow x \$\$ (i, j) = 0$*

**and** *a-noteq-b: a  $\neq$  b for a::'n and b::'n*

**proof** –

**have** *to-nat a  $\neq$  to-nat b using a-noteq-b by auto*

**hence** *distinct: Bij-Nat.to-nat a  $\neq$  Bij-Nat.to-nat b by (rule Bij-Nat-to-nat-neq)*

**moreover** **have** *Bij-Nat.to-nat a < dim-row x and Bij-Nat.to-nat b < dim-col*

*x*

**using** *Bij-Nat.to-nat-less-card dim-row-transfer-rule rel-xy dim-col-transfer-rule*

```

    by fastforce+
    ultimately have b: x $$ (Bij-Nat.to-nat a, Bij-Nat.to-nat b) = 0 using all0
  by auto
    have [transfer-rule]: HMA-I (Bij-Nat.to-nat a) a by (simp add: HMA-I-def)
    have [transfer-rule]: HMA-I (Bij-Nat.to-nat b) b by (simp add: HMA-I-def)
    have index-hma y a b = 0 using b by (transfer', auto)
    thus ?thesis unfolding index-hma-def .
  qed
  moreover have x $$ (i, j) = 0
    if all0:  $\forall a b. a \neq b \longrightarrow y \$h a \$h b = 0$ 
    and ij:  $i \neq j$  and i:  $i < \dim\text{-row } x$  and j:  $j < \dim\text{-col } x$  for i j
  proof -
    have i-n:  $i < \text{CARD}('n)$  and j-n:  $j < \text{CARD}('n)$ 
    using i j rel-xy dim-row-transfer-rule dim-col-transfer-rule
    by fastforce+
    let ?i' = Bij-Nat.from-nat i::'n
    let ?j' = Bij-Nat.from-nat j::'n
    have i'-neq-j':  $?i' \neq ?j'$  using ij i-n j-n Bij-Nat.from-nat-inj by blast
    hence y0: index-hma y ?i' ?j' = 0 using all0 unfolding index-hma-def by
  auto
    have [transfer-rule]: HMA-I i ?i' unfolding HMA-I-def
    by (simp add: Bij-Nat.to-nat-from-nat-id i-n)
    have [transfer-rule]: HMA-I j ?j' unfolding HMA-I-def
    by (simp add: Bij-Nat.to-nat-from-nat-id j-n)
    show ?thesis using y0 by (transfer, auto)
  qed
  ultimately show ?case unfolding isDiagonal-mat-def isDiagonal-def
  by auto
  qed

```

Indeed, we can prove the transfer rules with the new connection based on the *mod-type* class, which was developed in the *Mod-Type-Connect* file

This is the same lemma as the one presented above, but now using the *to-nat* function defined in the *mod-type* class and then we can prove it for non-square matrices, which is very useful since our algorithms are not restricted to square matrices.

**lemma** *HMA-isDiagonal-Mod-Type*[transfer-rule]: (*Mod-Type-Connect.HMA-M*  $\implies$   $(=)$ )

*isDiagonal-mat (isDiagonal::('a::{\text{zero}} ^'n::{\text{mod-type}} ^'m::{\text{mod-type}} => bool))*

**proof** (*intro rel-funI, goal-cases*)

**case** (1 x y)

**note** rel-xy [transfer-rule] = 1

**have** y \$h a \$h b = 0

**if** all0:  $\forall i j. i \neq j \wedge i < \dim\text{-row } x \wedge j < \dim\text{-col } x \longrightarrow x \$\$ (i, j) = 0$

**and** a-noteq-b: *to-nat a*  $\neq$  *to-nat b* for a::'m and b::'n

**proof** -

**have** distinct: *to-nat a*  $\neq$  *to-nat b* using a-noteq-b by auto

**moreover** have *to-nat a* < *dim-row x* and *to-nat b* < *dim-col x*

```

    using to-nat-less-card rel-xy
    using Mod-Type-Connect.dim-row-transfer-rule Mod-Type-Connect.dim-col-transfer-rule

    by fastforce+
    ultimately have b: x $$ (to-nat a, to-nat b) = 0 using all0 by auto
    have [transfer-rule]: Mod-Type-Connect.HMA-I (to-nat a) a
      by (simp add: Mod-Type-Connect.HMA-I-def)
    have [transfer-rule]: Mod-Type-Connect.HMA-I (to-nat b) b
      by (simp add: Mod-Type-Connect.HMA-I-def)
    have index-hma y a b = 0 using b by (transfer', auto)
    thus ?thesis unfolding index-hma-def .
  qed
  moreover have x $$ (i, j) = 0
    if all0:  $\forall a b. \text{to-nat } a \neq \text{to-nat } b \longrightarrow y \$h a \$h b = 0$ 
    and ij:  $i \neq j$  and i:  $i < \text{dim-row } x$  and j:  $j < \text{dim-col } x$  for i j
  proof -
    have i-n:  $i < \text{CARD}('m)$ 
      using i rel-xy by (simp add: Mod-Type-Connect.dim-row-transfer-rule)
    have j-n:  $j < \text{CARD}('n)$ 
      using j rel-xy by (simp add: Mod-Type-Connect.dim-col-transfer-rule)
    let ?i' = from-nat i::'m
    let ?j' = from-nat j::'n
    have to-nat ?i'  $\neq$  to-nat ?j'
      by (simp add: i-n ij j-n mod-type-class.to-nat-from-nat-id)
    hence y0: index-hma y ?i' ?j' = 0 using all0 unfolding index-hma-def by
  auto
    have [transfer-rule]: Mod-Type-Connect.HMA-I i ?i'
      unfolding Mod-Type-Connect.HMA-I-def
      by (simp add: to-nat-from-nat-id i-n)
    have [transfer-rule]: Mod-Type-Connect.HMA-I j ?j'
      unfolding Mod-Type-Connect.HMA-I-def
      by (simp add: to-nat-from-nat-id j-n)
    show ?thesis using y0 by (transfer, auto)
  qed
  ultimately show ?case unfolding isDiagonal-mat-def isDiagonal-def
    by auto
  qed

```

We state the transfer rule using the relations developed in the new bride of the file *Mod-Type-Connect*.

**lemma** *HMA-SNF*[*transfer-rule*]:  $(\text{Mod-Type-Connect.HMA-M} \implies (=)) \text{Smith-normal-form-mat}$

$(\text{Smith-normal-form}::'a::\{\text{comm-ring-1}\}^n::\{\text{mod-type}\}^m::\{\text{mod-type}\} \implies \text{bool})$

**proof** (*intro rel-funI, goal-cases*)

**case** (1 x y)

**note** *rel-xy*[*transfer-rule*] = 1

**have**  $y \$h a \$h b \text{ dvd } y \$h (a + 1) \$h (b + 1)$

**if** *SNF-condition*:  $\forall a. \text{Suc } a < \text{dim-row } x \wedge \text{Suc } a < \text{dim-col } x$   
 $\longrightarrow x \$\$ (a, a) \text{ dvd } x \$\$ (\text{Suc } a, \text{Suc } a)$

**and**  $a1: \text{Suc } (\text{to-nat } a) < \text{nrows } y$  **and**  $a2: \text{Suc } (\text{to-nat } b) < \text{ncols } y$   
**and**  $ab: \text{to-nat } a = \text{to-nat } b$  **for**  $a::'m$  **and**  $b::'n$

**proof** –

**have**  $[\text{transfer-rule}]: \text{Mod-Type-Connect.HMA-I } (\text{to-nat } a) a$   
**by**  $(\text{simp add: Mod-Type-Connect.HMA-I-def})$   
**have**  $[\text{transfer-rule}]: \text{Mod-Type-Connect.HMA-I } (\text{to-nat } (a+1)) (a+1)$   
**by**  $(\text{simp add: Mod-Type-Connect.HMA-I-def})$   
**have**  $[\text{transfer-rule}]: \text{Mod-Type-Connect.HMA-I } (\text{to-nat } b) b$   
**by**  $(\text{simp add: Mod-Type-Connect.HMA-I-def})$   
**have**  $[\text{transfer-rule}]: \text{Mod-Type-Connect.HMA-I } (\text{to-nat } (b+1)) (b+1)$   
**by**  $(\text{simp add: Mod-Type-Connect.HMA-I-def})$   
**have**  $\text{Suc } (\text{to-nat } a) < \text{dim-row } x$  **using**  $a1$   
**by**  $(\text{metis Mod-Type-Connect.dim-row-transfer-rule nrows-def rel-xy})$   
**moreover have**  $\text{Suc } (\text{to-nat } b) < \text{dim-col } x$   
**by**  $(\text{metis Mod-Type-Connect.dim-col-transfer-rule a2 ncols-def rel-xy})$   
**ultimately have**  $x \text{ $$$ } (\text{to-nat } a, \text{to-nat } b) \text{ dvd } x \text{ $$$ } (\text{Suc } (\text{to-nat } a), \text{Suc } (\text{to-nat } b))$   
**using**  $\text{SNF-condition}$  **by**  $(\text{simp add: ab})$   
**also have**  $\dots = x \text{ $$$ } (\text{to-nat } (a+1), \text{to-nat } (b+1))$   
**by**  $(\text{metis Suc-eq-plus1 a1 a2 nrows-def ncols-def to-nat-suc})$   
**finally have**  $\text{SNF-cond}: x \text{ $$$ } (\text{to-nat } a, \text{to-nat } b) \text{ dvd } x \text{ $$$ } (\text{to-nat } (a + 1), \text{to-nat } (b + 1))$  .  
**have**  $x \text{ $$$ } (\text{to-nat } a, \text{to-nat } b) = \text{index-hma } y a b$  **by**  $(\text{transfer, simp})$   
**moreover have**  $x \text{ $$$ } (\text{to-nat } (a + 1), \text{to-nat } (b + 1)) = \text{index-hma } y (a+1) (b+1)$   
**by**  $(\text{transfer, simp})$   
**ultimately show**  $?thesis$  **using**  $\text{SNF-cond}$  **unfolding**  $\text{index-hma-def}$  **by**  $\text{auto}$   
**qed**  
**moreover have**  $x \text{ $$$ } (a, a) \text{ dvd } x \text{ $$$ } (\text{Suc } a, \text{Suc } a)$   
**if**  $\text{SNF}: \forall a b. \text{to-nat } a = \text{to-nat } b \wedge \text{Suc } (\text{to-nat } a) < \text{nrows } y \wedge \text{Suc } (\text{to-nat } b) < \text{ncols } y$   
 $\rightarrow y \text{ $$$ } a \text{ $$$ } b \text{ dvd } y \text{ $$$ } (a + 1) \text{ $$$ } (b + 1)$   
**and**  $a1: \text{Suc } a < \text{dim-row } x$  **and**  $a2: \text{Suc } a < \text{dim-col } x$  **for**  $a$

**proof** –

**have**  $\text{dim-row-CARD}: \text{dim-row } x = \text{CARD}('m)$   
**using**  $\text{Mod-Type-Connect.dim-row-transfer-rule rel-xy}$  **by**  $\text{blast}$   
**have**  $\text{dim-col-CARD}: \text{dim-col } x = \text{CARD}('n)$   
**using**  $\text{Mod-Type-Connect.dim-col-transfer-rule rel-xy}$  **by**  $\text{blast}$   
**let**  $?a' = \text{from-nat } a::'m$   
**let**  $?b' = \text{from-nat } a::'n$   
**have**  $\text{Suc-a-less-CARD}: a + 1 < \text{CARD}('m)$  **using**  $a1$   $\text{dim-row-CARD}$  **by**  $\text{auto}$   
**have**  $\text{Suc-b-less-CARD}: a + 1 < \text{CARD}('n)$  **using**  $a2$   
**by**  $(\text{metis Mod-Type-Connect.dim-col-transfer-rule Suc-eq-plus1 rel-xy})$   
**have**  $aa'[\text{transfer-rule}]: \text{Mod-Type-Connect.HMA-I } a ?a'$   
**unfolding**  $\text{Mod-Type-Connect.HMA-I-def}$   
**by**  $(\text{metis Suc-a-less-CARD add-lessD1 mod-type-class.to-nat-from-nat-id})$   
**have**  $[\text{transfer-rule}]: \text{Mod-Type-Connect.HMA-I } (a+1) (?a' + 1)$   
**unfolding**  $\text{Mod-Type-Connect.HMA-I-def}$   
**unfolding**  $\text{from-nat-suc[symmetric]}$  **using**  $\text{to-nat-from-nat-id[OF Suc-a-less-CARD]}$

```

by auto
  have ab'[transfer-rule]: Mod-Type-Connect.HMA-I a ?b'
    unfolding Mod-Type-Connect.HMA-I-def
    by (metis Suc-b-less-CARD add-lessD1 mod-type-class.to-nat-from-nat-id)
  have [transfer-rule]: Mod-Type-Connect.HMA-I (a+1) (?b' + 1)
    unfolding Mod-Type-Connect.HMA-I-def
    unfolding from-nat-suc[symmetric] using to-nat-from-nat-id[OF Suc-b-less-CARD]
by auto
  have aa'1: a = to-nat ?a' using aa' by (simp add: Mod-Type-Connect.HMA-I-def)
  have ab'1: a = to-nat ?b' using ab' by (simp add: Mod-Type-Connect.HMA-I-def)
  have Suc (to-nat ?a') < nrows y using a1 dim-row-CARD
    by (simp add: mod-type-class.to-nat-from-nat-id nrows-def)
  moreover have Suc (to-nat ?b') < ncols y using a2 dim-col-CARD
    by (simp add: mod-type-class.to-nat-from-nat-id ncols-def)
  ultimately have SNF': y $h ?a' $h ?b' dvd y $h (?a' + 1) $h (?b' + 1)
    using SNF ab'1 aa'1 by auto
  have index-hma y ?a' ?b' = x $$ (a, a) by (transfer, simp)
  moreover have index-hma y (?a'+1) (?b'+1) = x $$ (a+1, a+1) by
(transfer, simp)
  ultimately show ?thesis using SNF' unfolding index-hma-def by auto
qed
ultimately show ?case unfolding Smith-normal-form-mat-def Smith-normal-form-def
using rel-xy by (auto) (transfer', auto)+
qed

```

```

lemma HMA-admits-SNF [transfer-rule]:
  ((Mod-Type-Connect.HMA-M :: - => 'a :: comm-ring-1 ^ 'n::{mod-type} ^ 'n::{mod-type}
=> -) ==> (=))
  admits-SNF-JNF admits-SNF-HA
proof (intro rel-funI, goal-cases)
  case (1 x y)
  note [transfer-rule] = this
  hence id: dim-row x = CARD('n) by (auto simp: Mod-Type-Connect.HMA-M-def)
  then show ?case unfolding admits-SNF-JNF-def admits-SNF-HA-def
    by (transfer, auto, metis 1 Mod-Type-Connect.dim-col-transfer-rule)
qed
end

```

Here we have a problem when trying to apply local type definitions

```

lemma diagonal-admits-SNF-imp-bezout-ring:
  assumes admits-SNF:  $\forall A::'a::comm-ring-1 \wedge 'n::\{mod-type\} \wedge 'n::\{mod-type\}. is-$ 
    Diagonal A
   $\longrightarrow (\exists P Q. invertible (P::'a::comm-ring-1 \wedge 'n::\{mod-type\} \wedge 'n::\{mod-type\})$ 
     $\wedge invertible (Q::'a::comm-ring-1 \wedge 'n::\{mod-type\} \wedge 'n::\{mod-type\})$ 
     $\wedge Smith-normal-form (P**A**Q))$ 
  shows OFCLASS('a::comm-ring-1, bezout-ring-class)
proof (rule diagonal-admits-SNF-imp-bezout-ring-JNF, auto)

```



```

fix A::'a mat and n
  assume A: A ∈ carrier-mat n n and diag-A: isDiagonal-mat A
  have a: ∀ A::'a::comm-ring-1^n::{mod-type}^n::{mod-type}. admits-SNF-HA A

  using admits-SNF unfolding admits-SNF-HA-def .
  have JNF: ∀ (A::'a mat) ∈ carrier-mat CARD('n) CARD('n). admits-SNF-JNF
A

```

```

proof
  fix A::'a mat
  assume A: A ∈ carrier-mat CARD('n) CARD('n)
  let ?B = (Mod-Type-Connect.to-hmam A::'a::comm-ring-1^n::{mod-type}^n::{mod-type})
  have [transfer-rule]: Mod-Type-Connect.HMA-M A ?B
  using A unfolding Mod-Type-Connect.HMA-M-def by auto
  have b: admits-SNF-HA ?B using a by auto
  show admits-SNF-JNF A using b by transfer
qed

```

```

thus ∃ P. P ∈ carrier-mat n n ∧
  (∃ Q. Q ∈ carrier-mat n n ∧ invertible-mat P
  ∧ invertible-mat Q ∧ Smith-normal-form-mat (P * A * Q))
  using JNF A diag-A unfolding admits-SNF-JNF-def unfolding square-mat.simps
oops

```

This means that the  $\implies$  implication cannot be proven in HA, since we cannot quantify over type variables in Isabelle/HOL. We then prove both implications in JNF.

## 9.5 Transferring the $\Leftarrow$ implication from HA to JNF using transfer rules and local type definitions

```

lemma bezout-ring-imp-diagonal-admits-SNF-mod-ring:
  assumes of: OFCLASS('a::comm-ring-1, bezout-ring-class)
  shows ∀ A::'a^n::nontriv mod-ring^n::nontriv mod-ring. isDiagonal A
  → (∃ P Q.
    invertible (P::'a^n::nontriv mod-ring^n::nontriv mod-ring) ∧
    invertible (Q::'a^n::nontriv mod-ring^n::nontriv mod-ring) ∧
    Smith-normal-form (P**A**Q))
  using bezout-ring-imp-diagonal-admits-SNF[OF assms] by auto

```

```

lemma bezout-ring-imp-diagonal-admits-SNF-mod-ring-admits:
  assumes of: class.bezout-ring (*) (1::'a::comm-ring-1) (+) 0 (-) uminus
  shows ∀ A::'a^n::nontriv mod-ring^n::nontriv mod-ring. admits-SNF-HA A
  using bezout-ring-imp-diagonal-admits-SNF
  [OF bezout-ring.intro-of-class[OF of]]
  unfolding admits-SNF-HA-def by auto

```

I start here to apply local type definitions

**context**

```

fixes  $p::nat$ 
assumes  $local\text{-}typedef: \exists (Rep :: ('b \Rightarrow int)) Abs. type\text{-}definition\ Rep\ Abs\ \{0..<p$ 
 $:: int\}$ 
and  $p: p>1$ 
begin

```

```

lemma  $type\text{-}to\text{-}set:$ 
shows  $class.nontriv\ TYPE('b)\ (is\ ?a)\ \mathbf{and}\ p=CARD('b)\ (is\ ?b)$ 
proof  $-$ 
from  $local\text{-}typedef$  obtain  $Rep::('b \Rightarrow int)$  and  $Abs$ 
where  $t: type\text{-}definition\ Rep\ Abs\ \{0..<p :: int\}$  by  $auto$ 
have  $card\ (UNIV :: 'b\ set) = card\ \{0..<p\}$  using  $t\ type\text{-}definition.card$  by
 $fastforce$ 
also have  $... = p$  by  $auto$ 
finally show  $?b ..$ 
then show  $?a$  unfolding  $class.nontriv\text{-}def$  using  $p$  by  $auto$ 
qed

```

I transfer the lemma from HA to JNF, substituting  $CARD('n)$  by  $p$ . I apply  $internalize\text{-}sort$  to  $'n$  and get rid of the  $nontriv$  restriction.

```

lemma  $bezout\text{-}ring\text{-}imp\text{-}diagonal\text{-}admits\text{-}SNF\text{-}mod\text{-}ring\text{-}admits\text{-}aux:$ 
assumes  $class.bezout\text{-}ring\ (*)\ (1::'a::comm\text{-}ring\text{-}1)\ (+)\ 0\ (-)\ uminus$ 
shows  $Ball\ \{A::'a::comm\text{-}ring\text{-}1\ mat.\ A \in carrier\text{-}mat\ p\ p\}$   $admits\text{-}SNF\text{-}JNF$ 
using  $bezout\text{-}ring\text{-}imp\text{-}diagonal\text{-}admits\text{-}SNF\text{-}mod\text{-}ring\text{-}admits[untransferred,\ un-$ 
 $folded\ CARD\text{-}mod\text{-}ring,$ 
 $internalize\text{-}sort\ 'n::nontriv,\ \mathbf{where}\ ?'a='b]$ 
unfolding  $type\text{-}to\text{-}set(2)[symmetric]$  using  $type\text{-}to\text{-}set(1)\ assms$  by  $auto$ 
end

```

The  $\Leftarrow$  implication in JNF

Since  $nontriv$  imposes the type to have more than one element, the cases  $n = 0$  ( $A \in carrier\text{-}mat\ 0\ 0$ ) and  $n = 1$  ( $A \in carrier\text{-}mat\ 1\ 1$ ) must be treated separately.

```

lemma  $bezout\text{-}ring\text{-}imp\text{-}diagonal\text{-}admits\text{-}SNF\text{-}mod\text{-}ring\text{-}admits\text{-}aux2:$ 
assumes  $of: class.bezout\text{-}ring\ (*)\ (1::'a::comm\text{-}ring\text{-}1)\ (+)\ 0\ (-)\ uminus$ 
shows  $\forall (A::'a\ mat) \in carrier\text{-}mat\ n\ n. admits\text{-}SNF\text{-}JNF\ A$ 
proof  $(cases\ n = 0)$ 
case  $True$ 
show  $?thesis$ 
by  $(rule,\ unfold\ True\ admits\text{-}SNF\text{-}JNF\text{-}def\ isDiagonal\text{-}mat\text{-}def\ invertible\text{-}mat\text{-}def$ 
 $Smith\text{-}normal\text{-}form\text{-}mat\text{-}def\ carrier\text{-}mat\text{-}def\ inverts\text{-}mat\text{-}def,\ fastforce)$ 
next
case  $False$  note  $not0 = False$ 
show  $?thesis$ 
proof  $(cases\ n=1)$ 
case  $True$ 
show  $?thesis$ 

```

```

by (rule, unfold True admits-SNF-JNF-def isDiagonal-mat-def invertible-mat-def
      Smith-normal-form-mat-def carrier-mat-def inverts-mat-def, auto)
      (metis dvd-1-left index-one-mat(2) index-one-mat(3) less-Suc0 nat-dvd-not-less
          right-mult-one-mat' zero-less-Suc)
next
  case False
  then have  $n > 1$  using not0 by auto
  then show ?thesis
  using bezout-ring-imp-diagonal-admits-SNF-mod-ring-admits-aux[cancel-type-definition,
of n] of
    by auto
  qed
qed

```

Alternative statements

```

lemma bezout-ring-imp-diagonal-admits-SNF-JNF:
  assumes of: class.bezout-ring (*) (1::a::comm-ring-1) (+) 0 (-) uminus
  shows  $\forall A::'a \text{ mat. admits-SNF-JNF } A$ 
proof
  fix  $A::'a \text{ mat}$ 
  have  $A \in \text{carrier-mat } (\text{dim-row } A) (\text{dim-col } A)$  unfolding carrier-mat-def by
  auto
  thus admits-SNF-JNF  $A$ 
  using bezout-ring-imp-diagonal-admits-SNF-mod-ring-admits-aux2[OF of]
  by (metis admits-SNF-JNF-def square-mat.elims(2))
qed

```

```

lemma admits-SNF-JNF-alt-def:
  ( $\forall A::'a::\text{comm-ring-1 mat. admits-SNF-JNF } A$ )
  = ( $\forall A \ n. (A::'a \text{ mat}) \in \text{carrier-mat } n \ n \wedge \text{isDiagonal-mat } A$ 
     $\longrightarrow (\exists P \ Q. P \in \text{carrier-mat } n \ n \wedge Q \in \text{carrier-mat } n \ n \wedge \text{invertible-mat } P \wedge$ 
     $\text{invertible-mat } Q$ 
     $\wedge \text{Smith-normal-form-mat } (P * A * Q))$ ) (is ? $a = ?b$ )
  by (auto simp add: admits-SNF-JNF-def, metis carrier-matD(1) carrier-matD(2),
    blast)

```

## 9.6 Final theorem in JNF

Final theorem using *class.bezout-ring*

```

theorem diagonal-admits-SNF-iff-bezout-ring:
  shows class.bezout-ring (*) (1::a::comm-ring-1) (+) 0 (-) uminus
   $\longleftrightarrow (\forall A::'a \text{ mat. admits-SNF-JNF } A)$  (is ? $a \longleftrightarrow ?b$ )
proof
  assume ? $a$ 
  thus ? $b$  using bezout-ring-imp-diagonal-admits-SNF-JNF by auto
next

```

```

assume  $b$ : ? $b$ 
have  $rw$ :  $\forall A\ n.$  ( $A::'a\ mat$ )  $\in$  carrier-mat  $n\ n$   $\wedge$  isDiagonal-mat  $A \longrightarrow$ 
  ( $\exists P\ Q.$   $P \in$  carrier-mat  $n\ n$   $\wedge$   $Q \in$  carrier-mat  $n\ n$   $\wedge$  invertible-mat  $P$ 
   $\wedge$  invertible-mat  $Q$   $\wedge$  Smith-normal-form-mat ( $P * A * Q$ ))
  using admits-SNF-JNF-alt-def  $b$  by auto
show ? $a$ 
  using diagonal-admits-SNF-imp-bezout-ring-JNF[OF  $rw$ ]
  using OFCLASS-bezout-ring-imp-class-bezout-ring[where ? $a='a$ ]
  by auto
qed

```

Final theorem using *OFCLASS*

```

theorem diagonal-admits-SNF-iff-bezout-ring':
  shows OFCLASS(' $a::comm-ring-1$ , bezout-ring-class)  $\equiv$  ( $\bigwedge A::'a\ mat.$  admits-SNF-JNF
   $A$ )
proof
  fix  $A::'a\ mat$ 
  assume  $a$ : OFCLASS(' $a$ , bezout-ring-class)
  show admits-SNF-JNF  $A$ 
  using OFCLASS-bezout-ring-imp-class-bezout-ring[OF  $a$ ] diagonal-admits-SNF-iff-bezout-ring
  by auto
next
  assume ( $\bigwedge A::'a\ mat.$  admits-SNF-JNF  $A$ )
  hence *: class.bezout-ring (*) ( $1::'a$ ) (+) 0 (-) uminus
  using diagonal-admits-SNF-iff-bezout-ring by auto
  show OFCLASS(' $a$ , bezout-ring-class)
  by (rule bezout-ring.intro-of-class, rule *)
qed
end

```

## 10 Uniqueness of the Smith normal form

```

theory SNF-Uniqueness
imports
  Cauchy-Binet
  Smith-Normal-Form-JNF
  Admits-SNF-From-Diagonal-Iff-Bezout-Ring
begin

```

```

lemma dvd-associated1:
  fixes  $a::'a::comm-ring-1$ 
  assumes  $\exists u.$   $u\ dvd\ 1 \wedge a = u*b$ 
  shows  $a\ dvd\ b \wedge b\ dvd\ a$ 
  using assms by auto

```

This is a key lemma. It demands the type class to be an integral domain. This means that the uniqueness result will be obtained for GCD domains, instead of rings.

**lemma** *dvd-associated2*:

**fixes**  $a::'a::idom$

**assumes**  $ab: a \text{ dvd } b$  **and**  $ba: b \text{ dvd } a$  **and**  $a: a \neq 0$

**shows**  $\exists u. u \text{ dvd } 1 \wedge a = u * b$

**proof** –

**obtain**  $k$  **where**  $a \text{ -} kb: a = k * b$  **using**  $ab$  **unfolding** *dvd-def*

**by** (*metis Groups.mult-ac(2) ba dvdE*)

**obtain**  $q$  **where**  $b \text{ -} qa: b = q * a$  **using**  $ba$  **unfolding** *dvd-def*

**by** (*metis Groups.mult-ac(2) ab dvdE*)

**have**  $1: a = k * q * a$  **using**  $a \text{ -} kb$   $b \text{ -} qa$  **by** *auto*

**hence**  $k * q = 1$  **using**  $a$  **by** *simp*

**thus** *?thesis* **using**  $1$  **by** (*metis a-kb dvd-triv-left*)

**qed**

**corollary** *dvd-associated*:

**fixes**  $a::'a::idom$

**assumes**  $a \neq 0$

**shows**  $(a \text{ dvd } b \wedge b \text{ dvd } a) = (\exists u. u \text{ dvd } 1 \wedge a = u * b)$

**using** *assms dvd-associated1 dvd-associated2* **by** *metis*

**lemma** *exists-inj-ge-index*:

**assumes**  $S: S \subseteq \{0..<n\}$  **and**  $Sk: \text{card } S = k$

**shows**  $\exists f. \text{inj-on } f \{0..<k\} \wedge f'\{0..<k\} = S \wedge (\forall i \in \{0..<k\}. i \leq f i)$

**proof** –

**have**  $\exists h. \text{bij-betw } h \{0..<k\} S$

**using**  $S Sk \text{ ex-bij-betw-nat-finite subset-eq-atLeast0-lessThan-finite}$  **by** *blast*

**from** *this* **obtain**  $g$  **where**  $\text{inj-on-g}: \text{inj-on } g \{0..<k\}$  **and**  $gk\text{-}S: g'\{0..<k\} = S$

**unfolding** *bij-betw-def* **by** *blast*

**let**  $?f = \text{strict-from-inj } k g$

**have** *strict-mono-on*  $\{0..<k\} ?f$  **by** (*rule strict-strict-from-inj[OF inj-on-g]*)

**hence**  $1: \text{inj-on } ?f \{0..<k\}$  **using** *strict-mono-on-imp-inj-on* **by** *blast*

**have**  $2: ?f'\{0..<k\} = S$  **by** (*simp add: strict-from-inj-image' inj-on-g gk-S*)

**have**  $3: \forall i \in \{0..<k\}. i \leq ?f i$

**proof**

**fix**  $i$  **assume**  $i: i \in \{0..<k\}$

**let**  $?xs = \text{sorted-list-of-set } (g'\{0..<k\})$

**have** *strict-from-inj*  $k g i = ?xs ! i$  **unfolding** *strict-from-inj-def* **using**  $i$  **by**

*auto*

**moreover** **have**  $i \leq ?xs ! i$

**proof** (*rule sorted-wrt-less-idx, rule sorted-distinct-imp-sorted-wrt*)

**show** *sorted*  $?xs$

**using** *sorted-sorted-list-of-set* **by** *blast*

**show** *distinct*  $?xs$  **using** *distinct-sorted-list-of-set* **by** *blast*

**show**  $i < \text{length } ?xs$

**by** (*metis S Sk atLeast0LessThan distinct-card distinct-sorted-list-of-set gk-S*)

*i*

*lessThan-iff set-sorted-list-of-set subset-eq-atLeast0-lessThan-finite*)

**qed**

ultimately show  $i \leq ?f i$  by auto  
qed  
show *?thesis* using 1 2 3 by auto  
qed

## 10.1 More specific results about submatrices

**lemma** *diagonal-imp-submatrix0*:  
**assumes** *dA*: *diagonal-mat A* **and** *A-carrier*:  $A \in \text{carrier-mat } n \ m$   
**and** *Ik*:  $\text{card } I = k$  **and** *Jk*:  $\text{card } J = k$   
**and** *r*:  $\forall \text{ row-index} \in I. \text{ row-index} < n$   
**and** *c*:  $\forall \text{ col-index} \in J. \text{ col-index} < m$   
**and** *a*:  $a < k$  **and** *b*:  $b < k$   
**shows**  $\text{submatrix } A \ I \ J \ \$\$ (a, b) = 0 \vee \text{submatrix } A \ I \ J \ \$\$ (a, b) = A \ \$\$ (\text{pick } I \ a, \ \text{pick } J \ b)$   
**proof** (*cases submatrix A I J \$\$\$ (a, b) = 0*)  
**case** *True*  
**then show** *?thesis* by auto  
**next**  
**case** *False* **note** *not0 = False*  
**have** *aux*:  $\text{submatrix } A \ I \ J \ \$\$ (a, b) = A \ \$\$ (\text{pick } I \ a, \ \text{pick } J \ b)$   
**proof** (*rule submatrix-index*)  
**have**  $\text{card } \{i. i < \text{dim-row } A \wedge i \in I\} = k$   
**by** (*smt A-carrier Ik carrier-matD(1) equalityI mem-Collect-eq r subsetI*)  
**moreover have**  $\text{card } \{i. i < \text{dim-col } A \wedge i \in J\} = k$   
**by** (*metis (no-types, lifting) A-carrier Jk c carrier-matD(2) carrier-mat-def equalityI mem-Collect-eq subsetI*)  
**ultimately show**  $a < \text{card } \{i. i < \text{dim-row } A \wedge i \in I\}$   
**and**  $b < \text{card } \{i. i < \text{dim-col } A \wedge i \in J\}$  **using** *a b* **by** auto  
**qed**  
**thus** *?thesis*  
**proof** (*cases pick I a = pick J b*)  
**case** *True*  
**then show** *?thesis* using *aux* by auto  
**next**  
**case** *False*  
**then show** *?thesis*  
**by** (*metis aux A-carrier Ik Jk a b c carrier-matD dA diagonal-mat-def pick-in-set-le r*)  
**qed**  
**qed**

**lemma** *diagonal-imp-submatrix-element-not0*:  
**assumes** *dA*: *diagonal-mat A*  
**and** *A-carrier*:  $A \in \text{carrier-mat } n \ m$   
**and** *Ik*:  $\text{card } I = k$  **and** *Jk*:  $\text{card } J = k$   
**and** *I*:  $I \subseteq \{0..<n\}$

**and**  $J: J \subseteq \{0..<m\}$   
**and**  $b: b < k$   
**and**  $ex-not0: \exists i. i < k \wedge submatrix\ A\ I\ J\ \$\$ (i, b) \neq 0$   
**shows**  $\exists ! i. i < k \wedge submatrix\ A\ I\ J\ \$\$ (i, b) \neq 0$   
**proof** –  
**have**  $I-eq: I = \{i. i < dim-row\ A \wedge i \in I\}$  **using**  $I\ A\text{-carrier}$  **unfolding**  $carrier\text{-mat-def}$  **by**  $auto$   
**have**  $J-eq: J = \{i. i < dim-col\ A \wedge i \in J\}$  **using**  $J\ A\text{-carrier}$  **unfolding**  $carrier\text{-mat-def}$  **by**  $auto$   
**obtain**  $a$  **where**  $sub-ab: submatrix\ A\ I\ J\ \$\$ (a, b) \neq 0$  **and**  $ak: a < k$  **using**  $ex-not0$  **by**  $auto$   
**moreover** **have**  $i = a$  **if**  $sub-ib: submatrix\ A\ I\ J\ \$\$ (i, b) \neq 0$  **and**  $ik: i < k$   
**for**  $i$   
**proof** –  
**have**  $1: pick\ I\ i < dim-row\ A$   
**using**  $I-eq\ Ik\ ik\ pick-in-set-le$  **by**  $auto$   
**have**  $2: pick\ J\ b < dim-col\ A$   
**using**  $J-eq\ Jk\ b\ pick-le$  **by**  $auto$   
**have**  $3: pick\ I\ a < dim-row\ A$   
**using**  $I-eq\ Ik\ calculation(2)\ pick-le$  **by**  $auto$   
**have**  $submatrix\ A\ I\ J\ \$\$ (i, b) = A\ \$\$ (pick\ I\ i, pick\ J\ b)$   
**by**  $(rule\ submatrix-index, insert\ I-eq\ Ik\ ik\ J-eq\ Jk\ b, auto)$   
**hence**  $pick-Ii-Jb: pick\ I\ i = pick\ J\ b$  **using**  $dA\ sub-ib\ 1\ 2$  **unfolding**  $diagonal\text{-mat-def}$  **by**  $auto$   
**have**  $submatrix\ A\ I\ J\ \$\$ (a, b) = A\ \$\$ (pick\ I\ a, pick\ J\ b)$   
**by**  $(rule\ submatrix-index, insert\ I-eq\ Ik\ ak\ J-eq\ Jk\ b, auto)$   
**hence**  $pick-Ia-Jb: pick\ I\ a = pick\ J\ b$  **using**  $dA\ sub-ab\ 3\ 2$  **unfolding**  $diagonal\text{-mat-def}$  **by**  $auto$   
**have**  $pick-Ia-Ii: pick\ I\ a = pick\ I\ i$  **using**  $pick-Ii-Jb\ pick-Ia-Jb$  **by**  $simp$   
**thus**  $?thesis$  **by**  $(metis\ Ik\ ak\ ik\ nat-neq-iff\ pick-mono-le)$   
**qed**  
**ultimately** **show**  $?thesis$  **by**  $auto$   
**qed**

**lemma**  $submatrix-index-exists$ :

**assumes**  $A\text{-carrier}: A \in carrier\text{-mat}\ n\ m$   
**and**  $Ik: card\ I = k$  **and**  $Jk: card\ J = k$   
**and**  $a: a \in I$  **and**  $b: b \in J$  **and**  $k: k > 0$   
**and**  $I: I \subseteq \{0..<n\}$  **and**  $J: J \subseteq \{0..<m\}$   
**shows**  $\exists a' b'. a' < k \wedge b' < k \wedge submatrix\ A\ I\ J\ \$\$ (a', b') = A\ \$\$ (a, b)$   
 $\wedge a = pick\ I\ a' \wedge b = pick\ J\ b'$

**proof** –  
**let**  $?xs = sorted-list-of-set\ I$   
**let**  $?ys = sorted-list-of-set\ J$   
**have**  $finI: finite\ I$  **and**  $finJ: finite\ J$  **using**  $k\ Ik\ Jk\ card-ge-0-finite$  **by**  $metis+$   
**have**  $set-xs: set\ ?xs = I$  **by**  $(rule\ set-sorted-list-of-set[OF\ finI])$   
**have**  $set-ys: set\ ?ys = J$  **by**  $(rule\ set-sorted-list-of-set[OF\ finJ])$   
**have**  $a-in-xs: a \in set\ ?xs$  **and**  $b-in-ys: b \in set\ ?ys$  **using**  $set-xs\ a\ set-ys\ b$  **by**

*auto*  
**have** *length-xs*:  $\text{length } ?xs = k$  **by** (*metis*  $Ik$  *distinct-card set-xs sorted-list-of-set*(3))  
**have** *length-ys*:  $\text{length } ?ys = k$  **by** (*metis*  $Jk$  *distinct-card set-ys sorted-list-of-set*(3))  
**obtain** *a'* **where**  $a': ?xs ! a' = a$  **and** *a'-length*:  $a' < \text{length } ?xs$   
**by** (*meson* *a-in-xs in-set-conv-nth*)  
**obtain** *b'* **where**  $b': ?ys ! b' = b$  **and** *b'-length*:  $b' < \text{length } ?ys$   
**by** (*meson* *b-in-ys in-set-conv-nth*)  
**have** *pick-a*:  $a = \text{pick } I a'$  **using** *a'* *a'-length* *finI sorted-list-of-set-eq-pick* **by**  
*auto*  
**have** *pick-b*:  $b = \text{pick } J b'$  **using** *b'* *b'-length* *finJ sorted-list-of-set-eq-pick* **by**  
*auto*  
**have** *I-rw*:  $I = \{i. i < \text{dim-row } A \wedge i \in I\}$  **and** *J-rw*:  $J = \{i. i < \text{dim-col } A \wedge i \in J\}$   
**using** *I A-carrier J* **by** *auto*  
**have** *a'k*:  $a' < k$  **using** *a'-length length-xs* **by** *auto*  
**moreover** **have** *b'k*:  $b' < k$  **using** *b'-length length-ys* **by** *auto*  
**moreover** **have** *sub-eq*: *submatrix*  $A I J$   $\$ \$ (a', b') = A \$ \$ (a, b)$   
**unfolding** *pick-a pick-b*  
**by** (*rule submatrix-index, insert J-rw I-rw Ik Jk a'-length length-xs b'-length length-ys, auto*)  
**ultimately show** *?thesis* **using** *pick-a pick-b* **by** *auto*  
**qed**

**lemma** *mat-delete-submatrix-insert*:

**assumes** *A-carrier*:  $A \in \text{carrier-mat } n m$   
**and** *Ik*:  $\text{card } I = k$  **and** *Jk*:  $\text{card } J = k$   
**and** *I*:  $I \subseteq \{0..<n\}$  **and** *J*:  $J \subseteq \{0..<m\}$   
**and** *a*:  $a < n$  **and** *b*:  $b < m$   
**and** *k*:  $k < \min n m$   
**and** *a-notin-I*:  $a \notin I$  **and** *b-notin-J*:  $b \notin J$   
**and** *a'k*:  $a' < \text{Suc } k$  **and** *b'k*:  $b' < \text{Suc } k$   
**and** *a-def*:  $\text{pick } (\text{insert } a I) a' = a$   
**and** *b-def*:  $\text{pick } (\text{insert } b J) b' = b$   
**shows** *mat-delete* (*submatrix*  $A (\text{insert } a I) (\text{insert } b J)$ )  $a' b' = \text{submatrix } A I J$   
**(is ?lhs = ?rhs)**  
**proof** (*rule eq-matI*)  
**have** *I-eq*:  $I = \{i. i < \text{dim-row } A \wedge i \in I\}$   
**using** *I A-carrier unfolding carrier-mat-def* **by** *auto*  
**have** *J-eq*:  $J = \{i. i < \text{dim-col } A \wedge i \in J\}$   
**using** *J A-carrier unfolding carrier-mat-def* **by** *auto*  
**have** *insert-I-eq*:  $\text{insert } a I = \{i. i < \text{dim-row } A \wedge i \in \text{insert } a I\}$   
**using** *I A-carrier a k unfolding carrier-mat-def* **by** *auto*  
**have** *card-Suc-k*:  $\text{card } \{i. i < \text{dim-row } A \wedge i \in \text{insert } a I\} = \text{Suc } k$   
**using** *insert-I-eq Ik a-notin-I*  
**by** (*metis* *I card-insert-disjoint finite-atLeastLessThan finite-subset*)  
**have** *insert-J-eq*:  $\text{insert } b J = \{i. i < \text{dim-col } A \wedge i \in \text{insert } b J\}$   
**using** *J A-carrier b k unfolding carrier-mat-def* **by** *auto*  
**have** *card-Suc-k'*:  $\text{card } \{i. i < \text{dim-col } A \wedge i \in \text{insert } b J\} = \text{Suc } k$



```

    using insert-J-eq Jk b-notin-J
    by (metis J card-insert-disjoint finite-atLeastLessThan finite-subset)
  show dim-row ?lhs = dim-row ?rhs
    unfolding mat-delete-dim unfolding dim-submatrix using card-Suc-k I-eq Jk
  by auto
  show dim-col ?lhs = dim-col ?rhs
    unfolding mat-delete-dim unfolding dim-submatrix using card-Suc-k' J-eq Jk
  by auto
  fix i j assume i: i < dim-row (submatrix A I J)
    and j: j < dim-col (submatrix A I J)
  have ik: i < k by (metis I-eq Ik dim-submatrix(1) i)
  have jk: j < k by (metis J-eq Jk dim-submatrix(2) j)
  show ?lhs $$ (i, j) = ?rhs $$ (i, j)
  proof -
    have index-eq1: pick (insert a I) (insert-index a' i) = pick I i
      by (rule pick-insert-index[OF Ik a-notin-I ik a-def], simp add: Ik a'k)
    have index-eq2: pick (insert b J) (insert-index b' j) = pick J j
      by (rule pick-insert-index[OF Jk b-notin-J jk b-def], simp add: Jk b'k)
    have ?lhs $$ (i, j)
      = (submatrix A (insert a I) (insert b J)) $$ (insert-index a' i, insert-index
    b' j)
    proof (rule mat-delete-index[symmetric, OF - a'k b'k ik jk])
      show submatrix A (insert a I) (insert b J) ∈ carrier-mat (Suc k) (Suc k)
        by (metis card-Suc-k card-Suc-k' carrier-matI dim-submatrix(1) dim-submatrix(2))
      qed
      also have ... = A $$ (pick (insert a I) (insert-index a' i), pick (insert b J)
      (insert-index b' j))
      proof (rule submatrix-index)
        show insert-index a' i < card {i. i < dim-row A ∧ i ∈ insert a I}
          using card-Suc-k ik insert-index-def by auto
        show insert-index b' j < card {j. j < dim-col A ∧ j ∈ insert b J}
          using card-Suc-k' insert-index-def jk by auto
        qed
        also have ... = A $$ (pick I i, pick J j) unfolding index-eq1 index-eq2 by auto
        also have ... = submatrix A I J $$ (i, j)
          by (rule submatrix-index[symmetric], insert ik I-eq Ik Jk J-eq jk, auto)
        finally show ?thesis .
      qed
    qed
  qed

```

## 10.2 On the minors of a diagonal matrix

lemma *det-minors-diagonal*:

assumes *dA*: diagonal-mat *A* and *A-carrier*:  $A \in \text{carrier-mat } n \ m$

and *Ik*:  $\text{card } I = k$  and *Jk*:  $\text{card } J = k$

and *r*:  $I \subseteq \{0..<n\}$

and *c*:  $J \subseteq \{0..<m\}$  and *k*:  $k > 0$

shows  $\det (\text{submatrix } A \ I \ J) = 0$

$\vee (\exists xs. (\det (\text{submatrix } A \ I \ J) = \text{prod-list } xs \vee \det (\text{submatrix } A \ I \ J) = -$

```

prod-list xs)
  ∧ set xs ⊆ {A$$ (i,i) | i. i < min n m ∧ A$$ (i,i) ≠ 0} ∧ length xs = k)
  using Ik Jk r c k
proof (induct k arbitrary: I J)
  case 0
  then show ?case by auto
next
  case (Suc k)
  note cardI = Suc.premis(1)
  note cardJ = Suc.premis(2)
  note I = Suc.premis(3)
  note J = Suc.premis(4)
  have *: {i. i < dim-row A ∧ i ∈ I} = I using I Ik A-carrier carrier-mat-def
  by auto
  have **: {j. j < dim-col A ∧ j ∈ J} = J using J Jk A-carrier carrier-mat-def
  by auto
  show ?case
  proof (cases k = 0)
    case True note k0 = True
    from this obtain a where aI: I = {a} using True cardI card-1-singletonE by
    auto
    from this obtain b where bJ: J = {b} using True cardJ card-1-singletonE
    by auto
    have an: a < n using aI I by auto
    have bm: b < m using bJ J by auto
    have sub-carrier: submatrix A {a} {b} ∈ carrier-mat 1 1
      unfolding carrier-mat-def submatrix-def
      using * ** aI bJ by auto
    have 1: det (submatrix A {a} {b}) = (submatrix A {a} {b}) $$ (0,0)
      by (rule det-singleton[OF sub-carrier])
    have 2: ... = A $$ (a,b)
      by (rule submatrix-singleton-index[OF A-carrier an bm])
    show ?thesis
  proof (cases A $$ (a,b) ≠ 0)
    let ?xs = [submatrix A {a} {b} $$ (0,0)]
    case True
      hence a = b using dA A-carrier an bm unfolding diagonal-mat-def car-
      rier-mat-def by auto
      hence set ?xs ⊆ {A $$ (i, i) | i. i < min n m ∧ A $$ (i, i) ≠ 0}
        using 2 True an bm by auto
      moreover have det (submatrix A {a} {b}) = prod-list ?xs using 1 by auto
      moreover have length ?xs = Suc k using k0 by auto
      ultimately show ?thesis using an bm unfolding aI bJ by blast
    next
      case False
      then show ?thesis using 1 2 aI bJ by auto
  qed
next
  case False

```

hence  $k0: 0 < k$  by *simp*  
 have  $k: k < \min n m$   
 by (*metis I J cardI cardJ le-imp-less-Suc less-Suc-eq-le min.commute*  
*min-def not-less subset-eq-atLeast0-lessThan-card*)  
 have *subIJ-carrier*:  $(\text{submatrix } A \ I \ J) \in \text{carrier-mat } (\text{Suc } k) \ (\text{Suc } k)$   
 unfolding *carrier-mat-def* using *\*\* cardI cardJ*  
 unfolding *submatrix-def* by *auto*  
 obtain  $b'$  where  $b'k: b' < \text{Suc } k$  by *auto*  
 let  $?f = \lambda i. \text{submatrix } A \ I \ J \ \$\$ (i, b') * \text{cofactor } (\text{submatrix } A \ I \ J) \ i \ b'$   
 have *det-rw*:  $\det (\text{submatrix } A \ I \ J)$   
 =  $(\sum_{i < \text{Suc } k} \text{submatrix } A \ I \ J \ \$\$ (i, b') * \text{cofactor } (\text{submatrix } A \ I \ J) \ i \ b')$   
 by (*rule laplace-expansion-column[OF subIJ-carrier b'k]*)  
 show *?thesis*  
 proof (*cases*  $\exists a' < \text{Suc } k. \text{submatrix } A \ I \ J \ \$\$ (a', b') \neq 0$ )  
 case *True*  
 obtain  $a'$  where *sub-IJ-0*:  $\text{submatrix } A \ I \ J \ \$\$ (a', b') \neq 0$   
 and  $a'k: a' < \text{Suc } k$   
 and *unique*:  $\forall j. j < \text{Suc } k \wedge \text{submatrix } A \ I \ J \ \$\$ (j, b') \neq 0 \longrightarrow j = a'$   
 using *diagonal-imp-submatrix-element-not0[OF dA A-carrier cardI cardJ I*  
*J b'k True]* by *auto*  
 have  $\text{submatrix } A \ I \ J \ \$\$ (a', b') = A \ \$\$ (\text{pick } I \ a', \text{pick } J \ b')$   
 by (*rule submatrix-index, auto simp add: \* a'k cardI \*\* b'k cardJ*)  
 from *this* obtain  $a \ b$  where *an*:  $a < n$  and *bm*:  $b < m$   
 and *sub-index*:  $\text{submatrix } A \ I \ J \ \$\$ (a', b') = A \ \$\$ (a, b)$   
 and *pick-a*:  $\text{pick } I \ a' = a$  and *pick-b*:  $\text{pick } J \ b' = b$   
 using *\*\* A-carrier a'k b'k cardI cardJ pick-le* by *fastforce*  
 obtain  $I'$  where  $aI': I = \text{insert } a \ I'$  and *a-notin*:  $a \notin I'$   
 by (*metis Set.set-insert a'k cardI pick-a pick-in-set-le*)  
 obtain  $J'$  where  $bJ': J = \text{insert } b \ J'$  and *b-notin*:  $b \notin J'$   
 by (*metis Set.set-insert b'k cardJ pick-b pick-in-set-le*)  
 have *Suc-k0*:  $0 < \text{Suc } k$  by *simp*  
 have  $aI: a \in I$  using  $aI'$  by *auto*  
 have  $bJ: b \in J$  using  $bJ'$  by *auto*  
 have  $\text{card } I': \text{card } I' = k$   
 by (*metis aI' a-notin cardI card.infinite card-insert-disjoint*  
*finite-insert nat.inject nat.simps(3)*)  
 have  $\text{card } J': \text{card } J' = k$   
 by (*metis bJ' b-notin cardJ card.infinite card-insert-disjoint*  
*finite-insert nat.inject nat.simps(3)*)  
 have  $I': I' \subseteq \{0..<n\}$  using  $I \ aI'$  by *blast*  
 have  $J': J' \subseteq \{0..<m\}$  using  $J \ bJ'$  by *blast*  
 have *det-sub-I'J'*:  $\text{Determinant.det } (\text{submatrix } A \ I' \ J') = 0 \vee$   
 ( $\exists xs. (\det (\text{submatrix } A \ I' \ J') = \text{prod-list } xs \vee \det (\text{submatrix } A \ I' \ J') = -$   
*prod-list } xs)  
 $\wedge \text{set } xs \subseteq \{A \ \$\$ (i, i) \mid i. i < \min n m \wedge A \ \$\$ (i, i) \neq 0\} \wedge \text{length } xs = k$ )  
 proof (*rule Suc.hyps[OF cardI' cardJ' - - k0]*)  
 show  $I' \subseteq \{0..<n\}$  using  $I \ aI'$  by *blast*  
 show  $J' \subseteq \{0..<m\}$  using  $J \ bJ'$  by *blast*  
 qed*

**have** *mat-delete-sub*:  
 $mat\text{-}delete\ (submatrix\ A\ (insert\ a\ I')\ (insert\ b\ J'))\ a'\ b' = submatrix\ A\ I'\ J'$   
**by** (*rule mat-delete-submatrix-insert*[*OF A-carrier cardI' cardJ' I' J' an bm k*]  
 $a\text{-notin}\ b\text{-notin}\ a'k\ b'k], insert\ pick\text{-}a\ pick\text{-}b\ aI'\ bJ', auto)$   
**have** *set-rw*:  $\{0..<Suc\ k\} = insert\ a'\ (\{0..<Suc\ k\} - \{a'\})$   
**by** (*simp add: a'k insert-absorb*)  
**have** *rw0*:  $sum\ ?f\ (\{0..<Suc\ k\} - \{a'\}) = 0$  **by** (*rule sum.neutral, insert unique, auto*)  
**have** *det (submatrix A I J)*  
 $= (\sum\ i < Suc\ k.\ submatrix\ A\ I\ J\ \$\$ (i, b') * cofactor\ (submatrix\ A\ I\ J)\ i\ b')$   
**by** (*rule laplace-expansion-column*[*OF subIJ-carrier b'k*])  
**also have**  $... = ?f\ a' + sum\ ?f\ (\{0..<Suc\ k\} - \{a'\})$   
**by** (*metis (no-types, lifting) Diff-iff atLeast0LessThan finite-atLeastLessThan finite-insert set-rw singletonI sum.insert*)  
**also have**  $... = ?f\ a'$  **using** *rw0 unfolding cofactor-def* **by** *auto*  
**also have**  $... = submatrix\ A\ I\ J\ \$\$ (a', b') * ((-1) ^ (a' + b')) * det\ (submatrix\ A\ I'\ J')$   
**unfolding cofactor-def using mat-delete-sub aI' bJ' by simp**  
**finally have** *det-submatrix-IJ*:  $det\ (submatrix\ A\ I\ J)$   
 $= A\ \$\$ (a, b) * ((-1) ^ (a' + b')) * det\ (submatrix\ A\ I'\ J')$  **unfolding**  
*sub-index .*  
**show** *?thesis*  
**proof** (*cases det (submatrix A I' J') = 0*)  
**case** *True*  
**then show** *?thesis using det-submatrix-IJ by auto*  
**next**  
**case** *False note det-not0 = False*  
**from this obtain** *xs where prod-list-xs: det (submatrix A I' J') = prod-list*  
*xs*  
 $\vee det\ (submatrix\ A\ I'\ J') = -\ prod\text{-list}\ xs$   
**and** *xs: set xs*  $\subseteq \{A\ \$\$ (i, i) \mid i. i < \min\ n\ m \wedge A\ \$\$ (i, i) \neq 0\}$   
**and** *length-xs: length xs = k*  
**using** *det-sub-I'J' by blast*  
**let** *?ys = A\$\$ (a,b) # xs*  
**have** *length-ys: length ?ys = Suc k* **using** *length-xs by auto*  
**have** *a-eq-b: a=b*  
**using** *A-carrier an bm sub-IJ-0 sub-index dA unfolding diagonal-mat-def*  
**by** *auto*  
**have** *A-aa-in: A\$\$ (a,a) ∈ {A \$\$ (i, i) | i. i < min n m ∧ A \$\$ (i, i) ≠ 0}*  
**using** *a-eq-b an bm sub-IJ-0 sub-index by auto*  
**have** *ys: set ?ys*  $\subseteq \{A\ \$\$ (i, i) \mid i. i < \min\ n\ m \wedge A\ \$\$ (i, i) \neq 0\}$   
**using** *xs A-aa-in a-eq-b by auto*  
**show** *?thesis*  
**proof** (*cases even (a'+b')*)  
**case** *True*  
**have** *det-submatrix-IJ: det (submatrix A I J) = A \$\$ (a, b) \* det (submatrix*  
*A I' J')*

```

      using det-submatrix-IJ True by auto
    show ?thesis
    proof (cases det (submatrix A I' J') = prod-list xs)
      case True
      have det (submatrix A I J) = prod-list ?ys
        using det-submatrix-IJ unfolding True by auto
      then show ?thesis using ys length-ys by blast
    next
      case False
      hence det (submatrix A I' J') = - prod-list xs using prod-list-xs by
simp
      hence det (submatrix A I J) = - prod-list ?ys using det-submatrix-IJ
by auto
      then show ?thesis using ys length-ys by blast
    qed
  next
  case False
  have det-submatrix-IJ: det (submatrix A I J) = A $$ (a, b) * - det
(submatrix A I' J')
  using det-submatrix-IJ False by auto
  show ?thesis
  proof (cases det (submatrix A I' J') = prod-list xs)
    case True
    have det (submatrix A I J) = - prod-list ?ys
      using det-submatrix-IJ unfolding True by auto
    then show ?thesis using ys length-ys by blast
  next
    case False
    hence det (submatrix A I' J') = - prod-list xs using prod-list-xs by
simp
    hence det (submatrix A I J) = prod-list ?ys using det-submatrix-IJ by
auto
    then show ?thesis using ys length-ys by blast
  qed
  qed
next
case False
have sum ?f {0..<Suc k} = 0 by (rule sum.neutral, insert False, auto)
thus ?thesis using det-rw
  by (simp add: atLeast0LessThan)
qed
qed
qed

```

**definition**  $minors\ A\ k = \{det\ (submatrix\ A\ I\ J) \mid I\ J,\ I \subseteq \{0..<dim\text{-row}\ A\} \wedge J \subseteq \{0..<dim\text{-col}\ A\} \wedge card\ I = k \wedge card\ J = k\}$

**lemma** *Gcd-minors-dvd*:

**fixes**  $A::'a::\{\text{semiring-Gcd,comm-ring-1}\}$  *mat*

**assumes**  $PAQ=B: P * A * Q = B$

**and**  $P: P \in \text{carrier-mat } m \ m$

**and**  $A: A \in \text{carrier-mat } m \ n$

**and**  $Q: Q \in \text{carrier-mat } n \ n$

**and**  $I: I \subseteq \{0..<\text{dim-row } A\}$  **and**  $J: J \subseteq \{0..<\text{dim-col } A\}$

**and**  $I_k: \text{card } I = k$  **and**  $J_k: \text{card } J = k$

**shows** *Gcd (minors A k) dvd det (submatrix B I J)*

**proof** –

**let**  $?subPA = \text{submatrix } (P * A) \ I \ UNIV$

**let**  $?subQ = \text{submatrix } Q \ UNIV \ J$

**have**  $subPA: ?subPA \in \text{carrier-mat } k \ n$

**proof** –

**have**  $I = \{i. i < \text{dim-row } P \wedge i \in I\}$  **using**  $P \ I \ A$  **by** *auto*

**hence**  $\text{card } \{i. i < \text{dim-row } P \wedge i \in I\} = k$  **using**  $I_k$  **by** *auto*

**thus**  $?thesis$  **using**  $A$  **unfolding** *submatrix-def* **by** *auto*

**qed**

**have**  $subQ: \text{submatrix } Q \ UNIV \ J \in \text{carrier-mat } n \ k$

**proof** –

**have**  $J\text{-eq}: J = \{j. j < \text{dim-col } Q \wedge j \in J\}$  **using**  $Q \ J \ A$  **by** *auto*

**hence**  $\text{card } \{j. j < \text{dim-col } Q \wedge j \in J\} = k$  **using**  $J_k$  **by** *auto*

**moreover** **have**  $\text{card } \{i. i < \text{dim-row } Q \wedge i \in UNIV\} = n$  **using**  $Q$  **by** *auto*

**ultimately show**  $?thesis$  **unfolding** *submatrix-def* **by** *auto*

**qed**

**have**  $sub\text{-}sub\text{-}PA: (\text{submatrix } ?subPA \ UNIV \ I') = \text{submatrix } (P * A) \ I \ I'$  **for**  $I'$

**using** *submatrix-split2[symmetric]* **by** *auto*

**have**  $\text{det}\text{-}subPA\text{-}rw: \text{det } (\text{submatrix } (P * A) \ I \ I') =$

$(\sum J' \mid J' \subseteq \{0..<m\} \wedge \text{card } J' = k. \text{det } ((\text{submatrix } P \ I \ J'))) * \text{det } (\text{submatrix}$

$A \ J' \ I')$

**if**  $I'1: I' \subseteq \{0..<n\}$  **and**  $I'2: \text{card } I' = k$  **for**  $I'$

**proof** –

**have**  $\text{submatrix } (P * A) \ I \ I' = \text{submatrix } P \ I \ UNIV * \text{submatrix } A \ UNIV \ I'$

**unfolding** *submatrix-mult ..*

**also have**  $\text{det } \dots = (\sum C \mid C \subseteq \{0..<m\} \wedge \text{card } C = k.$

$\text{det } (\text{submatrix } (\text{submatrix } P \ I \ UNIV) \ UNIV \ C)) * \text{det } (\text{submatrix } (\text{submatrix}$

$A \ UNIV \ I') \ C \ UNIV))$

**proof** (*rule Cauchy-Binet*)

**have**  $I = \{i. i < \text{dim-row } P \wedge i \in I\}$  **using**  $P \ I \ A$  **by** *auto*

**thus**  $\text{submatrix } P \ I \ UNIV \in \text{carrier-mat } k \ m$  **using**  $I_k \ P$  **unfolding** *submatrix-def* **by** *auto*

**have**  $I' = \{j. j < \text{dim-col } A \wedge j \in I'\}$  **using**  $I'1 \ A$  **by** *auto*

**thus**  $\text{submatrix } A \ UNIV \ I' \in \text{carrier-mat } m \ k$  **using**  $I'2 \ A$  **unfolding** *submatrix-def* **by** *auto*

**qed**

**also have**  $\dots = (\sum J' \mid J' \subseteq \{0..<m\} \wedge \text{card } J' = k.$

$\text{det } (\text{submatrix } P \ I \ J')) * \text{det } (\text{submatrix } A \ J' \ I')$

**unfolding** *submatrix-split2[symmetric]* *submatrix-split[symmetric]* **by** *simp*

**finally show** *?thesis* .  
**qed**  
**have**  $\det (\text{submatrix } B \ I \ J) = \det (\text{submatrix } (P * A * Q) \ I \ J)$  **using** *PAQ-B* **by**  
*simp*  
**also have**  $\dots = \det (?subPA * ?subQ)$  **unfolding** *submatrix-mult* **by** *auto*  
**also have**  $\dots = (\sum I' \mid I' \subseteq \{0..<n\} \wedge \text{card } I' = k. \det (\text{submatrix } ?subPA \ UNIV \ I'))$   
 $* \det (\text{submatrix } ?subQ \ I' \ UNIV)$   
**by** *(rule Cauchy-Binet[OF subPA subQ])*  
**also have**  $\dots = (\sum I' \mid I' \subseteq \{0..<n\} \wedge \text{card } I' = k.$   
 $\det (\text{submatrix } (P * A) \ I \ I') * \det (\text{submatrix } Q \ I' \ J))$   
**using** *submatrix-split[symmetric, of Q]* *submatrix-split2[symmetric, of P\*A]* **by**  
*presburger*  
**also have**  $\dots = (\sum I' \mid I' \subseteq \{0..<n\} \wedge \text{card } I' = k. \sum J' \mid J' \subseteq \{0..<m\} \wedge \text{card } J' = k.$   
 $\det (\text{submatrix } P \ I \ J') * \det (\text{submatrix } A \ J' \ I') * \det (\text{submatrix } Q \ I' \ J))$   
**using** *det-subPA-rw* **by** *(simp add: semiring-0-class.sum-distrib-right)*  
**finally have** *det-rw*:  $\det (\text{submatrix } B \ I \ J) = (\sum I' \mid I' \subseteq \{0..<n\} \wedge \text{card } I' = k.$   
 $\sum J' \mid J' \subseteq \{0..<m\} \wedge \text{card } J' = k.$   
 $\det (\text{submatrix } P \ I \ J') * \det (\text{submatrix } A \ J' \ I') * \det (\text{submatrix } Q \ I' \ J)) .$   
**show** *?thesis*  
**proof** *(unfold det-rw, (rule dvd-sum)+)*  
**fix**  $I' \ J'$   
**assume**  $I': I' \in \{I'. I' \subseteq \{0..<n\} \wedge \text{card } I' = k\}$   
**and**  $J': J' \in \{J'. J' \subseteq \{0..<m\} \wedge \text{card } J' = k\}$   
**have**  $Gcd (\text{minors } A \ k) \ \text{dvd} \ \det (\text{submatrix } A \ J' \ I')$   
**by** *(rule Gcd-dvd, unfold minors-def, insert A I' J', auto)*  
**then show**  $Gcd (\text{minors } A \ k) \ \text{dvd} \ \det (\text{submatrix } P \ I \ J') * \det (\text{submatrix } A \ J' \ I')$   
 $* \det (\text{submatrix } Q \ I' \ J)$  **by** *auto*  
**qed**  
**qed**

**lemma** *det-minors-diagonal2*:

**assumes** *dA*: *diagonal-mat A* **and** *A-carrier*:  $A \in \text{carrier-mat } n \ m$   
**and**  $Ik: \text{card } I = k$  **and**  $Jk: \text{card } J = k$   
**and**  $r: I \subseteq \{0..<n\}$   
**and**  $c: J \subseteq \{0..<m\}$  **and**  $k: k > 0$   
**shows**  $\det (\text{submatrix } A \ I \ J) = 0 \vee (\exists S. S \subseteq \{0..<\min \ n \ m\} \wedge \text{card } S = k \wedge S=I \wedge$   
 $(\det (\text{submatrix } A \ I \ J) = (\prod_{i \in S}. A \ \$(i,i)) \vee \det (\text{submatrix } A \ I \ J) = -$   
 $(\prod_{i \in S}. A \ \$(i,i))))$   
**using**  $Ik \ Jk \ r \ c \ k$   
**proof** *(induct k arbitrary: I J)*  
**case**  $0$   
**then show** *?case* **by** *auto*  
**next**

```

case (Suc k)
note cardI = Suc.premis(1)
note cardJ = Suc.premis(2)
note I = Suc.premis(3)
note J = Suc.premis(4)
have *: {i. i < dim-row A ∧ i ∈ I} = I using I Ik A-carrier carrier-mat-def
by auto
have **: {j. j < dim-col A ∧ j ∈ J} = J using J Jk A-carrier carrier-mat-def
by auto
show ?case
proof (cases k = 0)
  case True note k0 = True
  from this obtain a where aI: I = {a} using True cardI card-1-singletonE by
  auto
  from this obtain b where bJ: J = {b} using True cardJ card-1-singletonE
  by auto
  have an: a < n using aI I by auto
  have bm: b < m using bJ J by auto
  have sub-carrier: submatrix A {a} {b} ∈ carrier-mat 1 1
  unfolding carrier-mat-def submatrix-def
  using * ** aI bJ by auto
  have 1: det (submatrix A {a} {b}) = (submatrix A {a} {b}) $$ (0,0)
  by (rule det-singleton[OF sub-carrier])
  have 2: ... = A $$ (a,b)
  by (rule submatrix-singleton-index[OF A-carrier an bm])
  show ?thesis
  proof (cases A $$ (a,b) ≠ 0)
    let ?S={a}
    case True
    hence ab: a = b using dA A-carrier an bm unfolding diagonal-mat-def
    carrier-mat-def by auto
    hence ?S ⊆ {0..using an bm by auto
    moreover have det (submatrix A {a} {b}) = (∏ i ∈ ?S. A $$ (i, i)) using 1
    2 ab by auto
    moreover have card ?S = Suc k using k0 by auto
    ultimately show ?thesis using an bm unfolding aI bJ by blast
  next
  case False
  then show ?thesis using 1 2 aI bJ by auto
  qed
next
case False
hence k0: 0 < k by simp
have k: k < min n m
  by (metis I J cardI cardJ le-imp-less-Suc less-Suc-eq-le min.commute
  min-def not-less subset-eq-atLeast0-lessThan-card)
have subIJ-carrier: (submatrix A I J) ∈ carrier-mat (Suc k) (Suc k)
  unfolding carrier-mat-def using * ** cardI cardJ
  unfolding submatrix-def by auto

```



**obtain**  $b'$  **where**  $b'k$ :  $b' < \text{Suc } k$  **by** *auto*  
**let**  $?f = \lambda i. \text{submatrix } A \ I \ J \ \$\$ (i, b') * \text{cofactor } (\text{submatrix } A \ I \ J) \ i \ b'$   
**have**  $\text{det-rw}$ :  $\text{det } (\text{submatrix } A \ I \ J)$   
 $= (\sum i < \text{Suc } k. \text{submatrix } A \ I \ J \ \$\$ (i, b') * \text{cofactor } (\text{submatrix } A \ I \ J) \ i \ b')$   
**by** (*rule laplace-expansion-column*[*OF subIJ-carrier b'k*])  
**show** *?thesis*  
**proof** (*cases*  $\exists a' < \text{Suc } k. \text{submatrix } A \ I \ J \ \$\$ (a', b') \neq 0$ )  
**case** *True*  
**obtain**  $a'$  **where**  $\text{sub-IJ-0}$ :  $\text{submatrix } A \ I \ J \ \$\$ (a', b') \neq 0$   
**and**  $a'k$ :  $a' < \text{Suc } k$   
**and**  $\text{unique}$ :  $\forall j. j < \text{Suc } k \wedge \text{submatrix } A \ I \ J \ \$\$ (j, b') \neq 0 \longrightarrow j = a'$   
**using** *diagonal-imp-submatrix-element-not0*[*OF dA A-carrier cardI cardJ I*  
*J b'k True*] **by** *auto*  
**have**  $\text{submatrix } A \ I \ J \ \$\$ (a', b') = A \ \$\$ (\text{pick } I \ a', \text{pick } J \ b')$   
**by** (*rule submatrix-index, auto simp add: \* a'k cardI \*\* b'k cardJ*)  
**from** *this* **obtain**  $a \ b$  **where**  $an$ :  $a < n$  **and**  $bm$ :  $b < m$   
**and**  $\text{sub-index}$ :  $\text{submatrix } A \ I \ J \ \$\$ (a', b') = A \ \$\$ (a, b)$   
**and**  $\text{pick-a}$ :  $\text{pick } I \ a' = a$  **and**  $\text{pick-b}$ :  $\text{pick } J \ b' = b$   
**using**  $* ** A\text{-carrier } a'k \ b'k \ \text{cardI } \ \text{cardJ } \ \text{pick-le}$  **by** *fastforce*  
**obtain**  $I'$  **where**  $aI'$ :  $I = \text{insert } a \ I'$  **and**  $a\text{-notin}$ :  $a \notin I'$   
**by** (*metis Set.set-insert a'k cardI pick-a pick-in-set-le*)  
**obtain**  $J'$  **where**  $bJ'$ :  $J = \text{insert } b \ J'$  **and**  $b\text{-notin}$ :  $b \notin J'$   
**by** (*metis Set.set-insert b'k cardJ pick-b pick-in-set-le*)  
**have**  $\text{Suc-k0}$ :  $0 < \text{Suc } k$  **by** *simp*  
**have**  $aI$ :  $a \in I$  **using**  $aI'$  **by** *auto*  
**have**  $bJ$ :  $b \in J$  **using**  $bJ'$  **by** *auto*  
**have**  $\text{cardI}'$ :  $\text{card } I' = k$   
**by** (*metis aI' a-notin cardI card.infinite card-insert-disjoint*  
*finite-insert nat.inject nat.simps(3)*)  
**have**  $\text{cardJ}'$ :  $\text{card } J' = k$   
**by** (*metis bJ' b-notin cardJ card.infinite card-insert-disjoint*  
*finite-insert nat.inject nat.simps(3)*)  
**have**  $I'$ :  $I' \subseteq \{0..<n\}$  **using**  $I \ aI'$  **by** *blast*  
**have**  $J'$ :  $J' \subseteq \{0..<m\}$  **using**  $J \ bJ'$  **by** *blast*  
**have**  $\text{det-sub-I'J}'$ :  $\text{det } (\text{submatrix } A \ I' \ J') = 0 \vee (\exists S \subseteq \{0..<\min \ n \ m\}. \text{card}$   
 $S = k \wedge S = I'$   
 $\wedge (\text{det } (\text{submatrix } A \ I' \ J') = (\prod i \in S. A \ \$\$ (i, i))$   
 $\vee \text{det } (\text{submatrix } A \ I' \ J') = - (\prod i \in S. A \ \$\$ (i, i))))$   
**proof** (*rule Suc.hyps*[*OF cardI' cardJ' - - k0*])  
**show**  $I' \subseteq \{0..<n\}$  **using**  $I \ aI'$  **by** *blast*  
**show**  $J' \subseteq \{0..<m\}$  **using**  $J \ bJ'$  **by** *blast*  
**qed**  
**have**  $\text{mat-delete-sub}$ :  
 $\text{mat-delete } (\text{submatrix } A \ (\text{insert } a \ I') \ (\text{insert } b \ J')) \ a' \ b' = \text{submatrix } A \ I'$   
 $J'$   
**by** (*rule mat-delete-submatrix-insert*[*OF A-carrier cardI' cardJ' I' J' an bm*  
 $k$   
 $a\text{-notin } b\text{-notin } a'k \ b'k, \text{insert pick-a pick-b } aI' \ bJ', \text{auto}$ ])  
**have**  $\text{set-rw}$ :  $\{0..<\text{Suc } k\} = \text{insert } a' \ (\{0..<\text{Suc } k\} - \{a'\})$

**by** (*simp add: a'k insert-absorb*)  
**have**  $rw0: \text{sum } ?f (\{0..<Suc\ k\} - \{a'\}) = 0$  **by** (*rule sum.neutral, insert unique, auto*)  
**have**  $\text{det } (\text{submatrix } A\ I\ J)$   
 $= (\sum i < Suc\ k. \text{submatrix } A\ I\ J\ \$\$ (i, b') * \text{cofactor } (\text{submatrix } A\ I\ J)\ i\ b')$   
**by** (*rule laplace-expansion-column[OF subIJ-carrier b'k]*)  
**also have**  $\dots = ?f\ a' + \text{sum } ?f (\{0..<Suc\ k\} - \{a'\})$   
**by** (*metis (no-types, lifting) Diff-iff atLeast0LessThan finite-atLeastLessThan finite-insert set-rw singletonI sum.insert*)  
**also have**  $\dots = ?f\ a'$  **using**  $rw0$  **unfolding** *cofactor-def* **by** *auto*  
**also have**  $\dots = \text{submatrix } A\ I\ J\ \$\$ (a', b') * ((-1) ^ (a' + b')) * \text{det } (\text{submatrix } A\ I'\ J')$   
**unfolding** *cofactor-def* **using** *mat-delete-sub aI' bJ'* **by** *simp*  
**finally have**  $\text{det-submatrix-IJ: } \text{det } (\text{submatrix } A\ I\ J)$   
 $= A\ \$\$ (a, b) * ((-1) ^ (a' + b')) * \text{det } (\text{submatrix } A\ I'\ J')$  **unfolding**  
*sub-index* .  
**show** *?thesis*  
**proof** (*cases det (submatrix A I' J') = 0*)  
**case** *True*  
**then show** *?thesis* **using** *det-submatrix-IJ* **by** *auto*  
**next**  
**case** *False* **note**  $\text{det-not0} = \text{False}$   
**from this obtain**  $xs$  **where** *prod-list-xs: det (submatrix A I' J') = ( $\prod i \in xs.$*   
 $A\ \$\$ (i, i))$   
 $\vee \text{det } (\text{submatrix } A\ I'\ J') = - (\prod i \in xs. A\ \$\$ (i, i))$   
**and**  $xs \subseteq \{0..<min\ n\ m\}$   
**and** *length-xs: card xs = k*  
**and**  $xs-I': xs = I'$   
**using** *det-sub-I'J'* **by** *blast*  
**let**  $?ys = \text{insert } a\ xs$   
**have**  $a\ \text{notin-}xs: a \notin xs$   
**by** (*simp add: xs-I' a-notin*)  
**have**  $\text{length-ys: } \text{card } ?ys = Suc\ k$   
**using**  $\text{length-xs } a\ \text{notin-}xs$  **by** (*simp add: card-ge-0-finite k0*)  
**have**  $a\ \text{eq-b: } a = b$   
**using**  $A\ \text{carrier } an\ bm\ \text{sub-IJ-0 } \text{sub-index } dA$  **unfolding** *diagonal-mat-def*  
**by** *auto*  
**have**  $A\ \text{aa-in: } A\ \$\$ (a, a) \in \{A\ \$\$ (i, i) \mid i. i < min\ n\ m \wedge A\ \$\$ (i, i) \neq 0\}$   
**using**  $a\ \text{eq-b } an\ bm\ \text{sub-IJ-0 } \text{sub-index}$  **by** *auto*  
**show** *?thesis*  
**proof** (*cases even (a'+b')*)  
**case** *True*  
**have**  $\text{det-submatrix-IJ: } \text{det } (\text{submatrix } A\ I\ J) = A\ \$\$ (a, b) * \text{det } (\text{submatrix } A\ I'\ J')$   
**using** *det-submatrix-IJ True* **by** *auto*  
**show** *?thesis*  
**proof** (*cases det (submatrix A I' J') = ( $\prod i \in xs. A\ \$\$ (i, i)$ )*)  
**case** *True*  
**have**  $\text{det } (\text{submatrix } A\ I\ J) = (\prod i \in ?ys. A\ \$\$ (i, i))$

```

    using det-submatrix-IJ unfolding True a-eq-b
    by (metis (no-types, lifting) a-notin-xs a-eq-b
        card-ge-0-finite k0 length-xs prod.insert)
    then show ?thesis using length-ys
    using a-eq-b an bm xs xs-I'
    by (simp add: aI')
  next
    case False
    hence det (submatrix A I' J') = - (∏ i∈xs. A $$ (i, i)) using prod-list-xs
  by simp
    hence det (submatrix A I J) = -(∏ i∈?ys. A $$ (i, i)) using
  det-submatrix-IJ a-eq-b
    by (metis (no-types, lifting) a-notin-xs card-ge-0-finite k0
        length-xs mult-minus-right prod.insert)
    then show ?thesis using length-ys
    using a-eq-b an bm xs aI' xs-I' by force
  qed
  next
    case False
    have det-submatrix-IJ: det (submatrix A I J) = A $$ (a, b) * - det
  (submatrix A I' J')
    using det-submatrix-IJ False by auto
    show ?thesis
    proof (cases det (submatrix A I' J') = (∏ i∈xs. A $$ (i, i)))
      case True
      have det (submatrix A I J) = - (∏ i∈?ys. A $$ (i, i))
      using det-submatrix-IJ unfolding True
      by (metis (no-types, lifting) a-eq-b a-notin-xs card-ge-0-finite k0
          length-xs mult-minus-right prod.insert)
      then show ?thesis using length-ys
      using a-eq-b an bm xs aI' xs-I' by force
    next
      case False
      hence det (submatrix A I' J') = - (∏ i∈xs. A $$ (i, i)) using prod-list-xs
    by simp
      hence det (submatrix A I J) = (∏ i∈?ys. A $$ (i, i)) using det-submatrix-IJ
      by (metis (mono-tags, lifting) a-eq-b a-notin-xs card-ge-0-finite
          equation-minus-iff k0 length-xs prod.insert)
      then show ?thesis using length-ys
      using a-eq-b an bm xs aI' xs-I' by force
    qed
  qed
  next
    case False
    have sum ?f {0..<Suc k} = 0 by (rule sum.neutral, insert False, auto)
    thus ?thesis using det-rw
    by (simp add: atLeast0LessThan)
  qed

```

qed  
qed

### 10.3 Relating minors and GCD

lemma *diagonal-dvd-Gcd-minors*:

```

  fixes A::'a::{semiring-Gcd,comm-ring-1} mat
  assumes A: A ∈ carrier-mat n m
  and SNF-A: Smith-normal-form-mat A
shows (∏ i=0..<k. A $$ (i,i)) dvd Gcd (minors A k)
proof (cases k=0)
  case True
  then show ?thesis by auto
next
  case False
  hence k: 0<k by simp
  show ?thesis
  proof (rule Gcd-greatest)
    have diag-A: diagonal-mat A
    using SNF-A unfolding Smith-normal-form-mat-def isDiagonal-mat-def di-
agonal-mat-def by auto
    fix b assume b-in-minors: b ∈ minors A k
    show (∏ i = 0..<k. A $$ (i, i)) dvd b
    proof (cases b=0)
      case True
      then show ?thesis by auto
    next
      case False
      obtain I J where b: b = det (submatrix A I J) and I: I ⊆ {0..<dim-row A}

and J: J ⊆ {0..<dim-col A} and Ik: card I = k and Jk: card J = k
      using b-in-minors unfolding minors-def by blast
      obtain S where S: S ⊆ {0..<min n m} and Sk: card S = k
      and det-subS: det (submatrix A I J) = (∏ i∈S. A $$ (i,i))
      ∨ det (submatrix A I J) = -(∏ i∈S. A $$ (i,i))
      using det-minors-diagonal2[OF diag-A A Ik Jk - - k] I J A False b by auto
      obtain f where inj-f: inj-on f {0..<k} and fk-S: f'{0..<k} = S
      and i-fi: (∀ i∈{0..<k}. i ≤ f i) using exists-inj-ge-index[OF S Sk] by blast
      have (∏ i = 0..<k. A $$ (i, i)) dvd (∏ i∈{0..<k}. A $$ (f i, f i))
      by (rule prod-dvd-prod, rule SNF-divides-diagonal[OF A SNF-A], insert fk-S
S i-fi, force+)
      also have ... = (∏ i∈f'{0..<k}. A $$ (i,i))
      by (rule prod.reindex[symmetric, unfolded o-def, OF inj-f])
      also have ... = (∏ i∈S. A $$ (i, i)) using fk-S by auto
      finally have *: (∏ i = 0..<k. A $$ (i, i)) dvd (∏ i∈S. A $$ (i, i)) .
      show (∏ i = 0..<k. A $$ (i, i)) dvd b using det-subS b * by auto
    qed
  qed
qed

```

**lemma** *Gcd-minors-dvd-diagonal*:

**fixes**  $A::'a::\{\text{semiring-Gcd,comm-ring-1}\}$  *mat*

**assumes**  $A: A \in \text{carrier-mat } n \ m$

**and**  $\text{SNF-A: Smith-normal-form-mat } A$

**and**  $k: k \leq \min \ n \ m$

**shows**  $\text{Gcd}(\text{minors } A \ k) \ \text{dvd} \ (\prod_{i=0..<k.} A \ \text{\$ \$ } (i,i))$

**proof** (*rule Gcd-dvd*)

**define**  $I$  **where**  $I = \{0..<k\}$

**have**  $(\prod_{i=0..<k.} A \ \text{\$ \$ } (i, i)) = \text{det}(\text{submatrix } A \ I \ I)$

**proof** –

**have** *sub-eq*:  $\text{submatrix } A \ I \ I = \text{mat } k \ k \ (\lambda(i, j). A \ \text{\$ \$ } (i, j))$

**proof** (*rule eq-matI, auto*)

**have**  $I = \{i. i < \text{dim-row } A \wedge i \in I\}$  **unfolding** *I-def* **using**  $A \ k$  **by** *auto*

**hence** *ck*:  $\text{card } \{i. i < \text{dim-row } A \wedge i \in I\} = k$

**unfolding** *I-def* **using** *card-atLeastLessThan* **by** *presburger*

**have**  $I = \{i. i < \text{dim-col } A \wedge i \in I\}$  **unfolding** *I-def* **using**  $A \ k$  **by** *auto*

**hence** *ck2*:  $\text{card } \{j. j < \text{dim-col } A \wedge j \in I\} = k$

**unfolding** *I-def* **using** *card-atLeastLessThan* **by** *presburger*

**show** *dr*:  $\text{dim-row}(\text{submatrix } A \ I \ I) = k$  **using** *ck* **unfolding** *submatrix-def*

**by** *auto*

**show** *dc*:  $\text{dim-col}(\text{submatrix } A \ I \ I) = k$  **using** *ck2* **unfolding** *submatrix-def*

**by** *auto*

**fix**  $i \ j$  **assume**  $i: i < k$  **and**  $j: j < k$

**have** *p1*:  $\text{pick } I \ i = i$

**proof** –

**have**  $\{0..<i\} = \{a \in I. a < i\}$  **using** *I-def i* **by** *auto*

**hence** *i-eq*:  $i = \text{card } \{a \in I. a < i\}$

**by** (*metis card-atLeastLessThan diff-zero*)

**have**  $\text{pick } I \ i = \text{pick } I \ (\text{card } \{a \in I. a < i\})$  **using** *i-eq* **by** *simp*

**also have**  $\dots = i$  **by** (*rule pick-card-in-set, insert i I-def, simp*)

**finally show** *?thesis* .

**qed**

**have** *p2*:  $\text{pick } I \ j = j$

**proof** –

**have**  $\{0..<j\} = \{a \in I. a < j\}$  **using** *I-def j* **by** *auto*

**hence** *j-eq*:  $j = \text{card } \{a \in I. a < j\}$

**by** (*metis card-atLeastLessThan diff-zero*)

**have**  $\text{pick } I \ j = \text{pick } I \ (\text{card } \{a \in I. a < j\})$  **using** *j-eq* **by** *simp*

**also have**  $\dots = j$  **by** (*rule pick-card-in-set, insert j I-def, simp*)

**finally show** *?thesis* .

**qed**

**have**  $\text{submatrix } A \ I \ I \ \text{\$ \$ } (i, j) = A \ \text{\$ \$ } (\text{pick } I \ i, \text{pick } I \ j)$

**proof** (*rule submatrix-index*)

**show**  $i < \text{card } \{i. i < \text{dim-row } A \wedge i \in I\}$  **by** (*metis dim-submatrix(1) dr*

*i*)

**show**  $j < \text{card } \{j. j < \text{dim-col } A \wedge j \in I\}$  **by** (*metis dim-submatrix(2) dc j*)

**qed**

**also have**  $\dots = A \text{ $$$ } (i, j)$  **using**  $p1\ p2$  **by** *simp*  
**finally show**  $\text{submatrix } A \ I \ I \ \text{$$$ } (i, j) = A \ \text{$$$ } (i, j)$  .  
**qed**  
**hence**  $\det (\text{submatrix } A \ I \ I) = \det (\text{mat } k \ k \ (\lambda(i, j). \ A \ \text{$$$ } (i, j)))$  **by** *simp*  
**also have**  $\dots = \text{prod-list } (\text{diag-mat } (\text{mat } k \ k \ (\lambda(i, j). \ A \ \text{$$$ } (i, j))))$   
**proof** (*rule det-upper-triangular*)  
**show**  $\text{mat } k \ k \ (\lambda(i, j). \ A \ \text{$$$ } (i, j)) \in \text{carrier-mat } k \ k$  **by** *auto*  
**show** *upper-triangular* (*Matrix.mat*  $k \ k \ (\lambda(i, j). \ A \ \text{$$$ } (i, j))$ )  
**using** *SNF-A* *A* *k* **unfolding** *Smith-normal-form-mat-def* *isDiagonal-mat-def*  
**by** *auto*  
**qed**  
**also have**  $\dots = (\prod i = 0..<k. \ A \ \text{$$$ } (i, i))$   
**by** (*metis* (*mono-tags*, *lifting*) *atLeastLessThan-iff dim-row-mat(1) index-mat(1)*  
*prod.cong prod-list-diag-prod split-conv*)  
**finally show** *?thesis ..*  
**qed**  
**moreover have**  $I \subseteq \{0..<\text{dim-row } A\}$  **using**  $k \ A \ I\text{-def}$  **by** *auto*  
**moreover have**  $I \subseteq \{0..<\text{dim-col } A\}$  **using**  $k \ A \ I\text{-def}$  **by** *auto*  
**moreover have**  $\text{card } I = k$  **using**  $I\text{-def}$  **by** *auto*  
**ultimately show**  $(\prod i = 0..<k. \ A \ \text{$$$ } (i, i)) \in \text{minors } A \ k$  **unfolding** *minors-def*  
**by** *auto*  
**qed**

**lemma** *Gcd-minors-A-dvd-Gcd-minors-PAQ*:  
**fixes**  $A::'a::\{\text{semiring-Gcd, comm-ring-1}\}$  *mat*  
**assumes**  $A: A \in \text{carrier-mat } m \ n$   
**and**  $P: P \in \text{carrier-mat } m \ m$  **and**  $Q: Q \in \text{carrier-mat } n \ n$   
**shows**  $\text{Gcd } (\text{minors } A \ k) \ \text{dvd} \ \text{Gcd } (\text{minors } (P * A * Q) \ k)$   
**proof** (*rule Gcd-greatest*)  
**let**  $?B = (P * A * Q)$   
**fix**  $b$  **assume**  $b \in \text{minors } ?B \ k$   
**from this obtain**  $I \ J$  **where**  $b = \det (\text{submatrix } ?B \ I \ J)$  **and**  $I: I \subseteq \{0..<\text{dim-row } ?B\}$   
**and**  $J: J \subseteq \{0..<\text{dim-col } ?B\}$  **and**  $I_k: \text{card } I = k$  **and**  $J_k: \text{card } J = k$   
**unfolding** *minors-def* **by** *blast*  
**have**  $\text{Gcd } (\text{minors } A \ k) \ \text{dvd} \ \det (\text{submatrix } ?B \ I \ J)$   
**by** (*rule Gcd-minors-dvd* [*OF - P A Q - - I k J k*], *insert A I J P Q, auto*)  
**thus**  $\text{Gcd } (\text{minors } A \ k) \ \text{dvd} \ b$  **using**  $b$  **by** *simp*  
**qed**

**lemma** *Gcd-minors-PAQ-dvd-Gcd-minors-A*:  
**fixes**  $A::'a::\{\text{semiring-Gcd, comm-ring-1}\}$  *mat*  
**assumes**  $A: A \in \text{carrier-mat } m \ n$   
**and**  $P: P \in \text{carrier-mat } m \ m$   
**and**  $Q: Q \in \text{carrier-mat } n \ n$   
**and** *inv-P: invertible-mat P*

**and** *inv-Q*: *invertible-mat Q*  
**shows** *Gcd (minors (P\*A\*Q) k) dvd Gcd (minors A k)*  
**proof** (*rule Gcd-greatest*)  
**let** *?B = P \* A \* Q*  
**fix** *b* **assume** *b ∈ minors A k*  
**from** *this* **obtain** *I J* **where** *b = det (submatrix A I J)* **and** *I ⊆ {0..<dim-row A}*  
**and** *J: J ⊆ {0..<dim-col A}* **and** *Ik: card I = k* **and** *Jk: card J = k*  
**unfolding** *minors-def* **by** *blast*  
**obtain** *P'* **where** *PP': inverts-mat P P'* **and** *P'P: inverts-mat P' P*  
**using** *inv-P* **unfolding** *invertible-mat-def* **by** *auto*  
**obtain** *Q'* **where** *QQ': inverts-mat Q Q'* **and** *Q'Q: inverts-mat Q' Q*  
**using** *inv-Q* **unfolding** *invertible-mat-def* **by** *auto*  
**have** *P': P' ∈ carrier-mat m m* **using** *PP' P'P* **unfolding** *inverts-mat-def*  
**by** (*metis P carrier-matD(1) carrier-matD(2) carrier-matI index-mult-mat(3)*)  
*index-one-mat(3)*  
**have** *Q': Q' ∈ carrier-mat n n*  
**using** *QQ' Q'Q* **unfolding** *inverts-mat-def*  
**by** (*metis Q carrier-matD(1) carrier-matD(2) carrier-matI index-mult-mat(3)*)  
*index-one-mat(3)*  
**have** *rw: P' \* ?B \* Q' = A*  
**proof** –  
**have** *f1: P' \* P = 1\_m m*  
**by** (*metis (no-types) P' P'P carrier-matD(1) inverts-mat-def*)  
**have** *\**: *P' \* P \* A = P' \* (P \* A)*  
**by** (*meson A P P' assoc-mult-mat*)  
**have** *P' \* (P \* A \* Q) \* Q' = P' \* P \* A \* Q \* Q'*  
**by** (*smt A P P' Q assoc-mult-mat mult-carrier-mat*)  
**also have** *... = P' \* P \* (A \* Q \* Q')*  
**using** *A P P' Q Q' f1 \** **by** *auto*  
**also have** *... = A \* Q \* Q'* **using** *P'P A P'* **unfolding** *inverts-mat-def* **by**  
*auto*  
**also have** *... = A* **using** *QQ' A Q' Q* **unfolding** *inverts-mat-def* **by** *auto*  
**finally show** *?thesis* .  
**qed**  
**have** *Gcd (minors ?B k) dvd det (submatrix (P'\*?B\*Q') I J)*  
**by** (*rule Gcd-minors-dvd[OF - P' - Q' - - Ik Jk], insert P A Q I J, auto*)  
**also have** *... = det (submatrix A I J)* **using** *rw* **by** *simp*  
**finally show** *Gcd (minors ?B k) dvd b* **using** *b* **by** *simp*  
**qed**

**lemma** *Gcd-minors-dvd-diag-PAQ*:

**fixes** *P A Q::'a:: {semiring-Gcd, comm-ring-1} mat*  
**assumes** *A: A ∈ carrier-mat m n*  
**and** *P: P ∈ carrier-mat m m*  
**and** *Q: Q ∈ carrier-mat n n*  
**and** *SNF: Smith-normal-form-mat (P\*A\*Q)*  
**and** *k: k ≤ min m n*  
**shows** *Gcd (minors A k) dvd (∏ i=0..<k. (P \* A \* Q) \$\$ (i,i))*

**proof** –  
**have**  $Gcd (minors A k) \text{ dvd } Gcd (minors (P * A * Q) k)$   
**by** (rule *Gcd-minors-A-dvd-Gcd-minors-PAQ*[*OF A P Q*])  
**also have** ...  $\text{dvd } (\prod_{i=0..<k}. (P*A*Q) \text{ \$(\$ (i,i))})$   
**by** (rule *Gcd-minors-dvd-diagonal*[*OF - SNF k*], *insert P A Q, auto*)  
**finally show** ?thesis .  
**qed**

**lemma** *diag-PAQ-dvd-Gcd-minors*:  
**fixes**  $P A Q::'a::\{\text{semiring-Gcd, comm-ring-1}\}$  *mat*  
**assumes**  $A: A \in \text{carrier-mat } m \ n$   
**and**  $P: P \in \text{carrier-mat } m \ m$   
**and**  $Q: Q \in \text{carrier-mat } n \ n$   
**and** *inv-P*: *invertible-mat P*  
**and** *inv-Q*: *invertible-mat Q*  
**and** *SNF*: *Smith-normal-form-mat (P\*A\*Q)*  
**shows**  $(\prod_{i=0..<k}. (P * A * Q) \text{ \$(\$ (i,i))}) \text{ dvd } Gcd (minors A k)$   
**proof** –  
**have**  $(\prod_{i=0..<k}. (P*A*Q) \text{ \$(\$ (i,i))}) \text{ dvd } Gcd (minors (P * A * Q) k)$   
**by** (rule *diagonal-dvd-Gcd-minors*[*OF - SNF*], *auto*)  
**also have** ...  $\text{dvd } Gcd (minors A k)$   
**by** (rule *Gcd-minors-PAQ-dvd-Gcd-minors-A*[*OF - - - inv-P inv-Q*], *insert P A Q, auto*)  
**finally show** ?thesis .  
**qed**

**lemma** *Smith-prod-zero-imp-last-zero*:  
**fixes**  $A::'a::\{\text{semidom, comm-ring-1}\}$  *mat*  
**assumes**  $A: A \in \text{carrier-mat } m \ n$   
**and** *SNF*: *Smith-normal-form-mat A*  
**and** *prod-0*:  $(\prod_{j=0..<Suc \ i}. A \text{ \$(\$ (j,j))}) = 0$   
**and**  $i: i < \min \ m \ n$   
**shows**  $A \text{ \$(\$ (i,i))} = 0$   
**proof** –  
**obtain**  $j$  **where**  $A_{jj}: A \text{ \$(\$ (j,j))} = 0$  **and**  $j: j < Suc \ i$  **using** *prod-0 prod-zero-iff* **by** *auto*  
**show**  $A \text{ \$(\$ (i,i))} = 0$  **by** (rule *Smith-zero-imp-zero*[*OF A SNF A\_{jj} i*], *insert j, auto*)  
**qed**

## 10.4 Final theorem

**lemma** *Smith-normal-form-uniqueness-aux*:  
**fixes**  $P A Q::'a::\{\text{idom, semiring-Gcd}\}$  *mat*  
**assumes**  $A: A \in \text{carrier-mat } m \ n$



**and**  $P$ :  $P \in \text{carrier-mat } m \ m$   
**and**  $Q$ :  $Q \in \text{carrier-mat } n \ n$   
**and**  $\text{inv-}P$ :  $\text{invertible-mat } P$   
**and**  $\text{inv-}Q$ :  $\text{invertible-mat } Q$   
**and**  $PAQ\text{-}B$ :  $P * A * Q = B$   
**and**  $SNF$ :  $\text{Smith-normal-form-mat } B$

**and**  $P'$ :  $P' \in \text{carrier-mat } m \ m$   
**and**  $Q'$ :  $Q' \in \text{carrier-mat } n \ n$   
**and**  $\text{inv-}P'$ :  $\text{invertible-mat } P'$   
**and**  $\text{inv-}Q'$ :  $\text{invertible-mat } Q'$   
**and**  $P'AQ'\text{-}B'$ :  $P' * A * Q' = B'$   
**and**  $SNF\text{-}B'$ :  $\text{Smith-normal-form-mat } B'$   
**and**  $k$ :  $k < \min \ m \ n$

**shows**  $\forall i \leq k. B \ \$\$ (i, i) \ \text{dvd} \ B' \ \$\$ (i, i) \wedge B' \ \$\$ (i, i) \ \text{dvd} \ B \ \$\$ (i, i)$   
**proof** (*rule allI, rule impI*)  
**fix**  $i$  **assume**  $ik$ :  $i \leq k$   
**show**  $B \ \$\$ (i, i) \ \text{dvd} \ B' \ \$\$ (i, i) \wedge B' \ \$\$ (i, i) \ \text{dvd} \ B \ \$\$ (i, i)$   
**proof** –  
**let**  $\ ?\Pi B i = (\prod i=0..<i. B \ \$\$ (i, i))$   
**let**  $\ ?\Pi B' i = (\prod i=0..<i. B' \ \$\$ (i, i))$   
**have**  $\ ?\Pi B' i \ \text{dvd} \ \text{Gcd} (\text{minors } A \ i)$   
**by** (*unfold P'AQ'-B'[symmetric], rule diag-PAQ-dvd-Gcd-minors[OF A P' Q' inv-P' inv-Q]*,  
*insert P'AQ'-B' SNF-B' ik k, auto* )  
**also have** ...  $\ \text{dvd} \ ?\Pi B i$   
**by** (*unfold PAQ-B[symmetric], rule Gcd-minors-dvd-diag-PAQ[OF A P Q]*,  
*insert PAQ-B SNF ik k, auto* )  
**finally have**  $B'\text{-}i\text{-dvd-}B\text{-}i$ :  $\ ?\Pi B' i \ \text{dvd} \ ?\Pi B i$  .  
**have**  $\ ?\Pi B i \ \text{dvd} \ \text{Gcd} (\text{minors } A \ i)$   
**by** (*unfold PAQ-B[symmetric], rule diag-PAQ-dvd-Gcd-minors[OF A P Q inv-P inv-Q]*,  
*insert PAQ-B SNF ik k, auto* )  
**also have** ...  $\ \text{dvd} \ ?\Pi B' i$   
**by** (*unfold P'AQ'-B'[symmetric], rule Gcd-minors-dvd-diag-PAQ[OF A P' Q']*,  
*insert P'AQ'-B' SNF-B' ik k, auto* )  
**finally have**  $B\text{-}i\text{-dvd-}B'\text{-}i$ :  $\ ?\Pi B i \ \text{dvd} \ ?\Pi B' i$  .  
**let**  $\ ?\Pi B\text{-}Suc = (\prod i=0..<Suc \ i. B \ \$\$ (i, i))$   
**let**  $\ ?\Pi B'\text{-}Suc = (\prod i=0..<Suc \ i. B' \ \$\$ (i, i))$   
**have**  $\ ?\Pi B'\text{-}Suc \ \text{dvd} \ \text{Gcd} (\text{minors } A \ (Suc \ i))$   
**by** (*unfold P'AQ'-B'[symmetric], rule diag-PAQ-dvd-Gcd-minors[OF A P' Q' inv-P' inv-Q]*,  
*insert P'AQ'-B' SNF-B' ik k, auto* )  
**also have** ...  $\ \text{dvd} \ ?\Pi B\text{-}Suc$   
**by** (*unfold PAQ-B[symmetric], rule Gcd-minors-dvd-diag-PAQ[OF A P Q]*,  
*insert PAQ-B SNF ik k, auto* )  
**finally have**  $\ ?\Pi B'\text{-}Suc \ \text{dvd} \ ?\Pi B\text{-}Suc$  .  
**have**  $\ ?\Pi B\text{-}Suc \ \text{dvd} \ \text{Gcd} (\text{minors } A \ (Suc \ i))$

```

    by (unfold PAQ-B[symmetric], rule diag-PAQ-dvd-Gcd-minors[OF A P Q
inv-P inv-Q],
        insert PAQ-B SNF ik k, auto )
  also have ... dvd ? $\Pi$ B'-Suc
    by (unfold P'AQ'-B'[symmetric], rule Gcd-minors-dvd-diag-PAQ[OF A P'
Q'],
        insert P'AQ'-B' SNF-B' ik k, auto)
  finally have  $\downarrow$ : ? $\Pi$ B-Suc dvd ? $\Pi$ B'-Suc .
  show ?thesis
  proof (cases ? $\Pi$ B-Suc = 0)
    case True
      have True2: ? $\Pi$ B'-Suc = 0 using  $\downarrow$  True by fastforce
      have B$$$(i,i) = 0
        by (rule Smith-prod-zero-imp-last-zero[OF - SNF True], insert ik k PAQ-B
P Q, auto)
      moreover have B'$$$(i,i) = 0
        by (rule Smith-prod-zero-imp-last-zero[OF - SNF-B' True2],
            insert ik k P'AQ'-B' P' Q', auto)
      ultimately show ?thesis by auto
    next
      case False
        have  $\exists u. u \text{ dvd } 1 \wedge ?\Pi B' i = u * ?\Pi B i$ 
          by (rule dvd-associated2[OF B'-i-dvd-B-i B-i-dvd-B'-i], insert False B'-i-dvd-B-i,
force)
        from this obtain u where eq1:  $(\prod_{i=0..<i} B' \$\$ (i,i)) = u * (\prod_{i=0..<i} B \$\$ (i,i))$ 
          and u-dvd-1: u dvd 1 by blast
        have  $\exists u. u \text{ dvd } 1 \wedge ?\Pi B-Suc = u * ?\Pi B'-Suc$ 
          by (rule dvd-associated2[OF  $\downarrow$  3 False])
        from this obtain w where eq2:  $(\prod_{i=0..<Suc\ i} B \$\$ (i,i)) = w * (\prod_{i=0..<Suc\ i} B' \$\$ (i,i))$ 
          and w-dvd-1: w dvd 1 by blast
        have  $B \$\$ (i, i) * (\prod_{i=0..<i} B \$\$ (i,i)) = (\prod_{i=0..<Suc\ i} B \$\$ (i,i))$ 
          by (simp add: prod.atLeast0-lessThan-Suc ik)
        also have ... = w *  $(\prod_{i=0..<Suc\ i} B' \$\$ (i,i))$  unfolding eq2 by auto
        also have ... = w *  $(B' \$\$ (i,i) * (\prod_{i=0..<i} B' \$\$ (i,i)))$ 
          by (simp add: prod.atLeast0-lessThan-Suc ik)
        also have ... = w *  $(B' \$\$ (i,i) * u * (\prod_{i=0..<i} B \$\$ (i,i)))$ 
          unfolding eq1 by auto
        finally have  $B \$\$ (i,i) = w * u * B' \$\$ (i,i)$ 
          using False by auto
        moreover have w*u dvd 1 using u-dvd-1 w-dvd-1 by auto
        ultimately have  $\exists u. \text{is-unit } u \wedge B \$\$ (i, i) = u * B' \$\$ (i, i)$  by auto
        thus ?thesis using dvd-associated2 by force
      qed
    qed
  qed

```

**lemma** *Smith-normal-form-uniqueness*:  
**fixes**  $P A Q::'a::\{idom,semiring-Gcd\}$  *mat*  
**assumes**  $A: A \in carrier\text{-}mat\ m\ n$

**and**  $P: P \in carrier\text{-}mat\ m\ m$   
**and**  $Q: Q \in carrier\text{-}mat\ n\ n$   
**and**  $inv\text{-}P: invertible\text{-}mat\ P$   
**and**  $inv\text{-}Q: invertible\text{-}mat\ Q$   
**and**  $PAQ\text{-}B: P*A*Q = B$   
**and**  $SNF: Smith\text{-}normal\text{-}form\text{-}mat\ B$

**and**  $P': P' \in carrier\text{-}mat\ m\ m$   
**and**  $Q': Q' \in carrier\text{-}mat\ n\ n$   
**and**  $inv\text{-}P': invertible\text{-}mat\ P'$   
**and**  $inv\text{-}Q': invertible\text{-}mat\ Q'$   
**and**  $P'AQ'\text{-}B': P'*A*Q' = B'$   
**and**  $SNF\text{-}B': Smith\text{-}normal\text{-}form\text{-}mat\ B'$   
**and**  $i: i < \min\ m\ n$

**shows**  $\exists u. u\ dvd\ 1 \wedge B\ \$\$ (i,i) = u * B'\ \$\$ (i,i)$

**proof** (*cases*  $B\ \$\$ (i,i) = 0$ )

**case** *True*

**let**  $?PIB\text{-}Suc = (\prod_{i=0..<Suc\ i} B\ \$\$ (i,i))$   
**let**  $?PIB'\text{-}Suc = (\prod_{i=0..<Suc\ i} B'\ \$\$ (i,i))$   
**have**  $?PIB\text{-}Suc\ dvd\ Gcd\ (minors\ A\ (Suc\ i))$   
**by** (*unfold*  $PAQ\text{-}B[symmetric]$ , *rule*  $diag\text{-}PAQ\text{-}dvd\text{-}Gcd\text{-}minors[OF\ A\ P\ Q\ inv\text{-}P\ inv\text{-}Q]$ ,  
*insert*  $PAQ\text{-}B\ SNF\ i,$  *auto*)

**also have** ... *dvd*  $?PIB'\text{-}Suc$   
**by** (*unfold*  $P'AQ'\text{-}B'[symmetric]$ , *rule*  $Gcd\text{-}minors\text{-}dvd\text{-}diag\text{-}PAQ[OF\ A\ P'\ Q']$ ,  
*insert*  $P'AQ'\text{-}B'\ SNF\text{-}B'\ i,$  *auto*)

**finally have**  $\_4: ?PIB\text{-}Suc\ dvd\ ?PIB'\text{-}Suc$  .

**have** *prod0*:  $?PIB\text{-}Suc = 0$  **using** *True* **by** *auto*

**have** *True2*:  $?PIB'\text{-}Suc = 0$  **using**  $\_4$  **by** (*metis*  $dvd\text{-}0\text{-}left\text{-}iff\ prod0$ )

**have**  $B'\ \$\$ (i,i) = 0$   
**by** (*rule*  $Smith\text{-}prod\text{-}zero\text{-}imp\text{-}last\text{-}zero[OF\ \text{-}\ SNF\text{-}B'\ True2]$ ,  
*insert*  $i\ P'AQ'\text{-}B'\ P'\ Q',$  *auto*)

**thus** *thesis* **using** *True* **by** *auto*

**next**

**case** *False*

**have**  $\forall a \leq i. B\ \$\$ (a,a)\ dvd\ B'\ \$\$ (a,a) \wedge B'\ \$\$ (a,a)\ dvd\ B\ \$\$ (a,a)$   
**by** (*rule*  $Smith\text{-}normal\text{-}form\text{-}uniqueness\text{-}aux[OF\ assms]$ )

**hence**  $B\ \$\$ (i,i)\ dvd\ B'\ \$\$ (i,i) \wedge B'\ \$\$ (i,i)\ dvd\ B\ \$\$ (i,i)$  **using**  $i$  **by** *auto*

**thus** *thesis* **using**  $dvd\text{-}associated2\ False$  **by** *blast*

**qed**

The final theorem, moved to HOL Analysis

**lemma** *Smith-normal-form-uniqueness-HOL-Analysis*:  
**fixes**  $A::'a::\{idom,semiring-Gcd\}$   $\sim m::mod\text{-}type\ \sim n::mod\text{-}type$   
**and**  $P P'::'a\ \sim n::mod\text{-}type\ \sim n::mod\text{-}type$

```

and Q Q'::'a^m::mod-type^m::mod-type
assumes

  inv-P: invertible P
  and inv-Q: invertible Q
  and PAQ-B: P**A**Q = B
  and SNF: Smith-normal-form B

  and inv-P': invertible P'
  and inv-Q': invertible Q'
  and P'AQ'-B': P'*A**Q' = B'
  and SNF-B': Smith-normal-form B'
  and i: i < min (nrows A) (ncols A)
shows  $\exists u. u \text{ dvd } 1 \wedge B \text{ \$h Mod-Type.from-nat } i \text{ \$h Mod-Type.from-nat } i$ 
= u * B' \$h Mod-Type.from-nat i \$h Mod-Type.from-nat i
proof -
  let ?P = Mod-Type-Connect.from-hmam P
  let ?A = Mod-Type-Connect.from-hmam A
  let ?Q = Mod-Type-Connect.from-hmam Q
  let ?B = Mod-Type-Connect.from-hmam B
  let ?P' = Mod-Type-Connect.from-hmam P'
  let ?Q' = Mod-Type-Connect.from-hmam Q'
  let ?B' = Mod-Type-Connect.from-hmam B'
  let ?i = (Mod-Type.from-nat i)::'n
  let ?i' = (Mod-Type.from-nat i)::'m
  have [transfer-rule]: Mod-Type-Connect.HMA-M ?P P by (simp add: Mod-Type-Connect.HMA-M-def)
  have [transfer-rule]: Mod-Type-Connect.HMA-M ?A A by (simp add: Mod-Type-Connect.HMA-M-def)
  have [transfer-rule]: Mod-Type-Connect.HMA-M ?Q Q by (simp add: Mod-Type-Connect.HMA-M-def)
  have [transfer-rule]: Mod-Type-Connect.HMA-M ?B B by (simp add: Mod-Type-Connect.HMA-M-def)
  have [transfer-rule]: Mod-Type-Connect.HMA-M ?P' P' by (simp add: Mod-Type-Connect.HMA-M-def)
  have [transfer-rule]: Mod-Type-Connect.HMA-M ?Q' Q' by (simp add: Mod-Type-Connect.HMA-M-def)
  have [transfer-rule]: Mod-Type-Connect.HMA-M ?B' B' by (simp add: Mod-Type-Connect.HMA-M-def)
  have [transfer-rule]: Mod-Type-Connect.HMA-I i ?i
    by (metis Mod-Type-Connect.HMA-I-def i min.strict-boundedE
        mod-type-class.to-nat-from-nat-id nrows-def)
  have [transfer-rule]: Mod-Type-Connect.HMA-I i ?i'
    by (metis Mod-Type-Connect.HMA-I-def i min.strict-boundedE
        mod-type-class.to-nat-from-nat-id ncols-def)
  have i2: i < min CARD('m) CARD('n) using i unfolding nrows-def ncols-def
by auto
  have  $\exists u. u \text{ dvd } 1 \wedge ?B \text{ \$\$}(i,i) = u * ?B' \text{ \$\$}(i,i)$ 
proof (rule Smith-normal-form-uniqueness[of - CARD('n) CARD('m)])
  show ?P*?A*?Q=?B using PAQ-B by (transfer', auto)
  show Smith-normal-form-mat ?B using SNF by (transfer', auto)
  show ?P'*?A*?Q'=?B' using P'AQ'-B' by (transfer', auto)
  show Smith-normal-form-mat ?B' using SNF-B' by (transfer', auto)
  show invertible-mat ?P using inv-P by (transfer, auto)
  show invertible-mat ?P' using inv-P' by (transfer, auto)
  show invertible-mat ?Q using inv-Q by (transfer, auto)

```

```

    show invertible-mat ?Q' using inv-Q' by (transfer, auto)
  qed (insert i2, auto)
  hence  $\exists u. u \text{ dvd } 1 \wedge (\text{index-hma } B \text{ ?i ?i}') = u * (\text{index-hma } B' \text{ ?i ?i}')$  by
  (transfer', rule)
  thus ?thesis unfolding index-hma-def by simp
  qed

```

## 10.5 Uniqueness fixing a complete set of non-associates

**definition** *Smith-normal-form-wrt*  $A \mathcal{Q} = ($   
 $(\forall a b. \text{Mod-Type.to-nat } a = \text{Mod-Type.to-nat } b \wedge \text{Mod-Type.to-nat } a + 1 <$   
 $nrows A$   
 $\wedge \text{Mod-Type.to-nat } b + 1 < ncols A \longrightarrow A \ \$h \ a \ \$h \ b \ \text{dvd} \ A \ \$h \ (a+1)$   
 $\ \$h \ (b+1))$   
 $\wedge \text{isDiagonal } A \wedge \text{Complete-set-non-associates } \mathcal{Q}$   
 $\wedge (\forall a b. \text{Mod-Type.to-nat } a = \text{Mod-Type.to-nat } b \wedge \text{Mod-Type.to-nat } a < \min$   
 $(nrows A) (ncols A)$   
 $\wedge \text{Mod-Type.to-nat } b < \min (nrows A) (ncols A) \longrightarrow A \ \$h \ a \ \$h \ b \in \mathcal{Q})$   
 $)$

**lemma** *Smith-normal-form-wrt-uniqueness-HOL-Analysis:*

```

  fixes  $A::'a::\{idom, semiring-Gcd\}^m::\text{mod-type}^n::\text{mod-type}$ 
  and  $P P'::'a^m::\text{mod-type}^n::\text{mod-type}$ 
  and  $Q Q'::'a^m::\text{mod-type}^m::\text{mod-type}$ 
  assumes

```

```

  P: invertible P
  and Q: invertible Q
  and PAQ-S:  $P**A**Q = S$ 
  and SNF: Smith-normal-form-wrt  $S \ \mathcal{Q}$ 

```

```

  and P': invertible P'
  and Q': invertible Q'
  and P'AQ'-S':  $P'*A**Q' = S'$ 
  and SNF-S': Smith-normal-form-wrt  $S' \ \mathcal{Q}$ 

```

shows  $S = S'$

**proof** –

```

  have  $S \ \$h \ i \ \$h \ j = S' \ \$h \ i \ \$h \ j$  for  $i \ j$ 
  proof (cases  $\text{Mod-Type.to-nat } i \neq \text{Mod-Type.to-nat } j$ )
  case True

```

```

    then show ?thesis using SNF SNF-S' unfolding Smith-normal-form-wrt-def
    isDiagonal-def by auto

```

next

```

  case False
  let ?i =  $\text{Mod-Type.to-nat } i$ 
  let ?j =  $\text{Mod-Type.to-nat } j$ 
  have complete-set: Complete-set-non-associates  $\mathcal{Q}$ 
    using SNF-S' unfolding Smith-normal-form-wrt-def by simp
  have  $ij: ?i = ?j$  using False by auto

```

```

show ?thesis
proof (rule ccontr)
  assume d: S $h i $h j ≠ S' $h i $h j
  have n: normalize (S $h i $h j) ≠ normalize (S' $h i $h j)
  proof (rule in-Ass-not-associated[OF complete-set - - d])
    show S $h i $h j ∈ Q using SNF unfolding Smith-normal-form-wrt-def
      by (metis False min-less-iff-conj mod-type-class.to-nat-less-card ncols-def
nrows-def)
    show S' $h i $h j ∈ Q using SNF-S' unfolding Smith-normal-form-wrt-def
      by (metis False min-less-iff-conj mod-type-class.to-nat-less-card ncols-def
nrows-def)
    qed
  have ∃ u. u dvd 1 ∧ S $h i $h j = u * S' $h i $h j
  proof -
    have ∃ u. u dvd 1 ∧ S $h Mod-Type.from-nat ?i $h Mod-Type.from-nat ?i
      = u * S' $h Mod-Type.from-nat ?i $h Mod-Type.from-nat ?i
    proof (rule Smith-normal-form-uniqueness-HOL-Analysis[OF P Q PAQ-S -
P' Q' P'AQ'-S' -])
      show Smith-normal-form S and Smith-normal-form S'
        using SNF SNF-S' Smith-normal-form-def Smith-normal-form-wrt-def
by blast+
      show ?i < min (nrows A) (ncols A)
        by (metis ij min-less-iff-conj mod-type-class.to-nat-less-card ncols-def
nrows-def)
      qed
    thus ?thesis using False by auto
    qed
  from this obtain u where is-unit u and S $h i $h j = u * S' $h i $h j by
auto
  thus False using n
    by (simp add: normalize-1-iff normalize-mult)
  qed
qed
thus ?thesis by vector
qed

end

```

## 11 The Cauchy–Binet formula in HOL Analysis

```

theory Cauchy-Binet-HOL-Analysis
imports
  Cauchy-Binet
  Perron-Frobenius.HMA-Connect
begin

```

## 11.1 Definition of submatrices in HOL Analysis

**definition** *submatrix-hma* :: 'a<sup>nc</sup>nr ⇒ nat set ⇒ nat set ⇒ ('a<sup>nc2</sup>nr2)  
**where** *submatrix-hma* A I J = (χ a b. A \$h (from-nat (pick I (to-nat a))) \$h  
(from-nat (pick J (to-nat b))))

**context includes** *lifting-syntax*  
**begin**

**context**

**fixes** I::nat set **and** J::nat set

**assumes** I: card {i. i < CARD('nr::finite) ∧ i ∈ I} = CARD('nr2::finite)

**assumes** J: card {i. i < CARD('nc::finite) ∧ i ∈ J} = CARD('nc2::finite)

**begin**

**lemma** *HMA-submatrix[transfer-rule]*: (HMA-M ==> HMA-M) (λA. *submatrix*  
A I J)

((λA. *submatrix-hma* A I J):: 'a<sup>nc</sup>nr ⇒ 'a<sup>nc2</sup>nr2)

**proof** (*intro rel-funI, goal-cases*)

**case** (1 A B)

**note** *relAB[transfer-rule]* = *this*

**show** ?case **unfolding** *HMA-M-def*

**proof** (*rule eq-matI, auto*)

**show** *dim-row* (*submatrix* A I J) = CARD('nr2)

**unfolding** *submatrix-def*

**using** I *dim-row-transfer-rule relAB* **by force**

**show** *dim-col* (*submatrix* A I J) = CARD('nc2)

**unfolding** *submatrix-def*

**using** J *dim-col-transfer-rule relAB* **by force**

**let** ?B=(*submatrix-hma* B I J)::'a<sup>nc2</sup>nr2

**fix** i j **assume** i: i < CARD('nr2) **and**

j: j < CARD('nc2)

**have** i2: i < card {i. i < *dim-row* A ∧ i ∈ I}

**using** I *dim-row-transfer-rule i relAB* **by fastforce**

**have** j2: j < card {j. j < *dim-col* A ∧ j ∈ J}

**using** J *dim-col-transfer-rule j relAB* **by fastforce**

**let** ?i = (*from-nat* (*pick* I i))::'nr

**let** ?j = (*from-nat* (*pick* J j))::'nc

**let** ?i' = *Bij-Nat.to-nat* ((*Bij-Nat.from-nat* i)::'nr2)

**let** ?j' = *Bij-Nat.to-nat* ((*Bij-Nat.from-nat* j)::'nc2)

**have** i': ?i' = i **by** (*rule to-nat-from-nat-id[OF i]*)

**have** j': ?j' = j **by** (*rule to-nat-from-nat-id[OF j]*)

**let** ?f = (λ(i, j).

B \$h *Bij-Nat.from-nat* (*pick* I (*Bij-Nat.to-nat* ((*Bij-Nat.from-nat* i)::'nr2))))

\$h

*Bij-Nat.from-nat* (*pick* J (*Bij-Nat.to-nat* ((*Bij-Nat.from-nat* j)::'nc2))))

**have** [*transfer-rule*]: *HMA-I* (*pick* I i) ?i

**by** (*simp add: Bij-Nat.to-nat-from-nat-id I i pick-le HMA-I-def*)

**have** [*transfer-rule*]: *HMA-I* (*pick* J j) ?j

**by** (*simp add: Bij-Nat.to-nat-from-nat-id J j pick-le HMA-I-def*)

```

have submatrix A I J $$ (i, j) = A $$ (pick I i, pick J j) by (rule subma-
trix-index[OF i2 j2])
also have ... = index-hma B ?i ?j by (transfer, simp)
also have ... = B $h Bij-Nat.from-nat (pick I (Bij-Nat.to-nat ((Bij-Nat.from-nat
i)::'nr2))) $h
    Bij-Nat.from-nat (pick J (Bij-Nat.to-nat ((Bij-Nat.from-nat j)::'nc2)))
unfolding i' j' index-hma-def by auto
also have ... = ?f (i,j) by auto
also have ... = Matrix.mat CARD('nr2) CARD('nc2) ?f $$ (i, j)
by (rule index-mat[symmetric, OF i j])
also have ... = from-hmam ?B $$ (i, j)
unfolding from-hmam-def submatrix-hma-def by auto
finally show submatrix A I J $$ (i, j) = from-hmam ?B $$ (i, j) .
qed
qed

end
end

```

## 11.2 Transferring the proof from JNF to HOL Analysis

**lemma** *Cauchy-Binet-HOL-Analysis*:

**fixes** A::'a::comm-ring-1<sup>m</sup><sup>n</sup> **and** B::'a<sup>n</sup><sup>m</sup>

**shows** Determinants.det (A\*\*B) = ( $\sum I \in \{I. I \subseteq \{0..<ncols A\} \wedge card I = nrows A\}$ ).

Determinants.det ((submatrix-hma A UNIV I)::'a<sup>n</sup><sup>n</sup>) \*  
Determinants.det ((submatrix-hma B I UNIV)::'a<sup>n</sup><sup>n</sup>)

**proof** –

**let** ?A = (from-hma<sub>m</sub> A)

**let** ?B = (from-hma<sub>m</sub> B)

**have** relA[transfer-rule]: HMA-M ?A A **unfolding** HMA-M-def **by** simp

**have** relB[transfer-rule]: HMA-M ?B B **unfolding** HMA-M-def **by** simp

**have** ( $\sum I \in \{I. I \subseteq \{0..<ncols A\} \wedge card I = nrows A\}$ ).

Determinants.det ((submatrix-hma A UNIV I)::'a<sup>n</sup><sup>n</sup>) \*  
Determinants.det ((submatrix-hma B I UNIV)::'a<sup>n</sup><sup>n</sup>) =

( $\sum I \in \{I. I \subseteq \{0..<ncols A\} \wedge card I = nrows A\}$ ). det (submatrix ?A UNIV

I)

\* det (submatrix ?B I UNIV))

**proof** (rule sum.cong)

**fix** I **assume** I: I ∈ {I. I ⊆ {0..<ncols A} ∧ card I = nrows A}

**let** ?sub-A = ((submatrix-hma A UNIV I)::'a<sup>n</sup><sup>n</sup>)

**let** ?sub-B = ((submatrix-hma B I UNIV)::'a<sup>n</sup><sup>n</sup>)

**have** c1: card {i. i < CARD('n) ∧ i ∈ UNIV} = CARD('n) **using** I **by** auto

**have** c2: card {i. i < CARD('m) ∧ i ∈ I} = CARD('n)

**proof** –

**have** I = {i. i < CARD('m) ∧ i ∈ I} **using** I **unfolding** nrows-def ncols-def

**by** auto

**thus** ?thesis **using** I nrows-def **by** auto

**qed**



```

have [transfer-rule]: HMA-M (submatrix ?A UNIV I) ?sub-A
  using HMA-submatrix[OF c1 c2] relA unfolding rel-fun-def by auto
have [transfer-rule]: HMA-M (submatrix ?B I UNIV) ?sub-B
  using HMA-submatrix[OF c2 c1] relB unfolding rel-fun-def by auto
show Determinants.det ?sub-A * Determinants.det ?sub-B
  = det (submatrix ?A UNIV I) * det (submatrix ?B I UNIV) by (transfer',
auto)
qed (auto)
also have ... = det (?A*?B)
  by (rule Cauchy-Binet[symmetric], unfold nrows-def ncols-def, auto)
also have ... = Determinants.det (A**B) by (transfer', auto)
finally show ?thesis ..
qed

end

```

## 12 Diagonalizing matrices in JNF and HOL Analysis

```

theory Diagonalize
  imports Admits-SNF-From-Diagonal-Iff-Bezout-Ring
begin

```

This section presents a *locale* that assumes a sound operation to make a matrix diagonal. Then, the result is transferred to HOL Analysis.

### 12.1 Diagonalizing matrices in JNF

We assume a *diagonalize-JNF* operation in JNF, which is applied to matrices over a Bézout ring. However, probably a more restrictive type class is required.

```

locale diagonalize =
  fixes diagonalize-JNF :: 'a::bezout-ring mat  $\Rightarrow$  'a bezout  $\Rightarrow$  ('a mat  $\times$  'a mat  $\times$  'a mat)
  assumes soundness-diagonalize-JNF:
     $\forall A \text{ bezout. } A \in \text{carrier-mat } m \ n \wedge \text{is-bezout-ext } \text{bezout} \longrightarrow$ 
    (case diagonalize-JNF A bezout of (P,S,Q)  $\Rightarrow$ 
      P  $\in$  carrier-mat m m  $\wedge$  Q  $\in$  carrier-mat n n  $\wedge$  S  $\in$  carrier-mat m n
       $\wedge$  invertible-mat P  $\wedge$  invertible-mat Q  $\wedge$  isDiagonal-mat S  $\wedge$  S = P*A*Q)
begin

```

```

lemma soundness-diagonalize-JNF':
  fixes A::'a mat
  assumes is-bezout-ext bezout and A  $\in$  carrier-mat m n
  and diagonalize-JNF A bezout = (P,S,Q)
  shows P  $\in$  carrier-mat m m  $\wedge$  Q  $\in$  carrier-mat n n  $\wedge$  S  $\in$  carrier-mat m n
     $\wedge$  invertible-mat P  $\wedge$  invertible-mat Q  $\wedge$  isDiagonal-mat S  $\wedge$  S = P*A*Q

```

**using** *soundness-diagonalize-JNF* *assms* **unfolding** *case-prod-beta* **by** (*metis fst-conv snd-conv*)

## 12.2 Implementation and soundness result moved to HOL Analysis.

**definition** *diagonalize* :: 'a::bezout-ring ^ 'nc :: mod-type ^ 'nr :: mod-type  
 $\Rightarrow$  'a bezout  $\Rightarrow$   
 (('a ^ 'nr :: mod-type ^ 'nr :: mod-type)  
 $\times$  ('a ^ 'nc :: mod-type ^ 'nr :: mod-type)  
 $\times$  ('a ^ 'nc :: mod-type ^ 'nc :: mod-type))  
**where** *diagonalize* A bezout = (  
 let (P,S,Q) = *diagonalize-JNF* (Mod-Type-Connect.from-hma<sub>m</sub> A) bezout  
 in (Mod-Type-Connect.to-hma<sub>m</sub> P, Mod-Type-Connect.to-hma<sub>m</sub> S, Mod-Type-Connect.to-hma<sub>m</sub>  
 Q)  
 )

**lemma** *soundness-diagonalize*:

**assumes** b: *is-bezout-ext* bezout

**and** d: *diagonalize* A bezout = (P,S,Q)

**shows** *invertible* P  $\wedge$  *invertible* Q  $\wedge$  *isDiagonal* S  $\wedge$  S = P\*\*A\*\*Q

**proof** –

**define** A' **where** A' = Mod-Type-Connect.from-hma<sub>m</sub> A

**obtain** P' S' Q' **where** d-JNF: (P',S',Q') = *diagonalize-JNF* A' bezout

**by** (*metis prod-cases3*)

**define** m **and** n **where** m = *dim-row* A' **and** n = *dim-col* A'

**hence** A': A'  $\in$  *carrier-mat* m n **by** *auto*

**have** *res-JNF*: P'  $\in$  *carrier-mat* m m  $\wedge$  Q'  $\in$  *carrier-mat* n n  $\wedge$  S'  $\in$  *carrier-mat*  
 m n

$\wedge$  *invertible-mat* P'  $\wedge$  *invertible-mat* Q'  $\wedge$  *isDiagonal-mat* S'  $\wedge$  S' = P'\*A'\*Q'

**by** (*rule soundness-diagonalize-JNF'*[OF b A' d-JNF[*symmetric*]])

**have** Mod-Type-Connect.to-hma<sub>m</sub> P' = P **using** d **unfolding** *diagonalize-def*  
*Let-def*

**by** (*metis A'-def d-JNF fst-conv old.prod.case*)

**hence** P' = Mod-Type-Connect.from-hma<sub>m</sub> P **using** A'-def m-def *res-JNF* **by**  
*auto*

**hence** [*transfer-rule*]: Mod-Type-Connect.HMA-M P' P

**unfolding** Mod-Type-Connect.HMA-M-def **by** *auto*

**have** Mod-Type-Connect.to-hma<sub>m</sub> Q' = Q **using** d **unfolding** *diagonalize-def*  
*Let-def*

**by** (*metis A'-def d-JNF snd-conv old.prod.case*)

**hence** Q' = Mod-Type-Connect.from-hma<sub>m</sub> Q **using** A'-def n-def *res-JNF* **by**  
*auto*

**hence** [*transfer-rule*]: Mod-Type-Connect.HMA-M Q' Q

**unfolding** Mod-Type-Connect.HMA-M-def **by** *auto*

**have** Mod-Type-Connect.to-hma<sub>m</sub> S' = S **using** d **unfolding** *diagonalize-def*  
*Let-def*

**by** (*metis A'-def d-JNF snd-conv old.prod.case*)

**hence** S' = Mod-Type-Connect.from-hma<sub>m</sub> S **using** A'-def m-def n-def *res-JNF*

```

by auto
hence [transfer-rule]: Mod-Type-Connect.HMA-M S' S
  unfolding Mod-Type-Connect.HMA-M-def by auto
have [transfer-rule]: Mod-Type-Connect.HMA-M A' A
  using A'-def unfolding Mod-Type-Connect.HMA-M-def by auto
have invertible P using res-JNF by (transfer, simp)
moreover have invertible Q using res-JNF by (transfer, simp)
moreover have isDiagonal S using res-JNF by (transfer, simp)
moreover have S = P**A**Q using res-JNF by (transfer, simp)
ultimately show ?thesis by simp
qed
end

end

```

## 13 Smith normal form algorithm based on two steps in HOL Analysis

```

theory SNF-Algorithm-Two-Steps
  imports Diagonalize
begin

```

This file contains an algorithm to transform a matrix to its Smith normal form, based on two steps: first it is converted into a diagonal matrix and then transformed from diagonal to Smith.

We assume the existence of a diagonalize operation, and then we just have to connect it to the existing algorithm (in HOL Analysis) to transform a diagonal matrix into its Smith normal form.

### 13.1 The implementation

```

context diagonalize
begin

```

```

definition Smith-normal-form-of A bezout = (
  let (P'',D,Q'') = diagonalize A bezout;
      (P',S,Q') = diagonal-to-Smith-PQ D bezout
  in (P'**P'',S,Q'**Q')
)

```

### 13.2 Soundness in HOL Analysis

```

lemma Smith-normal-form-of-soundness:
  fixes A::'a::{bezout-ring} ^ cols::{'mod-type} ^ rows::{'mod-type}
  assumes b: is-bezout-ext bezout
  assumes PSQ: (P,S,Q) = Smith-normal-form-of A bezout
  shows S = P**A**Q  $\wedge$  invertible P  $\wedge$  invertible Q  $\wedge$  Smith-normal-form S

```

```

proof –
  obtain  $P'' D Q''$  where  $PDQ\text{-diag}: (P'', D, Q'') = \text{diagonalize } A \text{ bezout}$ 
    by (metis prod-cases3)
  have 1:  $\text{invertible } P'' \wedge \text{invertible } Q'' \wedge \text{isDiagonal } D \wedge D = P''**A**Q''$ 
    by (rule soundness-diagonalize[OF b PDQ-diag[symmetric]])
  obtain  $P' Q'$  where  $PSQ\text{-D}: (P', S, Q') = \text{diagonal-to-Smith-PQ } D \text{ bezout}$ 
    using  $PSQ PDQ\text{-diag}$  unfolding  $\text{Smith-normal-form-of-def}$ 
    unfolding  $\text{Let-def}$  by (smt Pair-inject case-prod-beta' surjective-pairing)
  have 2:  $\text{invertible } P' \wedge \text{invertible } Q' \wedge \text{Smith-normal-form } S \wedge S = P'**D**Q'$ 
    using  $\text{diagonal-to-Smith-PQ}' 1 b PSQ\text{-D}$  by blast
  have  $P: P = P'**P''$ 
    by (metis (mono-tags, lifting) PDQ-diag PSQ-D Pair-inject
      Smith-normal-form-of-def PSQ old.prod.case)
  have  $Q: Q = Q'**Q'$ 
    by (metis (mono-tags, lifting) PDQ-diag PSQ-D Pair-inject
      Smith-normal-form-of-def PSQ old.prod.case)
  have  $S = P**A**Q$  using 1 2 by (simp add: P Q matrix-mul-assoc)
  moreover have  $\text{invertible } P$  using  $P$  by (simp add: 1 2 invertible-mult)
  moreover have  $\text{invertible } Q$  using  $Q$  by (simp add: 1 2 invertible-mult)
  ultimately show ?thesis using 2 by auto
qed

end
end

```

## 14 Algorithm to transform a diagonal matrix into its Smith normal form in JNF

```

theory Diagonal-To-Smith-JNF
  imports Admits-SNF-From-Diagonal-Iff-Bezout-Ring
begin

```

In this file, we implement an algorithm to transform a diagonal matrix into its Smith normal form, using the JNF library.

There are, at least, three possible options:

1. Implement and prove the soundness of the algorithm from scratch in JNF
2. Implement it in JNF and connect it to the HOL Analysis version by means of transfer rules. Thus, we could obtain the soundness lemma in JNF.
3. Implement it in JNF, with calls to the HOL Analysis version by means of the functions *from-hma<sub>m</sub>* and *to-hma<sub>m</sub>*. That is, transform the matrix to HOL Analysis, apply the existing algorithm in HOL Analysis to get the Smith normal form and then transform the output to JNF.

Then, we could try to get the soundness theorem in JNF by means of transfer rules and local type definitions.

The first option requires much effort. As we will see, the third option is not possible.

### 14.1 Attempt with the third option: definitions and conditional transfer rules

**context**

**fixes**  $A::'a::\text{bezout-ring mat}$

**assumes**  $A \in \text{carrier-mat } \text{CARD}('nr::\text{mod-type}) \text{ CARD}('nc::\text{mod-type})$

**begin**

**private definition** *diagonal-to-Smith-PQ-JNF'*  $\text{bezout} = ($

*let*  $A' = \text{Mod-Type-Connect.to-hma}_m A::'a \wedge 'nc::\text{mod-type} \wedge 'nr::\text{mod-type};$

$(P,S,Q) = (\text{diagonal-to-Smith-PQ } A' \text{ bezout})$

*in*  $(\text{Mod-Type-Connect.from-hma}_m P, \text{Mod-Type-Connect.from-hma}_m S, \text{Mod-Type-Connect.from-hma}_m Q))$

**end**

This approach will not work. The type is necessary in the definition of the function. That is, outside the context, the function will be:

*diagonal-to-Smith-PQ-JNF'*  $\text{TYPE}('nc) \text{TYPE}('nr) A \text{bezout}$

And we cannot get rid of such  $\text{TYPE}('nc)$ .

That is, we could get a lemma like:

**lemma assumes**  $A \in \text{carrier-mat } m \ n$  **and**  $(P,S,Q) = \text{diagonal-to-Smith-PQ-JNF}'$

$\text{TYPE}('nr::\text{mod-type}) \text{TYPE}('nc::\text{mod-type}) A \text{bezout}$  **shows** *invertible-mat*

$P \wedge \text{invertible-mat } Q \wedge S = P * A * Q \wedge \text{Smith-normal-form-mat } S$

But we wouldn't be able to get rid of such types.

### 14.2 Attempt with the second option: implementation and soundness in JNF

**definition** *diagonal-step-JNF*  $A \ i \ j \ d \ v =$

*Matrix.mat*  $(\text{dim-row } A) (\text{dim-col } A) (\lambda (a,b). \text{if } a = i \wedge b = i \text{ then } d$

*else*

*if*  $a = j \wedge b = j$

*then*  $v * (A \ \$\$ (j,j))$  *else*  $A \ \$\$ (a,b)$

Conditional transfer rules are required, so I prove them within context with assumptions.

**context**

**includes** *lifting-syntax*

**fixes**  $i$  **and**  $j::\text{nat}$

```

assumes  $i: i < \min (CARD('nr::mod-type)) (CARD('nc::mod-type))$ 
and  $j: j < \min (CARD('nr::mod-type)) (CARD('nc::mod-type))$ 
begin

lemma HMA-diagonal-step[transfer-rule]:
  ((Mod-Type-Connect.HMA-M :: -  $\Rightarrow$  'a :: comm-ring-1  $\wedge$  'nc :: mod-type  $\wedge$  'nr ::
  mod-type  $\Rightarrow$  -)
  =====> (=) =====> (=) =====> Mod-Type-Connect.HMA-M)
  ( $\lambda A.$  diagonal-step-JNF A i j) ( $\lambda B.$  diagonal-step B i j)
  by (intro rel-funI, goal-cases, auto simp add: Mod-Type-Connect.HMA-M-def
  diagonal-step-JNF-def diagonal-step-def)
  (rule eq-matI, auto simp add: Mod-Type-Connect.from-hmam-def, insert from-nat-eq-imp-eq
  i j, auto)

end

```

```

definition diagonal-step-PQ-JNF ::
  'a::{bezout-ring} mat  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  'a bezout  $\Rightarrow$  ('a mat  $\times$  ('a mat))
  where diagonal-step-PQ-JNF A i k bezout =
  (let m = dim-row A; n = dim-col A;
    (p, q, u, v, d) = bezout (A $$ (i,i)) (A $$ (k,k));
    P = addrow (-v) k i (swaprows i k (addrow p k i (1m m)));
    Q = multcol k (-1) (addcol u k i (addcol q i k (1m n)))
    in (P,Q)
  )

```

```

context
  includes lifting-syntax
  fixes i and k::nat
  assumes  $i: i < \min (CARD('nr::mod-type)) (CARD('nc::mod-type))$ 
  and  $k: k < \min (CARD('nr::mod-type)) (CARD('nc::mod-type))$ 
begin

```

```

lemma HMA-diagonal-step-PQ[transfer-rule]:
  ((Mod-Type-Connect.HMA-M :: -  $\Rightarrow$  'a :: bezout-ring  $\wedge$  'nc :: mod-type  $\wedge$  'nr ::
  mod-type  $\Rightarrow$  -)
  =====> (=) =====> rel-prod Mod-Type-Connect.HMA-M Mod-Type-Connect.HMA-M)

```

```

  ( $\lambda A$  bezout. diagonal-step-PQ-JNF A i k bezout) ( $\lambda A$  bezout. diagonal-step-PQ
  A i k bezout)

```

```

proof (intro rel-funI, goal-cases)
  case (1 A A' bezout bezout')
  note HMA-M-AA'[transfer-rule] = 1(1)
  let ?d-JNF = (diagonal-step-PQ-JNF A i k bezout)
  let ?d-HA = (diagonal-step-PQ A' i k bezout')
  have [transfer-rule]: Mod-Type-Connect.HMA-I k (from-nat k::'nc)
  and [transfer-rule]: Mod-Type-Connect.HMA-I k (from-nat k::'nr)
  by (metis Mod-Type-Connect.HMA-I-def k min.strict-boundedE to-nat-from-nat-id) +
  have [transfer-rule]: Mod-Type-Connect.HMA-I i (from-nat i::'nc)

```

```

    and [transfer-rule]: Mod-Type-Connect.HMA-I i (from-nat i::'nr)
    by (metis Mod-Type-Connect.HMA-I-def i min.strict-boundedE to-nat-from-nat-id)+
    have [transfer-rule]: A $$ (i,i) = A' $h from-nat i $h from-nat i
    proof -
      have A $$ (i,i) = index-hma A' (from-nat i) (from-nat i) by (transfer, simp)
      also have ... = A' $h from-nat i $h from-nat i unfolding index-hma-def by
    auto
    finally show ?thesis .
  qed
  have [transfer-rule]: A $$ (k,k) = A' $h from-nat k $h from-nat k
  proof -
    have A $$ (k,k) = index-hma A' (from-nat k) (from-nat k) by (transfer, simp)
    also have ... = A' $h from-nat k $h from-nat k unfolding index-hma-def by
  auto
  finally show ?thesis .
  qed
  have dim-row-CARD: dim-row A = CARD('nr)
  using HMA-M-AA' Mod-Type-Connect.dim-row-transfer-rule by blast
  have dim-col-CARD: dim-col A = CARD('nc)
  using HMA-M-AA' Mod-Type-Connect.dim-col-transfer-rule by blast
  let ?p = fst (bezout (A' $h from-nat i $h from-nat i) (A' $h from-nat k $h
from-nat k))
  let ?v = fst (snd (snd (snd (bezout (A $$ (i, i)) (A $$ (k, k))))))
  have Mod-Type-Connect.HMA-M (fst ?d-JNF) (fst ?d-HA)
  unfolding diagonal-step-PQ-JNF-def diagonal-step-PQ-def Mod-Type-Connect.HMA-M-def

  unfolding Let-def split-beta dim-row-CARD
  by (auto, transfer, auto simp add: Mod-Type-Connect.HMA-M-def Rel-def
rel-funI)
  moreover have Mod-Type-Connect.HMA-M (snd ?d-JNF) (snd ?d-HA)
  unfolding diagonal-step-PQ-JNF-def diagonal-step-PQ-def Mod-Type-Connect.HMA-M-def

  unfolding Let-def split-beta dim-col-CARD
  by (auto, transfer, auto simp add: Mod-Type-Connect.HMA-M-def Rel-def
rel-funI)
  ultimately show ?case unfolding rel-prod-conv using 1
  by (simp add: split-beta)
  qed
end

fun diagonal-to-Smith-i-PQ-JNF ::
  nat list ⇒ nat ⇒ ('a::{bezout-ring} bezout)
  ⇒ ('a mat × 'a mat × 'a mat) ⇒ ('a mat × 'a mat × 'a mat)
  where
  diagonal-to-Smith-i-PQ-JNF [] i bezout (P,A,Q) = (P,A,Q) |
  diagonal-to-Smith-i-PQ-JNF (j#xs) i bezout (P,A,Q) = (
  if A $$ (i,i) dvd A $$ (j,j)

```

```

    then diagonal-to-Smith-i-PQ-JNF xs i bezout (P,A,Q)
  else let (p, q, u, v, d) = bezout (A $$ (i,i)) (A $$ (j,j));
    A' = diagonal-step-JNF A i j d v;
    (P',Q') = diagonal-step-PQ-JNF A i j bezout
    in diagonal-to-Smith-i-PQ-JNF xs i bezout (P'*P,A',Q*Q') — Apply the step
  )

```

**context**

```

  includes lifting-syntax
  fixes i and xs
  assumes i: i < min (CARD('nr::mod-type)) (CARD('nc::mod-type))
  and xs: ∀ j ∈ set xs. j < min (CARD('nr::mod-type)) (CARD('nc::mod-type))
begin

```

**declare** *diagonal-step-PQ.simps*[simp del]

**lemma** *HMA-diagonal-to-Smith-i-PQ-aux*: *HMA-M3* (P,A,Q)

```

(P' :: 'a :: bezout-ring ^ 'nr :: mod-type ^ 'nr :: mod-type,
 A' :: 'a :: bezout-ring ^ 'nc :: mod-type ^ 'nr :: mod-type,
 Q' :: 'a :: bezout-ring ^ 'nc :: mod-type ^ 'nc :: mod-type)
⇒ HMA-M3 (diagonal-to-Smith-i-PQ-JNF xs i bezout (P,A,Q))
  (diagonal-to-Smith-i-PQ xs i bezout (P',A',Q'))

```

**using** *i xs*

**proof** (*induct xs i bezout (P',A',Q')* *arbitrary: P' A' Q' P A Q rule: diagonal-to-Smith-i-PQ.induct*)

**case** (*1 i bezout P' A' Q'*)

**then show** *?case by auto*

**next**

**case** (*2 j xs i bezout P' A' Q'*)

**note** *HMA-M3*[*transfer-rule*] = *2.prem1*

**note** *i* = *2(4)*

**note** *j* = *2(5)*

**note** *IH1* = *2.hyps(1)*

**note** *IH2* = *2.hyps(2)*

**have** *j-min*: *j < min CARD('nr) CARD('nc)* **using** *j by auto*

**have** *HMA-M-AA'*[*transfer-rule*]: *Mod-Type-Connect.HMA-M A A'* **using** *HMA-M3*  
**by auto**

**have** [*transfer-rule*]: *Mod-Type-Connect.HMA-I j (from-nat j::'nc)*

**and** [*transfer-rule*]: *Mod-Type-Connect.HMA-I j (from-nat j::'nr)*

**by** (*metis Mod-Type-Connect.HMA-I-def j-min min.strict-boundedE to-nat-from-nat-id*) +

**have** [*transfer-rule*]: *Mod-Type-Connect.HMA-I i (from-nat i::'nc)*

**and** [*transfer-rule*]: *Mod-Type-Connect.HMA-I i (from-nat i::'nr)*

**by** (*metis Mod-Type-Connect.HMA-I-def i min.strict-boundedE to-nat-from-nat-id*) +

**have** [*transfer-rule*]: *A \$\$ (i, i) = A' \$h from-nat i \$h from-nat i*

**proof** –

**have** *A \$\$ (i,i) = index-hma A' (from-nat i) (from-nat i)* **by** (*transfer, simp*)

**also have** ... = *A' \$h from-nat i \$h from-nat i* **unfolding** *index-hma-def* **by**

*auto*



```

    finally show ?thesis .
  qed
  have [transfer-rule]: A $$ (j, j) = A' $h from-nat j $h from-nat j
  proof -
    have A $$ (j,j) = index-hma A' (from-nat j) (from-nat j) by (transfer, simp)
    also have ... = A' $h from-nat j $h from-nat j unfolding index-hma-def by
  auto
    finally show ?thesis .
  qed
  show ?case
  proof (cases A $$ (i, i) dvd A $$ (j, j))
    case True
      hence A' $h from-nat i $h from-nat i dvd A' $h from-nat j $h from-nat j by
  transfer
    then show ?thesis using True IH1 HMA-M3 i j by auto
  next
    case False
      obtain p q u v d where b: (p, q, u, v, d) = bezout (A $$ (i,i)) (A $$ (j,j))
      by (metis prod-cases5)
      let ?A'-JNF = diagonal-step-JNF A i j d v
      obtain P''-JNF Q''-JNF where P''Q''-JNF: (P''-JNF, Q''-JNF) = diago-
  nal-step-PQ-JNF A i j bezout
      by (metis surjective-pairing)
      have not-dvd: ¬ A' $h from-nat i $h from-nat i dvd A' $h from-nat j $h from-nat
  j using False by transfer
      let ?A' = diagonal-step A' i j d v
      obtain P'' Q'' where P''Q'': (P'', Q'') = diagonal-step-PQ A' i j bezout
      by (metis surjective-pairing)
      have b2: (p, q, u, v, d) = bezout (A' $h from-nat i $h from-nat i) (A' $h
  from-nat j $h from-nat j)
      using b by (transfer, auto)
      let ?D-HA = diagonal-to-Smith-i-PQ xs i bezout (P''**P', ?A', Q''**Q'')
      let ?D-JNF = diagonal-to-Smith-i-PQ-JNF xs i bezout (P''-JNF*P, ?A'-JNF, Q*Q''-JNF)
      have rw-1: diagonal-to-Smith-i-PQ-JNF (j # xs) i bezout (P, A, Q) = ?D-JNF

      using False b P''Q''-JNF
      by (auto, unfold split-beta, metis fst-conv snd-conv)
      have rw-2: diagonal-to-Smith-i-PQ (j # xs) i bezout (P', A', Q') = ?D-HA
      using not-dvd b2 P''Q'' by (auto, unfold split-beta, metis fst-conv snd-conv)
      have HMA-M3 ?D-JNF ?D-HA
      proof (rule IH2[OF not-dvd b2], auto)
        have j: j < min CARD('nr) CARD('nc) using j by auto
        have [transfer-rule]: rel-prod Mod-Type-Connect.HMA-M Mod-Type-Connect.HMA-M

        (diagonal-step-PQ-JNF A i j bezout) (diagonal-step-PQ A' i j bezout)
          using HMA-diagonal-step-PQ[OF i j] HMA-M-AA' unfolding rel-fun-def
  by auto
        hence [transfer-rule]: Mod-Type-Connect.HMA-M P''-JNF P''
          and [transfer-rule]: Mod-Type-Connect.HMA-M Q''-JNF Q''

```

```

    using P''Q'' P''Q''-JNF unfolding rel-prod-conv split-beta
    by (metis fst-conv, metis snd-conv)
    have [transfer-rule]: Mod-Type-Connect.HMA-M P P' using HMA-M3 by
auto
    show Mod-Type-Connect.HMA-M (P''-JNF * P) (P'' ** P')

    by (transfer-prover-start, transfer-step+, auto)

```

```

    show Mod-Type-Connect.HMA-M (diagonal-step-JNF A i j d v) (diagonal-step
A' i j d v)
    using HMA-diagonal-step[OF i j] HMA-M-AA' unfolding rel-fun-def by
auto
    have [transfer-rule]: Mod-Type-Connect.HMA-M Q Q' using HMA-M3 by
auto
    show Mod-Type-Connect.HMA-M (Q * Q''-JNF) (Q' ** Q'')
    by (transfer-prover-start, transfer-step+, auto)
    qed (insert i j P''Q'', auto)
    then show ?thesis using rw-1 rw-2 by auto
    qed
    qed

```

```

lemma HMA-diagonal-to-Smith-i-PQ[transfer-rule]:
  ((=)
  ==> (HMA-M3 :: (- => (- × ('a :: bezout-ring ^ 'nc :: mod-type ^ 'nr :: mod-type)
× -) =>-))
  ==> HMA-M3) (diagonal-to-Smith-i-PQ-JNF xs i) (diagonal-to-Smith-i-PQ
xs i)
proof (intro rel-funI, goal-cases)
  case (1 x y bezout bezout')
  then show ?case using HMA-diagonal-to-Smith-i-PQ-aux
  by (auto, smt (verit) HMA-M3.elims(2))
qed

```

end

```

fun Diagonal-to-Smith-row-i-PQ-JNF
  where Diagonal-to-Smith-row-i-PQ-JNF i bezout (P,A,Q)
  = diagonal-to-Smith-i-PQ-JNF [i + 1..<min (dim-row A) (dim-col A)] i bezout
(P,A,Q)

```

```

declare Diagonal-to-Smith-row-i-PQ-JNF.simps[simp del]
lemmas Diagonal-to-Smith-row-i-PQ-JNF-def = Diagonal-to-Smith-row-i-PQ-JNF.simps

```

```

context
  includes lifting-syntax
  fixes i
  assumes i: i < min (CARD('nr::mod-type)) (CARD('nc::mod-type))
begin

```

```

lemma HMA-Diagonal-to-Smith-row-i-PQ[transfer-rule]:
  ((=) ==> (HMA-M3 :: (- => (- × ('a::bezout-ring ^nc::mod-type ^nr::mod-type)
× -) => -)) ==> HMA-M3)
  (Diagonal-to-Smith-row-i-PQ-JNF i) (Diagonal-to-Smith-row-i-PQ i)
proof (intro rel-funI, clarify, goal-cases)
  case (1 - bezout P A Q P' A' Q')
  note HMA-M3[transfer-rule] = 1
  let ?xs1=[i + 1..<min (dim-row A) (dim-col A)]
  let ?xs2=[i + 1..<min (nrows A') (ncols A')]
  have xs-eq[transfer-rule]: ?xs1 = ?xs2
  using HMA-M3
  by (auto intro: arg-cong2[where f = upt]
  simp: Mod-Type-Connect.dim-col-transfer-rule Mod-Type-Connect.dim-row-transfer-rule
  nrows-def ncols-def)
  have j-xs:  $\forall j \in \text{set } ?xs1. j < \min \text{CARD}('nr) \text{CARD}('nc)$  using i
  by (metis atLeastLessThan-iff ncols-def nrows-def set-upt xs-eq)
  have rel: HMA-M3 (diagonal-to-Smith-i-PQ-JNF ?xs1 i bezout (P,A,Q))
  (diagonal-to-Smith-i-PQ ?xs1 i bezout (P',A',Q'))
  using HMA-diagonal-to-Smith-i-PQ[OF i j-xs] HMA-M3 unfolding rel-fun-def
by blast
  then show ?case
  unfolding Diagonal-to-Smith-row-i-PQ-JNF-def Diagonal-to-Smith-row-i-PQ-def
  by (metis Suc-eq-plus1 xs-eq)
qed

end

fun diagonal-to-Smith-aux-PQ-JNF
  where
  diagonal-to-Smith-aux-PQ-JNF [] bezout (P,A,Q) = (P,A,Q) |
  diagonal-to-Smith-aux-PQ-JNF (i#xs) bezout (P,A,Q)
  = diagonal-to-Smith-aux-PQ-JNF xs bezout (Diagonal-to-Smith-row-i-PQ-JNF
i bezout (P,A,Q))

context
  includes lifting-syntax
  fixes xs
  assumes xs:  $\forall j \in \text{set } xs. j < \min (\text{CARD}('nr::\text{mod-type})) (\text{CARD}('nc::\text{mod-type}))$ 
begin

lemma HMA-diagonal-to-Smith-aux-PQ-JNF[transfer-rule]:
  ((=) ==> (HMA-M3 :: (- => (- × ('a::bezout-ring ^nc::mod-type ^nr::mod-type)
× -) => -)) ==> HMA-M3)
  (diagonal-to-Smith-aux-PQ-JNF xs) (diagonal-to-Smith-aux-PQ xs)
proof (intro rel-funI, clarify, goal-cases)
  case (1 - bezout P A Q P' A' Q')
  note HMA-M3[transfer-rule] = 1
  show ?case

```

```

    using xs HMA-M3
  proof (induct xs arbitrary: P' A' Q' P A Q)
    case Nil
    then show ?case by auto
  next
    case (Cons i xs)
    note IH = Cons(1)
    note HMA-M3 = Cons.prem(2)
    have i: i < min CARD('nr) CARD('nc) using Cons.prem by auto
    let ?D-JNF = (Diagonal-to-Smith-row-i-PQ-JNF i bezout (P, A, Q))
    let ?D-HA = (Diagonal-to-Smith-row-i-PQ i bezout (P', A', Q'))
    have rw-1: diagonal-to-Smith-aux-PQ-JNF (i # xs) bezout (P, A, Q)
      = diagonal-to-Smith-aux-PQ-JNF xs bezout ?D-JNF by auto
    have rw-2: diagonal-to-Smith-aux-PQ (i # xs) bezout (P', A', Q')
      = diagonal-to-Smith-aux-PQ xs bezout ?D-HA by auto
    have HMA-M3 ?D-JNF ?D-HA
    using HMA-Diagonal-to-Smith-row-i-PQ[OF i] HMA-M3 unfolding rel-fun-def
  by blast
    then show ?case
      by (auto, smt (verit) Cons.hyps HMA-M3.elims(2) list.set-intros(2) local.Cons(2))
    qed
  qed
end

fun diagonal-to-Smith-PQ-JNF
  where diagonal-to-Smith-PQ-JNF A bezout
    = diagonal-to-Smith-aux-PQ-JNF [0..<min (dim-row A) (dim-col A) - 1]
      bezout (1m (dim-row A), A, 1m (dim-col A))

declare diagonal-to-Smith-PQ-JNF.simps[simp del]
lemmas diagonal-to-Smith-PQ-JNF-def = diagonal-to-Smith-PQ-JNF.simps

lemma diagonal-step-PQ-JNF-dim:
  assumes A: A ∈ carrier-mat m n
    and d: diagonal-step-PQ-JNF A i j bezout = (P, Q)
  shows P ∈ carrier-mat m m ∧ Q ∈ carrier-mat n n
  using A d unfolding diagonal-step-PQ-JNF-def split-beta Let-def by auto

lemma diagonal-step-JNF-dim:
  assumes A: A ∈ carrier-mat m n
  shows diagonal-step-JNF A i j d v ∈ carrier-mat m n
  using A unfolding diagonal-step-JNF-def by auto

lemma diagonal-to-Smith-i-PQ-JNF-dim:
  assumes P' ∈ carrier-mat m m ∧ A' ∈ carrier-mat m n ∧ Q' ∈ carrier-mat n n
    and diagonal-to-Smith-i-PQ-JNF xs i bezout (P', A', Q') = (P, A, Q)

```

**shows**  $P \in \text{carrier-mat } m \ m \wedge A \in \text{carrier-mat } m \ n \wedge Q \in \text{carrier-mat } n \ n$   
**using** *assms*  
**proof** (*induct xs i bezout (P',A',Q')* arbitrary:  $P \ A \ Q \ P' \ A' \ Q'$  rule: *diagonal-to-Smith-i-PQ-JNF.induct*)  
  **case** ( $1 \ i \ \text{bezout } P \ A \ Q$ )  
  **then show** *?case* **by** *auto*  
**next**  
  **case** ( $2 \ j \ xs \ i \ \text{bezout } P' \ A' \ Q'$ )  
  **show** *?case*  
  **proof** (*cases A' \$\$ (i, i) dvd A' \$\$ (j, j)*)  
  **case** *True*  
  **then show** *?thesis using 2* **by** *auto*  
**next**  
  **case** *False*  
  **obtain**  $p \ q \ u \ v \ d$  **where**  $b: (p, q, u, v, d) = \text{bezout } (A' \ \$\$ (i, i)) \ (A' \ \$\$ (j, j))$   
  **by** (*metis prod-cases5*)  
  **let**  $?A' = \text{diagonal-step-JNF } A' \ i \ j \ d \ v$   
  **obtain**  $P'' \ Q''$  **where**  $P''Q'': (P'', Q'') = \text{diagonal-step-PQ-JNF } A' \ i \ j \ \text{bezout}$   
  **by** (*metis surjective-pairing*)  
  **let**  $?A' = \text{diagonal-step-JNF } A' \ i \ j \ d \ v$   
  **let**  $?D\text{-JNF} = \text{diagonal-to-Smith-i-PQ-JNF } xs \ i \ \text{bezout } (P'' * P', ?A', Q' * Q'')$   
  **have**  $rw\text{-1}: \text{diagonal-to-Smith-i-PQ-JNF } (j \ \# \ xs) \ i \ \text{bezout } (P', A', Q') =$   
 $?D\text{-JNF}$   
  **using** *False b P''Q''*  
  **by** (*auto, unfold split-beta, metis fst-conv snd-conv*)  
  **show** *?thesis*  
  **proof** (*rule 2.hyps(2)[OF False b]*)  
  **show**  $?D\text{-JNF} = (P, A, Q)$  **using** *rw-1 2* **by** *auto*  
  **have**  $P'' \in \text{carrier-mat } m \ m$  **and**  $Q'' \in \text{carrier-mat } n \ n$   
  **using** *diagonal-step-PQ-JNF-dim[OF - P''Q''[symmetric]] 2.prem* **by** *auto*  
  **thus**  $P'' * P' \in \text{carrier-mat } m \ m \wedge ?A' \in \text{carrier-mat } m \ n \wedge Q' * Q'' \in$   
 $\text{carrier-mat } n \ n$   
  **using** *diagonal-step-JNF-dim 2* **by** (*metis mult-carrier-mat*)  
  **qed** (*insert P''Q'', auto*)  
**qed**  
**qed**

**lemma** *Diagonal-to-Smith-row-i-PQ-JNF-dim:*

**assumes**  $P' \in \text{carrier-mat } m \ m \wedge A' \in \text{carrier-mat } m \ n \wedge Q' \in \text{carrier-mat } n \ n$   
  **and** *Diagonal-to-Smith-row-i-PQ-JNF i bezout (P',A',Q') = (P,A,Q)*  
**shows**  $P \in \text{carrier-mat } m \ m \wedge A \in \text{carrier-mat } m \ n \wedge Q \in \text{carrier-mat } n \ n$   
**by** (*rule diagonal-to-Smith-i-PQ-JNF-dim, insert assms,*  
  *auto simp add: Diagonal-to-Smith-row-i-PQ-JNF-def*)

**lemma** *diagonal-to-Smith-aux-PQ-JNF-dim:*

**assumes**  $P' \in \text{carrier-mat } m \ m \wedge A' \in \text{carrier-mat } m \ n \wedge Q' \in \text{carrier-mat } n \ n$   
  **and** *diagonal-to-Smith-aux-PQ-JNF xs bezout (P',A',Q') = (P,A,Q)*  
**shows**  $P \in \text{carrier-mat } m \ m \wedge A \in \text{carrier-mat } m \ n \wedge Q \in \text{carrier-mat } n \ n$   
**using** *assms*

```

proof (induct xs bezout (P',A',Q') arbitrary: P A Q P' A' Q' rule: diagonal-to-Smith-aux-PQ-JNF.induct)
  case (1 bezout P A Q)
  then show ?case by simp
next
  case (2 i xs bezout P' A' Q')
  let ?D=(Diagonal-to-Smith-row-i-PQ-JNF i bezout (P', A', Q'))
  have diagonal-to-Smith-aux-PQ-JNF (i # xs) bezout (P', A', Q') =
    diagonal-to-Smith-aux-PQ-JNF xs bezout ?D by auto
  hence *: ... = (P,A,Q) using 2 by auto
  let ?P=fst ?D
  let ?S=fst (snd ?D)
  let ?Q=snd (snd ?D)
  show ?case
  proof (rule 2.hyps)
    show Diagonal-to-Smith-row-i-PQ-JNF i bezout (P', A', Q') = (?P,?S,?Q)
by auto
    show diagonal-to-Smith-aux-PQ-JNF xs bezout (?P, ?S, ?Q) = (P, A, Q)
using * by simp
    show ?P ∈ carrier-mat m m ∧ ?S ∈ carrier-mat m n ∧ ?Q ∈ carrier-mat n
n
    by (rule Diagonal-to-Smith-row-i-PQ-JNF-dim, insert 2, auto)
  qed
qed

```

```

lemma diagonal-to-Smith-PQ-JNF-dim:
  assumes A ∈ carrier-mat m n
  and PSQ: diagonal-to-Smith-PQ-JNF A bezout = (P,S,Q)
  shows P ∈ carrier-mat m m ∧ S ∈ carrier-mat m n ∧ Q ∈ carrier-mat n n
  by (rule diagonal-to-Smith-aux-PQ-JNF-dim, insert assms,
    auto simp add: diagonal-to-Smith-PQ-JNF-def)

```

```

context
  includes lifting-syntax
begin

```

```

lemma HMA-diagonal-to-Smith-PQ-JNF[transfer-rule]:
  ((Mod-Type-Connect.HMA-M) ==> (=) ==> HMA-M3) (diagonal-to-Smith-PQ-JNF)
  (diagonal-to-Smith-PQ)
proof (intro rel-funI, clarify, goal-cases)
  case (1 A A' - bezout)
  let ?xs1 = [0.. $\min$  (dim-row A) (dim-col A) - 1]
  let ?xs2 = [0.. $\min$  (nrows A') (ncols A') - 1]
  let ?PAQ=(1m (dim-row A), A, 1m (dim-col A))
  have dr: dim-row A = CARD('c)
    using 1 Mod-Type-Connect.dim-row-transfer-rule by blast
  have dc: dim-col A = CARD('b)
    using 1 Mod-Type-Connect.dim-col-transfer-rule by blast
  have xs-eq: ?xs1 = ?xs2

```

```

  by (simp add: dc dr ncols-def nrows-def)
  have j-xs:  $\forall j \in \text{set } ?xs1. j < \min \text{CARD}('c) \text{CARD}('b)$ 
  using dc dr less-imp-diff-less by auto
  let ?D-JNF = diagonal-to-Smith-aux-PQ-JNF ?xs1 bezout ?PAQ
  let ?D-HA = diagonal-to-Smith-aux-PQ ?xs1 bezout (mat 1, A', mat 1)
  have mat-rel-init: HMA-M3 ?PAQ (mat 1, A', mat 1)
  proof -
    have Mod-Type-Connect.HMA-M ( $1_m (\text{dim-row } A)$ ) ( $\text{mat } 1 :: 'a \wedge 'c :: \text{mod-type} \wedge 'c :: \text{mod-type}$ )

      unfolding dr by (transfer-prover-start, transfer-step, auto)
    moreover have Mod-Type-Connect.HMA-M ( $1_m (\text{dim-col } A)$ ) ( $\text{mat } 1 :: 'a \wedge 'b :: \text{mod-type} \wedge 'b :: \text{mod-type}$ )
      unfolding dc by (transfer-prover-start, transfer-step, auto)
    ultimately show ?thesis using 1 by auto
  qed
  have HMA-M3 ?D-JNF ?D-HA
  using HMA-diagonal-to-Smith-aux-PQ-JNF[OF j-xs] mat-rel-init unfolding
rel-fun-def by blast
  then show ?case using xs-eq unfolding diagonal-to-Smith-PQ-JNF-def diagonal-to-Smith-PQ-def
  by auto
  qed
end

```

### 14.3 Applying local type definitions

Now we get the soundness lemma in JNF, via the one in HOL Analysis. I need transfer rules and local type definitions.

**context**

includes *lifting-syntax*

begin

**private lemma** *diagonal-to-Smith-PQ-JNF-with-types:*

assumes  $A: A \in \text{carrier-mat } \text{CARD}('nr :: \text{mod-type}) \text{CARD}('nc :: \text{mod-type})$

and  $S: S \in \text{carrier-mat } \text{CARD}('nr) \text{CARD}('nc)$

and  $P: P \in \text{carrier-mat } \text{CARD}('nr) \text{CARD}('nr)$

and  $Q: Q \in \text{carrier-mat } \text{CARD}('nc) \text{CARD}('nc)$

and  $PSQ: \text{diagonal-to-Smith-PQ-JNF } A \text{ bezout} = (P, S, Q)$

and  $d: \text{isDiagonal-mat } A$  and  $ib: \text{is-bezout-ext } \text{bezout}$

shows  $S = P * A * Q \wedge \text{invertible-mat } P \wedge \text{invertible-mat } Q \wedge \text{Smith-normal-form-mat } S$

**proof** -

let ?P = Mod-Type-Connect.to-hma<sub>m</sub>  $P :: 'a \wedge 'nr :: \text{mod-type} \wedge 'nr :: \text{mod-type}$

let ?A = Mod-Type-Connect.to-hma<sub>m</sub>  $A :: 'a \wedge 'nc :: \text{mod-type} \wedge 'nr :: \text{mod-type}$

let ?Q = Mod-Type-Connect.to-hma<sub>m</sub>  $Q :: 'a \wedge 'nc :: \text{mod-type} \wedge 'nc :: \text{mod-type}$

let ?S = Mod-Type-Connect.to-hma<sub>m</sub>  $S :: 'a \wedge 'nc :: \text{mod-type} \wedge 'nr :: \text{mod-type}$

have [transfer-rule]: Mod-Type-Connect.HMA-M A ?A

by (simp add: Mod-Type-Connect.HMA-M-def A)

**moreover have** [transfer-rule]: *Mod-Type-Connect.HMA-M P ?P*  
**by** (*simp add: Mod-Type-Connect.HMA-M-def P*)  
**moreover have** [transfer-rule]: *Mod-Type-Connect.HMA-M Q ?Q*  
**by** (*simp add: Mod-Type-Connect.HMA-M-def Q*)  
**moreover have** [transfer-rule]: *Mod-Type-Connect.HMA-M S ?S*  
**by** (*simp add: Mod-Type-Connect.HMA-M-def S*)  
**ultimately have** [transfer-rule]: *HMA-M3 (P,S,Q) (?P,?S,?Q)* **by** *simp*  
**have** [transfer-rule]: *bezout = bezout ..*  
**have** *PSQ2: (?P,?S,?Q) = diagonal-to-Smith-PQ ?A bezout* **by** (*transfer, insert PSQ, auto*)  
**have** *?S = ?P\*\*?A\*\*?Q ∧ invertible ?P ∧ invertible ?Q ∧ Smith-normal-form ?S*  
**by** (*rule diagonal-to-Smith-PQ'[OF - ib PSQ2], transfer, auto simp add: d*)  
**with this[untransferred]** **show** *?thesis* **by** *auto*  
**qed**

**private lemma** *diagonal-to-Smith-PQ-JNF-mod-ring-with-types:*  
**assumes** *A: A ∈ carrier-mat CARD('nr::nontriv mod-ring) CARD('nc::nontriv mod-ring)*  
**and** *S: S ∈ carrier-mat CARD('nr mod-ring) CARD('nc mod-ring)*  
**and** *P: P ∈ carrier-mat CARD('nr mod-ring) CARD('nr mod-ring)*  
**and** *Q: Q ∈ carrier-mat CARD('nc mod-ring) CARD('nc mod-ring)*  
**and** *PSQ: diagonal-to-Smith-PQ-JNF A bezout = (P, S, Q)*  
**and** *d:isDiagonal-mat A and ib: is-bezout-ext bezout*  
**shows** *S = P \* A \* Q ∧ invertible-mat P ∧ invertible-mat Q ∧ Smith-normal-form-mat S*  
**by** (*rule diagonal-to-Smith-PQ-JNF-with-types[OF assms]*)

**thm** *diagonal-to-Smith-PQ-JNF-mod-ring-with-types[unfolded CARD-mod-ring, internalize-sort 'nr::nontriv]*

**private lemma** *diagonal-to-Smith-PQ-JNF-internalized-first:*  
*class.nontriv TYPE('a::type) ⇒*  
*A ∈ carrier-mat CARD('a) CARD('nc::nontriv) ⇒*  
*S ∈ carrier-mat CARD('a) CARD('nc) ⇒*  
*P ∈ carrier-mat CARD('a) CARD('a) ⇒*  
*Q ∈ carrier-mat CARD('nc) CARD('nc) ⇒*  
*diagonal-to-Smith-PQ-JNF A bezout = (P, S, Q) ⇒*  
*isDiagonal-mat A ⇒ is-bezout-ext bezout ⇒*  
*S = P \* A \* Q ∧ invertible-mat P ∧ invertible-mat Q ∧ Smith-normal-form-mat S*  
**using** *diagonal-to-Smith-PQ-JNF-mod-ring-with-types[unfolded CARD-mod-ring, internalize-sort 'nr::nontriv]* **by** *blast*



```

private lemma diagonal-to-Smith-PQ-JNF-internalized:
  class.nontriv TYPE('c::type)  $\implies$ 
  class.nontriv TYPE('a::type)  $\implies$ 
  A  $\in$  carrier-mat CARD('a) CARD('c)  $\implies$ 
  S  $\in$  carrier-mat CARD('a) CARD('c)  $\implies$ 
  P  $\in$  carrier-mat CARD('a) CARD('a)  $\implies$ 
  Q  $\in$  carrier-mat CARD('c) CARD('c)  $\implies$ 
  diagonal-to-Smith-PQ-JNF A bezout = (P, S, Q)  $\implies$ 
  isDiagonal-mat A  $\implies$  is-bezout-ext bezout  $\implies$ 
  S = P * A * Q  $\wedge$  invertible-mat P  $\wedge$  invertible-mat Q  $\wedge$  Smith-normal-form-mat S
  using diagonal-to-Smith-PQ-JNF-internalized-first[internalize-sort 'nc::nontriv]
by blast

```

**context**

```

  fixes m::nat and n::nat
  assumes local-typedef1:  $\exists$  (Rep :: ('b  $\Rightarrow$  int)) Abs. type-definition Rep Abs {0.. $m$  :: int}
  assumes local-typedef2:  $\exists$  (Rep :: ('c  $\Rightarrow$  int)) Abs. type-definition Rep Abs {0.. $n$  :: int}
  and m:  $m > 1$ 
  and n:  $n > 1$ 
begin

```

**lemma** *type-to-set1*:

```

  shows class.nontriv TYPE('b) (is ?a) and  $m = \text{CARD}('b)$  (is ?b)
proof -
  from local-typedef1 obtain Rep::('b  $\Rightarrow$  int) and Abs
  where t: type-definition Rep Abs {0.. $m$  :: int} by auto
  have card (UNIV :: 'b set) = card {0.. $m$ } using t type-definition.card by fastforce
  also have ... = m by auto
  finally show ?b ..
  then show ?a unfolding class.nontriv-def using m by auto
qed

```

**lemma** *type-to-set2*:

```

  shows class.nontriv TYPE('c) (is ?a) and  $n = \text{CARD}('c)$  (is ?b)
proof -
  from local-typedef2 obtain Rep::('c  $\Rightarrow$  int) and Abs
  where t: type-definition Rep Abs {0.. $n$  :: int} by blast
  have card (UNIV :: 'c set) = card {0.. $n$ } using t type-definition.card by force
  also have ... = n by auto
  finally show ?b ..
  then show ?a unfolding class.nontriv-def using n by auto
qed

```

**lemma** *diagonal-to-Smith-PQ-JNF-local-typedef*:

**assumes**  $A$ : *isDiagonal-mat*  $A$  **and**  $ib$ : *is-bezout-ext* *bezout*  
**and**  $A$ -*dim*:  $A \in \text{carrier-mat } m \ n$   
**assumes**  $PSQ$ :  $(P,S,Q) = \text{diagonal-to-Smith-PQ-JNF } A \ \text{bezout}$   
**shows**  $S = P * A * Q \wedge \text{invertible-mat } P \wedge \text{invertible-mat } Q \wedge \text{Smith-normal-form-mat } S$   
 $\wedge P \in \text{carrier-mat } m \ m \wedge S \in \text{carrier-mat } m \ n \wedge Q \in \text{carrier-mat } n \ n$   
**proof** –  
**have**  $\text{dim-matrices}$ :  $P \in \text{carrier-mat } m \ m \wedge S \in \text{carrier-mat } m \ n \wedge Q \in \text{carrier-mat } n \ n$   
**by** (*rule diagonal-to-Smith-PQ-JNF-dim*[*OF A-dim PSQ*[*symmetric*]])  
**show** *?thesis*  
**using** *diagonal-to-Smith-PQ-JNF-internalized*[**where** *?'c='c*, **where** *?'a='b*,  
*OF type-to-set2(1) type-to-set(1)*, *of m A S P Q*]  
**unfolding** *type-to-set1(2)*[*symmetric*] *type-to-set2(2)*[*symmetric*]  
**using** *assms m dim-matrices local-typedef1* **by** *auto*  
**qed**  
**end**  
**end**

**context**

**begin**

**private lemma** *diagonal-to-Smith-PQ-JNF-canceled-first*:

$\exists \text{Rep Abs. type-definition Rep Abs } \{0..<\text{int } n\} \implies \{0..<\text{int } m\} \neq \{\} \implies$   
 $1 < m \implies 1 < n \implies \text{isDiagonal-mat } A \implies \text{is-bezout-ext } \text{bezout} \implies$   
 $A \in \text{carrier-mat } m \ n \implies (P, S, Q) = \text{diagonal-to-Smith-PQ-JNF } A \ \text{bezout} \implies$   
 $S = P * A * Q \wedge \text{invertible-mat } P \wedge \text{invertible-mat } Q \wedge \text{Smith-normal-form-mat } S$

$\wedge P \in \text{carrier-mat } m \ m \wedge S \in \text{carrier-mat } m \ n \wedge Q \in \text{carrier-mat } n \ n$   
**using** *diagonal-to-Smith-PQ-JNF-local-typedef*[*cancel-type-definition*] **by** *blast*

**private lemma** *diagonal-to-Smith-PQ-JNF-canceled-both*:

$\{0..<\text{int } n\} \neq \{\} \implies \{0..<\text{int } m\} \neq \{\} \implies 1 < m \implies 1 < n \implies$   
 $\text{isDiagonal-mat } A \implies \text{is-bezout-ext } \text{bezout} \implies A \in \text{carrier-mat } m \ n \implies$   
 $(P, S, Q) = \text{diagonal-to-Smith-PQ-JNF } A \ \text{bezout} \implies S = P * A * Q \wedge$   
 $\text{invertible-mat } P \wedge \text{invertible-mat } Q \wedge \text{Smith-normal-form-mat } S$   
 $\wedge P \in \text{carrier-mat } m \ m \wedge S \in \text{carrier-mat } m \ n \wedge Q \in \text{carrier-mat } n \ n$   
**using** *diagonal-to-Smith-PQ-JNF-canceled-first*[*cancel-type-definition*] **by** *blast*

## 14.4 The final result

**lemma** *diagonal-to-Smith-PQ-JNF*:

**assumes**  $A$ : *isDiagonal-mat*  $A$  **and**  $ib$ : *is-bezout-ext* *bezout*  
**and**  $A \in \text{carrier-mat } m \ n$   
**and**  $PBQ$ :  $(P,S,Q) = \text{diagonal-to-Smith-PQ-JNF } A \ \text{bezout}$

**and**  $n$ :  $n > 1$  **and**  $m$ :  $m > 1$

**shows**  $S = P * A * Q \wedge \text{invertible-mat } P \wedge \text{invertible-mat } Q \wedge \text{Smith-normal-form-mat } S$

```

S
  ∧ P ∈ carrier-mat m m ∧ S ∈ carrier-mat m n ∧ Q ∈ carrier-mat n n
  using diagonal-to-Smith-PQ-JNF-canceled-both[OF - - m n]
  by (smt (verit, best) assms(1) assms(3) assms(4) assms(6) atLeastLessThan-empty-iff
gr-zeroI ib n not-less-iff-gr-or-eq of-nat-0-less-iff)
end
end

```

## 15 Smith normal form algorithm based on two steps in JNF

```

theory SNF-Algorithm-Two-Steps-JNF
  imports
    Diagonalize
    Diagonal-To-Smith-JNF
begin

```

### 15.1 Moving the result from HOL Analysis to JNF

```

context diagonalize
begin

```

```

definition Smith-normal-form-of-JNF A bezout = (
  let (P'',D,Q'') = diagonalize-JNF A bezout;
    (P',S,Q') = diagonal-to-Smith-PQ-JNF D bezout
  in (P'*P'',S,Q''*Q')
)

```

**lemma** *Smith-normal-form-of-JNF-soundness:*

```

assumes b: is-bezout-ext bezout and A: A ∈ carrier-mat m n
and n: 1 < n and m: 1 < m
and PSQ: Smith-normal-form-of-JNF A bezout = (P,S,Q)
shows S = P*A*Q ∧ invertible-mat P ∧ invertible-mat Q ∧ Smith-normal-form-mat S
  ∧ P ∈ carrier-mat m m ∧ S ∈ carrier-mat m n ∧ Q ∈ carrier-mat n n
proof –
  obtain P'' D Q'' where PDQ-diag: (P'',D,Q'') = diagonalize-JNF A bezout
  by (metis prod-cases3)
  have 1: invertible-mat P'' ∧ invertible-mat Q'' ∧ isDiagonal-mat D ∧ D =
P''*A*Q''
    ∧ P'' ∈ carrier-mat m m ∧ Q'' ∈ carrier-mat n n ∧ D ∈ carrier-mat m n
  using soundness-diagonalize-JNF[OF b A PDQ-diag[symmetric]] by auto
  obtain P' Q' where PSQ-D: (P',S,Q') = diagonal-to-Smith-PQ-JNF D bezout
  using PSQ PDQ-diag unfolding Smith-normal-form-of-JNF-def Let-def split-beta
  by (metis Pair-inject prod.collapse)
  have 2: invertible-mat P' ∧ invertible-mat Q' ∧ Smith-normal-form-mat S ∧ S

```

```

= P'*D*Q'
  ∧ P' ∈ carrier-mat m m ∧ Q' ∈ carrier-mat n n ∧ S ∈ carrier-mat m n
  using diagonal-to-Smith-PQ-JNF[OF - b - PSQ-D n m] 1 n m by auto
  have P: P = P'*P''
    by (metis (no-types, lifting) PDQ-diag PSQ PSQ-D Smith-normal-form-of-JNF-def
fst-conv prod.simps(2))
  have Q: Q = Q''*Q'
    by (metis (no-types, lifting) PDQ-diag PSQ PSQ-D Smith-normal-form-of-JNF-def
snd-conv prod.simps(2))
  have S = P'*(P''*A*Q')*Q' using 1 2 by auto
  also have ... = (P'*P'')*A*(Q''*Q')
    by (smt 1 2 A assoc-mult-mat carrier-matD carrier-mat-triv index-mult-mat)
  finally have S = (P' * P'') * A * (Q'' * Q') .
  moreover have invertible-mat P unfolding P by (rule invertible-mult-JNF,
insert 1 2, auto)
  moreover have invertible-mat Q unfolding Q by (rule invertible-mult-JNF,
insert 1 2, auto)
  ultimately show ?thesis using 1 2 P Q by auto
qed

end
end

```

## 16 A general algorithm to transform a matrix into its Smith normal form

```

theory SNF-Algorithm
  imports
    Smith-Normal-Form-JNF
begin

```

This theory presents an executable algorithm to transform a matrix to its Smith normal form.

### 16.1 Previous definitions and lemmas

```

definition is-SNF A R = (case R of (P,S,Q) ⇒
  P ∈ carrier-mat (dim-row A) (dim-row A) ∧
  Q ∈ carrier-mat (dim-col A) (dim-col A)
  ∧ invertible-mat P ∧ invertible-mat Q
  ∧ Smith-normal-form-mat S ∧ S = P * A * Q)

```

**lemma** *is-SNF-intro*:

```

assumes P ∈ carrier-mat (dim-row A) (dim-row A)
and Q ∈ carrier-mat (dim-col A) (dim-col A)
and invertible-mat P and invertible-mat Q
and Smith-normal-form-mat S and S = P * A * Q

```

**shows** *is-SNF*  $A (P, S, Q)$  **using** *assms unfolding is-SNF-def* **by** *auto*

**lemma** *Smith-1xn-two-matrices:*

**fixes**  $A :: 'a::comm-ring-1\ mat$

**assumes**  $A: A \in carrier\_mat\ 1\ n$

**and**  $PSQ: (P, S, Q) = (Smith-1xn\ A)$

**and** *is-SNF*: *is-SNF*  $A (Smith-1xn\ A)$

**shows**  $\exists Smith-1xn'. is-SNF\ A\ (1_m\ 1, (Smith-1xn'\ A))$

**proof** –

**let**  $?Q = P\ \$\ (0, 0) \cdot_m\ Q$

**have** *P00-dvd-1*:  $P\ \$\ (0, 0)\ dvd\ 1$

**by** (*metis (mono-tags, lifting) assms carrier-matD(1) determinant-one-element*

*invertible-iff-is-unit-JNF is-SNF-def prod.simps(2)*)

**have** *is-SNF*  $A\ (1_m\ 1, S, ?Q)$

**proof** (*rule is-SNF-intro*)

**show** *invertible-mat*  $(P\ \$\ (0, 0) \cdot_m\ Q)$

**by** (*rule invertible-mat-smult-mat, insert P00-dvd-1 assms, auto simp add: is-SNF-def*)

**show**  $S = 1_m\ 1 * A * (P\ \$\ (0, 0) \cdot_m\ Q)$

**by** (*smt (verit, ccfv-threshold) A PSQ is-SNF assoc-mult-mat carrier-matD(1) carrier-matD(2)*

*case-prodE is-SNF-def left-mult-one-mat mult-carrier-mat mult-smult-distrib prod.simps(1)*

*smult-mat-mat-one-element*)

**qed** (*insert assms, auto simp add: is-SNF-def*)

**thus** *?thesis* **by** *auto*

**qed**

**lemma** *Smith-1xn-two-matrices-all:*

**assumes** *is-SNF*:  $\forall (A::'a::comm-ring-1\ mat) \in carrier\_mat\ 1\ n. is-SNF\ A\ (Smith-1xn\ A)$

**shows**  $\exists Smith-1xn'. \forall (A::'a::comm-ring-1\ mat) \in carrier\_mat\ 1\ n. is-SNF\ A\ (1_m\ 1, (Smith-1xn'\ A))$

**proof** –

**let**  $?Smith-1xn' = \lambda A. let\ (P, S, Q) = (Smith-1xn\ A)\ in\ (S, P\ \$\ (0, 0) \cdot_m\ Q)$

**show** *?thesis*

**by** (*rule exI[of - ?Smith-1xn'] (smt (verit, ccfv-threshold) Smith-1xn-two-matrices assms carrier-matD*

*carrier-matI case-prodE determinant-one-element index-smult-mat(2,3) invertible-iff-is-unit-JNF*

*invertible-mat-smult-mat smult-mat-mat-one-element left-mult-one-mat is-SNF-def*

*mult-smult-assoc-mat mult-smult-distrib prod.simps(2)*)

**qed**

## 16.2 Previous operations

**context**

**assumes** *SORT-CONSTRAINT*('a::comm-ring-1)

**begin**

**definition** *is-div-op* :: ('a⇒'a⇒'a) ⇒ bool

**where** *is-div-op* *div-op* = (∀ a b. b dvd a → *div-op* a b \* b = a)

**lemma** *div-op-SOME*: *is-div-op* (λa b. (SOME k. k \* b = a))

**proof** (*unfold is-div-op-def, rule+*)

**fix** a b::'a **assume** *dvd*: b dvd a

**show** (SOME k. k \* b = a) \* b = a **by** (*rule someI-ex, insert dvd dvd-def*) (*metis dvdE mult.commute*)

**qed**

**fun** *reduce-column-aux* :: ('a⇒'a⇒'a) ⇒ nat list ⇒ 'a mat ⇒ ('a mat × 'a mat) ⇒ ('a mat × 'a mat)

**where** *reduce-column-aux* *div-op* [] H (P,K) = (P,K)

| *reduce-column-aux* *div-op* (i#xs) H (P,K) = (

— Reduce the i-th row

let k = *div-op* (H\$(i,0)) (H \$(0, 0));

P' = *addrow-mat* (*dim-row* H) (-k) i 0;

K' = *addrow* (-k) i 0 K

in *reduce-column-aux* *div-op* xs H (P'\*P,K')

)

**definition** *reduce-column* *div-op* H = *reduce-column-aux* *div-op* [2..*dim-row* H] H (1<sub>m</sub> (*dim-row* H),H)

**lemma** *reduce-column-aux*:

**assumes** H: H ∈ *carrier-mat* m n

**and** P-init: P-init ∈ *carrier-mat* m m

**and** K-init: K-init ∈ *carrier-mat* m n

**and** P-init-H-K-init: P-init \* H = K-init

**and** PK-H: (P,K) = *reduce-column-aux* *div-op* xs H (P-init,K-init)

**and** m: 0 < m

**and** inv-P: *invertible-mat* P-init

**and** xs: 0 ∉ set xs

**shows** P ∈ *carrier-mat* m m ∧ K ∈ *carrier-mat* m n ∧ P \* H = K ∧ *invertible-mat* P

**using** *assms*

**unfolding** *reduce-column-def*

**proof** (*induct* *div-op* xs H (P-init,K-init) *arbitrary*: P-init K-init *rule*: *reduce-column-aux.induct*)

**case** (1 *div-op* H P K)

**then show** ?*case* **by** *simp*

**next**

**case** (2 *div-op* i xs H P-init K-init)

```

show ?case
proof (rule 2.hyps)
  let ?x = div-op (H $$ (i, 0)) (H $$ (0, 0))
  let ?xa = addrow-mat (dim-row H) (- ?x) i 0
  let ?xb = addrow (- ?x) i 0 K-init
  show (P, K) = reduce-column-aux div-op xs H (?xa * P-init, ?xb)
    using 2.prem1 by (auto simp add: Let-def)
  show ?xa * P-init ∈ carrier-mat m m using 2(2) 2(3) by auto
  show 0 ∉ set xs using 2.prem1 by auto
  have ?xa * K-init = ?xb
    by (rule addrow-mat[symmetric], insert 2.prem1, auto)
  thus ?xa * P-init * H = ?xb
  by (metis (no-types, lifting) 2(5) 2.prem1(1) 2.prem1(2) addrow-mat-carrier

      assoc-mult-mat carrier-matD(1))
  show invertible-mat (?xa * P-init)
proof (rule invertible-mult-JNF)
  show xa: ?xa ∈ carrier-mat m m using 2(2) by auto
  have Determinant.det ?xa = 1 by (rule det-addrow-mat, insert 2.prem1,
auto)
  thus invertible-mat ?xa unfolding invertible-iff-is-unit-JNF[OF xa] by simp

  qed (auto simp add: 2.prem1)
qed(auto simp add: 2.prem1)
qed

```

**lemma** reduce-column-aux-preserves:

```

assumes H: H ∈ carrier-mat m n
  and P-init: P-init ∈ carrier-mat m m
  and K-init: K-init ∈ carrier-mat m n
and P-init-H-K-init: P-init * H = K-init
and PK-H: (P,K) = reduce-column-aux div-op xs H (P-init,K-init)
and m: 0 < m
and inv-P: invertible-mat P-init
and xs: 0 ∉ set xs and i: i ∉ set xs and im: i < m
shows Matrix.row K i = Matrix.row K-init i
  using PK-H inv-P H P-init K-init m xs i
  unfolding reduce-column-def
proof (induct div-op xs H (P-init,K-init) arbitrary: P-init K-init K rule: reduce-column-aux.induct)
  case (1 div-op H P K)
  then show ?case by auto
next
  case (2 div-op x xs H P-init K-init)
  thm 2.prem1
  2.hyps
  let ?x = div-op (H $$ (x, 0)) (H $$ (0, 0))
  let ?xa = addrow-mat (dim-row H) (- ?x) x 0
  let ?xb = addrow (- ?x) x 0 K-init

```

```

have IH: Matrix.row K i = Matrix.row ?xb i
proof (rule 2.hyps)
  show (P, K) = reduce-column-aux div-op xs H (?xa * P-init, ?xb)
    using 2.prem1 by (auto simp add: Let-def)
  show ?xa * P-init ∈ carrier-mat m m
    using 2(4) 2(5) by auto
  have ?xa * K-init = ?xb
    by (rule addrow-mat[symmetric], insert 2.prem1, auto)
  show invertible-mat (?xa * P-init)
  proof (rule invertible-mult-JNF)
    show xa: ?xa ∈ carrier-mat m m using 2.prem1 by auto
    have Determinant.det ?xa = 1 by (rule det-addrow-mat, insert 2.prem1,
auto)
    thus invertible-mat ?xa unfolding invertible-iff-is-unit-JNF[OF xa] by
simp
  qed (auto simp add: 2.prem1)
  show i ∉ set xs using 2(9) by auto
  show 0 ∉ set xs using 2(8) by auto
  qed(auto simp add: 2.prem1)
  also have ... = Matrix.row K-init i
    by (rule eq-vecI, auto, insert 2 2.prem1 im, auto)
  finally show ?case .
qed

lemma reduce-column-aux-index':
  assumes H: H ∈ carrier-mat m n
    and P-init: P-init ∈ carrier-mat m m
    and K-init: K-init ∈ carrier-mat m n
  and P-init-H-K-init: P-init * H = K-init
  and PK-H: (P,K) = reduce-column-aux div-op xs H (P-init,K-init)
  and m: 0 < m
  and inv-P: invertible-mat P-init
  and xs: 0 ∉ set xs
  and ∀ x∈set xs. x < m
  and distinct xs
shows (∀ i∈set xs. Matrix.row K i =
  Matrix.row (addrow (-(div-op (H $$ (i, 0)) (H $$ (0, 0)))) i 0 K-init) i)
  using assms
  unfolding reduce-column-def
proof (induct div-op xs H (P-init,K-init) arbitrary: P-init K-init K rule: reduce-column-aux.induct)
  case (1 div-op H P K)
  then show ?case by simp
next
  case (2 div-op i xs H P-init K-init)
  let ?x = div-op (H $$ (i, 0)) (H $$ (0, 0))
  let ?xa = addrow-mat (dim-row H) ?x i 0
  thm 2.prem1
  thm 2.hyps
  let ?xb = addrow (- ?x) i 0 K-init

```



```

let ?xa = addrow-mat (dim-row H) (- ?x) i 0
have reduce-column-aux div-op (i#xs) H (P-init,K-init)
  = reduce-column-aux div-op xs H (?xa*P-init,?xb)
by (auto simp add: Let-def)
hence PK: (P,K) = reduce-column-aux div-op xs H (?xa*P-init,?xb) using
2.premis by simp
  have xa-P-init: ?xa * P-init ∈ carrier-mat m m using 2(2) 2(3) by auto
  have zero-notin-xs: 0 ∉ set xs using 2.premis by auto
  have ?xa * K-init = ?xb
    by (rule addrow-mat[symmetric], insert 2.premis, auto)
  hence rw: ?xa * P-init * H = ?xb
  by (metis (no-types, lifting) 2(5) 2.premis(1) 2.premis(2) addrow-mat-carrier

      assoc-mult-mat carrier-matD(1))
  have inv-xa-P-init: invertible-mat (?xa * P-init)
  proof (rule invertible-mult-JNF)
    show xa: ?xa ∈ carrier-mat m m using 2(2) by auto
    have Determinant.det ?xa = 1 by (rule det-addrow-mat, insert 2.premis,
auto)
  thus invertible-mat ?xa unfolding invertible-iff-is-unit-JNF[OF xa] by simp

qed (auto simp add: 2.premis)
have i1: i≠0 using 2.premis(8) by auto
have i2: i<m by (simp add: 2.premis(9))
have i3: i∉set xs using 2 by auto
have d: distinct xs using 2 by auto
have ∀i∈set xs. Matrix.row K i = Matrix.row (addrow (- (div-op (H $$ (i,
0)) (H $$ (0, 0))))
  i 0 ?xb) i
  by (rule 2.hyps, insert xa-P-init zero-notin-xs rw inv-xa-P-init d,
auto simp add: 2.premis Let-def)
moreover have Matrix.row (addrow (- (div-op (H $$ (j, 0)) (H $$ (0, 0)))) j
0 ?xb) j
= Matrix.row (addrow (- (div-op (H $$ (j, 0)) (H $$ (0, 0)))) j 0 K-init) j
(is Matrix.row ?lhs j= Matrix.row ?rhs j)
if j: j ∈ set xs for j
proof (rule eq-vecI)
  fix ia assume ia: ia<dim-vec(Matrix.row ?rhs j)
  let ?k = div-op (H $$ (j, 0)) (H $$ (0, 0))
  let ?L = (addrow (- (div-op (H $$ (i, 0)) (H $$ (0, 0)))) i 0 K-init)
  have Matrix.row ?lhs j $v ia = ?lhs $$ (j,ia)
    by (metis (no-types, lifting) Matrix.row-def ia index-mat-addrow(5) in-
dex-row(2) index-vec)
  also have ... = (-?k) * ?L$(0,ia) + ?L$(j,ia)
  by (smt (verit) 2.premis(1) 2.premis(9) carrier-matD(1) ia index-mat-addrow(1,5)
index-row(2)
insert-iff list.set(2) mult-carrier-mat rw that xa-P-init)
  also have ... = ?rhs $$ (j,ia) using 2(10) 2(4) i1 i3 ia j by auto
  also have ... = Matrix.row ?rhs j $v ia using 2 ia j by auto

```

**finally show**  $Matrix.row \ ?lhs \ j \ \$v \ ia = Matrix.row \ ?rhs \ j \ \$v \ ia .$   
**qed** (*auto*)  
**ultimately have**  $\forall j \in set \ xs. Matrix.row \ K \ j =$   
 $Matrix.row \ (addrow \ (- \ (div-op \ (H \ \$\$ \ (j, 0)) \ (H \ \$\$ \ (0, 0)))) \ j \ 0 \ K-init) \ j$  **by**  
*auto*  
**moreover have**  $Matrix.row \ K \ i = Matrix.row \ ?xb \ i$   
**by** (*rule reduce-column-aux-preserves[OF - xa-P-init - rw PK - inv-xa-P-init*  
*zero-notin-xs*  
*i3 i2],insert 2.premis, auto*)  
**ultimately show** *?case by auto*  
**qed**

**corollary** *reduce-column-aux-index:*

**assumes**  $H: H \in carrier-mat \ m \ n$   
**and**  $P-init: P-init \in carrier-mat \ m \ m$   
**and**  $K-init: K-init \in carrier-mat \ m \ n$   
**and**  $P-init-H-K-init: P-init * H = K-init$   
**and**  $PK-H: (P,K) = reduce-column-aux \ div-op \ xs \ H \ (P-init,K-init)$   
**and**  $m: 0 < m$   
**and**  $inv-P: invertible-mat \ P-init$   
**and**  $xs: 0 \notin set \ xs$   
**and**  $\forall x \in set \ xs. x < m$   
**and** *distinct xs*  
**and**  $i \in set \ xs$

**shows**  $Matrix.row \ K \ i =$

$Matrix.row \ (addrow \ (- \ (div-op \ (H \ \$\$ \ (i, 0)) \ (H \ \$\$ \ (0, 0)))) \ i \ 0 \ K-init) \ i$   
**using** *reduce-column-aux-index' assms by simp*

**corollary** *reduce-column-aux-works:*

**assumes**  $H: H \in carrier-mat \ m \ n$   
**and**  $PK-H: (P,K) = reduce-column-aux \ div-op \ xs \ H \ (1_m \ (dim-row \ H), H)$   
**and**  $m: 0 < m$   
**and**  $xs: 0 \notin set \ xs$   
**and**  $xm: \forall x \in set \ xs. x < m$   
**and**  $d-xs: distinct \ xs$   
**and**  $i: i \in set \ xs$   
**and**  $dvd: H \ \$\$ \ (0, 0) \ dvd \ H \ \$\$ \ (i, 0)$   
**and**  $j0: \forall j \in \{1..<n\}. H \ \$\$ \ (0, j) = 0$   
**and**  $j1n: j \in \{1..<n\}$   
**and**  $n: 0 < n$   
**and**  $id: is-div-op \ div-op$

**shows**  $K \ \$\$ \ (i, 0) = 0$  **and**  $K \ \$\$ \ (i, j) = H \ \$\$ \ (i, j)$

**proof** –

**let**  $?k = div-op \ (H \ \$\$ \ (i, 0)) \ (H \ \$\$ \ (0, 0))$   
**let**  $?L = addrow \ (- ?k) \ i \ 0 \ H$   
**have**  $kH00-eq-Hi0: ?k * H \ \$\$ \ (0, 0) = H \ \$\$ \ (i, 0)$   
**using** *id dvd unfolding is-div-op-def by simp*  
**have**  $*$ :  $Matrix.row \ K \ i = Matrix.row \ ?L \ i$

by (rule reduce-column-aux-index[OF H - - - PK-H], insert assms, auto)  
 also have ...  $v\ 0 = ?L\ \$\$ (i,0)$  by (rule index-row, insert xm i H n, auto)  
 also have ...  $= (-\ ?k) * H\ \$\$ (0,0) + H\ \$\$ (i,0)$  by (rule index-mat-addrw, insert  
 i xm H n, auto)  
 also have ...  $= 0$  using kH00-eq-Hi0 by auto  
 finally show  $K\ \$\$ (i, 0) = 0$   
 by (metis H Matrix.row-def \* n carrier-matD(2) dim-vec index-mat-addrw(5)  
 index-vec)  
 have Matrix.row ?L i  $v\ j = ?L\ \$\$ (i,j)$  by (rule index-row, insert xm i H n j1n,  
 auto)  
 also have ...  $= (-\ ?k) * H\ \$\$ (0,j) + H\ \$\$ (i,j)$  by (rule index-mat-addrw, insert  
 xm i H j1n, auto)  
 also have ...  $= H\ \$\$ (i,j)$  using j1n j0 by auto  
 finally show  $K\ \$\$ (i,j) = H\ \$\$ (i,j)$  by (metis H \* Matrix.row-def atLeast-  
 LessThan-iff  
 carrier-matD(2) dim-vec index-mat-addrw(5) index-vec j1n)  
 qed

**lemma** reduce-column:

assumes  $H: H \in \text{carrier-mat } m\ n$   
 and PK-H:  $(P,K) = \text{reduce-column div-op } H$   
 and  $m: 0 < m$   
 shows  $P \in \text{carrier-mat } m\ m \wedge K \in \text{carrier-mat } m\ n \wedge P * H = K \wedge \text{invertible-mat } P$   
 by (rule reduce-column-aux[OF - - - - PK-H[unfolded reduce-column-def]], insert  
 assms, auto)

**lemma** reduce-column-preserves:

assumes  $H: H \in \text{carrier-mat } m\ n$   
 and PK-H:  $(P,K) = \text{reduce-column div-op } H$   
 and  $m: 0 < m$   
 and  $i \in \{0,1\}$   
 and  $i < m$   
 shows Matrix.row  $K\ i = \text{Matrix.row } H\ i$   
 by (rule reduce-column-aux-preserves[OF - - - - PK-H[unfolded reduce-column-def]],  
 insert assms, auto)

**lemma** reduce-column-preserves2:

assumes  $H: H \in \text{carrier-mat } m\ n$   
 and PK-H:  $(P,K) = \text{reduce-column div-op } H$   
 and  $m: 0 < m$  and  $i: i \in \{0,1\}$  and  $im: i < m$  and  $j: j < n$   
 shows  $K\ \$\$ (i,j) = H\ \$\$ (i,j)$   
 using reduce-column-preserves[OF H PK-H m i im]  
 by (metis H Matrix.row-def j carrier-matD(2) dim-vec index-vec)

**corollary** reduce-column-works:

assumes  $H: H \in \text{carrier-mat } m\ n$

```

and PK-H: (P,K) = reduce-column div-op H
and m: 0 < m
and dvd: H $$ (0, 0) dvd H $$ (i, 0)
and j0:  $\forall j \in \{1..<n\}. H \text{ } \$\$ (0, j) = 0$ 
and j1n:  $j \in \{1..<n\}$ 
and n: 0 < n
and i  $\in \{2..<m\}$ 
and id: is-div-op div-op
shows K $$ (i,0) = 0 and K $$ (i,j) = H $$ (i,j)
  by (rule reduce-column-aux-works[OF H PK-H[unfolded reduce-column-def]],
insert assms, auto)+

end

```

### 16.3 The implementation

We define a locale where we implement the algorithm. It has three fixed operations:

1. an operation to transform any  $1 \times 2$  matrix into its Smith normal form
2. an operation to transform any  $2 \times 2$  matrix into its Smith normal form
3. an operation that provides a witness for division (this operation always exists over a commutative ring with unit, but maybe we cannot provide a computable algorithm).

Since we are working in a commutative ring, we can easily get an operation for  $2 \times 1$  matrices via the  $1 \times 2$  operation.

```

locale Smith-Impl =
  fixes Smith-1x2 :: ('a::comm-ring-1) mat  $\Rightarrow$  ('a mat  $\times$  'a mat)
    and Smith-2x2 :: 'a mat  $\Rightarrow$  ('a mat  $\times$  'a mat  $\times$  'a mat)
    and div-op :: 'a  $\Rightarrow$  'a  $\Rightarrow$  'a
  assumes SNF-1x2-works:  $\forall (A::'a \text{ mat}) \in \text{carrier-mat } 1 \ 2. \text{ is-SNF } A (1_m \ 1, \text{Smith-1x2 } A)$ 
    and SNF-2x2-works:  $\forall (A::'a \text{ mat}) \in \text{carrier-mat } 2 \ 2. \text{ is-SNF } A (\text{Smith-2x2 } A)$ 
    and id: is-div-op div-op
begin

```

From a  $2 \times 2$  matrix (the  $B$ ), we construct the identity matrix of size  $n$  with the elements of  $B$  placed to modify the first element of a matrix and the element in position  $(k, k)$

```

definition make-mat n k (B::'a mat) = (Matrix.mat n n ( $\lambda(i,j). \text{ if } i = 0 \wedge j = 0$ 
then B $$ (0,0) else
  if  $i = 0 \wedge j = k$  then B $$ (0,1) else if  $i=k \wedge j = 0$ 
then B $$ (1,0) else if  $i=k \wedge j=k$  then B $$ (1,1)
else if  $i=j$  then 1 else 0))

```

**lemma** *make-mat-carrier*[simp]:  
 shows *make-mat*  $n$   $k$   $B \in \text{carrier-mat } n$   $n$   
 unfolding *make-mat-def* by *auto*

**lemma** *upper-triangular-mat-delete-make-mat*:  
 shows *upper-triangular* (*mat-delete* (*make-mat*  $n$   $k$   $B$ )  $0$   $0$ )  
**proof** –  
 { let  $?M = \text{make-mat } n$   $k$   $B$   
 fix  $i$   $j$   
 assume  $i < \text{dim-row } ?M - \text{Suc } 0$  and  $ji: j < i$   
 hence  $i-n1: i < n - 1$  by (*simp add: make-mat-def*)  
 hence  $\text{Suc-}i: \text{Suc } i < n$  by *linarith*  
 hence  $\text{Suc-}j: \text{Suc } j < n$  using  $ji$  by *auto*  
 have  $i1: \text{insert-index } 0$   $i = \text{Suc } i$  by (*rule insert-index, auto*)  
 have  $j1: \text{insert-index } 0$   $j = \text{Suc } j$  by (*rule insert-index, auto*)  
 have *mat-delete*  $?M$   $0$   $0$   $\$ \$ (i, j) = ?M$   $\$ \$ (\text{insert-index } 0$   $i, \text{insert-index } 0$   $j)$   
 by (*rule mat-delete-index*[*symmetric, OF - - i-n1*], *insert Suc-i Suc-j, auto*)  
 also have  $\dots = ?M$   $\$ \$ (\text{Suc } i, \text{Suc } j)$  unfolding  $i1$   $j1$  by *simp*  
 also have  $\dots = 0$  unfolding *make-mat-def* unfolding *index-mat*[*OF Suc-i Suc-j*]  
 using  $ji$  by *auto*  
 finally have *mat-delete*  $?M$   $0$   $0$   $\$ \$ (i, j) = 0$  .  
 }  
 thus *?thesis* unfolding *upper-triangular-def* by *auto*  
 qed

**lemma** *upper-triangular-mat-delete-make-mat2*:  
 assumes  $kn: k < n$   
 shows *upper-triangular* (*mat-delete* (*mat-delete* (*make-mat*  $n$   $k$   $B$ )  $0$   $k$ ) ( $k - 1$ )  
 $0$ )  
**proof** –  
 { let  $?M = \text{local.make-mat } n$   $k$   $B$   
 let  $?MD = \text{mat-delete } ?M$   $0$   $k$   
 fix  $i$   $j$  assume  $i: i < \text{dim-row } ?M - 2$  and  $ji: j < i$   
 have *insert-in: insert-index*  $0$   $i < n$  and *insert-Sucin: insert-index*  $0$  ( $\text{Suc } i$ )  $<$   
 $n$   
 using  $i$  *make-mat-def* by *auto*  
 have *insert-k-Sucj: insert-index*  $k$  ( $\text{Suc } j$ )  $< n$   
 using *insert-in insert-index-def*  $ji$  by *auto*  
 have *insert-j: insert-index*  $0$   $j = \text{Suc } j$  by *simp*  
 have *mat-delete*  $?MD$  ( $k - 1$ )  $0$   $\$ \$ (i, j) = ?MD$   $\$ \$ (\text{insert-index } (k-1)$   $i,$   
*insert-index*  $0$   $j)$   
**proof** (*rule mat-delete-index*[*symmetric*])  
 show  $i < n-2$  using  $i$  by (*simp add: make-mat-def*)  
 thus  $?MD \in \text{carrier-mat } (\text{Suc } (n - 2))$  ( $\text{Suc } (n - 2)$ )  
 by (*metis Suc-diff-Suc card-num-simps*(30) *make-mat-carrier mat-delete-carrier*  
  
*nat-diff-split-asm not-less0 not-less-eq numerals*(2))  
 show  $k - 1 < \text{Suc } (n - 2)$  using  $kn$  by *auto*  
 show  $0 < \text{Suc } (n - 2)$  by *blast*

```

    show  $j < n - 2$  using  $ji\ i$  by (simp add: make-mat-def)
  qed
  also have ... = ?MD $$ (insert-index (k-1) i, Suc j) unfolding insert-j by
auto
  also have ... = 0
  proof (cases  $i < (k-1)$ )
    case True
      hence insert-index (k-1) i = i by auto
      hence ?MD $$ (insert-index (k-1) i, Suc j) = ?MD $$ (i, Suc j) by auto
      also have ... = ?M $$ (insert-index 0 i, insert-index k (Suc j))
      proof (rule mat-delete-index[symmetric])
        show ?M  $\in$  carrier-mat (Suc (n-1)) (Suc (n-1)) using assms by auto
        show  $0 < \text{Suc } (n - 1)$ 
          by blast
        show  $k < \text{Suc } (n - 1)$  using  $kn$  by simp
        show  $i < n - 1$  using  $i$  using True assms by linarith
        thus  $\text{Suc } j < n - 1$  using  $ji$  less-trans-Suc by blast
      qed
    also have ... = 0 unfolding make-mat-def index-mat[OF insert-in insert-k-Sucj]
      using True  $ji$  by auto
    finally show ?thesis .
  next
    case False
      hence insert-index (k-1) i = Suc i by auto
      hence ?MD $$ (insert-index (k-1) i, Suc j) = ?MD $$ (Suc i, Suc j) by
auto
      also have ... = ?M $$ (insert-index 0 (Suc i), insert-index k (Suc j))
      proof (rule mat-delete-index[symmetric])
        show ?M  $\in$  carrier-mat (Suc (n-1)) (Suc (n-1)) using assms by auto
        thus  $\text{Suc } i < n - 1$  using  $i$  using False assms
        by (metis One-nat-def Suc-diff-Suc carrier-matD(1) diff-Suc-1 diff-Suc-eq-diff-pred

          diff-is-0-eq' linorder-not-less nat.distinct(1) numeral-2-eq-2)
        show  $0 < \text{Suc } (n - 1)$ 
          by blast
        show  $k < \text{Suc } (n - 1)$  using  $kn$  by simp
        show  $\text{Suc } j < n - 1$  using  $ji$  less-trans-Suc
          using  $\langle \text{Suc } i < n - 1 \rangle$  by linarith
      qed
    also have ... = 0 unfolding make-mat-def index-mat[OF insert-Sucin in-
sert-k-Sucj]
      using False  $ji$  by (simp add: insert-index-def)
    finally show ?thesis .
  qed
  finally have mat-delete ?MD (k - 1) 0 $$ (i, j) = 0 .
}
thus ?thesis unfolding upper-triangular-def by auto
qed

```

**corollary** *det-mat-delete-make-mat*:

**assumes**  $kn: k < n$   
**shows**  $\text{Determinant.det (mat-delete (mat-delete (make-mat n k B) 0 k) (k - 1) 0) = 1}$   
**proof** –  
**let**  $?M = \text{make-mat } n \ k \ B$   
**let**  $?MD = \text{mat-delete } ?M \ 0 \ k$   
**let**  $?MDMD = \text{mat-delete } ?MD \ (k - 1) \ 0$   
**have**  $eq1: ?MDMD \ \$\$ \ (i, i) = 1$  **if**  $i: i < n - 2$  **for**  $i$   
**proof** –  
**have**  $i1: \text{insert-index } 0 \ (\text{insert-index } (k - 1) \ i) < n$  **using**  $i$  **insert-index-def** **by** *auto*  
**have**  $i2: \text{insert-index } k \ (\text{insert-index } 0 \ i) < n$  **using**  $i$  **insert-index-def** **by** *auto*  
**have**  $?MDMD \ \$\$ \ (i, i) = ?MD \ \$\$ \ (\text{insert-index } (k - 1) \ i, \text{insert-index } 0 \ i)$   
**proof** (*rule mat-delete-index[symmetric, OF - - - i i]*)  
**show**  $\text{mat-delete (local.make-mat n k B) 0 k} \in \text{carrier-mat (Suc (n - 2)) (Suc (n - 2))}$   
**by** (*metis (mono-tags, opaque-lifting) Suc-diff-Suc card-num-simps(30) i make-mat-carrier*  
*mat-delete-carrier nat-diff-split-asm not-less0 not-less-eq numerals(2)*)  
**show**  $k - 1 < \text{Suc } (n - 2)$  **using**  $kn$  **by** *auto*  
**show**  $0 < \text{Suc } (n - 2)$  **using**  $kn$  **by** *auto*  
**qed**  
**also have**  $\dots = ?M \ \$\$ \ (\text{insert-index } 0 \ (\text{insert-index } (k - 1) \ i), \text{insert-index } k \ (\text{insert-index } 0 \ i))$   
**proof** (*rule mat-delete-index[symmetric]*)  
**show**  $\text{make-mat } n \ k \ B \in \text{carrier-mat (Suc (n - 1)) (Suc (n - 1))}$  **using**  $i$  **by** *auto*  
**show**  $\text{insert-index } (k - 1) \ i < n - 1$  **using**  $kn \ i$   
**by** (*metis diff-Suc-eq-diff-pred diff-commute insert-index-def nat-neq-iff not-less0 numeral-2-eq-2 zero-less-diff*)  
**show**  $\text{insert-index } 0 \ i < n - 1$  **using**  $i$  **by** *auto*  
**qed** (*insert kn, auto*)  
**also have**  $\dots = 1$  **unfolding** *make-mat-def index-mat[OF i1 i2]*  
**by** (*auto, metis One-nat-def diff-Suc-1 insert-index-exclude*  
*(metis One-nat-def diff-Suc-eq-diff-pred insert-index-def zero-less-diff)+*)  
**finally show** *?thesis* .  
**qed**  
**have**  $\text{Determinant.det } ?MDMD = \text{prod-list (diag-mat } ?MDMD)$   
**by** (*meson assms det-upper-triangular make-mat-carrier mat-delete-carrier upper-triangular-mat-delete-make-mat2*)  
**also have**  $\dots = 1$   
**proof** (*rule prod-list-neutral*)  
**fix**  $x$  **assume**  $x: x \in \text{set (diag-mat } ?MDMD)$   
**from this obtain**  $i$  **where**  $\text{index: } x = ?MDMD \ \$\$ \ (i, i)$  **and**  $i: i < \text{dim-row } ?MDMD$   
**unfolding** *diag-mat-def* **by** *auto*  
**have**  $?MDMD \ \$\$ \ (i, i) = 1$  **by** (*rule eq1, insert i, auto simp add: make-mat-def*)

```

    thus  $x=1$  using index by blast
  qed
  finally show ?thesis .
qed

lemma swaprows-make-mat:
  assumes  $B: B \in \text{carrier-mat } 2 \ 2$  and  $k0: k \neq 0$  and  $k: k < n$ 
  shows  $\text{swaprows } k \ 0 \ (\text{make-mat } n \ k \ B) = \text{make-mat } n \ k \ (\text{swaprows } 1 \ 0 \ B)$  (is
  ?lhs = ?rhs)
proof (cases  $n=0$ )
  case True
  then show ?thesis
    using make-mat-def by auto
next
  case False
  show ?thesis
  proof (rule eq-matI)
    show  $\text{dim-row } ?lhs = \text{dim-row } ?rhs$  and  $\text{dim-col } ?lhs = \text{dim-col } ?rhs$ 
      by (simp add: make-mat-def)+
  next
    let  $?M = (\text{make-mat } n \ k \ B)$ 
    fix  $i \ j$  assume  $i: i < \text{dim-row } ?rhs$  and  $j: j < \text{dim-col } ?rhs$ 
    hence  $i2: i < \text{dim-row } ?lhs$  and  $j2: j < \text{dim-col } ?lhs$  by (auto simp add:
    make-mat-def)
    then have  $i3: i < \text{dim-row } ?M$  and  $j3: j < \text{dim-col } ?M$  by auto
    then have  $i4: i < n$  and  $j4: j < n$  by (metis carrier-matD(1,2) make-mat-carrier)+
    have  $lhs: ?lhs \ \$\$ \ (i, j) =$ 
      (if  $k = i$  then  $?M \ \$\$ \ (0, j)$  else if  $0 = i$  then  $?M \ \$\$ \ (k, j)$  else  $?M \ \$\$ \ (i, j)$ )
      by (rule index-mat-swaprows, insert i3 j3, auto)
    also have  $\dots = ?rhs \ \$\$ \ (i, j)$  using  $B \ i4 \ j4 \ \text{False} \ k0 \ k$ 
      unfolding make-mat-def index-mat[OF i4 j4] by auto
    finally show  $?lhs \ \$\$ \ (i, j) = ?rhs \ \$\$ \ (i, j)$  .
  qed
qed

```

```

lemma cofactor-make-mat-00:
  assumes  $k: k < n$  and  $k0: k \neq 0$ 
  shows  $\text{cofactor } (\text{make-mat } n \ k \ B) \ 0 \ 0 = B \ \$\$ \ (1, 1)$ 
proof -
  let  $?M = \text{make-mat } n \ k \ B$ 
  let  $?MD = \text{mat-delete } ?M \ 0 \ 0$ 
  have  $MD\text{-rows: dim-row } ?MD = n - 1$  by (simp add: make-mat-def)
  have  $1: ?MD \ \$\$ \ (i, i) = 1$  if  $i: i < n - 1$  and  $ik: \text{Suc } i \neq k$  for  $i$ 
  proof -
    have Suc-i: Suc i < n using  $i$  by linarith
    have  $?MD \ \$\$ \ (i, i) = ?M \ \$\$ \ (\text{insert-index } 0 \ i, \text{insert-index } 0 \ i)$ 
      by (rule mat-delete-index[symmetric, OF - - - i], insert Suc-i, auto)
  qed

```



**also have** ... = ?M \$\$ (Suc i, Suc i) **by simp**  
**also have** ... = 1 **unfolding** make-mat-def index-mat[OF Suc-i Suc-i] **using**  
*ik by auto*  
**finally show** ?thesis .  
**qed**  
**have** ?2: ?MD \$\$ (i, i) = B\$\$\$(1,1) **if** i: i < n - 1 **and** ik: Suc i = k **for** i  
**proof** -  
**have** Suc-i: Suc i < n **using** i **by** linarith  
**have** ?MD \$\$ (i, i) = ?M \$\$ (insert-index 0 i, insert-index 0 i)  
**by** (rule mat-delete-index[symmetric, OF - - - i], insert Suc-i, auto)  
**also have** ... = ?M \$\$ (Suc i, Suc i) **by simp**  
**also have** ... = B\$\$\$(1,1) **unfolding** make-mat-def index-mat[OF Suc-i Suc-i]  
**using ik by auto**  
**finally show** ?thesis .  
**qed**  
**have** set-rw: insert (k-1) ({0..<dim-row ?MD}-{k-1}) = {0..<dim-row ?MD}  
  
**using** k k0 MD-rows **by auto**  
**have** up: upper-triangular ?MD **by** (rule upper-triangular-mat-delete-make-mat)  
**have** Determinant.cofactor (local.make-mat n k B) 0 0  
= Determinant.det (mat-delete (make-mat n k B) 0 0) **unfolding** cofactor-def  
**by auto**  
**also have** ... = prod-list (diag-mat ?MD) **using** up  
**using** det-upper-triangular make-mat-carrier mat-delete-carrier **by blast**  
**also have** ... = ( $\prod$  i = 0..<dim-row ?MD. ?MD \$\$ (i, i)) **unfolding** prod-list-diag-prod  
**by simp**  
**also have** ... = ( $\prod$  i  $\in$  insert (k-1) ({0..<dim-row ?MD}-{k-1}). ?MD \$\$ (i,  
i))  
**using** set-rw **by simp**  
**also have** ... = ?MD \$\$ (k-1, k-1) \* ( $\prod$  i  $\in$  {0..<dim-row ?MD} - {k-1}.  
?MD \$\$ (i, i))  
**by** (metis (no-types, lifting) Diff-iff finite-atLeastLessThan finite-insert prod.insert  
set-rw singletonI)  
**also have** ... = B\$\$\$(1,1)  
**by** (smt (verit) 1 2 DiffD1 DiffD2 Groups.mult-ac(2) MD-rows add-diff-cancel-left'  
add-diff-inverse-nat  
k0 atLeastLessThan-iff class-cring.finprod-all1 insertI1 less-one more-arith-simps(5)  
  
plus-1-eq-Suc set-rw)  
**finally show** ?thesis .  
**qed**

**lemma** cofactor-make-mat-0k:  
**assumes** kn: k < n **and** k0: k  $\neq$  0 **and** n0: 1 < n  
**shows** cofactor (make-mat n k B) 0 k = - B \$\$ (1,0)  
**proof** -  
**let** ?M = make-mat n k B

**let**  $?MD = \text{mat-delete } ?M \ 0 \ k$   
**have**  $n0: 0 < n-1$  **using**  $n0$  **by** *auto*  
**have**  $MD\text{-carrier}: ?MD \in \text{carrier-mat } (n-1) \ (n-1)$   
**using** *make-mat-carrier mat-delete-carrier* **by** *blast*  
**have**  $MD\text{-k1}: ?MD \ \$\$ \ (k-1, 0) = B \ \$\$ \ (1,0)$   
**proof**  $-$   
**have**  $n0': 0 < n$  **using**  $n0$  **by** *auto*  
**have**  $insert\text{-i}: \text{insert-index } 0 \ (k-1) = k$  **using**  $k0$  **by** *auto*  
**have**  $insert\text{-k}: \text{insert-index } k \ 0 = 0$  **using**  $k0$  **by** *auto*  
**have**  $?MD \ \$\$ \ (k-1, 0) = ?M \ \$\$ \ (\text{insert-index } 0 \ (k-1), \text{insert-index } k \ 0)$   
**by** (*rule mat-delete-index[symmetric, OF - - - n0], insert k0 kn, auto*)  
**also have**  $\dots = ?M \ \$\$ \ (k, 0)$  **unfolding**  $insert\text{-i} \ insert\text{-k}$  **by** *simp*  
**also have**  $\dots = B \ \$\$ \ (1,0)$  **using**  $k0$  **unfolding** *make-mat-def index-mat[OF*  
 $kn \ n0']$  **by** *auto*  
**finally show**  $?thesis$  .  
**qed**  
**have**  $MD0: ?MD \ \$\$ \ (i, 0) = 0$  **if**  $i: i < n-1$  **and**  $ik: \text{Suc } i \neq k$  **for**  $i$   
**proof**  $-$   
**have**  $i2: \text{Suc } i < n$  **using**  $i$  **by** *auto*  
**have**  $n0': 0 < n$  **using**  $n0$  **by** *auto*  
**have**  $insert\text{-i}: \text{insert-index } 0 \ i = \text{Suc } i$  **by** *simp*  
**have**  $insert\text{-k}: \text{insert-index } k \ 0 = 0$  **using**  $k0$  **by** *auto*  
**have**  $?MD \ \$\$ \ (i, 0) = ?M \ \$\$ \ (\text{insert-index } 0 \ i, \text{insert-index } k \ 0)$   
**by** (*rule mat-delete-index[symmetric, OF - - - i], insert i n0 kn, auto*)  
**also have**  $\dots = ?M \ \$\$ \ (\text{Suc } i, 0)$  **unfolding**  $insert\text{-i} \ insert\text{-k}$  **by** *simp*  
**also have**  $\dots = 0$  **using**  $ik$  **unfolding** *make-mat-def index-mat[OF i2 n0']* **by**  
*auto*  
**finally show**  $?thesis$  .  
**qed**  
**have**  $det\text{-cofactor}: \text{Determinant.cofactor } ?MD \ (k-1) \ 0 = (-1) \wedge (k-1)$   
**unfolding** *cofactor-def* **using** *det-mat-delete-make-mat[OF kn]* **by** *auto*  
**have**  $sum0: (\sum i \in \{0..<n-1\} - \{k-1\}. ?MD \ \$\$ \ (i, 0) * \text{Determinant.cofactor}$   
 $?MD \ i \ 0) = 0$   
**by** (*rule sum.neutral, insert MD0, fastforce*)  
**have**  $\text{Determinant.det } ?MD = (\sum i < n-1. ?MD \ \$\$ \ (i, 0) * \text{Determinant.cofactor}$   
 $?MD \ i \ 0)$   
**by** (*rule laplace-expansion-column[OF MD-carrier n0]*)  
**also have**  $\dots = ?MD \ \$\$ \ (k-1, 0) * \text{Determinant.cofactor } ?MD \ (k-1) \ 0$   
 $+ (\sum i \in \{0..<n-1\} - \{k-1\}. ?MD \ \$\$ \ (i, 0) * \text{Determinant.cofactor } ?MD \ i$   
 $0)$   
**by** (*metis (no-types, lifting) Suc-less-eq add-diff-inverse-nat atLeast0LessThan*  
*finite-atLeastLessThan*  
 $k0 \ kn \ \text{lessThan-iff less-one } n0 \ \text{nat-diff-split-asm plus-1-eq-Suc rel-simps(70)$   
 $\text{sum.remove}$ )  
**also have**  $\dots = ?MD \ \$\$ \ (k-1, 0) * \text{Determinant.cofactor } ?MD \ (k-1) \ 0$  **un-**  
**folding**  $sum0$  **by** *simp*  
**also have**  $\dots = ?MD \ \$\$ \ (k-1, 0) * (-1) \wedge (k-1)$  **unfolding** *det-cofactor* **by**  
*auto*  
**also have**  $\dots = (-1) \wedge (k-1) * B \ \$\$ \ (1,0)$  **using**  $MD\text{-k1}$  **by** *auto*

**finally show** *?thesis unfolding cofactor-def*  
**by** (*metis (no-types, lifting) arithmetic-simps(49) k0 left-minus-one-mult-self*  
*more-arith-simps(11) mult-minus1 power-eq-if*)  
**qed**

**lemma** *invertible-make-mat:*  
**assumes** *inv-B: invertible-mat B and B: B ∈ carrier-mat 2 2*  
**and** *kn: k < n and k0: k ≠ 0*  
**shows** *invertible-mat (make-mat n k B)*  
**proof** –  
**let** *?M = (make-mat n k B)*  
**have** *M-carrier: ?M ∈ carrier-mat n n by auto*  
**show** *?thesis*  
**proof** (*cases n=0*)  
**case** *True*  
**thus** *?thesis using M-carrier using invertible-mat-zero by blast*  
**next**  
**case** *False note n-not-0 = False*  
**show** *?thesis*  
**proof** (*cases n=1*)  
**case** *True*  
**then show** *?thesis using M-carrier using invertible-mat-zero assms by auto*  
**next**  
**case** *False*  
**hence** *n: 0 < n using n-not-0 by auto*  
**hence** *n1: 1 < n using False n-not-0 by auto*  
**have** *M00: ?M \$\$ (0,0) = B \$\$ (0,0) by (simp add: make-mat-def n)*  
**have** *M0k: ?M \$\$ (0,k) = B \$\$ (0,1) by (simp add: k0 kn make-mat-def n)*  
**have** *sum0: (∑ j ∈ ({0..<n} - {0} - {k}). ?M \$\$ (0, j) \* Determinant.cofactor*  
*?M 0 j) = 0*  
**proof** (*rule sum.neutral, rule ballI*)  
**fix** *x assume x: x ∈ {0..<n} - {0} - {k}*  
**have** *make-mat n k B \$\$ (0,x) = 0 unfolding make-mat-def using x by*  
*auto*  
**thus** *local.make-mat n k B \$\$ (0, x) \* Determinant.cofactor (local.make-mat*  
*n k B) 0 x = 0*  
**by** *simp*  
**qed**  
**have** *cofactor-M-00: Determinant.cofactor ?M 0 0 = B \$\$ (1,1)*  
**by** (*rule cofactor-make-mat-00[OF kn k0]*)  
**have** *cofactor-M-0k: Determinant.cofactor ?M 0 k = - B \$\$ (1,0)*  
**by** (*rule cofactor-make-mat-0k[OF kn k0 n1]*)  
**have** *Determinant.det ?M = (∑ j < n. ?M \$\$ (0, j) \* Determinant.cofactor*  
*?M 0 j)*  
**using** *laplace-expansion-row[OF M-carrier n] by auto*  
**also have** *... = (∑ j ∈ {0..<n}. ?M \$\$ (0, j) \* Determinant.cofactor ?M 0 j)*  
**by** (*rule sum.cong, auto*)  
**also have** *... = ?M \$\$ (0, 0) \* Determinant.cofactor ?M 0 0*

$+ ?M \text{ \textasciitilde{}} (0, k) * \text{Determinant.cofactor } ?M \ 0 \ k$   
 $+ (\sum_{j \in (\{0..<n\} - \{0\} - \{k\})}. ?M \ \text{ \textasciitilde{}} (0, j) * \text{Determinant.cofactor } ?M \ 0 \ j)$

**j)** **by** (*metis (no-types, lifting) add-cancel-right-right kn k0 atLeast0LessThan atLeast1-lessThan-eq-remove0 finite-atLeastLessThan insert-Diff-single insert-iff lessThan-iff n sum.atLeast-Suc-lessThan sum.remove sum0*)  
**also have**  $\dots = ?M \ \text{ \textasciitilde{}} (0, 0) * \text{Determinant.cofactor } ?M \ 0 \ 0$   
 $+ ?M \ \text{ \textasciitilde{}} (0, k) * \text{Determinant.cofactor } ?M \ 0 \ k$  **using** *sum0* **by** *auto*  
**also have**  $\dots = ?M \ \text{ \textasciitilde{}} (0, 0) * B \ \text{ \textasciitilde{}} (1,1) - ?M \ \text{ \textasciitilde{}} (0, k) * B \ \text{ \textasciitilde{}} (1,0)$   
**unfolding** *cofactor-M-00 cofactor-M-0k* **by** *auto*  
**also have**  $\dots = B \ \text{ \textasciitilde{}} (0, 0) * B \ \text{ \textasciitilde{}} (1,1) - B \ \text{ \textasciitilde{}} (0, 1) * B \ \text{ \textasciitilde{}} (1,0)$   
**unfolding** *M00 M0k* **by** *auto*  
**also have**  $\dots = \text{Determinant.det } B$  **unfolding** *det-2[OF B]* **by** *auto*  
**finally have**  $\text{Determinant.det } ?M = \text{Determinant.det } B$  .  
**thus** *?thesis* **unfolding** *cofactor-def*  
**using** *invertible-iff-is-unit-JNF* **by** (*metis B M-carrier inv-B*)  
**qed**  
**qed**  
**qed**

**lemma** *make-mat-index:*

**assumes** *i: i < n and j: j < n*  
**shows**  $\text{make-mat } n \ k \ B \ \text{ \textasciitilde{}} (i,j) = (\text{if } i = 0 \wedge j = 0 \text{ then } B \ \text{ \textasciitilde{}} (0,0) \text{ else}$   
 $\text{if } i = 0 \wedge j = k \text{ then } B \ \text{ \textasciitilde{}} (0,1) \text{ else if } i=k \wedge j = 0$   
 $\text{then } B \ \text{ \textasciitilde{}} (1,0) \text{ else if } i=k \wedge j=k \text{ then } B \ \text{ \textasciitilde{}} (1,1)$   
 $\text{else if } i=j \text{ then } 1 \text{ else } 0)$   
**unfolding** *make-mat-def index-mat[OF i j]* **by** *simp*

**lemma** *make-mat-works:*

**assumes** *A: A ∈ carrier-mat m n and Suc-i-less-n: Suc i < n*  
**and** *Q-step-def: Q-step = (make-mat n (Suc i) (snd (Smith-1x2*  
 $(\text{Matrix.mat } 1 \ 2 \ (\lambda(a,b). \text{if } b = 0 \text{ then } A \ \text{ \textasciitilde{}} (0,0) \text{ else } A \ \text{ \textasciitilde{}} (0, \text{Suc } i))))))$   
**shows**  $A \ \text{ \textasciitilde{}} (0,0) * Q\text{-step } \text{ \textasciitilde{}} (0, (\text{Suc } i)) + A \ \text{ \textasciitilde{}} (0, \text{Suc } i) * Q\text{-step } \text{ \textasciitilde{}} (\text{Suc } i, \text{Suc } i) = 0$   
**proof** –  
**have** *n0: 0 < n* **using** *Suc-i-less-n* **by** *simp*  
**let** *?A = (Matrix.mat 1 2 (λ(a, b). if b = 0 then A \text{ \textasciitilde{}} (0, 0) else A \text{ \textasciitilde{}} (0, Suc i)))*  
**let** *?S = fst (Smith-1x2 ?A)*  
**let** *?Q = snd (Smith-1x2 ?A)*  
**have** *1: (make-mat n (Suc i) ?Q) \text{ \textasciitilde{}} (0, Suc i) = ?Q \text{ \textasciitilde{}} (0,1)*  
**unfolding** *make-mat-index[OF n0 Suc-i-less-n]* **by** *auto*  
**have** *2: (make-mat n (Suc i) ?Q) \text{ \textasciitilde{}} (Suc i, Suc i) = ?Q \text{ \textasciitilde{}} (1,1)*  
**unfolding** *make-mat-index[OF Suc-i-less-n Suc-i-less-n]* **by** *auto*  
**have** *is-SNF-A': is-SNF ?A (1<sub>m</sub> 1, Smith-1x2 ?A)* **using** *SNF-1x2-works* **by** *auto*  
**have** *SNF-S: Smith-normal-form-mat ?S* **and** *S: ?S = 1<sub>m</sub> 1 \* ?A \* ?Q*  
**and** *Q: ?Q ∈ carrier-mat 2 2*

**using** *is-SNF-A'* **unfolding** *is-SNF-def* **by** *auto*  
**have**  $?S \text{ \textasciitilde{}}(0,1) = (?A * ?Q) \text{ \textasciitilde{}}(0,1)$  **unfolding** *S* **by** *auto*  
**also have**  $\dots = \text{Matrix.row } ?A \ 0 \cdot \text{col } ?Q \ 1$  **by** (*rule index-mult-mat, insert Q, auto*)  
**also have**  $\dots = (\sum ia = 0..<\text{dim-vec } (\text{col } ?Q \ 1). \text{Matrix.row } ?A \ 0 \ \$v \ ia * \text{col } ?Q \ 1 \ \$v \ ia)$   
**unfolding** *scalar-prod-def* **by** *auto*  
**also have**  $\dots = (\sum ia \in \{0,1\}. \text{Matrix.row } ?A \ 0 \ \$v \ ia * \text{col } ?Q \ 1 \ \$v \ ia)$   
**by** (*rule sum.cong, insert Q, auto*)  
**also have**  $\dots = \text{Matrix.row } ?A \ 0 \ \$v \ 0 * \text{col } ?Q \ 1 \ \$v \ 0 + \text{Matrix.row } ?A \ 0 \ \$v \ 1 * \text{col } ?Q \ 1 \ \$v \ 1$   
**using** *sum-two-elements* **by** *auto*  
**also have**  $\dots = A \text{ \textasciitilde{}}(0,0) * ?Q \text{ \textasciitilde{}}(0,1) + A \text{ \textasciitilde{}}(0,\text{Suc } i) * ?Q \text{ \textasciitilde{}}(1,1)$   
**by** (*smt (verit) One-nat-def Q carrier-matD(1) carrier-matD(2) dim-col-mat(1) dim-row-mat(1) index-col index-mat(1) index-row(1) lessI numeral-2-eq-2 pos2 prod.simps(2) rel-simps(93)*)  
**finally have**  $?S \text{ \textasciitilde{}}(0,1) = A \text{ \textasciitilde{}}(0,0) * ?Q \text{ \textasciitilde{}}(0,1) + A \text{ \textasciitilde{}}(0,\text{Suc } i) * ?Q \text{ \textasciitilde{}}(1,1)$  **by** *simp*  
**moreover have**  $?S \text{ \textasciitilde{}}(0,1) = 0$  **using** *SNF-S* **unfolding** *Smith-normal-form-mat-def isDiagonal-mat-def*  
**by** (*metis (no-types, lifting) Q S card-num-simps(30) carrier-matD(2) index-mult-mat(2) index-mult-mat(3) index-one-mat(2) lessI n-not-Suc-n numeral-2-eq-2*)  
**ultimately show** *?thesis* **using** *1 2* **unfolding** *Q-step-def* **by** *auto*  
**qed**

### 16.3.1 Case $1 \times n$

**fun** *Smith-1xn-aux* ::  $\text{nat} \Rightarrow 'a \ \text{mat} \Rightarrow ('a \ \text{mat} \times 'a \ \text{mat}) \Rightarrow ('a \ \text{mat} \times 'a \ \text{mat})$   
**where**  
*Smith-1xn-aux*  $0 \ A \ (S, Q) = (S, Q) \mid$   
*Smith-1xn-aux*  $(\text{Suc } i) \ A \ (S, Q) = (\text{let}$   
 $A\text{-step-1x2} = (\text{Matrix.mat } 1 \ 2 \ (\lambda(a,b). \text{if } b = 0 \ \text{then } S \text{ \textasciitilde{}}(0,0) \ \text{else } S \text{ \textasciitilde{}}(0,\text{Suc } i)));$   
 $(S\text{-step-1x2}, Q\text{-step-1x2}) = \text{Smith-1x2 } A\text{-step-1x2};$   
 $Q\text{-step} = \text{make-mat } (\text{dim-col } A) \ (\text{Suc } i) \ Q\text{-step-1x2};$   
 $S' = S * Q\text{-step}$   
 $\text{in } \text{Smith-1xn-aux } i \ A \ (S', Q * Q\text{-step}))$

**definition** *Smith-1xn*  $A = (\text{if } \text{dim-col } A = 0 \ \text{then } (A, 1_m \ (\text{dim-col } A))$   
 $\text{else } \text{Smith-1xn-aux } (\text{dim-col } A - 1) \ A \ (A, 1_m \ (\text{dim-col } A)))$

**lemma** *Smith-1xn-aux-Q-carrier*:

**assumes**  $r: (S', Q') = (\text{Smith-1xn-aux } i \ A \ (S, Q))$   
**assumes**  $A: A \in \text{carrier-mat } 1 \ n$  **and**  $Q: Q \in \text{carrier-mat } n \ n$   
**shows**  $Q' \in \text{carrier-mat } n \ n$   
**using**  $A \ r \ Q$

**proof** (*induct i A (S, Q) arbitrary: S Q rule: Smith-1xn-aux.induct*)  
**case**  $(1 \ A \ S \ Q)$

```

then show ?case by auto
next
case (2 i A S Q)
note A = 2.prem1(1)
note S'Q' = 2.prem1(2)
note Q = 2.prem1(3)
let ?A-step-1x2 = (Matrix.mat 1 2 (λ(a,b). if b = 0 then S $$ (0,0) else S
$$ (0,Suc i)))
let ?S-step-1x2 = fst (Smith-1x2 ?A-step-1x2)
let ?Q-step-1x2 = snd (Smith-1x2 ?A-step-1x2)
let ?Q-step = make-mat (dim-col A) (Suc i) ?Q-step-1x2
have rw: A * (Q * ?Q-step) = A * Q * ?Q-step
by (smt (verit) A Q assoc-mult-mat carrier-matD(2) make-mat-carrier)
have Smith-rw: Smith-1xn-aux (Suc i) A (S, Q) = Smith-1xn-aux i A (S *
?Q-step, Q * ?Q-step)
by (auto, metis (no-types, lifting) old.prod.exhaust snd-conv split-conv)
show ?case
proof (rule 2.hyps[of ?A-step-1x2 (?S-step-1x2, ?Q-step-1x2) ?S-step-1x2 ?Q-step-1x2])
show S * ?Q-step = S * ?Q-step ..
show A ∈ carrier-mat 1 n using A by auto
show (S', Q') = Smith-1xn-aux i A (S * ?Q-step, Q * ?Q-step) using 2.prem1
Smith-rw by auto
show Q * ?Q-step ∈ carrier-mat n n using A Q by auto
qed (auto)
qed

```

**lemma** *Smith-1xn-aux-invertible-Q:*

```

assumes r: (S', Q') = (Smith-1xn-aux i A (S, Q))
assumes A: A ∈ carrier-mat 1 n and Q: Q ∈ carrier-mat n n
and i: i < n and inv-Q: invertible-mat Q
shows invertible-mat Q'
using r A Q inv-Q i
proof (induct i A (S, Q) arbitrary: S Q rule: Smith-1xn-aux.induct)
case (1 A S Q)
then show ?case by auto
next
case (2 i A S Q)
let ?A-step-1x2 = (Matrix.mat 1 2 (λ(a,b). if b = 0 then S $$ (0,0) else S
$$ (0,Suc i)))
let ?S-step-1x2 = fst (Smith-1x2 ?A-step-1x2)
let ?Q-step-1x2 = snd (Smith-1x2 ?A-step-1x2)
let ?Q-step = make-mat (dim-col A) (Suc i) ?Q-step-1x2
have Smith-rw: Smith-1xn-aux (Suc i) A (S, Q) = Smith-1xn-aux i A (S *
?Q-step, Q * ?Q-step)
by (auto, metis (no-types, lifting) old.prod.exhaust snd-conv split-conv)
have i-col: Suc i < dim-col A
using 2.prem1 Suc-lessD by blast
have i-n: i < n by (simp add: 2.prem1 Suc-lessD)

```

```

show ?case
proof (rule 2.hyps[of ?A-step-1x2 (?S-step-1x2, ?Q-step-1x2) ?S-step-1x2 ?Q-step-1x2])
  show  $A \in \text{carrier-mat } 1 \ n$  using 2.prem1 by auto
  show  $Q * ?Q\text{-step} \in \text{carrier-mat } n \ n$  using 2.prem2 by auto
  show  $S * ?Q\text{-step} = S * ?Q\text{-step} ..$ 
  show  $(S', Q') = \text{Smith-1xn-aux } i \ A \ (S * ?Q\text{-step}, Q * ?Q\text{-step})$  using 2.prem3
Smith-rw by auto
  show  $Q \in \text{carrier-mat } n \ n$  using 2.prem4 by auto
  proof (rule invertible-mult-JNF)
    show  $Q \in \text{carrier-mat } n \ n$  using 2.prem5 by auto
    show  $?Q\text{-step} \in \text{carrier-mat } n \ n$  using 2.prem6 by auto
    show  $Q \in \text{invertible-mat } n$  using 2.prem7 by auto
    show  $?Q\text{-step} \in \text{invertible-mat } n$  using 2.prem8 by auto
    by (rule invertible-make-mat[OF - - i-col], insert SNF-1x2-works, unfold
  is-SNF-def, auto)
      (metis (no-types, lifting) case-prodE mat-carrier snd-conv)+
  qed
qed (auto simp add: i-n)
qed

```

**lemma** *Smith-1xn-aux-S'-AQ'*:

```

assumes r:  $(S', Q') = (\text{Smith-1xn-aux } i \ A \ (S, Q))$ 
assumes A:  $A \in \text{carrier-mat } 1 \ n$  and S:  $S \in \text{carrier-mat } 1 \ n$  and Q:  $Q \in \text{carrier-mat } n \ n$ 
and S-AQ:  $S = A * Q$  and i:  $i < n$ 
shows  $S' = A * Q'$ 
using A S r Q S-AQ
proof (induct i A (S, Q) arbitrary: S Q rule: Smith-1xn-aux.induct)
  case (1 A S Q)
    then show ?case by auto
  next
    case (2 i A S Q)
      let ?A-step-1x2 = (Matrix.mat 1 2 ( $\lambda(a,b).$  if  $b = 0$  then  $S \ \$\$ (0,0)$  else  $S \ \$\$ (0, \text{Suc } i)$ ))
      let ?S-step-1x2 = fst (Smith-1x2 ?A-step-1x2)
      let ?Q-step-1x2 = snd (Smith-1x2 ?A-step-1x2)
      let ?Q-step = make-mat (dim-col A) (Suc i) ?Q-step-1x2
      have rw:  $A * (Q * ?Q\text{-step}) = A * Q * ?Q\text{-step}$ 
        by (smt (verit) 2.prem9 assoc-mult-mat carrier-matD(2) make-mat-carrier)
      have Smith-rw:  $\text{Smith-1xn-aux } (\text{Suc } i) \ A \ (S, Q) = \text{Smith-1xn-aux } i \ A \ (S * ?Q\text{-step}, Q * ?Q\text{-step})$ 
        by (auto, metis (no-types, lifting) old.prod.exhaust snd-conv split-conv)
      show ?case
proof (rule 2.hyps[of ?A-step-1x2 (?S-step-1x2, ?Q-step-1x2) ?S-step-1x2 ?Q-step-1x2])
  show  $A \in \text{carrier-mat } 1 \ n$  using 2.prem10 by auto
  show  $Q * ?Q\text{-step} \in \text{carrier-mat } n \ n$  using 2.prem11 by auto
  show  $S * ?Q\text{-step} = S * ?Q\text{-step} ..$ 
  show  $(S', Q') = \text{Smith-1xn-aux } i \ A \ (S * ?Q\text{-step}, Q * ?Q\text{-step})$  using 2.prem12
Smith-rw by auto

```

```

show  $S * ?Q\text{-step} = A * (Q * ?Q\text{-step})$  using 2.prem $s$  rw by auto
show  $S * ?Q\text{-step} \in \text{carrier-mat } 1\ n$ 
using 2.prem $s$  by (smt (verit) carrier-matD(2) make-mat-carrier mult-carrier-mat)
qed (auto)
qed

```

**lemma** *Smith-1xn-aux-S'-works:*

```

assumes  $r: (S', Q') = (\text{Smith-1xn-aux } i\ A\ (S, Q))$ 
assumes  $A: A \in \text{carrier-mat } 1\ n$  and  $S: S \in \text{carrier-mat } 1\ n$  and  $Q: Q \in$ 
carrier-mat  $n\ n$ 
and  $S\text{-}AQ: S = A * Q$  and  $i: i < n$  and  $j0: 0 < j$  and  $jn: j < n$ 
and all-j-zero:  $\forall j \in \{i+1..<n\}. S\ \$\$ (0, j) = 0$ 
shows  $S'\ \$\$ (0, j) = 0$ 
using  $A\ S\ r\ Q\ i\ S\text{-}AQ$  all-j-zero  $j0\ jn$ 
proof (induct  $i\ A\ (S, Q)$  arbitrary:  $S\ Q$  rule: Smith-1xn-aux.induct)
case (1  $A\ S\ Q$ )
then show ?case using  $j0\ jn$  by auto
next
case (2  $i\ A\ S\ Q$ )
let ? $A\text{-step-1x2} = (\text{Matrix.mat } 1\ 2\ (\lambda(a,b). \text{if } b = 0 \text{ then } S\ \$\$ (0,0) \text{ else } S\$ 
 $\ \$\$ (0, \text{Suc } i)))$ 
let ? $S\text{-step-1x2} = \text{fst } (\text{Smith-1x2 } ?A\text{-step-1x2})$ 
let ? $Q\text{-step-1x2} = \text{snd } (\text{Smith-1x2 } ?A\text{-step-1x2})$ 
let ? $Q\text{-step} = \text{make-mat } (\text{dim-col } A)\ (\text{Suc } i)\ ?Q\text{-step-1x2}$ 
have i-less-n:  $i < n$  by (simp add: 2(6) Suc-lessD)
have rw:  $A * (Q * ?Q\text{-step}) = A * Q * ?Q\text{-step}$ 
by (smt (verit) 2.prem $s$  assoc-mult-mat carrier-matD(2) make-mat-carrier)
have Smith-rw:  $\text{Smith-1xn-aux } (\text{Suc } i)\ A\ (S, Q) = \text{Smith-1xn-aux } i\ A\ (S *$ 
 $?Q\text{-step}, Q * ?Q\text{-step})$ 
by (auto, metis (no-types, lifting) old.prod.exhaust snd-conv split-conv)
have  $S'\text{-}AQ': S' = A * Q'$ 
by (rule Smith-1xn-aux-S'-AQ', insert 2.prem $s$ , auto)
show ?case
proof (rule 2.hyps[of ? $A\text{-step-1x2}$  (? $S\text{-step-1x2}$ , ? $Q\text{-step-1x2}$ ) ? $S\text{-step-1x2}$  ? $Q\text{-step-1x2}$ ])
show  $A \in \text{carrier-mat } 1\ n$  using 2.prem $s$  by auto
show  $Q\text{-}Q\text{-step-carrier}: Q * ?Q\text{-step} \in \text{carrier-mat } n\ n$  using 2.prem $s$  by auto

show  $S * ?Q\text{-step} = S * ?Q\text{-step} ..$ 
show  $(S', Q') = \text{Smith-1xn-aux } i\ A\ (S * ?Q\text{-step}, Q * ?Q\text{-step})$  using 2.prem $s$ 
Smith-rw by auto
show  $S * ?Q\text{-step} = A * (Q * ?Q\text{-step})$  using 2.prem $s$  rw by auto
show  $S * ?Q\text{-step} \in \text{carrier-mat } 1\ n$ 
using 2.prem $s$  by (smt (verit) carrier-matD(2) make-mat-carrier mult-carrier-mat)

show  $\forall j \in \{i + 1..<n\}. (S * ?Q\text{-step})\ \$\$ (0, j) = 0$ 
proof (rule ballI)
fix  $j$  assume  $j: j \in \{i + 1..<n\}$ 
have  $(S * ?Q\text{-step})\ \$\$ (0, j) = \text{Matrix.row } S\ 0 \cdot \text{col } ?Q\text{-step } j$ 

```



by (rule index-mult-mat, insert j 2.premis, auto simp add: make-mat-def)  
 also have ... = 0  
 proof (cases j=Suc i)  
   case True  
     let ?f =  $\lambda x. \text{Matrix.row } S \ 0 \ \$v \ x * \text{col } ?Q\text{-step } j \ \$v \ x$   
     let ?set =  $\{0..<\text{dim-vec } (\text{col } ?Q\text{-step } j)\}$   
     have set-rw: ?set = insert 0 (insert j (?set - {0} - {j}))  
       using 2.premis True make-mat-def by auto  
     have sum0:  $(\sum x \in ?set - \{0\} - \{j\}. ?f \ x) = 0$   
     proof (rule sum.neutral, rule ballI)  
       fix x assume x:  $x \in ?set - \{0\} - \{j\}$   
       show ?f x = 0 using 2(6) 2.premis True make-mat-def x by auto  
     qed  
     have Matrix.row S 0 · col ?Q-step j =  $(\sum x = 0..<\text{dim-vec } (\text{col } ?Q\text{-step } j). ?f \ x)$   
       unfolding scalar-prod-def by simp  
     also have ... =  $(\sum x \in \text{insert } 0 \ (\text{insert } j \ (?set - \{0\} - \{j\})). ?f \ x)$  using  
   set-rw by auto  
     also have ... = ?f 0 +  $(\sum x \in \text{insert } j \ (?set - \{0\} - \{j\}). ?f \ x)$  by (simp  
   add: True)  
     also have ... = ?f 0 + ?f j +  $(\sum x \in ?set - \{0\} - \{j\}. ?f \ x)$   
       by (simp add: set-rw sum.insert-remove)  
     also have ... = ?f 0 + ?f j using sum0 by auto  
     also have ... = S \$\$ (0,0) \* ?Q-step \$\$ (0, Suc i) + S \$\$ (0,Suc i) \*  
   ?Q-step \$\$ (Suc i, Suc i)  
       using 2.premis True make-mat-def by auto  
     also have ... = 0 by (rule make-mat-works, insert 2.premis, auto)  
     finally show ?thesis .  
   next  
     case False note j-not-Suc-i = False  
     show ?thesis  
       unfolding scalar-prod-def  
     proof (rule sum.neutral, rule ballI)  
       fix x assume x:  $x \in \{0..<\text{dim-vec } (\text{col } ?Q\text{-step } j)\}$   
       have xn:  $x < n$  using 2(2) make-mat-def x by auto  
       have jn2:  $j < \text{dim-col } A$  using 2(2) j by auto  
       have xn2:  $x < \text{dim-col } A$  using 2.premis(1) xn by blast  
       have Matrix.row S 0 \$v x = S \$\$ (0,x) using 2.premis make-mat-def x  
   by auto  
     moreover have col ?Q-step j \$v x = ?Q-step \$\$ (x,j) using Q-Q-step-carrier  
   j x by auto  
       ultimately have eq: Matrix.row S 0 \$v x \* col ?Q-step j \$v x = S \$\$  
   (0,x) \* ?Q-step \$\$ (x,j) by auto  
       have S-0x: S \$\$ (0,x) = 0 if Suc i + 1 ≤ x using 2.premis xn that by  
   auto  
     moreover have ?Q-step \$\$ (x,j) = 0 if x ≤ Suc i  
       using that j j-not-Suc-i unfolding make-mat-def index-mat[OF xn2 jn2]

```

by auto
  ultimately show Matrix.row S 0 $v x * (col ?Q-step j) $v x = 0 using
eq by force
  qed
  qed
  finally show (S * ?Q-step) $$ (0, j) = 0 .
  qed
qed (auto simp add: 2.prem1 i-less-n)
qed

lemma Smith-1xn-works:
  assumes A: A ∈ carrier-mat 1 n
  and SQ: (S,Q) = Smith-1xn A
shows is-SNF A (1m 1, S,Q)
proof (cases n=0)
  case True
  thus ?thesis using assms
  unfolding is-SNF-def
  by (auto simp add: Smith-1xn-def)
next
  case False
  hence n0: 0 < n by auto
  show ?thesis
  proof (rule is-SNF-intro)
    have SQ-eq: (S,Q) = local.Smith-1xn-aux (dim-col A - 1) A (A,1m (dim-col
A))
    using SQ unfolding Smith-1xn-def by simp
    have col: dim-col A - 1 < dim-col A using n0 A by auto
    show 1m 1 ∈ carrier-mat (dim-row A) (dim-row A) using A by auto
    show Q: Q ∈ carrier-mat (dim-col A) (dim-col A)
    by (rule Smith-1xn-aux-Q-carrier[OF SQ-eq], insert A, auto)
    show invertible-mat (1m 1) by simp
    show invertible-mat Q by (rule Smith-1xn-aux-invertible-Q[OF SQ-eq], insert
A n0, auto)
    have S-AQ: S = A * Q
    by (rule Smith-1xn-aux-S'-AQ'[OF SQ-eq], insert A n0, auto)
    thus S = 1m 1 * A * Q using A by auto
    have S: S ∈ carrier-mat 1 n using S-AQ A Q by auto
    show Smith-normal-form-mat S
    proof (rule Smith-normal-form-mat-intro)
      show ∀ a. a + 1 < min (dim-row S) (dim-col S) → S $$ (a, a) dvd S $$ (a
+ 1, a + 1)
      using S by auto
      have S $$ (0, j) = 0 if j0: 0 < j and jn: j < n for j
      by (rule Smith-1xn-aux-S'-works[OF SQ-eq], insert A n0 j0 jn, auto)
      thus isDiagonal-mat S unfolding isDiagonal-mat-def using S by simp
    qed
  qed
qed
qed

```

### 16.3.2 Case $n \times 1$

**definition** *Smith-nx1*  $A =$

(let  $(S,P) = (\text{Smith-1xn-aux } (\text{dim-row } A - 1) (\text{transpose-mat } A) (\text{transpose-mat } A, 1_m (\text{dim-row } A)))$   
in  $(\text{transpose-mat } P, \text{transpose-mat } S)$ )

**lemma** *Smith-nx1-works*:

**assumes**  $A: A \in \text{carrier-mat } n \ 1$

**and**  $SQ: (P,S) = \text{Smith-nx1 } A$

**shows** *is-SNF*  $A (P, S, 1_m \ 1)$

**proof** (cases  $n=0$ )

**case** *True*

**thus** *?thesis* **using** *assms*

**unfolding** *is-SNF-def*

**by** (*auto simp add: Smith-nx1-def*)

**next**

**case** *False*

**hence**  $n0: 0 < n$  **by** *auto*

**show** *?thesis*

**proof** (*rule is-SNF-intro*)

**have**  $SQ\text{-eq}: (S^T, P^T) = (\text{Smith-1xn-aux } (\text{dim-row } A - 1) A^T (A^T, 1_m (\text{dim-row } A)))$

**using**  $SQ[\text{unfolded } \text{Smith-nx1-def}]$  **unfolding** *Let-def split-beta* **by** *auto*

**have** *is-SNF*  $(A^T) (1_m \ 1, S^T, P^T)$

**by** (*rule Smith-1xn-works[unfolded Smith-1xn-def, OF - -], insert SQ-eq A, auto*)

**have**  $Pt: P^T \in \text{carrier-mat } (\text{dim-col } (A^T)) (\text{dim-col } (A^T))$

**by** (*rule Smith-1xn-aux-Q-carrier[OF SQ-eq], insert A n0, auto*)

**thus**  $P: P \in \text{carrier-mat } (\text{dim-row } A) (\text{dim-row } A)$  **by** *auto*

**show**  $1_m \ 1 \in \text{carrier-mat } (\text{dim-col } A) (\text{dim-col } A)$  **using**  $A$  **by** *simp*

**have** *invertible-mat*  $(P^T)$

**by** (*rule Smith-1xn-aux-invertible-Q[OF SQ-eq], insert A n0, auto*)

**thus** *invertible-mat*  $P$  **by** (*metis det-transpose P Pt invertible-iff-is-unit-JNF*)

**show** *invertible-mat*  $(1_m \ 1)$  **by** *simp*

**have**  $S^T = A^T * P^T$

**by** (*rule Smith-1xn-aux-S'-AQ'[OF SQ-eq], insert A n0, auto*)

**hence**  $S = P * A$  **by** (*metis A transpose-mult transpose-transpose P carrier-matD(1)*)

**thus**  $S = P * A * 1_m \ 1$  **using**  $P \ A$  **by** *auto*

**hence**  $S: S \in \text{carrier-mat } n \ 1$  **using**  $P \ A$  **by** *auto*

**have** *is-SNF*  $(A^T) (1_m \ 1, S^T, P^T)$

**by** (*rule Smith-1xn-works[unfolded Smith-1xn-def, OF - -], insert SQ-eq A, auto*)

**hence** *Smith-normal-form-mat*  $(S^T)$  **unfolding** *is-SNF-def* **by** *auto*

**thus** *Smith-normal-form-mat*  $S$  **unfolding** *Smith-normal-form-mat-def isDiagonal-mat-def* **by** *auto*

**qed**

**qed**

### 16.3.3 Case $2 \times n$

**function** *Smith-2xn* :: 'a mat  $\Rightarrow$  ('a mat  $\times$  'a mat  $\times$  'a mat)

**where**

*Smith-2xn* *A* = (  
 if *dim-col* *A* = 0 then ( $1_m$  (*dim-row* *A*), *A*,  $1_m$  0) else  
 if *dim-col* *A* = 1 then let (*P*, *S*) = *Smith-nx1* *A* in (*P*, *S*,  $1_m$  (*dim-col* *A*)) else  
 if *dim-col* *A* = 2 then *Smith-2x2* *A*  
 else  
 let *A1* = *mat-of-cols* (*dim-row* *A*) [*col* *A* 0];  
   *A2* = *mat-of-cols* (*dim-row* *A*) [*col* *A* *i*. *i*  $\leftarrow$  [1..*dim-col* *A*]];  
   (*P1*, *D1*, *Q1*) = *Smith-2xn* *A2*;  
   *C* = (*P1* \* *A1*) @<sub>c</sub> (*P1* \* *A2* \* *Q1*);  
   *D* = *mat-of-cols* (*dim-row* *A*) [*col* *C* 0, *col* *C* 1];  
   *E* = *mat-of-cols* (*dim-row* *A*) [*col* *C* *i*. *i*  $\leftarrow$  [2..*dim-col* *A*]];  
   (*P2*, *D2*, *Q2*) = *Smith-2x2* *D*;  
   *H* = (*P2* \* *D* \* *Q2*) @<sub>c</sub> (*P2* \* *E*);  
   *k* = (*div-op* (*H* \$\$ (0,2)) (*H* \$\$ (0,0)));  
   *H2* = *addcol* (-*k*) 2 0 *H*;  
   (-, -, -, *H2-DR*) = *split-block* *H2* 1 1;  
   (*H-1xn*, *Q3*) = *Smith-1xn* *H2-DR*;  
   *S* = *four-block-mat* (*Matrix.mat* 1 1 ( $\lambda(a,b). H $$ (0,0)$ )) ( $0_m$  1 (*dim-col* *A* - 1)) ( $0_m$  1 1) *H-1xn*;  
   *Q1'* = *four-block-mat* ( $1_m$  1) ( $0_m$  1 (*dim-col* *A* - 1)) ( $0_m$  (*dim-col* *A* - 1) 1) *Q1*;  
   *Q2'* = *four-block-mat* *Q2* ( $0_m$  2 (*dim-col* *A* - 2)) ( $0_m$  (*dim-col* *A* - 2) 2) ( $1_m$  (*dim-col* *A* - 2));  
   *Q-div-k* = *addrow-mat* (*dim-col* *A*) (-*k*) 0 2;  
   *Q3'* = *four-block-mat* ( $1_m$  1) ( $0_m$  1 (*dim-col* *A* - 1)) ( $0_m$  (*dim-col* *A* - 1) 1) *Q3*  
 in (*P2* \* *P1*, *S*, *Q1'* \* *Q2'* \* *Q-div-k* \* *Q3'*)  
 by *pat-completeness auto*

**termination apply** (*relation measure* ( $\lambda A. \text{dim-col } A$ )) **by** *auto*

**lemma** *Smith-2xn-0*:

**assumes** *A*: *A*  $\in$  *carrier-mat* 2 0

**shows** *is-SNF* *A* (*Smith-2xn* *A*)

**proof** -

**have** *Smith-2xn* *A* = ( $1_m$  (*dim-row* *A*), *A*,  $1_m$  0)

**using** *A* **by** *auto*

**moreover have** *is-SNF* *A* ... **by** (*rule is-SNF-intro*, *insert* *A*, *auto*)

**ultimately show** *?thesis* **by** *simp*

**qed**

**lemma** *Smith-2xn-1*:

**assumes** *A*: *A*  $\in$  *carrier-mat* 2 1

**shows** *is-SNF* *A* (*Smith-2xn* *A*)

**proof** -

**obtain** *P S* **where** *PS*: *Smith-nx1* *A* = (*P*, *S*) **using** *prod.exhaust* **by** *blast*

**have** \*: *is-SNF*  $A$   $(P, S, 1_m \ 1)$  **by** (rule *Smith-nx1-works*[*OF*  $A$  *PS*[*symmetric*]])  
**moreover have** *Smith-2xn*  $A = (P, S, 1_m \ (\dim\text{-col } A))$   
**using**  $A$  *PS* **by** *auto*  
**moreover have** *is-SNF*  $A$  ... **using** \*  $A$  **by** *auto*  
**ultimately show** ?*thesis* **by** *simp*  
**qed**

**lemma** *Smith-2xn-2*:  
**assumes**  $A: A \in \text{carrier-mat } 2 \ 2$   
**shows** *is-SNF*  $A$  (*Smith-2xn*  $A$ )  
**proof** –  
**have** *Smith-2xn*  $A = \text{Smith-2x2 } A$  **using**  $A$  **by** *auto*  
**from** *this* **show** ?*thesis* **using** *SNF-2x2-works* **using**  $A$  **by** *auto*  
**qed**

**lemma** *is-SNF-Smith-2xn-n-ge-2*:  
**assumes**  $A: A \in \text{carrier-mat } 2 \ n$  **and**  $n: n > 2$   
**shows** *is-SNF*  $A$  (*Smith-2xn*  $A$ )  
**using**  $A$  *n id*  
**proof** (*induct*  $A$  *arbitrary: n* rule: *Smith-2xn.induct*)  
**case**  $(1 \ A)$   
**note**  $A = 1.\text{prems}(1)$   
**note**  $n\text{-ge-2} = 1.\text{prems}(2)$   
**have** *dim-col-A-g2*: *dim-col*  $A > 2$  **using**  $n\text{-ge-2 } A$  **by** *auto*  
**define**  $A1$  **where**  $A1 = \text{mat-of-cols } (\text{dim-row } A) [\text{col } A \ 0]$   
**define**  $A2$  **where**  $A2 = \text{mat-of-cols } (\text{dim-row } A) [\text{col } A \ i. \ i \leftarrow [1..<\text{dim-col } A]]$   
**obtain**  $P1 \ D1 \ Q1$  **where**  $P1D1Q1: (P1, D1, Q1) = \text{Smith-2xn } A2$  **by** (*metis* *prod-cases3*)  
**define**  $C$  **where**  $C = (P1 * A1) @_c (P1 * A2 * Q1)$   
**define**  $D$  **where**  $D = \text{mat-of-cols } (\text{dim-row } A) [\text{col } C \ 0, \ \text{col } C \ 1]$   
**define**  $E$  **where**  $E = \text{mat-of-cols } (\text{dim-row } A) [\text{col } C \ i. \ i \leftarrow [2..<\text{dim-col } A]]$   
**obtain**  $P2 \ D2 \ Q2$  **where**  $P2D2Q2: (P2, D2, Q2) = \text{Smith-2x2 } D$  **by** (*metis* *prod-cases3*)  
**define**  $H$  **where**  $H = (P2 * D * Q2) @_c (P2 * E)$   
**define**  $k$  **where**  $k = \text{div-op } (H \ \$\$ \ (0, 2)) \ (H \ \$\$ \ (0, 0))$   
**define**  $H2$  **where**  $H2 = \text{addcol } (-k) \ 2 \ 0 \ H$   
**obtain**  $H2\text{-UL } H2\text{-UR } H2\text{-DL } H2\text{-DR}$   
**where** *split-H2*:  $(H2\text{-UL}, H2\text{-UR}, H2\text{-DL}, H2\text{-DR}) = (\text{split-block } H2 \ 1 \ 1)$  **by** (*metis* *prod-cases4*)  
**obtain**  $H\text{-1xn } Q3$  **where**  $H\text{-1xn-Q3}: (H\text{-1xn}, Q3) = \text{Smith-1xn } H2\text{-DR}$  **by** (*metis* *surj-pair*)  
**define**  $S$  **where**  $S = \text{four-block-mat } (\text{Matrix.mat } 1 \ 1 \ (\lambda(a, b). \ H \ \$\$ \ (0, 0))) \ (0_m \ 1 \ (\text{dim-col } A - 1)) \ (0_m \ 1 \ 1) \ H\text{-1xn}$   
**define**  $Q1'$  **where**  $Q1' = \text{four-block-mat } (1_m \ 1) \ (0_m \ 1 \ (\text{dim-col } A - 1)) \ (0_m \ (\text{dim-col } A - 1) \ 1) \ Q1$   
**define**  $Q2'$  **where**  $Q2' = \text{four-block-mat } Q2 \ (0_m \ 2 \ (\text{dim-col } A - 2)) \ (0_m \ (\text{dim-col } A - 2) \ 2) \ (1_m \ (\text{dim-col } A - 2))$   
**define**  $Q\text{-div-k}$  **where**  $Q\text{-div-k} = \text{addrow-mat } (\text{dim-col } A) \ (-k) \ 0 \ 2$   
**define**  $Q3'$  **where**  $Q3' = \text{four-block-mat } (1_m \ 1) \ (0_m \ 1 \ (\text{dim-col } A - 1)) \ (0_m$

```

(dim-col A - 1) 1) Q3
  have Smith-2xn-rw: Smith-2xn A = (P2 * P1, S, Q1' * Q2' * Q-div-k * Q3')
  proof (rule prod3-intro)
    have P1-def: fst (Smith-2xn A2) = P1 and Q1-def: snd (snd (Smith-2xn A2))
    = Q1
    and P2-def: fst (Smith-2x2 D) = P2 and Q2-def: snd (snd (Smith-2x2 D)) =
    Q2
    and H-1xn-def: fst (Smith-1xn H2-DR) = H-1xn and Q3-def: snd (Smith-1xn
    H2-DR) = Q3
    and H2-DR-def: snd (snd (snd (split-block H2 1 1))) = H2-DR
    using P2D2Q2 P1D1Q1 H-1xn-Q3 split-H2 fstI sndI by metis+
  note aux= P1-def Q1-def Q1'-def Q2'-def Q-div-k-def Q3'-def S-def A1-def[symmetric]
    C-def[symmetric] P2-def Q2-def Q3-def D-def[symmetric] E-def[symmetric]
    H-def[symmetric]
    k-def[symmetric] H2-def[symmetric] H2-DR-def H-1xn-def A2-def[symmetric]
  show fst (Smith-2xn A) = P2 * P1
    using dim-col-A-g2 unfolding Smith-2xn.simps[of A] Let-def split-beta
    by (insert P1D1Q1 P2D2Q2 D-def C-def, unfold aux, auto simp del: Smith-2xn.simps)
  show fst (snd (Smith-2xn A)) = S
    using dim-col-A-g2 unfolding Smith-2xn.simps[of A] Let-def split-beta
    by (insert P1D1Q1 P2D2Q2, unfold aux, auto simp del: Smith-2xn.simps)
  show snd (snd (Smith-2xn A)) = Q1' * Q2' * Q-div-k * Q3'
    using dim-col-A-g2 unfolding Smith-2xn.simps[of A] Let-def split-beta
    by (insert P1D1Q1 P2D2Q2, unfold aux, auto simp del: Smith-2xn.simps)
  qed
  show ?case
  proof (unfold Smith-2xn-rw, rule is-SNF-intro)
    have is-SNF-A2: is-SNF A2 (Smith-2xn A2)
    proof (cases 2 < dim-col A2)
      case True
      show ?thesis
      by (rule 1.hyps, insert True A dim-col-A-g2 id, auto simp add: A2-def)
    next
      case False
      hence dim-col A2 = 2 using n-ge-2 A unfolding A2-def by auto
      hence A2: A2 ∈ carrier-mat 2 2 unfolding A2-def using A by auto
      hence *: Smith-2xn A2 = Smith-2x2 A2 by auto
      show ?thesis unfolding * using SNF-2x2-works A2 by auto
    qed
  have A1[simp]: A1 ∈ carrier-mat (dim-row A) 1 unfolding A1-def by auto
  have A2[simp]: A2 ∈ carrier-mat (dim-row A) (dim-col A - 1) unfolding
  A2-def by auto
  have P1[simp]: P1 ∈ carrier-mat (dim-row A) (dim-row A)
  and inv-P1: invertible-mat P1
  and Q1: Q1 ∈ carrier-mat (dim-col A2) (dim-col A2) and inv-Q1: invert-
  ible-mat Q1
  and SNF-P1A2Q1: Smith-normal-form-mat (P1*A2*Q1)
  using is-SNF-A2 P1D1Q1 A2 unfolding is-SNF-def by fastforce+
  have D[simp]: D ∈ carrier-mat 2 2 unfolding D-def

```

**by** (*metis* 1(2) *One-nat-def Suc-eq-plus1 carrier-matD(1) list.size(3)*  
*list.size(4) mat-of-cols-carrier(1) numerals(2)*)  
**have** *is-SNF-D: is-SNF D (Smith-2x2 D) using SNF-2x2-works D by auto*  
**hence** *P2[simp]: P2 ∈ carrier-mat (dim-row A) (dim-row A) and inv-P2:*  
*invertible-mat P2*  
**and** *Q2[simp]: Q2 ∈ carrier-mat (dim-col D) (dim-col D) and inv-Q2:*  
*invertible-mat Q2*  
**using** *P2D2Q2 D-def unfolding is-SNF-def by force+*  
**show** *P2-P1: P2 \* P1 ∈ carrier-mat (dim-row A) (dim-row A) by (rule*  
*mult-carrier-mat[OF P2 P1])*  
**show** *invertible-mat (P2 \* P1) by (rule invertible-mult-JNF[OF P2 P1 inv-P2*  
*inv-P1])*  
**have** *Q1': Q1' ∈ carrier-mat (dim-col A) (dim-col A) using Q1 unfolding*  
*Q1'-def*  
**by** (*auto, smt (verit) A2 One-nat-def add-diff-inverse-nat carrier-matD(1)*  
*carrier-matD(2) carrier-matI*  
*dim-col-A-g2 gr-implies-not0 index-mat-four-block(2) index-mat-four-block(3)*  
  
*index-one-mat(2) index-one-mat(3) less-Suc0*)  
**have** *Q2': Q2' ∈ carrier-mat (dim-col A) (dim-col A) using Q2 unfolding*  
*Q2'-def*  
**by** (*smt (verit) D One-nat-def Suc-lessD add-diff-inverse-nat carrier-matD(1)*  
*carrier-matD(2)*  
*carrier-matI dim-col-A-g2 gr-implies-not0 index-mat-four-block(2) in-*  
*dex-mat-four-block(3)*  
*index-one-mat(2) index-one-mat(3) less-2-cases numeral-2-eq-2 semir-*  
*ing-norm(138)*)  
**have** *H2[simp]: H2 ∈ carrier-mat (dim-row A) (dim-col A) using A P2 D*  
**unfolding** *H2-def H-def*  
**by** (*smt (verit) E-def Q2 Q2' Q2'-def append-cols-def arithmetic-simps(50)*  
*carrier-matD(1) carrier-matD(2)*  
*carrier-mat-triv index-mat-addcol(4) index-mat-addcol(5) index-mat-four-block(2)*  
  
*index-mat-four-block(3) index-mult-mat(2) index-mult-mat(3) index-one-mat(2)*  
*index-zero-mat(2)*  
*index-zero-mat(3) length-map length-upt mat-of-cols-carrier(3)*)  
**have** *H'[simp]: H2-DR ∈ carrier-mat 1 (n - 1)*  
**by** (*rule split-block(4)[OF split-H2[symmetric]], insert H2 A n-ge-2, auto*)  
**have** *is-SNF-H': is-SNF H2-DR (1<sub>m</sub> 1, H-1xn, Q3)*  
**by** (*rule Smith-1xn-works[OF H' H-1xn-Q3]*)  
**from this have** *Q3: Q3 ∈ carrier-mat (dim-col H2-DR) (dim-col H2-DR) and*  
*inv-Q3: invertible-mat Q3*  
**unfolding** *is-SNF-def by auto*  
**have** *Q3': Q3' ∈ carrier-mat (dim-col A) (dim-col A)*  
**by** (*metis A A2 H' Q1 Q1' Q1'-def Q3 Q3'-def carrier-matD(1) carrier-matD(2)*  
*carrier-matI*  
*index-mat-four-block(2) index-mat-four-block(3)*)  
**have** *Q-div-k[simp]: Q-div-k ∈ carrier-mat (dim-col A) (dim-col A) unfolding*  
*Q-div-k-def by auto*

```

have inv-Q-div-k: invertible-mat Q-div-k
by (metis Q-div-k Q-div-k-def det-addrw-mat det-one invertible-iff-is-unit-JNF

      invertible-mat-one nat.simps(3) numerals(2) one-carrier-mat)
show Q1' * Q2' * Q-div-k * Q3' ∈ carrier-mat (dim-col A) (dim-col A)
  using Q1' Q2' Q-div-k Q3' by auto
have inv-Q1': invertible-mat Q1'
proof -
  have invertible-mat (four-block-mat (1m 1) (0m 1 (n - 1)) (0m (n - 1) 1)
Q1)
    by (rule invertible-mat-four-block-mat-lower-right, insert Q1 inv-Q1 A2
1.premis, auto)
    thus ?thesis unfolding Q1'-def using A by auto
  qed
have inv-Q2': invertible-mat Q2'
  by (unfold Q2'-def, rule invertible-mat-four-block-mat-lower-right-id,
insert Q2 n-ge-2 inv-Q2 A D, auto)
have inv-Q3': invertible-mat Q3'
proof -
  have invertible-mat (four-block-mat (1m 1) (0m 1 (n - 1)) (0m (n - 1) 1)
Q3)
    by (rule invertible-mat-four-block-mat-lower-right, insert Q3 H' inv-Q3
1.premis, auto)
    thus ?thesis unfolding Q3'-def using A by auto
  qed
show invertible-mat (Q1' * Q2' * Q-div-k * Q3')
  using inv-Q1' inv-Q2' inv-Q-div-k inv-Q3'
  by (meson Q1' Q2' Q3' Q-div-k invertible-mult-JNF mult-carrier-mat)
have A-A1-A2: A = A1 @c A2 unfolding A1-def A2-def append-cols-def
proof (rule eq-matI, auto)
  fix i assume i: i < dim-row A show 1: A $$ (i, 0) = mat-of-cols (dim-row
A) [col A 0] $$ (i, 0)
  by (metis dim-col-A-g2 gr-zeroI i index-col mat-of-cols-Cons-index-0 not-less0)
  let ?xs = (map (col A) [Suc 0..fix j
  assume j1: j < Suc (dim-col A - Suc 0)
  and j2: 0 < j
  have mat-of-cols (dim-row A) ?xs $$ (i, j - Suc 0) = ?xs ! (j - Suc 0) $v i
  by (rule mat-of-cols-index, insert j1 j2 i, auto)
  also have ... = A $$ (i,j) using dim-col-A-g2 i j1 j2 by auto
  finally show A $$ (i, j) = mat-of-cols (dim-row A) ?xs $$ (i, j - Suc 0) ..

  next
  show dim-col A = Suc (dim-col A - Suc 0) using n-ge-2 A by auto
qed
have C-P1-A-Q1': C = P1 * A * Q1'
proof -
  have aux: P1 * (A1 @c A2) = ((P1 * A1) @c (P1 * A2))
  by (rule append-cols-mult-left, insert A1 A2 P1, auto)

```



**have**  $P1 * A * Q1' = P1 * (A1 @_c A2) * Q1'$  **using**  $A-A1-A2$  **by**  $simp$   
**also have**  $\dots = ((P1 * A1) @_c (P1 * A2)) * Q1'$  **unfolding**  $aux \dots$   
**also have**  $\dots = (P1 * A1) @_c ((P1 * A2) * Q1)$   
**by**  $(rule\ append\ cols\ mult\ right\ id, insert\ P1\ A1\ A2\ Q1'\text{-def}\ Q1, auto)$   
**finally show**  $?thesis$  **unfolding**  $C\text{-def}$  **by**  $auto$   
**qed**  
**have**  $E\ ij\ 0: E\ \$\$ (i,j) = 0$  **if**  $i: i < dim\ row\ E$  **and**  $j: j < dim\ col\ E$  **and**  $ij: (i,j) \neq (1,0)$   
**for**  $i\ j$   
**proof**  $-$   
**let**  $?ws = (map\ (col\ C)\ [2..<dim\ col\ A])$   
**have**  $E\ \$\$ (i,j) = ?ws\ !\ j\ \$v\ i$   
**by**  $(unfold\ E\text{-def}, rule\ mat\ of\ cols\ index, insert\ i\ j\ A\ E\text{-def}, auto)$   
**also have**  $\dots = (col\ C\ (j+2))\ \$v\ i$  **using**  $E\text{-def}\ j$  **by**  $auto$   
**also have**  $\dots = C\ \$\$ (i,j+2)$   
**by**  $(metis\ C\text{-}P1\text{-}A\text{-}Q1'\ P1\ Q1'\ E\text{-def}\ carrier\ matD(1)\ carrier\ matD(2)\ index\ col\ index\ mult\ mat(2)\ index\ mult\ mat(3)\ length\ map\ length\ upt\ less\ diff\ conv\ mat\ of\ cols\ carrier(2)\ mat\ of\ cols\ carrier(3)\ i\ j)$   
**also have**  $\dots = (if\ j + 2 < dim\ col\ (P1 * A1)\ then\ (P1 * A1)\ \$\$ (i, j + 2)\ else\ (P1 * A2 * Q1)\ \$\$ (i, (j+2) - 1))$   
**unfolding**  $C\text{-def}$   
**by**  $(rule\ append\ cols\ nth, insert\ i\ j\ P1\ A1\ A2\ Q1\ A, auto\ simp\ add: E\text{-def})$   
  
**also have**  $\dots = (P1 * A2 * Q1)\ \$\$ (i, j+1)$   
**by**  $(metis\ A1\ One\ nat\ def\ add.\ assoc\ add\ diff\ cancel\ right'\ add\ is\ 0\ arith\ special(3)\ carrier\ matD(2)\ index\ mult\ mat(3)\ less\ Suc0\ zero\ neq\ numeral)$   
**also have**  $\dots = 0$  **using**  $SNF\text{-}P1A2Q1$  **unfolding**  $Smith\ normal\ form\ mat\ def\ isDiagonal\ mat\ def$   
**by**  $(metis\ (no\ types, lifting)\ A\ A2\ P1\ Q1\ Suc\ diff\ Suc\ Suc\ mono\ E\text{-def}\ add\ Suc\ right\ add\ lessD1\ arith\ extra\_simps(6)\ carrier\ matD(1)\ carrier\ matD(2)\ dim\ col\ A\ g2\ gr\ implies\ not0\ index\ mult\ mat(2)\ index\ mult\ mat(3)\ length\ map\ length\ upt\ less\ Suc\ eq\ mat\ of\ cols\ carrier(2)\ mat\ of\ cols\ carrier(3)\ numeral\ 2\ eq\ 2\ plus\ 1\ eq\ Suc\ i\ j\ ij)$   
**finally show**  $?thesis$   $.$   
**qed**  
**have**  $C\text{-}D\text{-}E: C = D @_c E$   
**proof**  $(rule\ eq\ matI)$   
**have**  $C\ \$\$ (i, j) = mat\ of\ cols\ (dim\ row\ A)\ [col\ C\ 0, col\ C\ 1]\ \$\$ (i, j)$   
**if**  $i: i < dim\ row\ A$  **and**  $j: j < 2$  **for**  $i\ j$   
**proof**  $-$   
**let**  $?ws = [col\ C\ 0, col\ C\ 1]$   
**have**  $mat\ of\ cols\ (dim\ row\ A)\ [col\ C\ 0, col\ C\ 1]\ \$\$ (i, j) = ?ws\ !\ j\ \$v\ i$   
**by**  $(rule\ mat\ of\ cols\ index, insert\ i\ j, auto)$   
**also have**  $\dots = C\ \$\$ (i, j)$  **using**  $j\ index\ col$

**by** (*auto*, *smt* (*verit*) *A C-P1-A-Q1' P1 Q1' Suc-lessD carrier-matD i*  
*index-col index-mult-mat(2,3)*  
*less-2-cases n-ge-2 nth-Cons-0 nth-Cons-Suc numeral-2-eq-2*)  
**finally show** *?thesis by simp*  
**qed**  
**moreover have**  $C \text{ \textcircled{=} } (i, j) = \text{mat-of-cols } (\text{dim-row } A) (\text{map } (\text{col } C) [2..<\text{dim-col}$   
*A]) \text{ \textcircled{=} } (i, j - 2)  
**if**  $i < \text{dim-row } A$  **and**  $j1: j < \text{dim-col } A$  **and**  $j2: j \geq 2$  **for**  $i j$   
**proof** –  
**let**  $?ws = (\text{map } (\text{col } C) [2..<\text{dim-col } A])$   
**have**  $\text{mat-of-cols } (\text{dim-row } A) ?ws \text{ \textcircled{=} } (i, j - 2) = ?ws ! (j - 2) \text{ \textcircled{=} } v i$   
**by** (*rule mat-of-cols-index, insert i j1 j2, auto*)  
**also have**  $\dots = C \text{ \textcircled{=} } (i, j)$   
**by** (*metis C-P1-A-Q1' P1 Q1' add-diff-inverse-nat carrier-matD(1) carrier-matD(2) dim-col-A-g2*  
*i index-col index-mult-mat(2) index-mult-mat(3) less-diff-iff less-imp-le-nat*  
  
*linorder-not-less nth-map-upt j1 j2*)  
**finally show** *?thesis by auto*  
**qed**  
**ultimately show**  $\bigwedge i j. i < \text{dim-row } (D \text{ \textcircled{=} }_c E) \implies j < \text{dim-col } (D \text{ \textcircled{=} }_c E) \implies$   
 $C \text{ \textcircled{=} } (i, j) = (D \text{ \textcircled{=} }_c E) \text{ \textcircled{=} } (i, j)$   
**unfolding** *D-def E-def append-cols-def* **by** (*auto simp add: numerals*)  
**show**  $\text{dim-row } C = \text{dim-row } (D \text{ \textcircled{=} }_c E)$  **using** *P1 A* **unfolding** *C-def D-def*  
*E-def append-cols-def* **by** *auto*  
**show**  $\text{dim-col } C = \text{dim-col } (D \text{ \textcircled{=} }_c E)$  **using** *A1 Q1 A2 A n-ge-2*  
**unfolding** *C-def D-def E-def append-cols-def* **by** *auto*  
**qed**  
**have**  $E[\text{simp}]: E \in \text{carrier-mat } 2 (n - 2)$  **unfolding** *E-def* **using** *A* **by** *auto*  
**have**  $H[\text{simp}]: H \in \text{carrier-mat } (\text{dim-row } A) (\text{dim-col } A)$  **unfolding** *H-def*  
*append-cols-def* **using** *A*  
**by** (*smt* (*verit*) *E Groups.add-ac(1) One-nat-def P2-P1 Q2 Q2' Q2'-def carrier-matD*  
*index-mat-four-block*  
*plus-1-eq-Suc index-mult-mat index-one-mat index-zero-mat numeral-2-eq-2*  
*carrier-matI*)  
**have**  $H - P2 - P1 - A - Q1' - Q2': H = P2 * P1 * A * Q1' * Q2'$   
**proof** –  
**have**  $\text{aux}: (P2 * D \text{ \textcircled{=} }_c P2 * E) = P2 * (D \text{ \textcircled{=} }_c E)$   
**by** (*rule append-cols-mult-left[symmetric], insert D E P2 A, auto simp add:*  
*D-def E-def*)  
**have**  $H = P2 * D * Q2 \text{ \textcircled{=} }_c P2 * E$  **using** *H-def* **by** *auto*  
**also have**  $\dots = (P2 * D \text{ \textcircled{=} }_c P2 * E) * Q2'$  **by** (*rule append-cols-mult-right-id2[symmetric],*  
*insert Q2 D Q2'-def, auto simp add: D-def E-def*)  
**also have**  $\dots = (P2 * (D \text{ \textcircled{=} }_c E)) * Q2'$  **using** *aux* **by** *auto*  
**also have**  $\dots = P2 * C * Q2'$  **unfolding** *C-D-E* **by** *auto*  
**also have**  $\dots = P2 * P1 * A * Q1' * Q2'$  **unfolding** *C-P1-A-Q1'*  
**by** (*smt* (*verit*, *ccfv-threshold*) *P1 P2 Q1' assoc-mult-mat carrier-mat-triv*  
*mult-carrier-mat*)  
**finally show** *?thesis* .*

**qed**  
**have**  $H2-H-Q-div-k$ :  $H2 = H * Q-div-k$  **unfolding**  $H2-def$   $Q-div-k-def$   
**by** (*metis*  $H-P2-P1-A-Q1'-Q2'$   $Q2'$  *addcol-mat carrier-matD*(2) *dim-col-A-g2*  
*gr-implies-not0*  
*mat-carrier times-mat-def zero-order*(5))  
**hence**  $H2-P2-P1-A-Q1'-Q2'-Q-div-k$ :  $H2 = P2 * P1 * A * Q1' * Q2' * Q-div-k$   
**unfolding**  $H-P2-P1-A-Q1'-Q2'$  **by** *simp*  
**have**  $H2-as-four-block-mat$ :  $H2 = four-block-mat$   $H2-UL$   $H2-UR$   $H2-DL$   $H2-DR$   
  
**by** (*rule split-block*[*OF split-H2*[*symmetric*], *of - n-1*], *insert H2 A n-ge-2*,  
*auto*)  
**have**  $H2-UL$ :  $H2-UL \in carrier-mat$  1 1  
**by** (*rule split-block*[*OF split-H2*[*symmetric*], *of - n-1*], *insert H2 A n-ge-2*,  
*auto*)  
**have**  $H2-UR$ :  $H2-UR \in carrier-mat$  1 (*dim-col A - 1*)  
**by** (*rule split-block*(2)[*OF split-H2*[*symmetric*]], *insert H2 A n-ge-2*, *auto*)  
**have**  $H2-DL$ :  $H2-DL \in carrier-mat$  1 1  
**by** (*rule split-block*[*OF split-H2*[*symmetric*], *of - n-1*], *insert H2 A n-ge-2*,  
*auto*)  
**have**  $H2-DR$ :  $H2-DR \in carrier-mat$  1 (*dim-col A - 1*)  
**by** (*rule split-block*[*OF split-H2*[*symmetric*]], *insert H2 A n-ge-2*, *auto*)  
**have**  $H2-UR-00$ :  $H2-UR$  \$\$ (0,0) = 0  
**proof** -  
**have**  $H2-UR$  \$\$ (0,0) =  $H2$  \$\$ (0,1)  
**by** (*smt* (*verit*)  $A$   $H2-H-Q-div-k$   $H2-UL$   $H2-as-four-block-mat$   $H2-def$   $H-P2-P1-A-Q1'-Q2'$ )  
  
*Num.numeral-nat*(7)  $P2-P1$   $Q2'$  *add-diff-cancel-left'* *carrier-matD* *dim-col-A-g2*  
*index-mat-addcol*  
*index-mat-four-block index-mult-mat less-trans-Suc plus-1-eq-Suc pos2*  
*semiring-norm*(138)  
*zero-less-one-class.zero-less-one*)  
**also have** ... =  $H$  \$\$ (0,1)  
**unfolding**  $H2-def$  **by** (*rule index-mat-addcol*, *insert H A n-ge-2*, *auto*)  
**also have** ... = ( $P2 * D * Q2$ ) \$\$ (0,1)  
**by** (*smt* (*verit*)  $C-D-E$   $C-P1-A-Q1'$   $D$   $H2-H-Q-div-k$   $H2-UL$   $H2-as-four-block-mat$   
 $H-P2-P1-A-Q1'-Q2'$   $H-def$   $Q1'$   
 $Q2$  *add-lessD1* *append-cols-def* *carrier-matD*(1) *carrier-matD*(2) *dim-col-A-g2*  
  
*index-mat-four-block index-mult-mat*(2) *index-mult-mat*(3) *lessI numer-*  
*als*(2) *plus-1-eq-Suc zero-less-Suc*)  
**also have** ... = 0 **using** *is-SNF-D*  $P2D2Q2$   $D$   
**unfolding** *is-SNF-def* *Smith-normal-form-mat-def* *isDiagonal-mat-def* **by**  
*auto*  
**finally show**  $H2-UR$  \$\$ (0,0) = 0 .  
**qed**  
**have**  $H2-UR-0j$ :  $H2-UR$  \$\$ (0,j) = 0 **if**  $j-ge-1$ :  $j > 1$  **and**  $j$ :  $j < n-1$  **for**  $j$   
**proof** -  
**have**  $col-E-0$ :  $col E$  ( $j - 1$ ) =  $0_v$  2  
**by** (*rule eq-vecI*, *unfold col-def*, *insert E E-ij-0 j j-ge-1 n-ge-2*, *auto*)

(metis E Suc-diff-Suc Suc-lessD Suc-less-eq Suc-pred carrier-matD index-vec  
 numerals(2), insert E, blast)

**have** H2-UR \$\$ (0,j) = H2 \$\$ (0,j+1)

**by** (metis (no-types, lifting) A H2-P2-P1-A-Q1'-Q2'-Q-div-k H2-UL H2-as-four-block-mat  
 H2-def

H-P2-P1-A-Q1'-Q2' P2-P1 Q2' add-diff-cancel-right' carrier-matD in-  
 dex-mat-addcol(5)

index-mat-four-block index-mult-mat(2,3) less-diff-conv less-numeral-extra(1)

not-add-less2 pos2 j)

**also have** ... = H \$\$ (0,j+1) **unfolding** H2-def

**by** (metis A H2-P2-P1-A-Q1'-Q2'-Q-div-k H2-def H-P2-P1-A-Q1'-Q2' One-nat-def  
 P2-P1 Q-div-k-def

add-right-cancel carrier-matD(1) carrier-matD(2) index-mat-addcol(3)

index-mat-addcol(5)

index-mat-addrow-mat(3) index-mult-mat(2) index-mult-mat(3) less-diff-conv

less-not-refl2

numerals(2) plus-1-eq-Suc pos2 j j-ge-1)

**also have** ... = (if j+1 < dim-col (P2 \* D \* Q2)

then (P2 \* D \* Q2) \$\$ (0, j+1) else (P2\*E) \$\$ (0, (j+1) - 2))

**by** (unfold H-def, rule append-cols-nth, insert E P2 A Q2 D j, auto simp add:  
 E-def)

**also have** ... = (P2\*E) \$\$ (0, j - 1)

**by** (metis (no-types, lifting) D One-nat-def Q2 add-Suc-right add-lessD1  
 arithmetic-simps(50)

carrier-matD(2) diff-Suc-Suc index-mult-mat(3) not-less-eq numeral-2-eq-2  
 j-ge-1)

**also have** ... = Matrix.row P2 0 · col E (j - 1)

**by** (rule index-mult-mat, insert P2 j-ge-1 A j, auto simp add: E-def)

**also have** ... = 0 **unfolding** col-E-0 **by** (simp add: scalar-prod-def)

**finally show** ?thesis .

**qed**

**have** H00-dvd-D01: H\$\$ (0,0) dvd D\$\$ (0,1)

**proof** -

**have** H\$\$ (0,0) = (P2\*D\*Q2) \$\$ (0,0) **unfolding** H-def **using** append-cols-nth  
 D E

**by** (smt (verit, ccfv-SIG) A C-D-E C-P1-A-Q1' D H2-DR H2-H-Q-div-k  
 H2-UL H2-as-four-block-mat H-P2-P1-A-Q1'-Q2'  
 One-nat-def P1 Q1' Q2 Suc-lessD append-cols-def carrier-matD dim-col-A-g2

index-mat-four-block index-mult-mat numerals(2) plus-1-eq-Suc zero-less-Suc)

**also have** ... dvd D\$\$ (0,1) **by** (rule S00-dvd-all-A[OF D - - inv-P2 inv-Q2],  
 insert is-SNF-D P2D2Q2 P2 Q2 D, unfold is-SNF-def, auto)

**finally show** ?thesis .

**qed**

**have** D01-dvd-H02: D\$\$ (0,1) dvd H\$\$ (0,2) **and** D01-dvd-H12: D\$\$ (0,1) dvd  
 H\$\$ (1,2)

**proof** -

**have** D\$\$ (0,1) = C\$\$ (0,1) **unfolding** C-D-E

**by** (smt (verit) A C-D-E C-P1-A-Q1' D One-nat-def P1 Q1' append-cols-def)

*carrier-matD(1) carrier-matD(2)*  
*dim-col-A-g2 index-mat-four-block(1) index-mat-four-block(2) index-mat-four-block(3)*  
  
*index-mult-mat(2) index-mult-mat(3) lessI less-trans-Suc numerals(2)*  
*pos2)*  
**also have** ... =  $(P1 * A2 * Q1)$  \$\$ (0,0) **using** *C-def*  
**by** (*smt (verit) 1(2) A1 A-A1-A2 P1 Q1 add-diff-cancel-left' append-cols-def*  
*card-num-simps(30)*  
*carrier-matD dim-col-A-g2 index-mat-four-block index-mult-mat less-numeral-extra(4)*  
  
*less-trans-Suc plus-1-eq-Suc pos2)*  
**also have** ... *dvd*  $(P1 * A2 * Q1)$  \$\$ (1,1)  
**by** (*smt (verit) 1(2) A2 One-nat-def P1 Q1 S00-dvd-all-A SNF-P1A2Q1*  
*carrier-matD(1) carrier-matD(2) dim-col-A-g2*  
*dvd-elements-mult-matrix-left-right inv-P1 inv-Q1 lessI less-diff-conv*  
*numeral-2-eq-2 plus-1-eq-Suc)*  
**also have** ... = *C* \$\$ (1,2) **unfolding** *C-def*  
**by** (*smt (verit) 1(2) A1 A-A1-A2 One-nat-def P1 Q1 append-cols-def*  
*carrier-matD(1) carrier-matD(2) diff-Suc-1*  
*dim-col-A-g2 index-mat-four-block index-mult-mat lessI not-numeral-less-one*  
*numeral-2-eq-2)*  
**also have** ... = *E* \$\$ (1,0) **unfolding** *C-D-E*  
**by** (*smt (verit) 1(3) A C-D-E C-P1-A-Q1' D One-nat-def append-cols-def*  
*carrier-matD less-irrefl-nat*  
*P1 Q1' diff-Suc-1 diff-Suc-Suc index-mat-four-block index-mult-mat lessI*  
*numerals(2))*  
**finally have** \*:  $D$  \$\$ (0,1) *dvd*  $E$  \$\$ (1,0) **by** *auto*  
**also have** ... *dvd*  $(P2 * E)$  \$\$ (0,0)  
**by** (*smt (verit) 1(3) A E E-ij-0 P2 carrier-matD(1) carrier-matD(2)*  
*dvd-0-right*  
*dvd-elements-mult-matrix-left dvd-refl pos2 zero-less-diff)*  
**also have** ... =  $H$  \$\$ (0,2) **unfolding** *H-def*  
**by** (*smt (verit) 1(3) A C-D-E C-P1-A-Q1' D Groups.add-ac(1) H2-DR*  
*H2-H-Q-div-k H2-UL H2-as-four-block-mat*  
*H-P2-P1-A-Q1'-Q2' One-nat-def P1 Q1' Q2 add-diff-cancel-left' ap-*  
*pend-cols-def carrier-matD*  
*index-mat-four-block index-mult-mat less-irrefl-nat numerals(2) plus-1-eq-Suc*  
*pos2)*  
**finally show**  $D$  \$\$ (0, 1) *dvd*  $H$  \$\$ (0, 2) .  
**have**  $E$  \$\$ (1,0) *dvd*  $(P2 * E)$  \$\$ (1,0)  
**by** (*smt (verit) 1(3) A E E-ij-0 P2 carrier-matD(1) carrier-matD(2)*  
*dvd-0-right*  
*dvd-elements-mult-matrix-left dvd-refl rel-simps(49) semiring-norm(76)*  
*zero-less-diff)*  
**also have** ... =  $H$  \$\$ (1,2) **unfolding** *H-def*  
**by** (*smt (verit) A C-D-E C-P1-A-Q1' D H2-DR H2-H-Q-div-k H2-UL*  
*H2-as-four-block-mat H-P2-P1-A-Q1'-Q2'*  
*One-nat-def P1 Q1' Q2 add-diff-cancel-left' append-cols-def carrier-matD*  
*diff-Suc-eq-diff-pred*

*index-mat-four-block index-mult-mat lessI less-irrefl-nat n-ge-2 numerals(2)*  
*plus-1-eq-Suc)*  
**finally show**  $D\$(0,1) \text{ dvd } H\$(1,2)$  **using** \* **by** *auto*  
**qed**  
**have**  $kH00\text{-eq-}H02: k * H\$(0,0) = H\$(0,2)$   
**using** *id D01-dvd-H02 H00-dvd-D01* **unfolding** *k-def is-div-op-def* **by** *auto*  
**have**  $H2\text{-UR-}01: H2\text{-UR}\$(0,1) = 0$   
**proof** –  
**have**  $H2\text{-UR}\$(0,1) = H2\$(0,2)$   
**by** (*metis (no-types, lifting) A H2-P2-P1-A-Q1'-Q2'-Q-div-k H2-UL H2-as-four-block-mat*  
*One-nat-def*  
*P2-P1 Q-div-k-def carrier-matD diff-Suc-1 dim-col-A-g2 index-mat-addrow-mat(3)*)  
  
*index-mat-four-block index-mult-mat(2,3) numeral-2-eq-2 pos2 rel-simps(50)*  
*rel-simps(68))*  
**also have**  $\dots = (-k) * H\$(0,0) + H\$(0,2)$   
**by** (*unfold H2-def, rule index-mat-addcol[of - ], insert H A n-ge-2, auto*)  
**also have**  $\dots = 0$  **using**  $kH00\text{-eq-}H02$  **by** *auto*  
**finally show** *?thesis* .  
**qed**  
**have**  $H2\text{-UR-}0: H2\text{-UR} = (0_m \ 1 \ (n - 1))$   
**by** (*rule eq-matI, insert H2-UR-0j H2-UR-01 H2-UR-00 H2-UR A nat-neq-iff,*  
*auto*)  
**have**  $H2\text{-UL-}H: H2\text{-UL}\$(0,0) = H\$(0,0)$   
**proof** –  
**have**  $H2\text{-UL}\$(0,0) = H2\$(0,0)$   
**by** (*metis (no-types, lifting) Pair-inject index-mat(1) split-H2 split-block-def*  
*zero-less-one-class.zero-less-one*)  
**also have**  $\dots = H\$(0,0)$   
**unfolding**  $H2\text{-def}$  **by** (*rule index-mat-addcol, insert H A n-ge-2, auto*)  
**finally show** *?thesis* .  
**qed**  
**have**  $H2\text{-DL-}H-10: H2\text{-DL}\$(0,0) = H\$(1,0)$   
**proof** –  
**have**  $H2\text{-DL}\$(0,0) = H2\$(1,0)$   
**by** (*smt (verit, ccfv-threshold) H2-DL One-nat-def Pair-inject add.right-neutral*  
*add-Suc-right carrier-matD(1)*  
*dim-row-mat(1) index-mat(1) rel-simps(68) split-H2 split-block-def*  
*split-conv*)  
**also have**  $\dots = H\$(1,0)$  **unfolding**  $H2\text{-def}$  **by** (*rule index-mat-addcol, insert*  
*H A n-ge-2, auto*)  
**finally show** *?thesis* .  
**qed**  
**have**  $H-10: H\$(1,0) = 0$   
**proof** –  
**have**  $H\$(1,0) = (P2 * D * Q2)\$(1,0)$  **unfolding**  $H\text{-def}$   
**by** (*smt (verit) A C-D-E C-P1-A-Q1' D E One-nat-def P1 P2-P1 Q2 Q2'*  
*Q2'-def Suc-lessD append-cols-def*  
*carrier-matD dim-col-A-g2 index-mat-four-block index-mult-mat in-*)

```

dex-one-mat
  index-zero-mat lessI numerals(2)
  also have ... = 0 using is-SNF-D P2D2Q2 D
  unfolding is-SNF-def Smith-normal-form-mat-def isDiagonal-mat-def by
auto
  finally show ?thesis .
qed
have S-H2-Q3': S = H2 * Q3'
and S-as-four-block-mat: S = four-block-mat (H2-UL) (0m 1 (n - 1)) (H2-DL)
(H2-DR * Q3)
proof -
  have H2 * Q3' = four-block-mat (H2-UL * 1m 1 + H2-UR * 0m (dim-col A
- 1) 1)
(H2-UL * 0m 1 (dim-col A - 1) + H2-UR * Q3)
(H2-DL * 1m 1 + H2-DR * 0m (dim-col A - 1) 1) (H2-DL * 0m 1 (dim-col
A - 1) + H2-DR * Q3)
  unfolding H2-as-four-block-mat Q3'-def
  by (rule mult-four-block-mat[OF H2-UL H2-UR H2-DL H2-DR], insert Q3
A H', auto)
  also have ... = four-block-mat (H2-UL) (0m 1 (n - 1)) (H2-DL) (H2-DR *
Q3)
  by (rule cong-four-block-mat, insert H2-UR-0 H2-UL H2-UR H2-DL H2-DR
Q3, auto)
  also have *: ... = S unfolding S-def
  proof (rule cong-four-block-mat)
    show H2-UL = Matrix.mat 1 1 (λ(a, b). H $$ (0, 0))
    by (rule eq-matI, insert H2-UL H2-UL-H, auto)
    show H2-DR * Q3 = H-1xn using is-SNF-H' unfolding is-SNF-def by
auto
    show 0m 1 (n - 1) = 0m 1 (dim-col A - 1) using A by auto
    show H2-DL = 0m 1 1 using H2-DL H2-DL-H-10 H-10 by auto
  qed
  finally show S = H2 * Q3'
  and S = four-block-mat (H2-UL) (0m 1 (n - 1)) (H2-DL) (H2-DR * Q3)
  using * by auto
qed
thus S = P2 * P1 * A * (Q1' * Q2' * Q-div-k * Q3') unfolding H2-P2-P1-A-Q1'-Q2'-Q-div-k

  by (smt (verit, ccfv-threshold) Q1' Q2' Q2'-def Q3' Q3'-def Q-div-k as-
soc-mult-mat
  carrier-matD carrier-mat-triv index-mult-mat)
show Smith-normal-form-mat S
proof (rule Smith-normal-form-mat-intro)
  have Sij-0: S$$ (i, j) = 0 if ij: i ≠ j and i: i < dim-row S and j: j < dim-col
S for i j
  proof (cases i=1 ∧ j=0)
    case True
    have S$$ (1, 0) = 0 using S-as-four-block-mat
    by (metis (no-types, lifting) H2-DL-H-10 H2-UL H-10 One-nat-def True

```

```

carrier-matD diff-Suc-1
  index-mat-four-block rel-simps(71) that(2) that(3) zero-less-one-class.zero-less-one)
  then show ?thesis using True by auto
next
  case False note not-10 = False
  show ?thesis
  proof (cases i=0)
    case True
      hence j0: j>0 using ij by auto
      then show ?thesis using S-as-four-block-mat
        by (smt (verit) 1(2) H2-DR H2-H-Q-div-k H2-UL H-P2-P1-A-Q1'-Q2'
          Num.numeral-nat(7) P2-P1 Q3 S-H2-Q3'
            Suc-pred True carrier-matD index-mat-four-block index-mult-mat
              index-zero-mat(1)
                not-less-eq plus-1-eq-Suc pos2 that(3) zero-less-one-class.zero-less-one)
    next
      case False
        have SNF-H-1xn: Smith-normal-form-mat H-1xn using is-SNF-H' un-
          folding is-SNF-def by auto
        have i1: i=1 using False ij i H2-DR H2-UL S-as-four-block-mat by auto
        hence j1: j>1 using ij not-10 by auto thm is-SNF-H'
        have S$$ (i,j) = (if i < dim-row H2-UL then if j < dim-col H2-UL then
          H2-UL $$ (i, j)
            else (0m 1 (n - 1)) $$ (i, j - dim-col H2-UL)
              else if j < dim-col H2-UL then H2-DL $$ (i - dim-row H2-UL, j)
                else (H2-DR * Q3) $$ (i - dim-row H2-UL, j - dim-col H2-UL))
          unfolding S-as-four-block-mat
          by (rule index-mat-four-block, insert i j H2-UL H2-DR Q3 S-H2-Q3' H2
            Q3' A, auto)
        also have ... = (H2-DR * Q3) $$ (0, j - 1) using H2-UL i1 not-10 by
          auto
        also have ... = H-1xn $$ (0,j-1)
          using S-def calculation i1 j not-10 i by auto
        also have ... = 0 using SNF-H-1xn j1 i j
          unfolding Smith-normal-form-mat-def isDiagonal-mat-def
          by (simp add: S-def i1)
        finally show ?thesis .
      qed
    qed
  thus isDiagonal-mat S unfolding isDiagonal-mat-def by auto
  have S$$ (0,0) dvd S$$ (1,1)
  proof -
    have dvd-all:  $\forall i j. i < 2 \wedge j < n \longrightarrow H2-UL \$$(0,0) dvd (H2 * Q3') \$$(i, j)$ 
    proof (rule dvd-elements-mult-matrix-right)
      show H2':  $H2 \in \text{carrier-mat } 2 \ n$  using H2 A by auto
      show Q3':  $Q3' \in \text{carrier-mat } n \ n$  using Q3' A by auto
      have H2-UL  $\$(0, 0) \text{ dvd } H2 \ \$$(i, j) \text{ if } i: i < 2 \text{ and } j: j < n \text{ for } i \ j$ 
      proof (cases i=0)

```



```

case True
then show ?thesis
  by (metis (no-types, lifting) A H2-H-Q-div-k H2-UL H2-UR-0
H2-as-four-block-mat
H-P2-P1-A-Q1'-Q2' P2-P1 Q3 Q-div-k S-as-four-block-mat Sij-0
carrier-matD
dvd-0-right dvd-refl index-mat-four-block index-mult-mat(2,3) j
less-one pos2)
  next
    case False
      hence i1: i=1 using i by auto
      have H2-10-0: H2 $$ (1,0) = 0
      by (metis (no-types, lifting) H2-H-Q-div-k H2-def H-10 H-P2-P1-A-Q1'-Q2'
One-nat-def
Q2' H2' basic-trans-rules(19) carrier-matD dim-col-A-g2 in-
dex-mat-addcol(3)
index-mult-mat(2,3) lessI numeral-2-eq-2 rel-simps(76))
      moreover have H2-UL00-dvd-H211:H2-UL $$ (0, 0) dvd H2 $$ (1, 1)
      proof -
        have H2-UL $$ (0, 0) = H $$ (0, 0) by (simp add: H2-UL-H)
        also have ... = (P2*D*Q2) $$ (0,0) unfolding H-def using
append-cols-nth D E
        by (smt (verit, ccfv-threshold) A C-D-E C-P1-A-Q1' D H2-DR
H2-H-Q-div-k H2-UL H2-as-four-block-mat
H-P2-P1-A-Q1'-Q2' One-nat-def P1 Q1' Q2 Suc-lessD append-cols-def
carrier-matD
dim-col-A-g2 index-mat-four-block index-mult-mat numerals(2)
plus-1-eq-Suc zero-less-Suc)
        also have ... dvd (P2*D*Q2) $$ (1,1)
        using is-SNF-D P2D2Q2 unfolding is-SNF-def Smith-normal-form-mat-def
by auto
        (metis D Q2 carrier-matD index-mult-mat(1) index-mult-mat(2) lessI
numerals(2) pos2)
        also have ... = H $$ (1,1) unfolding H-def using append-cols-nth D
E
        by (smt (verit, ccfv-threshold) A C-D-E C-P1-A-Q1' H2-DR
H2-H-Q-div-k H2-UL H2-as-four-block-mat H-P2-P1-A-Q1'-Q2'
One-nat-def P1 Q1' Q2 append-cols-def carrier-matD(1) car-
rier-matD(2) dim-col-A-g2
index-mat-four-block index-mult-mat(2) index-mult-mat(3) lessI
less-trans-Suc
numerals(2) plus-1-eq-Suc pos2)
        also have ... = H2 $$ (1, 1)
        by (metis A H2-def H-P2-P1-A-Q1'-Q2' One-nat-def P2-P1 Q2'
carrier-matD dim-col-A-g2 i i1
index-mat-addcol(3) index-mult-mat(2) index-mult-mat(3)
less-trans-Suc nat-neq-iff pos2)
      finally show ?thesis .
qed

```

**moreover have**  $H2\text{-}UL00\text{-}dvd\text{-}H212: H2\text{-}UL \ \$\$ (0, 0) \ dvd \ H2 \ \$\$ (1, 2)$   
**proof** –  
**have**  $H2\text{-}UL \ \$\$ (0, 0) = H \ \$\$ (0, 0)$  **by** (*simp add: H2-UL-H*)  
**also have ... dvd H**  $\ \$\$ (1, 2)$  **using**  $D01\text{-}dvd\text{-}H12 \ H00\text{-}dvd\text{-}D01 \ dvd\text{-}trans$   
**by** *blast*  
**also have ... = (-k) \* H**  $\ \$\$ (1, 0) + H \ \$\$ (1, 2)$   
**using**  $H\text{-}10$  **by** *auto*  
**also have ... = H2**  $\ \$\$ (1, 2)$   
**unfolding**  $H2\text{-}def$  **by** (*rule index-mat-addcol[symmetric], insert H A*  
*n-ge-2, auto*)  
**finally show** *?thesis* .  
**qed**  
**moreover have**  $H2 \ \$\$ (1, j) = 0$  **if**  $j1: j > 2$  **and**  $j: j < n$   
**proof** –  
**let**  $?f = (\lambda(i, j). \sum ia = 0..<dim\text{-}vec \ (col \ E \ j). \ Matrix.\ row \ P2 \ i \ \$v \ ia$   
 $* \ col \ E \ j \ \$v \ ia)$   
**have**  $H2 \ \$\$ (1, j) = H \ \$\$ (1, j)$  **unfolding**  $H2\text{-}def$  **using**  $j \ j1 \ n\text{-}ge\text{-}2$   
**by** (*metis (mono-tags, lifting) 1(2) H2' H-10 H-P2-P1-A-Q1'-Q2'*  
 $Q2' \ arithmetic\text{-}simps(49)$   
 $carrier\text{-}matD \ i \ i1 \ index\text{-}mat\text{-}addcol(1) \ index\text{-}mult\text{-}mat \ semiring\text{-}norm(64) \ H2\text{-}H\text{-}Q\text{-}div\text{-}k$ )  
**also have ... = (P2 \* E)**  $\ \$\$ (1, j - 2)$  **unfolding**  $H\text{-}def$   
**by** (*smt A C-D-E C-P1-A-Q1' D H2' H2-H-Q-div-k H-P2-P1-A-Q1'-Q2'*  
 $P1 \ Q1' \ Q2 \ append\text{-}cols\text{-}def$   
 $basic\text{-}trans\text{-}rules(19) \ carrier\text{-}matD \ index\text{-}mat\text{-}four\text{-}block \ index\text{-}mult\text{-}mat(2)$   
 $index\text{-}mult\text{-}mat(3) \ j \ less\text{-}one \ nat\text{-}neq\text{-}iff \ not\text{-}less\text{-}less\text{-}Suc\text{-}eq$   
 $numerals(2) \ j1$ )  
**also have ... = Matrix.mat** ( $dim\text{-}row \ P2$ ) ( $dim\text{-}col \ E$ )  $?f \ \$\$ (1, j - 2)$   
**unfolding**  $times\text{-}mat\text{-}def \ scalar\text{-}prod\text{-}def$  **by** *simp*  
**also have ... = ?f** ( $1, j - 2$ ) **by** (*rule index-mat, insert P2 E E-def*  
 $n\text{-}ge\text{-}2 \ j \ j1 \ A, \ auto$ )  
**also have ... =** ( $\sum ia = 0..<2. \ Matrix.\ row \ P2 \ 1 \ \$v \ ia * \ col \ E \ (j - 2)$   
 $\$v \ ia)$   
**using**  $E \ A \ E\text{-}def \ j \ j1$  **by** *auto*  
**also have ... =** ( $\sum ia \in \{0, 1\}. \ Matrix.\ row \ P2 \ 1 \ \$v \ ia * \ col \ E \ (j - 2)$   
 $\$v \ ia)$   
**by** (*rule sum.cong, auto*)  
**also have ... = Matrix.row**  $P2 \ 1 \ \$v \ 0 * \ col \ E \ (j - 2) \ \$v \ 0$   
 $+ \ Matrix.\ row \ P2 \ 1 \ \$v \ 1 * \ col \ E \ (j - 2) \ \$v \ 1$   
**by** (*simp add: sum-two-elements[OF zero-neq-one]*)  
**also have ... = 0** **using**  $E\text{-}ij\text{-}0 \ E\text{-}def \ E \ A$   
**by** (*auto, smt (verit) D Q2 Q2' Q2'-def Suc-lessD add-cancel-right-right*  
 $add\text{-}diff\text{-}inverse\text{-}nat$   
 $arith\text{-}extra\text{-}simps(19) \ carrier\text{-}matD \ i \ i1 \ index\text{-}col \ index\text{-}mat\text{-}four\text{-}block(3)$   
 $index\text{-}one\text{-}mat(3) \ less\text{-}2\text{-}cases \ nat\text{-}add\text{-}left\text{-}cancel\text{-}less \ numeral\text{-}2\text{-}eq\text{-}2$   
 $semiring\text{-}norm(138) \ semiring\text{-}norm(160) \ j \ j1 \ zero\text{-}less\text{-}diff$ )

**finally show** *?thesis* .  
**qed**  
**ultimately show** *?thesis* **using** *i1 False*  
**by** (*metis One-nat-def dvd-0-right less-2-cases nat-neq-iff j*)  
**qed**  
**thus**  $\forall i j. i < 2 \wedge j < n \longrightarrow H2\text{-UL} \ \$\$ (0, 0) \text{ dvd } H2 \ \$\$ (i, j)$  **by** *auto*  
**qed**  
**have**  $S\ \$\$ (0,0) = H2\text{-UL} \ \$\$ (0,0)$  **using** *H2-UL S-as-four-block-mat* **by**  
*auto*  
**also have** ... *dvd (H2\*Q3')*  $\ \$\$ (1,1)$  **using** *dvd-all n-ge-2* **by** *auto*  
**also have** ... = *S*  $\ \$\$ (1,1)$  **using** *S-H2-Q3'* **by** *auto*  
**finally show** *?thesis* .  
**qed**  
**thus**  $\forall a. a + 1 < \min (\dim\text{-row } S) (\dim\text{-col } S) \longrightarrow S \ \$\$ (a, a) \text{ dvd } S \ \$\$ (a + 1, a + 1)$   
**by** (*metis 1(2) H2-H-Q-div-k H-P2-P1-A-Q1'-Q2' One-nat-def P2-P1 S-H2-Q3' Suc-eq-plus1 index-mult-mat(2) less-Suc-eq less-one min-less-iff-conj numeral-2-eq-2 carrier-matD(1)*)  
**qed**  
**qed**  
**qed**

**lemma** *is-SNF-Smith-2xn*:

**assumes** *A*:  $A \in \text{carrier-mat } 2 \ n$   
**shows** *is-SNF A (Smith-2xn A)*  
**proof** (*cases n > 2*)  
**case** *True*  
**then show** *?thesis* **using** *is-SNF-Smith-2xn-n-ge-2[OF A]* **by** *simp*  
**next**  
**case** *False*  
**hence**  $n=0 \vee n=1 \vee n=2$  **by** *auto*  
**then show** *?thesis* **using** *Smith-2xn-0 Smith-2xn-1 Smith-2xn-2 A* **by** *blast*  
**qed**

#### 16.3.4 Case $n \times 2$

**definition** *Smith-nx2 A* = (*let*  $(P,S,Q) = \text{Smith-2xn } A^T$  *in*  $(Q^T, S^T, P^T)$ )

**lemma** *is-SNF-Smith-nx2*:

**assumes** *A*:  $A \in \text{carrier-mat } n \ 2$   
**shows** *is-SNF A (Smith-nx2 A)*  
**proof** –  
**obtain** *P S Q* **where**  $PSQ: (P,S,Q) = \text{Smith-2xn } A^T$  **by** (*metis prod-cases3*)  
**hence** *rw*:  $\text{Smith-nx2 } A = (Q^T, S^T, P^T)$  **unfolding** *Smith-nx2-def* **by** (*metis split-conv*)

```

have is-SNF  $A^T$  (Smith-2xn  $A^T$ ) by (rule is-SNF-Smith-2xn, insert id A, auto)
hence is-SNF-PSQ: is-SNF  $A^T$  ( $P, S, Q$ ) using PSQ by auto
show ?thesis
proof (unfold rw, rule is-SNF-intro)
  show  $Q^T \in \text{carrier-mat } (\text{dim-row } A) (\text{dim-row } A)$ 
  and  $P^T \in \text{carrier-mat } (\text{dim-col } A) (\text{dim-col } A)$ 
  and invertible-mat  $Q^T$  and invertible-mat  $P^T$ 
  using is-SNF-PSQ invertible-mat-transpose unfolding is-SNF-def by auto
  have Smith-normal-form-mat  $S$  and PATQ:  $S = P * A^T * Q$ 
  using is-SNF-PSQ invertible-mat-transpose unfolding is-SNF-def by auto
  thus Smith-normal-form-mat  $S^T$  unfolding Smith-normal-form-mat-def isDiagonal-mat-def by auto
  show  $S^T = Q^T * A * P^T$  using PATQ
  by (smt (verit, ccfv-threshold) Matrix.transpose-mult Matrix.transpose-transpose Pt Qt assoc-mult-mat carrier-mat-triv index-mult-mat(2))
qed
qed

```

### 16.3.5 Case $m \times n$

```

declare Smith-2xn.simps[simp del]

```

```

function (domintros) Smith-mxn :: 'a mat  $\Rightarrow$  ('a mat  $\times$  'a mat  $\times$  'a mat)

```

```

where

```

```

  Smith-mxn  $A =$  (
    if  $\text{dim-row } A = 0 \vee \text{dim-col } A = 0$  then ( $1_m (\text{dim-row } A), A, 1_m (\text{dim-col } A)$ )
    else if  $\text{dim-row } A = 1$  then ( $1_m 1, \text{Smith-1xn } A$ )
    else if  $\text{dim-row } A = 2$  then Smith-2xn  $A$ 
    else if  $\text{dim-col } A = 1$  then let ( $P, S$ ) = Smith-nx1  $A$  in ( $P, S, 1_m 1$ )
    else if  $\text{dim-col } A = 2$  then Smith-nx2  $A$ 
    else
      let  $A1 = \text{mat-of-row } (\text{Matrix.row } A 0);$ 
       $A2 = \text{mat-of-rows } (\text{dim-col } A) [\text{Matrix.row } A i. i \leftarrow [1..<\text{dim-row } A]];$ 
      ( $P1, D1, Q1$ ) = Smith-mxn  $A2;$ 
       $C = (A1 * Q1) @_r (P1 * A2 * Q1);$ 
       $D = \text{mat-of-rows } (\text{dim-col } A) [\text{Matrix.row } C 0, \text{Matrix.row } C 1];$ 
       $E = \text{mat-of-rows } (\text{dim-col } A) [\text{Matrix.row } C i. i \leftarrow [2..<\text{dim-row } A]];$ 
      ( $P2, F, Q2$ ) = Smith-2xn  $D;$ 
       $H = (P2 * D * Q2) @_r (E * Q2);$ 
      ( $P-H2, H2$ ) = reduce-column div-op  $H;$ 
      ( $H2-UL, H2-UR, H2-DL, H2-DR$ ) = split-block  $H2 1 1;$ 
      ( $P3, S', Q3$ ) = Smith-mxn  $H2-DR;$ 
       $S = \text{four-block-mat } (\text{Matrix.mat } 1 1 (\lambda(a, b). H \$\$ (0, 0))) (0_m 1 (\text{dim-col } A - 1)) (0_m (\text{dim-row } A - 1) 1) S';$ 
       $P1' = \text{four-block-mat } (1_m 1) (0_m 1 (\text{dim-row } A - 1)) (0_m (\text{dim-row } A - 1) 1) P1;$ 
       $P2' = \text{four-block-mat } P2 (0_m 2 (\text{dim-row } A - 2)) (0_m (\text{dim-row } A - 2) 2) (1_m (\text{dim-row } A - 2));$ 

```

```

    P3' = four-block-mat (1m 1) (0m 1 (dim-row A - 1)) (0m (dim-row A -
1) 1) P3;
    Q3' = four-block-mat (1m 1) (0m 1 (dim-col A - 1)) (0m (dim-col A - 1)
1) Q3
    in (P3' * P-H2 * P2' * P1', S, Q1 * Q2 * Q3')
)
  by pat-completeness fast

```

```

declare Smith-2xn.simps[simp]

```

```

lemma Smith-mxn-dom-nm-less-2:

```

```

  assumes A: A ∈ carrier-mat m n and mn: n ≤ 2 ∨ m ≤ 2
  shows Smith-mxn-dom A
  by (rule Smith-mxn.domintros, insert assms, auto)

```

```

lemma Smith-mxn-pinduct-carrier-less-2:

```

```

  assumes A: A ∈ carrier-mat m n and mn: n ≤ 2 ∨ m ≤ 2
  shows fst (Smith-mxn A) ∈ carrier-mat m m
  ∧ fst (snd (Smith-mxn A)) ∈ carrier-mat m n
  ∧ snd (snd (Smith-mxn A)) ∈ carrier-mat n n

```

```

proof -

```

```

  have A-dom: Smith-mxn-dom A using Smith-mxn-dom-nm-less-2[OF assms] by
simp

```

```

  show ?thesis

```

```

proof (cases dim-row A = 0 ∨ dim-col A = 0)

```

```

  case True

```

```

  have Smith-mxn A = (1m (dim-row A), A, 1m (dim-col A))

```

```

  using Smith-mxn.psimps[OF A-dom] True by auto

```

```

  thus ?thesis using A by auto

```

```

next

```

```

  case False note 1 = False

```

```

  show ?thesis

```

```

  proof (cases dim-row A = 1)

```

```

    case True

```

```

    have Smith-mxn A = (1m 1, Smith-1xn A)

```

```

    using Smith-mxn.psimps[OF A-dom] True 1 by auto

```

```

    then show ?thesis using Smith-1xn-works unfolding is-SNF-def

```

```

    by (smt (verit) Smith-1xn-aux-Q-carrier Smith-1xn-aux-S'-AQ' Smith-1xn-def
True assms(1) carrier-matD

```

```

    carrier-matI diff-less fst-conv index-mult-mat not-gr0 one-carrier-mat
prod.collapse

```

```

    right-mult-one-mat' snd-conv zero-less-one-class.zero-less-one)

```

```

  next

```

```

    case False note 2 = False

```

```

    then show ?thesis

```

```

    proof (cases dim-row A = 2)

```

```

    case True
    hence A': A ∈ carrier-mat 2 n using A by auto
    have Smith-mxn A = Smith-2xn A using Smith-mxn.psimps[OF A-dom] True
1 2 by auto
    then show ?thesis using is-SNF-Smith-2xn[OF A'] A unfolding is-SNF-def
    by (metis (mono-tags, lifting) carrier-matD carrier-mat-triv case-prod-beta
index-mult-mat(2,3))
  next
  case False note 3 = False
  show ?thesis
  proof (cases dim-col A = 1)
    case True
    hence A': A ∈ carrier-mat m 1 using A by auto
    have Smith-mxn A = (let (P,S) = Smith-nx1 A in (P,S,1_m 1))
    using Smith-mxn.psimps[OF A-dom] True 1 2 3 by auto
    then show ?thesis using Smith-nx1-works[OF A'] A unfolding is-SNF-def
    by (metis (mono-tags, lifting) carrier-matD carrier-mat-triv case-prod-unfold

        index-mult-mat(2,3) surjective-pairing)
  next
  case False
  hence dim-col A = 2 using 1 2 3 mn A by auto
  hence A': A ∈ carrier-mat m 2 using A by auto
  hence Smith-mxn A = Smith-nx2 A
    using Smith-mxn.psimps[OF A-dom] 1 2 3 False by auto
  then show ?thesis using is-SNF-Smith-nx2[OF A'] A unfolding is-SNF-def
by force
  qed
  qed
  qed
  qed
  qed
  qed

```

**lemma** *Smith-mxn-pinduct-carrier-ge-2*:  $\llbracket \text{Smith-mxn-dom } A; A \in \text{carrier-mat } m \ n; m > 2; n > 2 \rrbracket \implies$

```

  fst (Smith-mxn A) ∈ carrier-mat m m
  ∧ fst (snd (Smith-mxn A)) ∈ carrier-mat m n
  ∧ snd (snd (Smith-mxn A)) ∈ carrier-mat n n
proof (induct arbitrary: m n rule: Smith-mxn.pinduct)
  case (1 A)
  note A-dom = 1(1)
  note A = 1.prem(1)
  note m = 1.prem(2)
  note n = 1.prem(3)
  define A1 where A1 = mat-of-row (Matrix.row A 0)
  define A2 where A2 = mat-of-rows (dim-col A) [Matrix.row A i. i ← [1..<dim-row
A]]
  obtain P1 D1 Q1 where P1D1Q1: (P1,D1,Q1) = Smith-mxn A2 by (metis
prod-cases3)

```

**define**  $C$  **where**  $C = (A1 * Q1) @_r (P1 * A2 * Q1)$   
**define**  $D$  **where**  $D = \text{mat-of-rows } (\text{dim-col } A) [\text{Matrix.row } C \ 0, \text{Matrix.row } C \ 1]$   
**define**  $E$  **where**  $E = \text{mat-of-rows } (\text{dim-col } A) [\text{Matrix.row } C \ i. \ i \leftarrow [2..<\text{dim-row } A]]$   
**obtain**  $P2 \ F \ Q2$  **where**  $P2FQ2: (P2, F, Q2) = \text{Smith-2xn } D$  **by**  $(\text{metis prod-cases3})$   
**define**  $H$  **where**  $H = (P2 * D * Q2) @_r (E * Q2)$   
**obtain**  $P\text{-}H2 \ H2$  **where**  $P\text{-}H2H2: (P\text{-}H2, H2) = \text{reduce-column div-op } H$  **by**  $(\text{metis surj-pair})$   
**obtain**  $H2\text{-}UL \ H2\text{-}UR \ H2\text{-}DL \ H2\text{-}DR$  **where**  $\text{split-}H2: (H2\text{-}UL, H2\text{-}UR, H2\text{-}DL, H2\text{-}DR) = \text{split-block } H2 \ 1 \ 1$   
**by**  $(\text{metis split-block-def})$   
**obtain**  $P3 \ S' \ Q3$  **where**  $P3S'Q3: (P3, S', Q3) = \text{Smith-mxn } H2\text{-}DR$  **by**  $(\text{metis prod-cases3})$   
**define**  $S$  **where**  $S = \text{four-block-mat } (\text{Matrix.mat } 1 \ 1 \ (\lambda(a, b). H \ \$\$ \ (0, 0)))$   
 $(0_m \ 1 \ (\text{dim-col } A - 1))$   
 $(0_m \ (\text{dim-row } A - 1) \ 1) \ S'$   
**define**  $P1'$  **where**  $P1' = \text{four-block-mat } (1_m \ 1) \ (0_m \ 1 \ (\text{dim-row } A - 1)) \ (0_m \ (\text{dim-row } A - 1) \ 1) \ P1$   
**define**  $P2'$  **where**  $P2' = \text{four-block-mat } P2 \ (0_m \ 2 \ (\text{dim-row } A - 2)) \ (0_m \ (\text{dim-row } A - 2) \ 2) \ (1_m \ (\text{dim-row } A - 2))$   
**define**  $P3'$  **where**  $P3' = \text{four-block-mat } (1_m \ 1) \ (0_m \ 1 \ (\text{dim-row } A - 1)) \ (0_m \ (\text{dim-row } A - 1) \ 1) \ P3$   
**define**  $Q3'$  **where**  $Q3' = \text{four-block-mat } (1_m \ 1) \ (0_m \ 1 \ (\text{dim-col } A - 1)) \ (0_m \ (\text{dim-col } A - 1) \ 1) \ Q3$   
**have**  $A1: A1 \in \text{carrier-mat } 1 \ n$  **unfolding**  $A1\text{-def}$  **using**  $A$  **by**  $\text{auto}$   
**have**  $A2: A2 \in \text{carrier-mat } (m-1) \ n$  **unfolding**  $A2\text{-def}$  **using**  $A$  **by**  $\text{auto}$   
**have**  $\text{fst } (\text{Smith-mxn } A2) \in \text{carrier-mat } (m-1) \ (m-1)$   
 $\wedge \text{fst } (\text{snd } (\text{Smith-mxn } A2)) \in \text{carrier-mat } (m-1) \ n$   
 $\wedge \text{snd } (\text{snd } (\text{Smith-mxn } A2)) \in \text{carrier-mat } n \ n$   
**proof**  $(\text{cases } 2 < m - 1)$   
**case**  $\text{True}$   
**show**  $?thesis$  **by**  $(\text{rule } 1.\text{hyps}(2), \text{insert } A \ m \ n \ A2\text{-def } A1\text{-def } \text{True } \text{id}, \text{auto})$   
**next**  
**case**  $\text{False}$   
**hence**  $m=3$  **using**  $m$  **by**  $\text{auto}$   
**hence**  $A2': A2 \in \text{carrier-mat } 2 \ n$  **using**  $A2$  **by**  $\text{auto}$   
**have**  $A2\text{-dom}: \text{Smith-mxn-dom } A2$  **using**  $A2'$   $\text{Smith-mxn-dom-nm-less-2}$  **by**  $\text{force}$   
**have**  $\text{dim-row } A2 = 2$  **using**  $A2 \ A2'$  **by**  $\text{fast}$   
**hence**  $\text{Smith-mxn } A2 = \text{Smith-2xn } A2$   
**using**  $n$  **unfolding**  $\text{Smith-mxn.psimps}[OF \ A2\text{-dom}]$  **by**  $\text{auto}$   
**then show**  $?thesis$  **using**  $\text{is-SNF-Smith-2xn}[OF \ A2'] \ m \ A2$  **unfolding**  $\text{is-SNF-def}$   $\text{split-beta}$   
**by**  $(\text{metis carrier-matD carrier-matI index-mult-mat}(2,3))$   
**qed**  
**hence**  $P1: P1 \in \text{carrier-mat } (m-1) \ (m-1)$   
**and**  $D1: D1 \in \text{carrier-mat } (m-1) \ n$   
**and**  $Q1: Q1 \in \text{carrier-mat } n \ n$  **using**  $P1D1Q1$  **by**  $(\text{metis fst-conv snd-conv})+$

```

have C : C ∈ carrier-mat (1 + (m-1)) n unfolding C-def
  by (rule carrier-append-rows, insert P1 D1 Q1 A1, auto)
hence C : C ∈ carrier-mat m n using m by simp
have D : D ∈ carrier-mat 2 n unfolding D-def using C A by auto
have E : E ∈ carrier-mat (m-2) n unfolding E-def using A by auto
have P2 : P2 ∈ carrier-mat 2 2 and Q2 : Q2 ∈ carrier-mat n n
  using is-SNF-Smith-2xn[OF D] P2FQ2 D unfolding is-SNF-def by auto
have H ∈ carrier-mat (2 + (m-2)) n unfolding H-def
  by (rule carrier-append-rows, insert P2 D Q2 E, auto)
hence H : H ∈ carrier-mat m n using m by auto
have H2 : H2 ∈ carrier-mat m n using m H P-H2H2 reduce-column by blast
have H2-DR : H2-DR ∈ carrier-mat (m - 1) (n - 1)
  by (rule split-block(4)[OF split-H2[symmetric]], insert H2 m n, auto)
have fst (Smith-mxn H2-DR) ∈ carrier-mat (m-1) (m-1)
  ∧ fst (snd (Smith-mxn H2-DR)) ∈ carrier-mat (m-1) (n-1)
  ∧ snd (snd (Smith-mxn H2-DR)) ∈ carrier-mat (n-1) (n-1)
proof (cases 2 < m-1 ∧ 2 < n-1)
  case True
    show ?thesis
  proof (rule 1.hyps(3)[OF - - - - A1-def A2-def P1D1Q1 - - C-def])
    show (P2,F,Q2) = Smith-2xn D using P2FQ2 by auto
  qed (insert A P1D1Q1 D-def E-def P2FQ2 P-H2H2 P3S'Q3 H-def split-H2
H2-DR True id, auto)
  next
    case False note m-eq-3-or-n-eq-3 = False
    show ?thesis
  proof (cases (2 < m - 1))
    case True
      hence n3 : n=3 using m-eq-3-or-n-eq-3 n m by auto
      have H2-DR-dom : Smith-mxn-dom H2-DR
        by (rule Smith-mxn.domintros, insert H2-DR n3, auto)
      have H2-DR' : H2-DR ∈ carrier-mat (m-1) 2 using H2-DR n3 by auto
      hence dim-col H2-DR = 2 by simp
      hence Smith-mxn H2-DR = Smith-nx2 H2-DR
        using n H2-DR' True unfolding Smith-mxn.psimps[OF H2-DR-dom] by
auto
      then show ?thesis using is-SNF-Smith-nx2[OF H2-DR'] m H2-DR unfolding
is-SNF-def by auto
    next
      case False
      hence m3 : m=3 using m-eq-3-or-n-eq-3 n m by auto
      have H2-DR-dom : Smith-mxn-dom H2-DR
        using False H2-DR Smith-mxn-dom-nm-less-2 not-less by blast
      have H2-DR' : H2-DR ∈ carrier-mat 2 (n-1) using H2-DR m3 by auto
      hence dim-row H2-DR = 2 by simp
      hence Smith-mxn H2-DR = Smith-2xn H2-DR
        using n H2-DR' unfolding Smith-mxn.psimps[OF H2-DR-dom] by auto
      then show ?thesis using is-SNF-Smith-2xn[OF H2-DR'] m H2-DR unfolding
is-SNF-def by force

```



**qed**  
**qed**  
**hence**  $P3$ :  $P3 \in \text{carrier-mat } (m-1) (m-1)$   
**and**  $S'$ :  $S' \in \text{carrier-mat } (m-1) (n-1)$   
**and**  $Q3$ :  $Q3 \in \text{carrier-mat } (n-1) (n-1)$  **using**  $P3S'Q3$  **by** (*metis fst-conv snd-conv*)  
**have**  $\text{Smith-final}$ :  $\text{Smith-mxn } A = (P3' * P-H2 * P2' * P1', S, Q1 * Q2 * Q3')$   
**proof** –  
**have**  $P1\text{-def}$ :  $P1 = \text{fst } (\text{Smith-mxn } A2)$  **and**  $D1\text{-def}$ :  $D1 = \text{fst } (\text{snd } (\text{Smith-mxn } A2))$   
**and**  $Q1\text{-def}$ :  $Q1 = \text{snd } (\text{snd } (\text{Smith-mxn } A2))$  **using**  $P1D1Q1$  **by** (*metis fstI sndI*)  
**have**  $P2\text{-def}$ :  $P2 = \text{fst } (\text{Smith-2xn } D)$  **and**  $F\text{-def}$ :  $F = \text{fst } (\text{snd } (\text{Smith-2xn } D))$   
  
**and**  $Q2\text{-def}$ :  $Q2 = \text{snd } (\text{snd } (\text{Smith-2xn } D))$  **using**  $P2FQ2$  **by** (*metis fstI sndI*)  
**have**  $P-H2\text{-def}$ :  $P-H2 = \text{fst } (\text{reduce-column div-op } H)$   
**and**  $H2\text{-def}$ :  $H2 = \text{snd } (\text{reduce-column div-op } H)$   
**using**  $P-H2H2$  **by** (*metis fstI sndI*)  
**have**  $H2\text{-UL-def}$ :  $H2\text{-UL} = \text{fst } (\text{split-block } H2 \ 1 \ 1)$   
**and**  $H2\text{-UR-def}$ :  $H2\text{-UR} = \text{fst } (\text{snd } (\text{split-block } H2 \ 1 \ 1))$   
**and**  $H2\text{-DL-def}$ :  $H2\text{-DL} = \text{fst } (\text{snd } (\text{snd } (\text{split-block } H2 \ 1 \ 1)))$   
**and**  $H2\text{-DR-def}$ :  $H2\text{-DR} = \text{snd } (\text{snd } (\text{snd } (\text{split-block } H2 \ 1 \ 1)))$   
**using**  $\text{split-H2}$  **by** (*metis fstI sndI*)  
**have**  $P3\text{-def}$ :  $P3 = \text{fst } (\text{Smith-mxn } H2\text{-DR})$   
**and**  $S'\text{-def}$ :  $S' = \text{fst } (\text{snd } (\text{Smith-mxn } H2\text{-DR}))$   
**and**  $Q3\text{-def}$ :  $Q3 = (\text{snd } (\text{snd } (\text{Smith-mxn } H2\text{-DR})))$  **using**  $P3S'Q3$  **by** (*metis fstI sndI*)  
**note**  $\text{aux} = \text{Smith-mxn.psimps}[OF \ A\text{-dom}] \ \text{Let-def split-beta}$   
 $A1\text{-def}[\text{symmetric}] \ A2\text{-def}[\text{symmetric}] \ P1\text{-def}[\text{symmetric}] \ D1\text{-def}[\text{symmetric}]$   
 $Q1\text{-def}[\text{symmetric}]$   
 $C\text{-def}[\text{symmetric}] \ D\text{-def}[\text{symmetric}] \ E\text{-def}[\text{symmetric}] \ P2\text{-def}[\text{symmetric}] \ Q2\text{-def}[\text{symmetric}]$   
 $F\text{-def}[\text{symmetric}] \ H\text{-def}[\text{symmetric}] \ P-H2\text{-def}[\text{symmetric}] \ H2\text{-def}[\text{symmetric}]$   
 $H2\text{-UL-def}[\text{symmetric}]$   
 $H2\text{-DL-def}[\text{symmetric}] \ H2\text{-UR-def}[\text{symmetric}] \ H2\text{-DR-def}[\text{symmetric}] \ P3\text{-def}[\text{symmetric}]$   
 $S'\text{-def}[\text{symmetric}]$   
 $Q3\text{-def}[\text{symmetric}] \ P1'\text{-def}[\text{symmetric}] \ P2'\text{-def}[\text{symmetric}] \ P3'\text{-def}[\text{symmetric}]$   
 $Q1\text{-def}[\text{symmetric}]$   
 $Q2\text{-def}[\text{symmetric}] \ Q3'\text{-def}[\text{symmetric}] \ S\text{-def}[\text{symmetric}]$   
**show**  $?thesis$  **by** (*rule prod3-intro, unfold aux, insert 1.prem, auto*)  
**qed**  
**have**  $P1'$ :  $P1' \in \text{carrier-mat } m \ m$  **unfolding**  $P1'\text{-def}$  **using**  $P1 \ m$  **by** *auto*  
**moreover** **have**  $P2'$ :  $P2' \in \text{carrier-mat } m \ m$  **unfolding**  $P2'\text{-def}$  **using**  $P2 \ m$   
 $A$  **by** *auto*  
**moreover** **have**  $P3'$ :  $P3' \in \text{carrier-mat } m \ m$  **unfolding**  $P3'\text{-def}$  **using**  $P3 \ m$   
**by** *auto*  
**moreover** **have**  $P-H2$ :  $P-H2 \in \text{carrier-mat } m \ m$  **using**  $\text{reduce-column}[OF \ H \ P-H2H2] \ m$  **by** *simp*  
**moreover** **have**  $S \in \text{carrier-mat } m \ n$  **unfolding**  $S\text{-def}$  **using**  $H \ A \ S'$

**by** (*auto*, *smt* (*verit*) *C One-nat-def Suc-pred*  $\langle C \in \text{carrier-mat } (1 + (m - 1))$   
 $n \rangle \text{carrier-matD carrier-matI}$   
 $\text{dim-col-mat}(1) \text{dim-row-mat}(1) \text{index-mat-four-block } n \text{neq0-conv plus-1-eq-Suc}$   
 $\text{zero-order}(3)$ )  
**moreover have**  $Q3' \in \text{carrier-mat } n \ n$  **unfolding**  $Q3'\text{-def}$  **using**  $Q3 \ n$  **by** *auto*  
**ultimately show** *?case* **using** *Smith-final Q1 Q2* **by** *auto*  
**qed**

**corollary** *Smith-mxn-pinduct-carrier*:  $\llbracket \text{Smith-mxn-dom } A; A \in \text{carrier-mat } m \ n \rrbracket$   
 $\implies$   
 $\text{fst } (\text{Smith-mxn } A) \in \text{carrier-mat } m \ m$   
 $\wedge \text{fst } (\text{snd } (\text{Smith-mxn } A)) \in \text{carrier-mat } m \ n$   
 $\wedge \text{snd } (\text{snd } (\text{Smith-mxn } A)) \in \text{carrier-mat } n \ n$   
**using** *Smith-mxn-pinduct-carrier-ge-2 Smith-mxn-pinduct-carrier-less-2*  
**by** (*meson linorder-not-le*)

**termination proof** (*relation measure*  $(\lambda A. \text{dim-row } A)$ )  
**fix**  $A \ A1 \ A2 \ xb \ P1 \ y \ D1 \ Q1 \ C \ D \ E \ xf \ P2 \ yb \ Q2 \ F \ yc \ H \ xj \ P\text{-}H2 \ H2 \ xl \ xm \ ye \ xn$   
 $yf \ xo \ yg$   
**assume**  $1: \neg (\text{dim-row } A = 0 \vee \text{dim-col } A = 0)$  **and**  $2: \text{dim-row } A \neq 1$   
**and**  $3: \text{dim-row } A \neq 2$  **and**  $4: \text{dim-col } A \neq 1$  **and**  $5: \text{dim-col } A \neq 2$   
**and**  $6: A1 = \text{mat-of-row } (\text{Matrix.row } A \ 0)$   
**and**  $xa\text{-def}: A2 = \text{mat-of-rows } (\text{dim-col } A) \ (\text{map } (\text{Matrix.row } A) \ [1..<\text{dim-row}$   
 $A])$   
**and**  $xb\text{-def}: xb = \text{Smith-mxn } A2$  **and**  $P1\text{-}y\text{-}xb: (P1, y) = xb$   
**and**  $D1\text{-}Q1\text{-}y: (D1, Q1) = y$  **and**  $C\text{-def}: C = A1 * Q1 @_r P1 * A2 * Q1$   
**and**  $D\text{-def}: D = \text{mat-of-rows } (\text{dim-col } A) \ [\text{Matrix.row } C \ 0, \text{Matrix.row } C \ 1]$   
**and**  $E\text{-def}: E = \text{mat-of-rows } (\text{dim-col } A) \ (\text{map } (\text{Matrix.row } C) \ [2..<\text{dim-row}$   
 $A])$   
**and**  $xf: xf = \text{Smith-2xn } D$  **and**  $P2\text{-}yb\text{-}xf: (P2, yb) = xf$  **and**  $F\text{-}Q2\text{-}yb: (F, Q2)$   
 $= yb$   
**and**  $H\text{-def}: H = P2 * D * Q2 @_r E * Q2$  **and**  $xj: xj = \text{reduce-column div-op}$   
 $H$   
**and**  $P\text{-}H2\text{-}H2: (P\text{-}H2, H2) = xj$  **and**  $b4: xl = \text{split-block } H2 \ 1 \ 1$   
**and**  $b1: (xm, ye) = xl$  **and**  $b2: (xn, yf) = ye$  **and**  $b3: (xo, yg) = yf$   
**and**  $A2\text{-dom}: \text{Smith-mxn-dom } A2$   
**let**  $?m = \text{dim-row } A$   
**let**  $?n = \text{dim-col } A$   
**have**  $m: 2 < ?m$  **and**  $n: 2 < ?n$  **using**  $1 \ 2 \ 3 \ 4 \ 5 \ 6$  **by** *auto*  
**have**  $A1: A1 \in \text{carrier-mat } 1 \ (\text{dim-col } A)$  **using**  $6$  **by** *auto*  
**have**  $A2: A2 \in \text{carrier-mat } (\text{dim-row } A - 1) \ (\text{dim-col } A)$  **using**  $xa\text{-def}$  **by** *auto*  
**have**  $\text{fst } (\text{Smith-mxn } A2) \in \text{carrier-mat } (?m - 1) \ (?m - 1)$   
 $\wedge \text{fst } (\text{snd } (\text{Smith-mxn } A2)) \in \text{carrier-mat } (?m - 1) \ ?n$   
 $\wedge \text{snd } (\text{snd } (\text{Smith-mxn } A2)) \in \text{carrier-mat } ?n \ ?n$   
**by** (*rule Smith-mxn-pinduct-carrier[OF A2-dom A2]*)  
**hence**  $P1: P1 \in \text{carrier-mat } (?m - 1) \ (?m - 1)$  **and**  $D1: D1 \in \text{carrier-mat } (?m - 1)$   
 $?n$

**and**  $Q1$ :  $Q1 \in \text{carrier-mat } ?n \ ?n$  **using**  $P1\text{-}y\text{-}xb \ D1\text{-}Q1\text{-}y \ xa\text{-}def \ xb\text{-}def$  **by**  
*(metis fstI sndI)*  
**have**  $C$ :  $C \in \text{carrier-mat } ?m \ ?n$  **unfolding**  $C\text{-}def$  **using**  $A1 \ Q1 \ P1 \ A2 \ Q1$   
**by** *(smt (verit) 1 Suc-pred card-num-simps(30) carrier-append-rows mult-carrier-mat  
neq0-conv plus-1-eq-Suc)*  
**have**  $D$ :  $D \in \text{carrier-mat } 2 \ ?n$  **unfolding**  $D\text{-}def$  **using**  $C$  **by** *auto*  
**have**  $E$ :  $E \in \text{carrier-mat } (?m-2) \ ?n$  **unfolding**  $E\text{-}def$  **using**  $C \ m$  **by** *auto*  
**have**  $P2FQ2$ :  $(P2, F, Q2) = \text{Smith-}2xn \ D$  **using**  $F\text{-}Q2\text{-}yb \ P2\text{-}yb\text{-}xf \ xf$  **by** *blast*  
**have**  $P2$ :  $P2 \in \text{carrier-mat } 2 \ 2$  **and**  $F$ :  $F \in \text{carrier-mat } 2 \ ?n$  **and**  $Q2$ :  $Q2 \in$   
 $\text{carrier-mat } ?n \ ?n$   
**using**  $is\text{-}SNF\text{-}Smith\text{-}2xn[OF \ D] \ D \ P2FQ2$  **unfolding**  $is\text{-}SNF\text{-}def$  **by** *auto*  
**have**  $H \in \text{carrier-mat } (2 + (?m-2)) \ ?n$   
**by** *(unfold H-def, rule carrier-append-rows, insert D Q2 P2 E, auto)*  
**hence**  $H$ :  $H \in \text{carrier-mat } ?m \ ?n$  **using**  $m$  **by** *auto*  
**have**  $H2$ :  $H2 \in \text{carrier-mat } (dim\text{-}row \ H) \ (dim\text{-}col \ H)$   
**and**  $P\text{-}H2$ :  $P\text{-}H2 \in \text{carrier-mat } (dim\text{-}row \ A) \ (dim\text{-}row \ A)$   
**using**  $reduce\text{-}column[OF \ H \ xj[unfolded \ P\text{-}H2\text{-}H2[symmetric]]] \ m \ H$  **by** *auto*  
**have**  $dim\text{-}row \ yg < dim\text{-}row \ H2$   
**by** *(rule split-block4-decreases-dim-row, insert b1 b2 b3 b4 m n H H2, auto)*  
**also** **have**  $\dots = dim\text{-}row \ A$  **using**  $H2 \ H$  **by** *auto*  
**finally** **show**  $(yg, A) \in \text{measure } dim\text{-}row$  **unfolding**  $in\text{-}measure$  .  
**qed** *(auto)*

**lemma**  $is\text{-}SNF\text{-}Smith\text{-}m \times n\text{-}less\text{-}2$ :

**assumes**  $A$ :  $A \in \text{carrier-mat } m \ n$  **and**  $mn$ :  $n \leq 2 \vee m \leq 2$

**shows**  $is\text{-}SNF \ A \ (\text{Smith-}m \times n \ A)$

**proof** –

**show**  $?thesis$

**proof** *(cases dim-row A = 0  $\vee$  dim-col A = 0)*

**case**  $True$

**have**  $\text{Smith-}m \times n \ A = (1_m \ (dim\text{-}row \ A), A, 1_m \ (dim\text{-}col \ A))$

**using**  $\text{Smith-}m \times n.\text{simps} \ True$  **by** *auto*

**thus**  $?thesis$  **using**  $A \ True$  **unfolding**  $is\text{-}SNF\text{-}def$  **by** *auto*

**next**

**case**  $False$  **note**  $1 = False$

**show**  $?thesis$

**proof** *(cases dim-row A = 1)*

**case**  $True$

**have**  $\text{Smith-}m \times n \ A = (1_m \ 1, \text{Smith-}1 \times n \ A)$

**using**  $\text{Smith-}m \times n.\text{simps} \ True \ 1$  **by** *auto*

**then** **show**  $?thesis$  **using**  $\text{Smith-}1 \times n\text{-}works$  **by** *(metis True carrier-mat-triv  
surj-pair)*

**next**

**case**  $False$  **note**  $2 = False$

**then** **show**  $?thesis$

**proof** *(cases dim-row A = 2)*

**case**  $True$

**hence**  $A'$ :  $A \in \text{carrier-mat } 2 \ n$  **using**  $A$  **by** *auto*

```

      have Smith-mxn A = Smith-2xn A using Smith-mxn.simps True 1 2 by
    auto
    then show ?thesis using is-SNF-Smith-2xn[OF A] A by auto
  next
  case False note 3 = False
  show ?thesis
  proof (cases dim-col A = 1)
    case True
    hence A': A ∈ carrier-mat m 1 using A by auto
    have Smith-mxn A = (let (P,S) = Smith-nx1 A in (P,S,1m 1))
      using Smith-mxn.simps True 1 2 3 by auto
    then show ?thesis using Smith-nx1-works[OF A] A by (auto simp add:
  case-prod-beta)
  next
  case False
  hence dim-col A = 2 using 1 2 3 mn A by auto
  hence A': A ∈ carrier-mat m 2 using A by auto
  hence Smith-mxn A = Smith-nx2 A
    using Smith-mxn.simps 1 2 3 False by auto
  then show ?thesis using is-SNF-Smith-nx2[OF A] A by force
  qed
  qed
  qed
  qed
  qed

```

```

lemma is-SNF-Smith-mxn-ge-2:
  assumes A: A ∈ carrier-mat m n and m: m > 2 and n: n > 2
  shows is-SNF A (Smith-mxn A)
  using A m n
  proof (induct A arbitrary: m n rule: Smith-mxn.induct)
    case (1 A)
    note A = 1.prem1
    note m = 1.prem2
    note n = 1.prem3
    have A-dim-not0: ¬ (dim-row A = 0 ∨ dim-col A = 0) and A-dim-row-not1:
  dim-row A ≠ 1
      and A-dim-row-not2: dim-row A ≠ 2 and A-dim-col-not1: dim-col A ≠ 1
      and A-dim-col-not2: dim-col A ≠ 2
    using A m n by auto
    note A-dim-intro = A-dim-not0 A-dim-row-not1 A-dim-row-not2 A-dim-col-not1
  A-dim-col-not2
    define A1 where A1 = mat-of-row (Matrix.row A 0)
    define A2 where A2 = mat-of-rows (dim-col A) [Matrix.row A i. i ← [1..<dim-row
  A]]
    obtain P1 D1 Q1 where P1D1Q1: (P1,D1,Q1) = Smith-mxn A2 by (metis
  prod-cases3)
    define C where C = (A1*Q1) @r (P1*A2*Q1)
  
```

```

define  $D$  where  $D = \text{mat-of-rows } (\text{dim-col } A) [\text{Matrix.row } C \ 0, \text{Matrix.row } C \ 1]$ 
define  $E$  where  $E = \text{mat-of-rows } (\text{dim-col } A) [\text{Matrix.row } C \ i. \ i \leftarrow [2..<\text{dim-row } A]]$ 
obtain  $P2 \ F \ Q2$  where  $P2FQ2: (P2, F, Q2) = \text{Smith-2xn } D$  by (metis prod-cases3)
define  $H$  where  $H = (P2 * D * Q2) @_r (E * Q2)$ 
obtain  $P\text{-}H2 \ H2$  where  $P\text{-}H2H2: (P\text{-}H2, H2) = \text{reduce-column div-op } H$  by
(metis surj-pair)
obtain  $H2\text{-}UL \ H2\text{-}UR \ H2\text{-}DL \ H2\text{-}DR$  where  $\text{split-}H2: (H2\text{-}UL, H2\text{-}UR, H2\text{-}DL, H2\text{-}DR) = \text{split-block } H2 \ 1 \ 1$ 
by (metis split-block-def)
obtain  $P3 \ S' \ Q3$  where  $P3S'Q3: (P3, S', Q3) = \text{Smith-mxn } H2\text{-}DR$  by (metis prod-cases3)
define  $S$  where  $S = \text{four-block-mat } (\text{Matrix.mat } 1 \ 1 \ (\lambda(a, b). \ H \ \$\$ \ (0, 0)))$ 
 $(0_m \ 1 \ (\text{dim-col } A - 1))$ 
 $(0_m \ (\text{dim-row } A - 1) \ 1) \ S'$ 
define  $P1'$  where  $P1' = \text{four-block-mat } (1_m \ 1) \ (0_m \ 1 \ (\text{dim-row } A - 1)) \ (0_m$ 
 $(\text{dim-row } A - 1) \ 1) \ P1$ 
define  $P2'$  where  $P2' = \text{four-block-mat } P2 \ (0_m \ 2 \ (\text{dim-row } A - 2)) \ (0_m$ 
 $(\text{dim-row } A - 2) \ 2) \ (1_m \ (\text{dim-row } A - 2))$ 
define  $P3'$  where  $P3' = \text{four-block-mat } (1_m \ 1) \ (0_m \ 1 \ (\text{dim-row } A - 1)) \ (0_m$ 
 $(\text{dim-row } A - 1) \ 1) \ P3$ 
define  $Q3'$  where  $Q3' = \text{four-block-mat } (1_m \ 1) \ (0_m \ 1 \ (\text{dim-col } A - 1)) \ (0_m$ 
 $(\text{dim-col } A - 1) \ 1) \ Q3$ 
have Smith-final:  $\text{Smith-mxn } A = (P3' * P\text{-}H2 * P2' * P1', S, Q1 * Q2 * Q3')$ 
proof -
have  $P1\text{-def}$ :  $P1 = \text{fst } (\text{Smith-mxn } A2)$  and  $D1\text{-def}$ :  $D1 = \text{fst } (\text{snd } (\text{Smith-mxn } A2))$ 
and  $Q1\text{-def}$ :  $Q1 = \text{snd } (\text{snd } (\text{Smith-mxn } A2))$  using  $P1D1Q1$  by (metis fstI sndI)+
have  $P2\text{-def}$ :  $P2 = \text{fst } (\text{Smith-2xn } D)$  and  $F\text{-def}$ :  $F = \text{fst } (\text{snd } (\text{Smith-2xn } D))$ 
and  $Q2\text{-def}$ :  $Q2 = \text{snd } (\text{snd } (\text{Smith-2xn } D))$  using  $P2FQ2$  by (metis fstI sndI)+
have  $P\text{-}H2\text{-def}$ :  $P\text{-}H2 = \text{fst } (\text{reduce-column div-op } H)$ 
and  $H2\text{-def}$ :  $H2 = \text{snd } (\text{reduce-column div-op } H)$ 
using  $P\text{-}H2H2$  by (metis fstI sndI)+
have  $H2\text{-}UL\text{-def}$ :  $H2\text{-}UL = \text{fst } (\text{split-block } H2 \ 1 \ 1)$ 
and  $H2\text{-}UR\text{-def}$ :  $H2\text{-}UR = \text{fst } (\text{snd } (\text{split-block } H2 \ 1 \ 1))$ 
and  $H2\text{-}DL\text{-def}$ :  $H2\text{-}DL = \text{fst } (\text{snd } (\text{snd } (\text{split-block } H2 \ 1 \ 1)))$ 
and  $H2\text{-}DR\text{-def}$ :  $H2\text{-}DR = \text{snd } (\text{snd } (\text{snd } (\text{split-block } H2 \ 1 \ 1)))$ 
using  $\text{split-}H2$  by (metis fstI sndI)+
have  $P3\text{-def}$ :  $P3 = \text{fst } (\text{Smith-mxn } H2\text{-}DR)$  and  $S'\text{-def}$ :  $S' = \text{fst } (\text{snd } (\text{Smith-mxn } H2\text{-}DR))$ 
and  $Q3\text{-def}$ :  $Q3 = (\text{snd } (\text{snd } (\text{Smith-mxn } H2\text{-}DR)))$  using  $P3S'Q3$  by (metis fstI sndI)+
note aux =  $\text{Smith-mxn.simps}[of \ A]$  Let-def split-beta
 $A1\text{-def}[\text{symmetric}] \ A2\text{-def}[\text{symmetric}] \ P1\text{-def}[\text{symmetric}] \ D1\text{-def}[\text{symmetric}]$ 
 $Q1\text{-def}[\text{symmetric}]$ 

```

$C$ -def[symmetric]  $D$ -def[symmetric]  $E$ -def[symmetric]  $P2$ -def[symmetric]  $Q2$ -def[symmetric]  
 $F$ -def[symmetric]  $H$ -def[symmetric]  $P$ - $H2$ -def[symmetric]  $H2$ -def[symmetric]  
 $H2$ - $UL$ -def[symmetric]  
 $H2$ - $DL$ -def[symmetric]  $H2$ - $UR$ -def[symmetric]  $H2$ - $DR$ -def[symmetric]  $P3$ -def[symmetric]  
 $S'$ -def[symmetric]  
 $Q3$ -def[symmetric]  $P1'$ -def[symmetric]  $P2'$ -def[symmetric]  $P3'$ -def[symmetric]  
 $Q1$ -def[symmetric]  
 $Q2$ -def[symmetric]  $Q3'$ -def[symmetric]  $S$ -def[symmetric]  
**show** ?thesis **by** (rule prod3-intro, unfold aux, insert 1.premis, auto)  
**qed**  
**show** ?case  
**proof** (unfold Smith-final, rule is-SNF-intro)  
**have**  $A1$ [simp]:  $A1 \in \text{carrier-mat } 1 \ n$  **unfolding**  $A1$ -def **using**  $A$  **by** auto  
**have**  $A2$ [simp]:  $A2 \in \text{carrier-mat } (m-1) \ n$  **unfolding**  $A2$ -def **using**  $A$  **by**  
auto  
**have** is-SNF- $A2$ : is-SNF  $A2$  (Smith-mxn  $A2$ )  
**proof** (cases  $n \leq 2 \vee m - 1 \leq 2$ )  
**case** True  
**then show** ?thesis **using** is-SNF-Smith-mxn-less-2[OF  $A2$ ] **by** simp  
**next**  
**case** False  
**hence**  $n1$ :  $2 < n$  **and**  $m1$ :  $2 < m - 1$  **by** auto  
**show** ?thesis **by** (rule 1.hyps(1)[OF  $A$ -dim-intro  $A1$ -def  $A2$ -def  $A2 \ m1 \ n1$ ])  
  
**qed**  
**have**  $P1$ [simp]:  $P1 \in \text{carrier-mat } (m-1) \ (m-1)$   
**and** inv- $P1$ : invertible-mat  $P1$   
**and**  $Q1$ :  $Q1 \in \text{carrier-mat } n \ n$  **and** inv- $Q1$ : invertible-mat  $Q1$   
**and** SNF- $P1A2Q1$ : Smith-normal-form-mat ( $P1 * A2 * Q1$ )  
**using** is-SNF- $A2 \ P1D1Q1 \ A2 \ A \ n \ m$  **unfolding** is-SNF-def **by** auto  
**have**  $C$ [simp]:  $C \in \text{carrier-mat } m \ n$  **unfolding**  $C$ -def **using**  $P1 \ Q1 \ A1 \ A2 \ m$   
**by** (smt (verit) 1(3)  $A$ -dim-not0 Suc-pred card-num-simps(30) carrier-append-rows  
carrier-matD  
carrier-mat-triv index-mult-mat(2,3) neq0-conv plus-1-eq-Suc)  
**have**  $D$ [simp]:  $D \in \text{carrier-mat } 2 \ n$  **unfolding**  $D$ -def **using**  $A \ m$  **by** auto  
**have** is-SNF- $D$ : is-SNF  $D$  (Smith-2xn  $D$ ) **by** (rule is-SNF-Smith-2xn[OF  $D$ ])  
**hence**  $P2$ [simp]:  $P2 \in \text{carrier-mat } 2 \ 2$  **and** inv- $P2$ : invertible-mat  $P2$   
**and**  $Q2$ [simp]:  $Q2 \in \text{carrier-mat } n \ n$  **and** inv- $Q2$ : invertible-mat  $Q2$   
**and**  $F$ [simp]:  $F \in \text{carrier-mat } 2 \ n$  **and**  $F$ - $P2DQ2$ :  $F = P2 * D * Q2$   
**and** SNF- $F$ : Smith-normal-form-mat  $F$   
**using**  $P2FQ2 \ D$ -def  $A$  **unfolding** is-SNF-def **by** auto  
**have**  $E$ [simp]:  $E \in \text{carrier-mat } (m-2) \ n$  **unfolding**  $E$ -def **using**  $A$  **by** auto  
**have**  $H$ -aux:  $H \in \text{carrier-mat } (2 + (m-2)) \ n$  **unfolding**  $H$ -def  
**by** (rule carrier-append-rows, insert  $P2 \ D \ Q2 \ E \ F$ - $P2DQ2 \ F \ A \ m \ n$  mult-carrier-mat,  
force)  
**hence**  $H$ [simp]:  $H \in \text{carrier-mat } m \ n$  **using**  $m$  **by** auto  
**have**  $H2$ [simp]:  $H2 \in \text{carrier-mat } m \ n$  **using**  $m \ H \ P$ - $H2H2 \ A$  reduce-column  
**by** blast  
**have**  $H2$ - $DR$ [simp]:  $H2$ - $DR \in \text{carrier-mat } (m - 1) \ (n - 1)$

**by** (rule *split-block(4)*[*OF split-H2[symmetric]*], insert *H2 m n A H*, auto, insert *H2*, blast+)

**have**  $P1'[simp]$ :  $P1' \in \text{carrier-mat } m \ m$  **unfolding**  $P1'$ -def **using**  $P1 \ m$  **by** auto

**have**  $P2'[simp]$ :  $P2' \in \text{carrier-mat } m \ m$  **unfolding**  $P2'$ -def **using**  $P2 \ m \ A \ m$  **by** (metis (no-types, lifting) *H H-aux carrier-matD carrier-mat-triv index-mat-four-block(2,3) index-one-mat(2,3)*)

**have** *is-SNF-H2-DR*: *is-SNF H2-DR (Smith-mxn H2-DR)*

**proof** (cases  $2 < m - 1 \wedge 2 < n - 1$ )

**case** *True*

**hence**  $m1: 2 < m - 1$  **and**  $n1: 2 < n - 1$  **by** *simp+*

**show** *?thesis*

**by** (rule *1.hyps(2)*[*OF A-dim-intro A1-def A2-def P1D1Q1 - - C-def D-def E-def P2FQ2 - - H-def P-H2H2 - split-H2 - - - H2-DR m1 n1*], auto)

**next**

**case** *False*

**hence**  $m-1 \leq 2 \vee n-1 \leq 2$  **by** auto

**then show** *?thesis using H2-DR is-SNF-Smith-mxn-less-2* **by** blast

**qed**

**hence**  $P3[simp]$ :  $P3 \in \text{carrier-mat } (m-1) \ (m-1)$  **and** *inv-P3*: *invertible-mat P3*

**and**  $Q3[simp]$ :  $Q3 \in \text{carrier-mat } (n-1) \ (n-1)$  **and** *inv-Q3*: *invertible-mat Q3*

**and**  $S'[simp]$ :  $S' \in \text{carrier-mat } (m-1) \ (n-1)$  **and**  $S'-P3H2-DRQ3$ :  $S' = P3 * H2-DR * Q3$

**and** *SNF-S'*: *Smith-normal-form-mat S'*

**using**  $A \ m \ n \ H2-DR \ P3S'Q3$  **unfolding** *is-SNF-def* **by** auto

**have**  $P3'[simp]$ :  $P3' \in \text{carrier-mat } m \ m$  **unfolding**  $P3'$ -def **using**  $P3 \ m$  **by** auto

**have**  $P-H2[simp]$ :  $P-H2 \in \text{carrier-mat } m \ m$  **using** *reduce-column*[*OF H P-H2H2*]  $m$  **by** *simp*

**have**  $S[simp]$ :  $S \in \text{carrier-mat } m \ n$  **unfolding**  $S$ -def **using**  $H \ A \ S'$

**by** (smt (verit) *A-dim-intro(1) One-nat-def Suc-pred carrier-matD carrier-matI dim-col-mat(1) dim-row-mat(1) index-mat-four-block(2,3) nat-neq-iff not-less-zero plus-1-eq-Suc*)

**have**  $Q3'[simp]$ :  $Q3' \in \text{carrier-mat } n \ n$  **unfolding**  $Q3'$ -def **using**  $Q3 \ n$  **by** auto

**show** *P-final-carrier*:  $P3' * P-H2 * P2' * P1' \in \text{carrier-mat } (\text{dim-row } A)$  (*dim-row A*)

**using**  $P3' \ P-H2 \ P2' \ P1' \ A$  **by** (metis *carrier-matD carrier-matI index-mult-mat(2,3)*)

**show** *Q-final-carrier*:  $Q1 * Q2 * Q3' \in \text{carrier-mat } (\text{dim-col } A)$  (*dim-col A*)

**using**  $Q1 \ Q2 \ Q3' \ A$  **by** (metis *carrier-matD carrier-matI index-mult-mat(2,3)*)

**have** *inv-P1'*: *invertible-mat P1'* **unfolding**  $P1'$ -def

**by** (rule *invertible-mat-four-block-mat-lower-right*[*OF - inv-P1*], insert  $A \ P1$ , auto)

**have** *inv-P2'*: *invertible-mat P2'* **unfolding**  $P2'$ -def

**by** (rule *invertible-mat-four-block-mat-lower-right-id*[*OF - - - - inv-P2*], insert

```

A m, auto)
  have inv-P3': invertible-mat P3' unfolding P3'-def
    by (rule invertible-mat-four-block-mat-lower-right[OF - inv-P3], insert A P3,
auto)
  have inv-P-H2: invertible-mat P-H2 using reduce-column[OF H P-H2H2] m
by simp
  show invertible-mat (P3' * P-H2 * P2' * P1') using inv-P1' inv-P2' inv-P3'
inv-P-H2
    by (meson P1' P2' P3' P-H2 invertible-mult-JNF mult-carrier-mat)
  have inv-Q3': invertible-mat Q3' unfolding Q3'-def
    by (rule invertible-mat-four-block-mat-lower-right[OF - inv-Q3], insert A Q3,
auto)
  show invertible-mat (Q1 * Q2 * Q3') using inv-Q1 inv-Q2 inv-Q3'
    by (meson Q1 Q2 Q3' invertible-mult-JNF mult-carrier-mat)
  have A-A1-A2: A = A1 @r A2 unfolding append-cols-def
  proof (rule eq-matI)
    have A1-A2': A1 @r A2 ∈ carrier-mat (1+(m-1)) n by (rule carrier-append-rows[OF
A1 A2])
    hence A1-A2: A1 @r A2 ∈ carrier-mat m n using m by simp
    thus dim-row A = dim-row (A1 @r A2) and dim-col A = dim-col (A1 @r
A2) using A by auto
    fix i j assume i: i < dim-row (A1 @r A2) and j: j < dim-col (A1 @r A2)
    show A $$ (i, j) = (A1 @r A2) $$ (i, j)
    proof (cases i=0)
      case True
        have (A1 @r A2) $$ (i, j) = (A1 @r A2) $$ (0, j) using True by simp
        also have ... = four-block-mat A1 (0m (dim-row A1) 0) A2 (0m (dim-row
A2) 0) $$ (0,j)
          unfolding append-rows-def ..
        also have ... = A1 $$ (0,j) using A1 A1-A2 j by auto
        also have ... = A $$ (0,j) unfolding A1-def using A1-A2 A i j by auto
        finally show ?thesis using True by simp
      case False
        let ?xs = (map (Matrix.row A) [1..<dim-row A])
        have (A1 @r A2) $$ (i, j) = four-block-mat A1 (0m (dim-row A1) 0) A2
(0m (dim-row A2) 0) $$ (i,j)
          unfolding append-rows-def ..
        also have ... = A2 $$ (i-1,j) using A1 A1-A2' A2 False i j by auto
        also have ... = mat-of-rows (dim-col A) ?xs $$ (i - 1, j) by (simp add:
A2-def)
        also have ... = ?xs ! (i-1) $v j by (rule mat-of-rows-index, insert i False
A j m A1-A2, auto)
        also have ... = A $$ (i,j) using False A A1-A2 i j by auto
        finally show ?thesis ..
    qed
  qed
  have C-eq: C = P1' * A * Q1
  proof -

```



```

have aux: (A1 @r A2) * Q1 = ((A1 * Q1) @r (A2*Q1))
  by (rule append-rows-mult-right, insert A1 A2 Q1, auto)
have P1' * A * Q1 = P1' * (A1 @r A2) * Q1 using A-A1-A2 by simp
  also have ... = P1' * ((A1 @r A2) * Q1) using A A-A1-A2 P1' Q1
assoc-mult-mat by blast
  also have ... = P1' * ((A1 * Q1) @r (A2*Q1)) by (simp add: aux)
  also have ... = (A1 * Q1) @r (P1 * (A2 * Q1))
    by (rule append-rows-mult-left-id, insert A1 Q1 A2 P1 P1'-def A, auto)
  also have ... = (A1 * Q1) @r (P1 * A2 * Q1) using A2 P1 Q1 by auto
  finally show ?thesis unfolding C-def ..
qed
have C-D-E: C = D @r E
proof -
  let ?xs = [Matrix.row C 0, Matrix.row C 1]
  let ?ys = (map (Matrix.row C) [0..<2])
  have xs-ys: ?xs = ?ys by (simp add: upt-conv-Cons)
  have D-rw: D = mat-of-rows (dim-col C) (map (Matrix.row C) [0..<2])
    unfolding D-def xs-ys using A C by (metis carrier-matD(2))
  have d1: dim-col A = dim-col C using A C by blast
  have d2: dim-row A = dim-row C using A C by blast
  show ?thesis unfolding D-rw E-def d1 d2 by (rule append-rows-split, insert
m C A d2, auto)
qed
have H-eq: H = P2' * P1' * A * Q1 * Q2
proof -
  have aux: ((P2 * D) @r E) = P2' * (D @r E)
    by (rule append-rows-mult-left-id2[symmetric, OF D E - P2], insert P2'-def
A, auto)
  have H = P2 * D * Q2 @r E * Q2 by (simp add: H-def)
  also have ... = (P2 * D @r E) * Q2
    by (rule append-rows-mult-right[symmetric, OF mult-carrier-mat[OF P2 D]
E Q2])
  also have ... = P2' * (D @r E) * Q2 by (simp add: aux)
  also have ... = P2' * C * Q2 unfolding C-D-E by simp
  also have ... = P2' * (P1' * A * Q1) * Q2 unfolding C-eq by simp
  also have ... = P2' * P1' * A * Q1 * Q2
    by (smt (verit) A P1' P2' Q1 <P2' * C * Q2 = P2' * (P1' * A * Q1) *
Q2> assoc-mult-mat mult-carrier-mat)
  finally show ?thesis .
qed
have P-H2-H-H2: P-H2 * H = H2 using reduce-column[OF H P-H2H2] m by
auto
  hence H2-eq: H2 = P-H2 * P2' * P1' * A * Q1 * Q2 unfolding H-eq
    by (smt (verit, ccfv-threshold) P1' P1'-def P2' P2'-def P-H2 P-final-carrier
Q1 Q2 Q-final-carrier assoc-mult-mat
carrier-matD carrier-mat-triv index-mult-mat(2,3))
  have H2-as-four-block-mat: H2 = four-block-mat H2-UL H2-UR H2-DL H2-DR

using split-H2 by (metis (no-types, lifting) H2 P1' P1'-def Q3' Q3'-def

```

```

carrier-matD
  index-mat-four-block(2) index-one-mat(2) split-block(5)
  have H2-UL: H2-UL ∈ carrier-mat 1 1
    by (rule split-block(1)[OF split-H2[symmetric], of m-1 n-1], insert H2 A m
n, auto, insert H2, blast+)
  have H2-UR: H2-UR ∈ carrier-mat 1 (n-1)
    by (rule split-block(2)[OF split-H2[symmetric], of m-1], insert H2 A m n,
auto, insert H2, blast+)
  have H2-DL: H2-DL ∈ carrier-mat (m-1) 1
    by (rule split-block(3)[OF split-H2[symmetric], of - n-1], insert H2 A m n,
auto, insert H2, blast+)
  have H2-DR: H2-DR ∈ carrier-mat (m-1) (n-1)
    by (rule split-block(4)[OF split-H2[symmetric], of - n-1], insert H2 A m n,
auto, insert H2, blast+)
  have H-ij-F-ij: H$(i,j) = F $(i,j) if i: i<2 and j: j<n for i j
  proof -
    have H$(i,j) = (if i < dim-row (P2*D*Q2) then (P2*D*Q2) $(i,j) else
(E*Q2) $(i-2,j))
    proof (unfold H-def, rule append-rows-nth)
      show P2 * D * Q2 ∈ carrier-mat 2 n using F F-P2DQ2 by blast
      show E * Q2 ∈ carrier-mat (m-2) n using E Q2 using mult-carrier-mat
by blast
    qed (insert m j i, auto)
    also have ... = F $(i,j) using F F-P2DQ2 i by auto
    finally show ?thesis .
  qed
  have isDiagonal-F: isDiagonal-mat F
    using is-SNF-D P2FQ2 unfolding is-SNF-def Smith-normal-form-mat-def
by auto
  have H-0j-0: H $(0,j) = 0 if j: j∈{1..<n} for j
  proof -
    have H $(0,j) = F $(0,j) using H-ij-F-ij j by auto
    also have ... = 0 using isDiagonal-F unfolding isDiagonal-mat-def using
F j by auto
    finally show ?thesis .
  qed
  have H2-0j: H2 $(0,j) = H $(0,j) if j: j<n for j
    by (rule reduce-column-preserves2[OF H P-H2H2 - - - j], insert m, auto)
  have H2-UR-0: H2-UR = (0_m 1 (n-1))
  proof (rule eq-matI)
    show dim-row H2-UR = dim-row (0_m 1 (n-1)) and dim-col H2-UR =
dim-col (0_m 1 (n-1))
    using H2-UR by auto
    fix i j assume i: i < dim-row (0_m 1 (n-1)) and j: j < dim-col (0_m 1 (n
- 1))
    have i0: i=0 using i by auto
    have 1: 0 < dim-row H2-UL + dim-row H2-DR using i H2-UL H2-DR by
auto
    have 2: j+1 < dim-col H2-UL + dim-col H2-DR using j H2-UL H2-DR by

```

*auto*  
**have**  $H2-UR$   $\$ \$ (i, j) = H2$   $\$ \$ (0, j+1)$   
**unfolding**  $i0$   $H2-as-four-block-mat$  **using**  $index-mat-four-block(1)[OF\ 1\ 2]$   
 $H2-UL$  **by** *auto*  
**also have**  $\dots = H$   $\$ \$ (0, j+1)$  **by** (*rule*  $H2-0j$ , *insert*  $j$ , *auto*)  
**also have**  $\dots = 0$  **using**  $H-0j-0$   $j$  **by** *auto*  
**finally show**  $H2-UR$   $\$ \$ (i, j) = 0_m$   $1$   $(n - 1)$   $\$ \$ (i, j)$  **using**  $i\ j$  **by** *auto*  
**qed**  
**have**  $H2-UL00-H00$ :  $H2-UL$   $\$ \$ (0, 0) = H$   $\$ \$ (0, 0)$   
**using**  $H2-UL$   $H2-as-four-block-mat$   $H2-0j\ n$  **by** *fastforce*  
**have**  $F00-dvd-Dij$ :  $F$   $\$ \$ (0, 0)$   $dvd$   $D$   $\$ \$ (i, j)$  **if**  $i: i < 2$  **and**  $j: j < n$  **for**  $i\ j$   
**by** (*rule*  $S00-dvd-all-A[OF\ D\ P2\ Q2\ inv-P2\ inv-Q2\ F-P2DQ2\ SNF-F\ i\ j]$ )  
  
**have**  $D10-dvd-Eij$ :  $D$   $\$ \$ (1, 0)$   $dvd$   $E$   $\$ \$ (i, j)$  **if**  $i: i < m - 2$  **and**  $j: j < n$  **for**  $i\ j$   
**proof** –  
**have**  $D$   $\$ \$ (1, 0) = C$   $\$ \$ (1, 0)$   
**by** (*smt* (*verit*)  $C\ C-D-E\ F\ F-P2DQ2\ H\ H-def\ One-nat-def\ Suc-lessD$   
 $add-diff-cancel-right'$   $append-rows-def$   
 $arith-special(3)$   $carrier-matD$   $index-mat-four-block$   $index-mult-mat(2)$   
 $lessI\ m\ n\ plus-1-eq-Suc$ )  
**also have**  $\dots = (P1 * A2 * Q1)$   $\$ \$ (0, 0)$   
**by** (*smt* (*verit*)  $1(3)$   $A1\ A2\ A-A1-A2\ A-dim-not0\ P1\ Q1\ Suc-eq-plus1$   
 $Suc-lessD$   $add-diff-cancel-right'$   
 $append-rows-def$   $arith-special(3)$   $card-num-simps(30)$   $carrier-matD$   $in-$   
 $dex-mat-four-block$   
 $index-mult-mat(2, 3)$   $less-not-refl2$   $local.C-def\ m\ neq0-conv$ )  
**also have**  $\dots dvd$   $(P1 * A2 * Q1)$   $\$ \$ (i+1, j)$   
**by** (*rule*  $SNF-first-divides-all[OF\ SNF-P1A2Q1\ -\ -\ j]$ , *insert*  $P1\ A2\ Q1\ i\ A$ ,  
*auto*)  
**also have**  $\dots = C$   $\$ \$ (i+2, j)$  **unfolding**  $C-def$  **using**  $append-rows-nth$   
**by** (*smt* (*verit*, *ccfv-threshold*)  $A\ A1\ A2\ A-A1-A2\ P1\ Q1\ Suc-lessD$   
 $add-Suc-right$   $add-diff-cancel-left'$   $append-rows-def$   
 $arith-special(3)$   $carrier-matD$   $index-mat-four-block$   $index-mult-mat(2, 3)$   
 $j\ less-diff-conv$   
 $not-add-less2$   $plus-1-eq-Suc$   $that(1)$ )  
**also have**  $\dots = E$   $\$ \$ (i, j)$   
**by** (*smt* (*verit*)  $C\ C-D-E\ D$   $add-diff-cancel-right'$   $append-rows-def$   $car-$   
 $rier-matD$   $index-mat-four-block\ j\ i$   
 $less-diff-conv$   $not-add-less2$ )  
**finally show** *?thesis* .  
**qed**  
**have**  $F00-H00$ :  $F$   $\$ \$ (0, 0) = H$   $\$ \$ (0, 0)$  **using**  $H-ij-F-ij\ n$  **by** *auto*  
**have**  $F00-dvd-Eij$ :  $F$   $\$ \$ (0, 0)$   $dvd$   $E$   $\$ \$ (i, j)$  **if**  $i: i < m - 2$  **and**  $j: j < n$  **for**  $i\ j$   
**by** (*metis* (*no-types*, *lifting*)  $A\ A-dim-not0\ D10-dvd-Eij\ F00-dvd-Dij$   $arith-special(3)$   
 $carrier-matD(2)$   
 $dvd-trans\ j\ lessI\ neq0-conv\ plus-1-eq-Suc\ i$ )  
**have**  $F00-dvd-EQ2ij$ :  $F$   $\$ \$ (0, 0)$   $dvd$   $(E * Q2)$   $\$ \$ (i, j)$  **if**  $i: i < m - 2$  **and**  $j: j < n$   
**for**  $i\ j$   
**using**  $dvd-elements-mult-matrix-right[OF\ E\ Q2]$   $F00-dvd-Eij\ i\ j$  **by** *auto*

```

have H00-dvd-all: H $$ (0, 0) dvd H $$ (i, j) if i: i<m and j: j<n for i j
proof (cases i<2)
  case True
  then show ?thesis by (metis F F00-H00 H-ij-F-ij SNF-F SNF-first-divides-all
j)
next
  case False
  have F $$ (0, 0) dvd (E*Q2) $$ (i-2,j) by (rule F00-dvd-EQ2ij, insert False
i j, auto)
  moreover have H $$ (i, j) = (E*Q2) $$ (i-2,j)
    by (smt (verit) C C-D-E D F F-P2DQ2 False H-def append-rows-def
carrier-matD i
index-mat-four-block index-mult-mat(2) j)
  ultimately show ?thesis using F00-H00 by simp
qed
have H-00-dvd-H-i0: H $$ (0, 0) dvd H $$ (i, 0) if i: i<m for i
  using H00-dvd-all[OF i] n by auto
have H2-DL-0: H2-DL = (0m (m - 1) 1)
proof (rule eq-matI)
  show dim-row (H2-DL) = dim-row (0m (m - 1) 1)
  and dim-col (H2-DL) = dim-col (0m (m - 1) 1) using P3 H2-DL A by
auto
  fix i j assume i: i < dim-row (0m (m - 1) 1) and j: j < dim-col (0m (m
- 1) 1)
  have j0: j=0 using j by auto
  have (H2-DL) $$ (i, j) = H2 $$ (i+1,0)
    using H2-UR H2-UR-0 n j0 H2 H2-UL H2-as-four-block-mat i by auto
  also have ... = 0
  proof (cases i=0)
    case True
    have H2 $$ (1,0) = H $$ (1,0) by (rule reduce-column-preserves2[OF H
P-H2H2], insert m n, auto)
    also have ... = F $$ (1,0) by (rule H-ij-F-ij, insert n, auto)
    also have ... = 0 using isDiagonal-F F n unfolding isDiagonal-mat-def
by auto
  finally show ?thesis by (simp add: True)
  next
  case False
  show ?thesis
  proof (rule reduce-column-works(1)[OF H P-H2H2])
    show H $$ (0, 0) dvd H $$ (i + 1, 0) using H-00-dvd-H-i0 False i by
simp
    show  $\forall j \in \{1..<n\}. H \ \$\$ (0, j) = 0$  using H-0j-0 by auto
    show  $i + 1 \in \{2..<m\}$  using i False by auto
  qed (insert m n id, auto)
  qed
  finally show (H2-DL) $$ (i, j) = 0m (m - 1) 1 $$ (i, j) using i j j0 by
auto
  qed

```

**have**  $P3' * H2 = \text{four-block-mat } H2\text{-UL } H2\text{-UR } (P3 * H2\text{-DL}) (P3 * H2\text{-DR})$   
**proof** –  
**have**  $P3' * H2 = \text{four-block-mat}$   
 $(1_m \ 1 * H2\text{-UL} + 0_m \ 1 (\text{dim-row } A - 1) * H2\text{-DL}) (1_m \ 1 * H2\text{-UR} + 0_m \ 1$   
 $(\text{dim-row } A - 1) * H2\text{-DR})$   
 $(0_m (\text{dim-row } A - 1) \ 1 * H2\text{-UL} + P3 * H2\text{-DL}) (0_m (\text{dim-row } A - 1) \ 1 * H2\text{-UR} + P3 * H2\text{-DR})$   
**unfolding**  $P3'\text{-def } H2\text{-as-four-block-mat}$   
**by**  $(\text{rule mult-four-block-mat}[OF \ - \ - \ P3 \ H2\text{-UL } H2\text{-UR } H2\text{-DL } H2\text{-DR}],$   
 $\text{insert } A, \text{ auto})$   
**also have**  $\dots = \text{four-block-mat } H2\text{-UL } H2\text{-UR } (P3 * H2\text{-DL}) (P3 * H2\text{-DR})$   
**by**  $(\text{rule cong-four-block-mat}, \text{insert } H2\text{-UL } A \ m \ H2\text{-DL } H2\text{-DR } H2\text{-UR } P3,$   
 $\text{auto})$   
**finally show**  $?thesis .$   
**qed**  
**hence**  $P3'\text{-H2-as-four-block-mat}: P3' * H2 = \text{four-block-mat } H2\text{-UL } (0_m \ 1$   
 $(n-1)) (0_m (m-1) \ 1) (P3 * H2\text{-DR})$   
**unfolding**  $H2\text{-UR-0 } H2\text{-DL-0}$  **using**  $P3$  **by**  $\text{auto}$   
**also have**  $\dots * Q3' = S$  **(is**  $?lhs = ?rhs)$   
**proof** –  
**have**  $?lhs = \text{four-block-mat } H2\text{-UL } (0_m \ 1 (n-1)) (0_m (m-1) \ 1) (P3 * H2\text{-DR})$   
 $* \text{four-block-mat } (1_m \ 1) (0_m \ 1 (n-1)) (0_m (n-1) \ 1) Q3$  **unfolding**  
 $Q3'\text{-def}$  **using**  $A$  **by**  $\text{auto}$   
**also have**  $\dots =$   
 $\text{four-block-mat } (H2\text{-UL} * 1_m \ 1 + (0_m \ 1 (n-1)) * 0_m (n-1) \ 1) (H2\text{-UL} * 0_m \ 1 (n-1) + (0_m \ 1 (n-1)) * Q3)$   
 $(0_m (m-1) \ 1 * 1_m \ 1 + P3 * H2\text{-DR} * 0_m (n-1) \ 1) (0_m (m-1) \ 1 * 0_m \ 1 (n-1) + P3 * H2\text{-DR} * Q3)$   
**by**  $(\text{rule mult-four-block-mat}[OF \ H2\text{-UL}], \text{insert } P3 \ H2\text{-DR } Q3, \text{ auto})$   
**also have**  $\dots = \text{four-block-mat } H2\text{-UL } (0_m \ 1 (n-1)) (0_m (m-1) \ 1) (P3 * H2\text{-DR} * Q3)$   
**by**  $(\text{rule cong-four-block-mat}, \text{insert } H2\text{-UL } A \ m \ H2\text{-DL } H2\text{-DR } H2\text{-UR } P3 \ Q3, \text{ auto})$   
**also have**  $\dots = \text{four-block-mat } (\text{Matrix.mat } 1 \ 1 (\lambda(a, b). H \ \$\$ (0, 0)))$   
 $(0_m \ 1 (\text{dim-col } A - 1)) (0_m (\text{dim-row } A - 1) \ 1) S'$   
**by**  $(\text{rule cong-four-block-mat}, \text{insert } A \ S'\text{-P3H2-DRQ3 } H2\text{-UL00-H00 } H2\text{-UL},$   
 $\text{auto})$   
**finally show**  $?thesis$  **unfolding**  $S\text{-def}$  **by**  $\text{simp}$   
**qed**  
**finally have**  $P3'\text{-H2-Q3'-S}: P3' * H2 * Q3' = S .$   
**have**  $S\text{-as-four-block-mat}: S = \text{four-block-mat } H2\text{-UL } (0_m \ 1 (n-1)) (0_m (m-1) \ 1) S'$   
**unfolding**  $S\text{-def}$  **by**  $(\text{rule cong-four-block-mat}, \text{insert } A \ S'\text{-P3H2-DRQ3 } H2\text{-UL00-H00 } H2\text{-UL}, \text{ auto})$   
**show**  $S = P3' * P\text{-H2} * P2' * P1' * A * (Q1 * Q2 * Q3')$  **using**  $P3'\text{-H2-Q3'-S}$   
**unfolding**  $H2\text{-eq}$   
**by**  $(\text{smt } P1 \ P1'\text{-def } P2' \ P2'\text{-def } P3 \ P3'\text{-def } P\text{-H2 } Q1 \ Q2 \ Q3' \ Q3'\text{-def } S \ Q\text{-final-carrier } P\text{-final-carrier})$

```

      assoc-mult-mat carrier-matD carrier-mat-triv index-mat-four-block(2,3)
index-mult-mat(2,3)
    have H00-dvd-all-H2:  $H \text{ $$$ } (0, 0) \text{ dvd } H2 \text{ $$$ } (i, j)$  if  $i: i < m$  and  $j: j < n$  for
i j
      using dvd-elements-mult-matrix-left[OF H P-H2] H00-dvd-all i j P-H2-H-H2
by blast
    hence H00-dvd-all-S:  $H \text{ $$$ } (0, 0) \text{ dvd } S \text{ $$$ } (i, j)$  if  $i: i < m$  and  $j: j < n$  for i j
      using dvd-elements-mult-matrix-left-right[OF H2 P3' Q3'] P3'-H2-Q3'-S i j
by auto
    show Smith-normal-form-mat S
    proof (rule Smith-normal-form-mat-intro)
      show isDiagonal-mat S
      proof (unfold isDiagonal-mat-def, rule+)
        fix i j assume  $i \neq j \wedge i < \text{dim-row } S \wedge j < \text{dim-col } S$ 
        hence ij:  $i \neq j$  and  $i: i < \text{dim-row } S$  and  $j: j < \text{dim-col } S$  by auto
        have i2:  $i < \text{dim-row } H2\text{-UL} + \text{dim-row } S'$  and j2:  $j < \text{dim-col } H2\text{-UL} +$ 
dim-col  $S'$ 
          using S-as-four-block-mat i j by auto
        have S  $\text{$$$ } (i, j) =$  (if  $i < \text{dim-row } H2\text{-UL}$  then if  $j < \text{dim-col } H2\text{-UL}$  then
H2-UL  $\text{$$$ } (i, j)$ 
          else  $(0_m \ 1 \ (n - 1)) \text{ $$$ } (i, j - \text{dim-col } H2\text{-UL})$  else if  $j < \text{dim-col } H2\text{-UL}$ 
          then  $(0_m \ (m - 1) \ 1) \text{ $$$ } (i - \text{dim-row } H2\text{-UL}, j)$  else  $S' \text{ $$$ } (i - \text{dim-row}$ 
H2-UL,  $j - \text{dim-col } H2\text{-UL})$ )
          by (unfold S-as-four-block-mat, rule index-mat-four-block(1)[OF i2 j2])
        also have ... = 0 (is ?lhs = 0)
        proof (cases  $i = 0 \vee j = 0$ )
          case True
            then show ?thesis unfolding S-def using ij i j S H2-UL by fastforce
          next
            case False
              have diag-S': isDiagonal-mat  $S'$  using SNF-S' unfolding Smith-normal-form-mat-def
by simp
                have i-not-0:  $i \neq 0$  and j-not-0:  $j \neq 0$  using False by auto
                hence ?lhs =  $S' \text{ $$$ } (i - \text{dim-row } H2\text{-UL}, j - \text{dim-col } H2\text{-UL})$  using i j
ij H2-UL by auto
                also have ... = 0 using diag-S'  $S' \text{ H2-UL } i\text{-not-0 } j\text{-not-0 } ij$  unfolding
isDiagonal-mat-def
                by (smt (verit) S-as-four-block-mat add-diff-inverse-nat add-less-cancel-left
carrier-matD i
index-mat-four-block(2,3) j less-one)
                finally show ?thesis .
              qed
            finally show S  $\text{$$$ } (i, j) = 0$  .
          qed
        show  $\forall a. a + 1 < \min(\text{dim-row } S) (\text{dim-col } S) \longrightarrow S \text{ $$$ } (a, a) \text{ dvd } S \text{ $$$ } (a$ 
+ 1,  $a + 1)$ 
        proof safe
          fix i assume  $i: i + 1 < \min(\text{dim-row } S) (\text{dim-col } S)$ 
          show S  $\text{$$$ } (i, i) \text{ dvd } S \text{ $$$ } (i + 1, i + 1)$ 

```

```

proof (cases i=0)
  case True
  have S $$ (0, 0) = H $$ (0,0) using H2-UL H2-UL00-H00 S-as-four-block-mat
by auto
  also have ... dvd S $$ (1,1) using H00-dvd-all-S i m n by auto
  finally show ?thesis using True by simp
next
  case False
  have S $$ (i, i) = S' $$ (i-1, i-1) using False S-def i by auto
  also have ... dvd S' $$ (i, i) using SNF-S' i S' S unfolding Smith-normal-form-mat-def
  by (smt (verit) False H2-UL S-as-four-block-mat add commute add-diff-inverse-nat
  carrier-matD
  index-mat-four-block(2,3) less-one min-less-iff-conj nat-add-left-cancel-less)
  also have ... = S $$ (i+1,i+1) using False S-def i by auto
  finally show ?thesis .
qed
qed
qed
qed
qed

```

## 16.4 Soundness theorem

**theorem** *is-SNF-Smith-mxn*:

**assumes** A:  $A \in \text{carrier-mat } m \ n$

**shows** *is-SNF* A (*Smith-mxn* A)

**using** *is-SNF-Smith-mxn-ge-2*[OF A] *is-SNF-Smith-mxn-less-2*[OF A] **by** *linarith*

**declare** *Smith-mxn.simps*[code]

**end**

**declare** *Smith-Impl.Smith-mxn.simps*[code-unfold]

**definition** *T-spec* :: ( $'a::\{\text{comm-ring-1}\} \Rightarrow 'a \Rightarrow ('a \times 'a \times 'a) \Rightarrow \text{bool}$ )

**where** *T-spec* T = ( $\forall a \ b::'a. \text{let } (a1,b1,d) = T \ a \ b \ \text{in}$

$a = a1*d \wedge b = b1*d \wedge \text{ideal-generated } \{a1,b1\} = \text{ideal-generated}$

$\{1\}$ )

**definition** *D'-spec* :: ( $'a::\{\text{comm-ring-1}\} \Rightarrow 'a \Rightarrow 'a \Rightarrow ('a \times 'a) \Rightarrow \text{bool}$ )

**where** *D'-spec* D' = ( $\forall a \ b \ c::'a. \text{let } (p,q) = D' \ a \ b \ c \ \text{in}$

$\text{ideal-generated}\{a,b,c\} = \text{ideal-generated}\{1\}$

$\longrightarrow \text{ideal-generated } \{p*a,p*b+q*c\} = \text{ideal-generated } \{1\}$ )

**end**

## 17 The Smith normal form algorithm in HOL Analysis

```

theory SNF-Algorithm-HOL-Analysis
  imports
    SNF-Algorithm
    Admits-SNF-From-Diagonal-Iff-Bezout-Ring
begin

```

### 17.1 Transferring the result from JNF to HOL Analysis

```

definition Smith-mxn-HMA :: (('a::comm-ring-1) ⇒ (('a) × ('a)))
  ⇒ (('a) ⇒ (('a) × ('a) × ('a))) ⇒ ('a ⇒ 'a ⇒ 'a) ⇒ ('a^n::mod-type ^ 'm::mod-type)
  ⇒ (('a ^ 'm::mod-type ^ 'm::mod-type) × ('a ^ 'n::mod-type ^ 'm::mod-type) × ('a ^ 'n::mod-type ^ 'n::mod-type))
  where
    Smith-mxn-HMA Smith-1x2 Smith-2x2 div-op A =
      (let Smith-1x2-JNF = (λA'. let (S',Q') = Smith-1x2 (Mod-Type-Connect.to-hma_v
        (Matrix.row A' 0))
          in (mat-of-row (Mod-Type-Connect.from-hma_v S'),
            Mod-Type-Connect.from-hma_m Q'));
        Smith-2x2-JNF = (λA'. let (P', S',Q') = Smith-2x2 (Mod-Type-Connect.to-hma_m
          A')
          in (Mod-Type-Connect.from-hma_m P', Mod-Type-Connect.from-hma_m
            S', Mod-Type-Connect.from-hma_m Q'));
          (P,S,Q) = Smith-Impl.Smith-mxn Smith-1x2-JNF Smith-2x2-JNF div-op
            (Mod-Type-Connect.from-hma_m A)
          in (Mod-Type-Connect.to-hma_m P, Mod-Type-Connect.to-hma_m S, Mod-Type-Connect.to-hma_m
            Q)
          )

```

```

definition is-SNF-HMA A R = (case R of (P,S,Q) ⇒
  invertible P ∧ invertible Q
  ∧ Smith-normal-form S ∧ S = P ** A ** Q)

```

### 17.2 Soundness in HOL Analysis

```

lemma is-SNF-Smith-mxn-HMA:
  fixes A::('a::comm-ring-1) ^ 'n::mod-type ^ 'm::mod-type
  assumes PSQ: (P,S,Q) = Smith-mxn-HMA Smith-1x2 Smith-2x2 div-op A
  and SNF-1x2-works: ∀ A. let (S',Q) = Smith-1x2 A in S' $h 1 = 0 ∧ invertible
    Q ∧ S' = A v* Q
  and SNF-2x2-works: ∀ A. is-SNF-HMA A (Smith-2x2 A)
  and d: is-div-op div-op
  shows is-SNF-HMA A (P,S,Q)
proof –
  let ?A = Mod-Type-Connect.from-hma_m A
  define Smith-1x2-JNF where Smith-1x2-JNF = (λA'. let (S',Q')

```



```

= Smith-1x2 (Mod-Type-Connect.to-hmav (Matrix.row A' 0))
in (mat-of-row (Mod-Type-Connect.from-hmav S'), Mod-Type-Connect.from-hmam
Q')
define Smith-2x2-JNF where Smith-2x2-JNF = ( $\lambda A'$ . let (P', S', Q') = Smith-2x2
(Mod-Type-Connect.to-hmam A')
in (Mod-Type-Connect.from-hmam P', Mod-Type-Connect.from-hmam S', Mod-Type-Connect.from-hmam
Q'))
obtain P' S' Q' where P'S'Q': (P', S', Q') = Smith-Impl.Smith-mxn Smith-1x2-JNF
Smith-2x2-JNF div-op ?A
by (metis prod-cases3)
have PSQ-P'S'Q': (P, S, Q) =
(Mod-Type-Connect.to-hmam P', Mod-Type-Connect.to-hmam S', Mod-Type-Connect.to-hmam
Q')
using PSQ P'S'Q' Smith-1x2-JNF-def Smith-2x2-JNF-def
unfolding Smith-mxn-HMA-def Let-def by (metis case-prod-conv)
have SNF-1x2-works':  $\forall (A::'a \text{ mat}) \in \text{carrier-mat } 1 \ 2. \text{ is-SNF } A (1_m \ 1, (\text{Smith-1x2-JNF } A))$ 
proof (rule+)
fix A'::'a mat assume A': A' ∈ carrier-mat 1 2
let ?A' = (Mod-Type-Connect.to-hmav (Matrix.row A' 0))::'a~2
obtain S2 Q2 where S'Q': (S2, Q2) = Smith-1x2 ?A'
by (metis surjective-pairing)
let ?S2 = (Mod-Type-Connect.from-hmav S2)
let ?S' = mat-of-row ?S2
let ?Q' = Mod-Type-Connect.from-hmam Q2
have [transfer-rule]: Mod-Type-Connect.HMA-V ?S2 S2
unfolding Mod-Type-Connect.HMA-V-def by auto
have [transfer-rule]: Mod-Type-Connect.HMA-M ?Q' Q2
unfolding Mod-Type-Connect.HMA-M-def by auto
have [transfer-rule]: Mod-Type-Connect.HMA-I 1 (1::2)
unfolding Mod-Type-Connect.HMA-I-def by (simp add: to-nat-1)
have c[transfer-rule]: Mod-Type-Connect.HMA-V ((Matrix.row A' 0)) ?A'
unfolding Mod-Type-Connect.HMA-V-def
by (rule from-hma-to-hmav[symmetric], insert A', auto simp add: Matrix.row-def)

have *: Smith-1x2-JNF A' = (?S', ?Q') by (metis Smith-1x2-JNF-def S'Q'
case-prod-conv)
show is-SNF A' (1_m 1, Smith-1x2-JNF A') unfolding *
proof (rule is-SNF-intro)
let ?row-A' = (Matrix.row A' 0)
have w: S2 $h 1 = 0 ∧ invertible Q2 ∧ S2 = ?A' v* Q2
using SNF-1x2-works by (metis (mono-tags, lifting) S'Q' fst-conv prod.case-eq-if
snd-conv)
have ?S2 $v 1 = 0 using w[untransferred] by auto
thus Smith-normal-form-mat ?S' unfolding Smith-normal-form-mat-def is-Diagonal-mat-def
by (auto simp add: less-2-cases-iff)
have S2-Q2-A: S2 = transpose Q2 *v ?A' using w transpose-matrix-vector
by auto

```

```

    have S2-Q2-A': ?S2 = transpose-mat ?Q' *_v ((Matrix.row A' 0)) using
S2-Q2-A by transfer'
    show 1_m 1 ∈ carrier-mat (dim-row A') (dim-row A') using A' by auto
    show ?Q' ∈ carrier-mat (dim-col A') (dim-col A') using A' by auto
    show invertible-mat (1_m 1) by auto
    show invertible-mat ?Q' using w[untransferred] by auto
    have ?S' = A' * ?Q'
    proof (rule eq-matI)
      show dim-row ?S' = dim-row (A' * ?Q') and dim-col ?S' = dim-col (A' *
?Q')
        using A' by auto
      fix i j assume i: i < dim-row (A' * ?Q') and j: j < dim-col (A' * ?Q')
      have ?S' $$ (i, j) = ?S' $$ (0, j)
        by (metis A' One-nat-def carrier-matD(1) i index-mult-mat(2) less-Suc0)
      also have ... = ?S2 $v j using j by auto
      also have ... = (transpose-mat ?Q' *_v ?row-A') $v j unfolding S2-Q2-A'
by simp
      also have ... = Matrix.row (transpose-mat ?Q') j · ?row-A'
        by (rule index-mult-mat-vec, insert j, auto)
      also have ... = Matrix.col ?Q' j · ?row-A' using j by auto
      also have ... = ?row-A' · Matrix.col ?Q' j
      by (metis (no-types, lifting) Mod-Type-Connect.HMA-V-def Mod-Type-Connect.from-hma_m-def

Mod-Type-Connect.from-hma_v-def c col-def comm-scalar-prod dim-row-mat(1)
vec-carrier)
      also have ... = (A' * ?Q') $$ (0, j) using A' j by auto
      finally show ?S' $$ (i, j) = (A' * ?Q') $$ (i, j) using i j A' by auto
    qed
    thus ?S' = 1_m 1 * A' * ?Q' using A' by auto
  qed
qed
have SNF-2x2-works': ∀ (A::'a mat) ∈ carrier-mat 2 2. is-SNF A (Smith-2x2-JNF
A)
proof
  fix A::'a mat assume A': A' ∈ carrier-mat 2 2
  let ?A' = Mod-Type-Connect.to-hma_m A::'a ^2 ^2
  obtain P2 S2 Q2 where P2S2Q2: (P2, S2, Q2) = Smith-2x2 ?A'
    by (metis prod-cases3)
  let ?P2 = Mod-Type-Connect.from-hma_m P2
  let ?S2 = Mod-Type-Connect.from-hma_m S2
  let ?Q2 = Mod-Type-Connect.from-hma_m Q2
  have [transfer-rule]: Mod-Type-Connect.HMA-M ?Q2 Q2
    and [transfer-rule]: Mod-Type-Connect.HMA-M ?P2 P2
    and [transfer-rule]: Mod-Type-Connect.HMA-M ?S2 S2
    and [transfer-rule]: Mod-Type-Connect.HMA-M A' ?A'
  unfolding Mod-Type-Connect.HMA-M-def using A' by auto
  have is-SNF A' (?P2, ?S2, ?Q2)
proof -
  have P2: ?P2 ∈ carrier-mat (dim-row A') (dim-row A') and

```

```

    Q2: ?Q2 ∈ carrier-mat (dim-col A') (dim-col A') using A' by auto
    have is-SNF-HMA ?A' (P2,S2,Q2) using SNF-2x2-works by (simp add:
P2S2Q2)
    hence invertible P2 ∧ invertible Q2 ∧ Smith-normal-form S2 ∧ S2 = P2 **
?A' ** Q2
    unfolding is-SNF-HMA-def by auto
    from this[untransferred] show ?thesis using P2 Q2 unfolding is-SNF-def
by auto
    qed
    thus is-SNF A' (Smith-2x2-JNF A') using P2S2Q2 by (metis Smith-2x2-JNF-def
case-prod-conv)
    qed
    interpret Smith-Impl Smith-1x2-JNF Smith-2x2-JNF div-op
    using SNF-2x2-works' SNF-1x2-works' d by (unfold-locales, auto)
    have A: ?A ∈ carrier-mat CARD('m) CARD('n) by auto
    have is-SNF ?A (Smith-Impl.Smith-mxn Smith-1x2-JNF Smith-2x2-JNF div-op
?A)
    by (rule is-SNF-Smith-mxn[OF A])
    hence inv-P': invertible-mat P'
    and Smith-S': Smith-normal-form-mat S' and inv-Q': invertible-mat Q'
    and S'-P'AQ': S' = P' * ?A * Q'
    and P': P' ∈ carrier-mat (dim-row ?A) (dim-row ?A)
    and Q': Q' ∈ carrier-mat (dim-col ?A) (dim-col ?A)
    unfolding is-SNF-def P'S'Q'[symmetric] by auto
    have S': S' ∈ carrier-mat (dim-row ?A) (dim-col ?A) using P' Q' S'-P'AQ' by
auto
    have [transfer-rule]: Mod-Type-Connect.HMA-M P' P
    and [transfer-rule]: Mod-Type-Connect.HMA-M S' S
    and [transfer-rule]: Mod-Type-Connect.HMA-M Q' Q
    and [transfer-rule]: Mod-Type-Connect.HMA-M ?A A
    unfolding Mod-Type-Connect.HMA-M-def using PSQ-P'S'Q'
    using from-hma-to-hma_m[symmetric] P' A Q' S' by auto
    have inv-Q: invertible Q using inv-Q' by transfer
    moreover have Smith-S: Smith-normal-form S using Smith-S' by transfer
    moreover have inv-P: invertible P using inv-P' by transfer
    moreover have S = P ** A ** Q using S'-P'AQ' by transfer
    thus ?thesis using inv-Q inv-P Smith-S unfolding is-SNF-HMA-def by auto
    qed
end

```

## 18 Elementary divisor rings

```

theory Elementary-Divisor-Rings
  imports
    SNF-Algorithm
    Rings2-Extended
begin

```

This theory contains the definition of elementary divisor rings and Hermite

rings, as well as the corresponding relation between both concepts. It also includes a complete characterization for elementary divisor rings, by means of an *if and only if*-statement.

The results presented here follows the article “Some remarks about elementary divisor rings” by Leonard Gillman and Melvin Henriksen.

## 18.1 Previous definitions and basic properties of Hermite ring

**definition** *admits-triangular-reduction*  $A =$   
 $(\exists U :: 'a :: \text{comm-ring-1 mat. } U \in \text{carrier-mat } (\text{dim-col } A) (\text{dim-col } A)$   
 $\wedge \text{invertible-mat } U \wedge \text{lower-triangular } (A * U))$

**class** *Hermite-ring* =  
**assumes**  $\forall (A :: 'a :: \text{comm-ring-1 mat}). \text{admits-triangular-reduction } A$

**lemma** *admits-triangular-reduction-intro*:  
**assumes** *invertible-mat*  $(U :: 'a :: \text{comm-ring-1 mat})$   
**and**  $U \in \text{carrier-mat } (\text{dim-col } A) (\text{dim-col } A)$   
**and** *lower-triangular*  $(A * U)$   
**shows** *admits-triangular-reduction*  $A$   
**using** *assms* **unfolding** *admits-triangular-reduction-def* **by** *auto*

**lemma** *OFCLASS-Hermite-ring-def*:  
 $\text{OFCLASS}('a :: \text{comm-ring-1}, \text{Hermite-ring-class})$   
 $\equiv (\wedge (A :: 'a :: \text{comm-ring-1 mat}). \text{admits-triangular-reduction } A)$

**proof**  
**fix**  $A :: 'a \text{ mat}$   
**assume**  $H : \text{OFCLASS}('a :: \text{comm-ring-1}, \text{Hermite-ring-class})$   
**have**  $\forall A. \text{admits-triangular-reduction } (A :: 'a \text{ mat})$   
**using** *conjunctionD2*[ $\text{OF } H[\text{unfolded } \text{Hermite-ring-class-def class.Hermite-ring-def}]$ ]  
**by** *auto*  
**thus** *admits-triangular-reduction*  $A$  **by** *auto*  
**next**  
**assume**  $i : (\wedge A :: 'a \text{ mat. } \text{admits-triangular-reduction } A)$   
**show**  $\text{OFCLASS}('a, \text{Hermite-ring-class})$   
**proof**  
**show**  $\forall A :: 'a \text{ mat. } \text{admits-triangular-reduction } A$  **using**  $i$  **by** *auto*  
**qed**  
**qed**

**definition** *admits-diagonal-reduction*  $:: 'a :: \text{comm-ring-1 mat} \Rightarrow \text{bool}$   
**where** *admits-diagonal-reduction*  $A = (\exists P Q. P \in \text{carrier-mat } (\text{dim-row } A)$   
 $(\text{dim-row } A) \wedge$   
 $Q \in \text{carrier-mat } (\text{dim-col } A) (\text{dim-col } A)$   
 $\wedge \text{invertible-mat } P \wedge \text{invertible-mat } Q$   
 $\wedge \text{Smith-normal-form-mat } (P * A * Q))$

**lemma** *admits-diagonal-reduction-intro*:  
**assumes**  $P \in \text{carrier-mat } (\text{dim-row } A) (\text{dim-row } A)$   
**and**  $Q \in \text{carrier-mat } (\text{dim-col } A) (\text{dim-col } A)$   
**and** *invertible-mat*  $P$  **and** *invertible-mat*  $Q$   
**and** *Smith-normal-form-mat*  $(P * A * Q)$   
**shows** *admits-diagonal-reduction*  $A$  **using** *assms unfolding admits-diagonal-reduction-def*  
**by** *fast*

**lemma** *admits-diagonal-reduction-imp-exists-algorithm-is-SNF*:  
**assumes**  $A \in \text{carrier-mat } m \ n$   
**and** *admits-diagonal-reduction*  $A$   
**shows**  $\exists \text{algorithm. is-SNF } A (\text{algorithm } A)$   
**using** *assms unfolding is-SNF-def admits-diagonal-reduction-def*  
**by** *auto*

**lemma** *exists-algorithm-is-SNF-imp-admits-diagonal-reduction*:  
**assumes**  $A \in \text{carrier-mat } m \ n$   
**and**  $\exists \text{algorithm. is-SNF } A (\text{algorithm } A)$   
**shows** *admits-diagonal-reduction*  $A$   
**using** *assms unfolding is-SNF-def admits-diagonal-reduction-def*  
**by** *auto*

**lemma** *admits-diagonal-reduction-eq-exists-algorithm-is-SNF*:  
**assumes**  $A: A \in \text{carrier-mat } m \ n$   
**shows** *admits-diagonal-reduction*  $A = (\exists \text{algorithm. is-SNF } A (\text{algorithm } A))$   
**using** *admits-diagonal-reduction-imp-exists-algorithm-is-SNF[OF A]*  
**using** *exists-algorithm-is-SNF-imp-admits-diagonal-reduction[OF A]*  
**by** *auto*

**lemma** *admits-diagonal-reduction-imp-exists-algorithm-is-SNF-all*:  
**assumes**  $(\forall (A::'a::\text{comm-ring-1 mat}) \in \text{carrier-mat } m \ n. \text{admits-diagonal-reduction } A)$   
**shows**  $(\exists \text{algorithm. } \forall (A::'a \text{ mat}) \in \text{carrier-mat } m \ n. \text{is-SNF } A (\text{algorithm } A))$   
**proof** –  
**let**  $?algorithm = \lambda A. \text{SOME } (P, S, Q). \text{is-SNF } A (P, S, Q)$   
**show** *?thesis*  
**by** (*rule exI[of - ?algorithm]*) (*metis (no-types, lifting)*)  
*admits-diagonal-reduction-imp-exists-algorithm-is-SNF assms case-prod-beta prod.collapse someI*  
**qed**

**lemma** *exists-algorithm-is-SNF-imp-admits-diagonal-reduction-all*:  
**assumes**  $(\exists \text{algorithm. } \forall (A::'a \text{ mat}) \in \text{carrier-mat } m \ n. \text{is-SNF } A (\text{algorithm } A))$   
**shows**  $(\forall (A::'a::\text{comm-ring-1 mat}) \in \text{carrier-mat } m \ n. \text{admits-diagonal-reduction } A)$

**using** *assms exists-algorithm-is-SNF-imp-admits-diagonal-reduction* **by** *blast*

**lemma** *admits-diagonal-reduction-eq-exists-algorithm-is-SNF-all*:  
**shows**  $(\forall (A::'a::\text{comm-ring-1 mat}) \in \text{carrier-mat } m \ n. \text{ admits-diagonal-reduction } A)$   
 $= (\exists \text{ algorithm}. \forall (A::'a \text{ mat}) \in \text{carrier-mat } m \ n. \text{ is-SNF } A \ (\text{algorithm } A))$   
**using** *exists-algorithm-is-SNF-imp-admits-diagonal-reduction-all*  
**using** *admits-diagonal-reduction-imp-exists-algorithm-is-SNF-all* **by** *auto*

## 18.2 The class that represents elementary divisor rings

**class** *elementary-divisor-ring* =  
**assumes**  $\forall (A::'a::\text{comm-ring-1 mat}). \text{ admits-diagonal-reduction } A$

**lemma** *dim-row-mat-diag[simp]*:  $\text{dim-row } (\text{mat-diag } n \ f) = n$  **and**  
*dim-col-mat-diag[simp]*:  $\text{dim-col } (\text{mat-diag } n \ f) = n$   
**using** *mat-diag-dim unfolding carrier-mat-def* **by** *auto+*

## 18.3 Hermite ring implies Bézout ring

To prove this fact, we make use of the alternative definition for Bézout rings: each finitely generated ideal is principal

**lemma** *Hermite-ring-imp-Bezout-ring*:  
**assumes**  $H: \text{ OFCLASS}('a::\text{comm-ring-1}, \text{ Hermite-ring-class})$   
**shows**  $\text{ OFCLASS}('a::\text{comm-ring-1}, \text{ bezout-ring-class})$   
**proof** (*rule all-fin-gen-ideals-are-principal-imp-bezout, rule+*)  
**fix**  $I::'a \text{ set}$  **assume**  $\text{ fin: finitely-generated-ideal } I$

**obtain**  $S$  **where** *ig-S: ideal-generated*  $S = I$  **and** *fin-S: finite*  $S$   
**using** *fin unfolding finitely-generated-ideal-def* **by** *auto*  
**obtain**  $xs$  **where** *set-xs: set*  $xs = S$  **and** *d: distinct*  $xs$   
**using** *finite-distinct-list[OF fin-S]* **by** *blast*  
**hence** *length-eq-card: length*  $xs = \text{card } S$  **using** *distinct-card* **by** *force*  
**define**  $n$  **where**  $n = \text{card } S$   
**define**  $A$  **where**  $A = \text{mat-of-rows } n \ [\text{vec-of-list } xs]$   
**have**  $A[\text{simp}]$ :  $A \in \text{carrier-mat } 1 \ n$  **unfolding** *A-def* **using** *mat-of-rows-carrier*  
**by** *auto*  
**have**  $\forall (A::'a::\text{comm-ring-1 mat}). \text{ admits-triangular-reduction } A$   
**using**  $H$  **unfolding** *OFCLASS-Hermite-ring-def* **by** *auto*  
**from this obtain**  $Q$  **where** *inv-Q: invertible-mat*  $Q$  **and** *t-AQ: lower-triangular*  
 $(A * Q)$   
**and**  $Q[\text{simp}]$ :  $Q \in \text{carrier-mat } n \ n$   
**unfolding** *admits-triangular-reduction-def* **using**  $A$  **by** *auto*  
**have**  $AQ[\text{simp}]$ :  $A * Q \in \text{carrier-mat } 1 \ n$  **using**  $A \ Q$  **by** *auto*  
**show** *principal-ideal*  $I$   
**proof** (*cases*  $xs = []$ )  
**case** *True*

```

then show ?thesis
  by (metis empty-set ideal-generated-0 ideal-generated-empty ig-S princi-
pal-ideal-def set-xs)
next
  case False
  have a:  $0 < \dim\text{-row } A$  using A by auto
  have  $0 < \text{length } xs$  using False by auto
  hence b:  $0 < \dim\text{-col } A$  using A n-def length-eq-card by auto
  have q0:  $0 < \dim\text{-col } Q$  by (metis A Q b carrier-matD(2))
  have n0:  $0 < n$  using  $\langle 0 < \text{length } xs \rangle$  length-eq-card n-def by linarith
  define d where  $d = (A * Q) \text{ } \$\$ (0, 0)$ 
  let ?h =  $(\lambda x. \text{THE } i. xs ! i = x \wedge i < n)$ 
  let ?u =  $\lambda i. xs ! i$ 
  have bij: bij-betw ?h (set xs) {0.. $n$ }
  proof (rule bij-betw-imageI)
    show inj-on ?h (set xs)
    proof -
      have  $x=y$  if  $x: x \in \text{set } xs$  and  $y: y \in \text{set } xs$ 
      and  $xy: (\text{THE } i. xs ! i = x \wedge i < n) = (\text{THE } i. xs ! i = y \wedge i < n)$ 
    for  $x y$ 
      proof -
        let ?i =  $(\text{THE } i. xs ! i = x \wedge i < n)$ 
        let ?j =  $(\text{THE } i. xs ! i = y \wedge i < n)$ 
        obtain  $i$  where  $xs\text{-}i: xs ! i = x \wedge i < n$  using  $x$ 
        by (metis in-set-conv-nth length-eq-card n-def)
        from  $this$  have 1:  $xs ! ?i = x \wedge ?i < n$ 
        by (rule theI, insert d  $xs\text{-}i$  length-eq-card n-def nth-eq-iff-index-eq,
fastforce)
        obtain  $j$  where  $xs\text{-}j: xs ! j = y \wedge j < n$  using  $y$ 
        by (metis in-set-conv-nth length-eq-card n-def)
        from  $this$  have 2:  $xs ! ?j = y \wedge ?j < n$ 
        by (rule theI, insert d  $xs\text{-}j$  length-eq-card n-def nth-eq-iff-index-eq,
fastforce)
        show ?thesis using 1 2 d  $xy$  by argo
      qed
      thus ?thesis unfolding inj-on-def by auto
    qed
  show  $(\lambda x. \text{THE } i. xs ! i = x \wedge i < n) \text{ } \text{' set } xs = \{0.. $n\}$$ 
  proof (auto)
    fix  $xa$  assume  $xa: xa \in \text{set } xs$ 
    let ?i =  $(\text{THE } i. xs ! i = xa \wedge i < n)$ 
    obtain  $i$  where  $xs\text{-}i: xs ! i = xa \wedge i < n$  using  $xa$ 
    by (metis in-set-conv-nth length-eq-card n-def)
    from  $this$  have 1:  $xs ! ?i = xa \wedge ?i < n$ 
    by (rule theI, insert d  $xs\text{-}i$  length-eq-card n-def nth-eq-iff-index-eq,
fastforce)
    thus  $(\text{THE } i. xs ! i = xa \wedge i < n) < n$  by simp
  next
    fix  $x$  assume  $x: x < n$ 

```

```

    have  $\exists xa \in \text{set } xs. x = (\text{THE } i. xs ! i = xa \wedge i < n)$ 
      by (rule bestI[of -  $xs ! x$ ], rule the-equality[symmetric], insert  $x d$ )
        (auto simp add: length-eq-card n-def nth-eq-iff-index-eq)+
    thus  $x \in (\lambda x. \text{THE } i. xs ! i = x \wedge i < n)$  ‘ set xs unfolding image-def
  by auto
    qed
  qed
  have  $i: \text{ideal-generated } \{d\} = \text{ideal-generated } S$ 
  proof -
    have ideal-S-explicit:  $\text{ideal-generated } S = \{y. \exists f. (\sum_{i \in S} f i * i) = y\}$ 
      unfolding ideal-explicit2[OF fin-S] by simp
    have  $\text{ideal-generated } \{d\} \subseteq \text{ideal-generated } S$ 
    proof (rule ideal-generated-subset2, auto simp add: ideal-S-explicit)
      have  $n: \text{dim-vec } (\text{col } Q 0) = n$  using  $Q n\text{-def}$  by auto
      have aux:  $\text{Matrix.row } A 0 \$v i = xs ! i$  if  $i: i < n$  for  $i$ 
      proof -
        have  $i2: i < \text{dim-col } A$ 
          by (simp add: A-def i)
        have  $\text{Matrix.row } A 0 \$v i = A \$\$ (0, i)$  by (rule index-row(1), auto simp
add:  $a b i2$ )
        also have  $\dots = [\text{vec-of-list } xs] ! 0 \$v i$ 
          unfolding A-def by (rule mat-of-rows-index, auto simp add:  $i$ )
        also have  $\dots = xs ! i$ 
          by (simp add: vec-of-list-index)
        finally show ?thesis .
      qed
    let  $?f = \lambda x. \text{let } i = (\text{THE } i. xs ! i = x \wedge i < n)$  in  $\text{col } Q 0 \$v i$ 
    let  $?g = (\lambda i. xs ! i * \text{col } Q 0 \$v i)$ 
    have  $d = (A * Q) \$\$ (0, 0)$  unfolding  $d\text{-def}$  by simp
    also have  $\dots = \text{Matrix.row } A 0 \cdot \text{col } Q 0$  by (rule index-mult-mat(1)[OF
 $a q0$ ])
    also have  $\dots = (\sum_{i = 0..<\text{dim-vec } (\text{col } Q 0)}. \text{Matrix.row } A 0 \$v i * \text{col } Q 0 \$v i)$ 
      unfolding scalar-prod-def by simp
    also have  $\dots = (\sum_{i = 0..<n}. \text{Matrix.row } A 0 \$v i * \text{col } Q 0 \$v i)$  unfolding
 $n$  by auto
    also have  $\dots = (\sum_{i = 0..<n}. xs ! i * \text{col } Q 0 \$v i)$ 
      by (rule sum.cong, auto simp add: aux)
    also have  $\dots = (\sum_{x \in \text{set } xs}. ?g (?h x))$ 
      by (rule sum.reindex-bij-betw[symmetric, OF bij])
    also have  $\dots = (\sum_{x \in \text{set } xs}. ?f x * x)$ 
    proof (rule sum.cong, auto simp add: Let-def)
      fix  $x$  assume  $x: x \in \text{set } xs$ 
      let  $?i = (\text{THE } i. xs ! i = x \wedge i < n)$ 
      obtain  $i$  where  $xs\text{-}i: xs ! i = x \wedge i < n$ 
        by (metis in-set-conv-nth x length-eq-card n-def)
      from  $xs\text{-}i$  have  $xs ! ?i = x \wedge ?i < n$ 
      by (rule theI, insert  $d xs\text{-}i$  length-eq-card n-def nth-eq-iff-index-eq, fastforce)

```



**thus**  $xs ! ?i * col Q 0 \$v ?i = col Q 0 \$v ?i * x$  **by** *auto*  
**qed**  
**also have**  $... = (\sum x \in S. ?f x * x)$  **using** *set-xs* **by** *auto*  
**finally show**  $\exists f. (\sum i \in S. f i * i) = d$  **by** *auto*  
**qed**  
**moreover have** *ideal-generated*  $S \subseteq$  *ideal-generated*  $\{d\}$   
**proof**  
**fix**  $x$  **assume**  $x: x \in$  *ideal-generated*  $S$  **thm** *Matrix.diag-mat-def*  
**hence**  $x$ - $xs: x \in$  *ideal-generated*  $(set\ xs)$  **by** *(simp add: set-xs)*  
**from this obtain**  $f$  **where**  $f: (\sum i \in (set\ xs). f i * i) = x$  **using**  $x$  *ideal-explicit2*  
**by** *auto*  
**define**  $B$  **where**  $B = Matrix.vec\ n\ (\lambda i. f\ (A\ \$\$ (0,i)))$   
**have**  $B: B \in$  *carrier-vec*  $n$  **unfolding**  $B$ -*def* **by** *auto*  
**have**  $(A *_v B)\ \$v\ 0 = Matrix.row\ A\ 0 \cdot B$  **by** *(rule index-mult-mat-vec[OF*  
*a])*  
**also have**  $... = sum\ (\lambda i. f\ (A\ \$\$ (0,i)) * A\ \$\$ (0,i))\ \{0..<n\}$   
**unfolding**  $B$ -*def* *Matrix.row-def* *scalar-prod-def* **by** *(rule sum.cong, auto*  
*simp add: A-def)*  
**also have**  $... = sum\ (\lambda i. f\ i * i)\ (set\ xs)$   
**proof** *(rule sum.reindex-bij-betw)*  
**have**  $1: inj$ -*on*  $(\lambda x. A\ \$\$ (0, x))\ \{0..<n\}$   
**proof** *(unfold inj-on-def, auto)*  
**fix**  $x\ y$  **assume**  $x: x < n$  **and**  $y: y < n$  **and**  $xy: A\ \$\$ (0, x) = A\ \$\$ (0,$   
*y)*  
**have**  $A\ \$\$ (0,x) = [vec-of-list\ xs] ! 0\ \$v\ x$   
**unfolding**  $A$ -*def* **by** *(rule mat-of-rows-index, insert x y, auto)*  
**also have**  $... = xs ! x$  **using**  $x$  **by** *(simp add: vec-of-list-index)*  
**finally have**  $1: A\ \$\$ (0,x) = xs ! x$  .  
**have**  $A\ \$\$ (0,y) = [vec-of-list\ xs] ! 0\ \$v\ y$   
**unfolding**  $A$ -*def* **by** *(rule mat-of-rows-index, insert x y, auto)*  
**also have**  $... = xs ! y$  **using**  $y$  **by** *(simp add: vec-of-list-index)*  
**finally have**  $2: A\ \$\$ (0,y) = xs ! y$  .  
**show**  $x = y$  **using**  $1\ 2\ x\ y\ d\ length$ -*eq-card*  $n$ -*def* *nth-eq-iff-index-eq*  $xy$   
**by** *fastforce*  
**qed**  
**have**  $2: A\ \$\$ (0, xa) \in set\ xs$  **if**  $xa: xa < n$  **for**  $xa$   
**proof** –  
**have**  $A\ \$\$ (0,xa) = [vec-of-list\ xs] ! 0\ \$v\ xa$   
**unfolding**  $A$ -*def* **by** *(rule mat-of-rows-index, insert xa, auto)*  
**also have**  $... = xs ! xa$  **using**  $xa$  **by** *(simp add: vec-of-list-index)*  
**finally show**  $?thesis$  **using**  $xa$  **by** *(simp add: length-eq-card n-def)*  
**qed**  
**have**  $3: x \in (\lambda x. A\ \$\$ (0, x))\ ' \{0..<n\}$  **if**  $x: x \in set\ xs$  **for**  $x$   
**proof** –  
**obtain**  $i$  **where**  $xs: xs ! i = x \wedge i < n$   
**by** *(metis in-set-conv-nth length-eq-card n-def x)*  
**have**  $A\ \$\$ (0,i) = [vec-of-list\ xs] ! 0\ \$v\ i$   
**unfolding**  $A$ -*def* **by** *(rule mat-of-rows-index, insert xs, auto)*  
**also have**  $... = xs ! i$  **using**  $xs$  **by** *(simp add: vec-of-list-index)*

**finally show** *?thesis* **using** *xs* **unfolding** *image-def* **by** *auto*  
**qed**  
**show** *bij-betw*  $(\lambda x. A \text{ \*\* } (0, x)) \{0..<n\}$  *(set xs)* **using** *1 2 3* **unfolding**  
*bij-betw-def* **by** *auto*  
**qed**  
**finally have** *AB00-sum*:  $(A \text{ \*_v } B) \$v 0 = \text{sum } (\lambda i. f i * i)$  *(set xs)* **by** *auto*  
**hence** *AB-00-x*:  $(A \text{ \*_v } B) \$v 0 = x$  **using** *f* **by** *auto*  
**obtain** *Q'* **where** *QQ'*: *inverts-mat* *Q Q'*  
**and** *Q'Q*: *inverts-mat* *Q' Q* **and** *Q'*: *Q' ∈ carrier-mat n n*  
**by** *(rule obtain-inverse-matrix[OF Q inv-Q], auto)*  
**have** *eq*:  $A = (A * Q) * Q'$  **using** *QQ'* **unfolding** *inverts-mat-def*  
**by** *(metis A Q Q' assoc-mult-mat carrier-matD(1) right-mult-one-mat)*  
  
**let** *?g* =  $\lambda i. \text{Matrix.row } (A * Q) 0 \$v i * (\text{Matrix.row } Q' i \cdot B)$   
**have** *sum0*:  $(\sum i = 1..<n. ?g i) = 0$   
**proof** *(rule sum.neutral, rule)*  
**fix** *x* **assume** *x*:  $x \in \{1..<n\}$   
**hence** *Matrix.row*  $(A * Q) 0 \$v x = 0$  **using** *t-AQ* **unfolding** *lower-triangular-def*  
**by** *(auto, metis Q Suc-le-lessD a carrier-matD(2) index-mult-mat(2,3) index-row(1))*  
**thus** *Matrix.row*  $(A * Q) 0 \$v x * (\text{Matrix.row } Q' x \cdot B) = 0$  **by** *simp*  
**qed**  
**have** *set-rw*:  $\{0..<n\} - \{0\} = \{1..<n\}$   
**by** *(simp add: atLeast0LessThan atLeast1-lessThan-eq-remove0)*  
**have** *mat-rw*:  $(A * Q * Q') \text{ \*_v } B = A * Q \text{ \*_v } (Q' \text{ \*_v } B)$   
**by** *(rule assoc-mult-mat-vec, insert Q Q' B A Q, auto)*  
**from** *eq* **have**  $A \text{ \*_v } B = (A * Q) \text{ \*_v } (Q' \text{ \*_v } B)$  **using** *mat-rw* **by** *auto*  
**from** *this* **have**  $(A \text{ \*_v } B) \$v 0 = (A * Q \text{ \*_v } (Q' \text{ \*_v } B)) \$v 0$  **by** *auto*  
**also** **have**  $\dots = \text{Matrix.row } (A * Q) 0 \cdot (Q' \text{ \*_v } B)$   
**by** *(rule index-mult-mat-vec, insert a B-def n0, auto)*  
**also** **have**  $\dots = (\sum i = 0..<n. ?g i)$  **using** *Q'* **by** *(auto simp add: scalar-prod-def)*  
**also** **have**  $\dots = ?g 0 + (\sum i \in \{0..<n\} - \{0\}. ?g i)$   
**by** *(metis (no-types, lifting) Q atLeast0LessThan carrier-matD(2) finite-atLeastLessThan lessThan-iff q0 sum.remove)*  
**also** **have**  $\dots = ?g 0 + (\sum i = 1..<n. ?g i)$  **using** *set-rw* **by** *simp*  
**also** **have**  $\dots = ?g 0$  **using** *sum0* **by** *auto*  
**also** **have**  $\dots = d * (\text{Matrix.row } Q' 0 \cdot B)$  **by** *(simp add: a d-def q0)*  
**finally show**  $x \in \text{ideal-generated } \{d\}$  **using** *AB-00-x* **unfolding** *ideal-generated-singleton*  
  
**using** *mult commute* **by** *auto*  
**qed**  
**ultimately show** *?thesis* **by** *auto*  
**qed**  
**thus** *principal-ideal I* **unfolding** *principal-ideal-def ig-S* **by** *blast*  
**qed**  
**qed**

## 18.4 Elementary divisor ring implies Hermite ring

**context**

**assumes** *SORT-CONSTRAINT('a::comm-ring-1)*

**begin**

**lemma** *triangularizable-m0:*

**assumes** *A: A ∈ carrier-mat m 0*

**shows**  $\exists U. U \in \text{carrier-mat } 0 \ 0 \wedge \text{invertible-mat } U \wedge \text{lower-triangular } (A * U)$

**using** *A unfolding lower-triangular-def carrier-mat-def invertible-mat-def inverts-mat-def*

**by auto** (*metis gr-implies-not0 index-one-mat(2) index-one-mat(3) right-mult-one-mat'*)

**lemma** *triangularizable-0n:*

**assumes** *A: A ∈ carrier-mat 0 n*

**shows**  $\exists U. U \in \text{carrier-mat } n \ n \wedge \text{invertible-mat } U \wedge \text{lower-triangular } (A * U)$

**using** *A unfolding lower-triangular-def carrier-mat-def invertible-mat-def inverts-mat-def*

**by auto** (*metis index-one-mat(2) index-one-mat(3) right-mult-one-mat'*)

**lemma** *diagonal-imp-triangular-1x2:*

**assumes** *A: A ∈ carrier-mat 1 2 and d: admits-diagonal-reduction (A::'a mat)*

**shows** *admits-triangular-reduction A*

**proof** –

**obtain** *P Q where P: P ∈ carrier-mat (dim-row A) (dim-row A)*

**and** *Q: Q ∈ carrier-mat (dim-col A) (dim-col A)*

**and** *inv-P: invertible-mat P and inv-Q: invertible-mat Q*

**and** *SNF: Smith-normal-form-mat (P \* A \* Q)*

**using** *d unfolding admits-diagonal-reduction-def by blast*

**have**  $(P * A * Q) = P * (A * Q)$  **using** *P Q assoc-mult-mat by blast*

**also have**  $\dots = P \ \$\$ (0,0) \cdot_m (A * Q)$  **by** (*rule smult-mat-mat-one-element, insert P A Q, auto*)

**also have**  $\dots = A * (P \ \$\$ (0,0) \cdot_m Q)$  **using** *Q by auto*

**finally have** *eq: (P \* A \* Q) = A \* (P \ \\$\\$ (0,0) \cdot\_m Q) .*

**have** *inv: invertible-mat (P \ \\$\\$ (0,0) \cdot\_m Q)*

**proof** –

**have** *d: Determinant.det P = P \ \\$\\$ (0, 0)* **by** (*rule determinant-one-element, insert P A, auto*)

**from this have** *P-dvd-1: P \ \\$\\$ (0, 0) dvd 1*

**using** *invertible-iff-is-unit-JNF[OF P] using inv-P by auto*

**have** *Q-dvd-1: Determinant.det Q dvd 1* **using** *inv-Q invertible-iff-is-unit-JNF[OF Q] by simp*

**have** *Determinant.det (P \ \\$\\$ (0, 0) \cdot\_m Q) = P \ \\$\\$ (0, 0) ^ dim-col Q \* Determinant.det Q*

**unfolding** *det-smult by auto*

**also have**  $\dots \text{ dvd } 1$  **using** *P-dvd-1 Q-dvd-1 unfolding is-unit-mult-iff*

**by** (*metis dvdE dvd-mult-left one-dvd power-mult-distrib power-one*)

**finally have**  $det: (Determinant.det (P \ \$\$ (0, 0) \cdot_m Q) \ dvd \ 1) \cdot$   
**have**  $PQ: P \ \$\$ (0,0) \cdot_m Q \in carrier\text{-}mat \ 2 \ 2$  **using**  $A \ P \ Q$  **by** *auto*  
**show**  $?thesis$  **using** *invertible-iff-is-unit-JNF[OF PQ]*  $det$  **by** *auto*  
**qed**  
**moreover have** *lower-triangular*  $(A * (P \ \$\$ (0,0) \cdot_m Q))$  **unfolding** *lower-triangular-def*  
**using** *SNF eq*  
**unfolding** *Smith-normal-form-mat-def isDiagonal-mat-def* **by** *auto*  
**moreover have**  $(P \ \$\$ (0,0) \cdot_m Q) \in carrier\text{-}mat (dim\text{-}col \ A) (dim\text{-}col \ A)$  **using**  
 $P \ Q \ A$  **by** *auto*  
**ultimately show**  $?thesis$  **unfolding** *admits-triangular-reduction-def* **by** *auto*  
**qed**

**lemma** *triangular-imp-diagonal-1x2*:  
**assumes**  $A: A \in carrier\text{-}mat \ 1 \ 2$  **and**  $t: admits\text{-}triangular\text{-}reduction (A::'a \ mat)$   
**shows** *admits-diagonal-reduction A*  
**proof** –  
**obtain**  $U$  **where**  $U: U \in carrier\text{-}mat (dim\text{-}col \ A) (dim\text{-}col \ A)$   
**and**  $inv\text{-}U: invertible\text{-}mat \ U$  **and**  $AU: lower\text{-}triangular (A * U)$   
**using**  $t$  **unfolding** *admits-triangular-reduction-def* **by** *blast*  
**have**  $SNF\text{-}AU: Smith\text{-}normal\text{-}form\text{-}mat (A * U)$   
**using**  $AU \ A$  **unfolding** *Smith-normal-form-mat-def lower-triangular-def isDiagonal-mat-def* **by** *auto*  
**have**  $A * U = (1_m \ 1) * A * U$  **using**  $A$  **by** *auto*  
**hence**  $SNF: Smith\text{-}normal\text{-}form\text{-}mat ((1_m \ 1) * A * U)$  **using**  $SNF\text{-}AU$  **by** *auto*  
**moreover have** *invertible-mat*  $(1_m \ 1)$   
**using** *invertible-mat-def inverts-mat-def* **by** *fastforce*  
**ultimately show**  $?thesis$  **using**  $inv\text{-}U$  **unfolding** *admits-diagonal-reduction-def*  
**by**  $(smt (verit) U \ assms(1) carrier\text{-}matD(1) one\text{-}carrier\text{-}mat)$   
**qed**

**lemma** *triangular-eq-diagonal-1x2*:  
 $(\forall A \in carrier\text{-}mat \ 1 \ 2. admits\text{-}triangular\text{-}reduction (A::'a \ mat))$   
 $= (\forall A \in carrier\text{-}mat \ 1 \ 2. admits\text{-}diagonal\text{-}reduction (A::'a \ mat))$   
**using** *triangular-imp-diagonal-1x2 diagonal-imp-triangular-1x2* **by** *auto*

**lemma** *admits-triangular-mat-1x1*:  
**assumes**  $A: A \in carrier\text{-}mat \ 1 \ 1$   
**shows** *admits-triangular-reduction (A::'a mat)*  
**by**  $(rule \ admits\text{-}triangular\text{-}reduction\text{-}intro[of \ 1_m \ 1], \ insert \ A,$   
 $auto \ simp \ add: \ admits\text{-}triangular\text{-}reduction\text{-}def \ lower\text{-}triangular\text{-}def)$

**lemma** *admits-diagonal-mat-1x1*:  
**assumes**  $A: A \in carrier\text{-}mat \ 1 \ 1$   
**shows** *admits-diagonal-reduction (A::'a mat)*  
**by**  $(rule \ admits\text{-}diagonal\text{-}reduction\text{-}intro[of \ (1_m \ 1) - (1_m \ 1)],$   
 $insert \ A, \ auto \ simp \ add: \ Smith\text{-}normal\text{-}form\text{-}mat\text{-}def \ isDiagonal\text{-}mat\text{-}def)$

```

lemma admits-diagonal-imp-admits-triangular-1xn:
  assumes a:  $\forall A \in \text{carrier-mat } 1 \ 2. \text{ admits-diagonal-reduction } (A::'a \text{ mat})$ 
  shows  $\forall A \in \text{carrier-mat } 1 \ n. \text{ admits-triangular-reduction } (A::'a \text{ mat})$ 
proof
  fix A::'a mat assume A:  $A \in \text{carrier-mat } 1 \ n$ 
  have  $\exists U. U \in \text{carrier-mat } (\text{dim-col } A) (\text{dim-col } A)$ 
     $\wedge \text{invertible-mat } U \wedge \text{lower-triangular } (A * U)$ 
  using A
  proof (induct n arbitrary: A rule: less-induct)
  case (less n)
  note  $A = \text{less.prem}(1)$ 
  show ?case
  proof (cases n=0)
  case True
  then show ?thesis using triangularizable-m0 triangularizable-0n less.prem
by auto
  next
  case False note nm-not-0 = False
  from this have n-not-0:  $n \neq 0$  by auto
  show ?thesis
  proof (cases n>2)
  case False note n-less-2 = False
  show ?thesis using admits-triangular-mat-1x1 a diagonal-imp-triangular-1x2

  unfolding admits-triangular-reduction-def
  by (metis (full-types) admits-triangular-mat-1x1 Suc-1 admits-triangular-reduction-def

  less(2) less-Suc-eq less-one linorder-neqE-nat n-less-2 nm-not-0 trian-
  gular-eq-diagonal-1x2)
  next
  case True note n-ge-2 = True
  let ?B = mat-of-row (vec-last (Matrix.row A 0) (n - 1))
  have  $\exists V. V \in \text{carrier-mat } (\text{dim-col } ?B) (\text{dim-col } ?B)$ 
     $\wedge \text{invertible-mat } V \wedge \text{lower-triangular } (?B * V)$ 
  proof (rule less.hyps)
  show  $n-1 < n$  using n-not-0 by auto
  show  $\text{mat-of-row } (\text{vec-last } (\text{Matrix.row } A \ 0) \ (n - 1)) \in \text{carrier-mat } 1 \ (n$ 
  - 1)
  using A by simp
  qed
from this obtain V where inv-V: invertible-mat V and BV: lower-triangular
  (?B * V)
  and V':  $V \in \text{carrier-mat } (\text{dim-col } ?B) (\text{dim-col } ?B)$ 
  by fast
  have  $V \in \text{carrier-mat } (n-1) \ (n-1)$  using V' by auto
  have BV-0:  $\forall j \in \{1..<n-1\}. (?B * V) \ \S\S \ (0,j) = 0$ 
  by (rule, rule lower-triangular-index[OF BV], insert V, auto)

```

```

define b where  $b = (?B * V) \$\$ (0,0)$ 
define a where  $a = A \$\$ (0,0)$ 
define ab::'a mat where  $ab = Matrix.mat\ 1\ 2\ (\lambda(i,j). \text{if } i=0 \wedge j=0 \text{ then } a$ 
else b)
  have ab[simp]:  $ab \in carrier\text{-}mat\ 1\ 2$  unfolding ab-def by simp
  hence admits-diagonal-reduction ab using a by auto
  hence admits-triangular-reduction ab using diagonal-imp-triangular-1x2[OF
ab] by auto
  from this obtain W where inv-W: invertible-mat W and ab-W: lower-triangular
(ab * W)
    and W:  $W \in carrier\text{-}mat\ 2\ 2$ 
    unfolding admits-triangular-reduction-def using ab by auto
    have id-n2-carrier[simp]:  $1_m\ (n-2) \in carrier\text{-}mat\ (n-2)\ (n-2)$  by auto
    define U where  $U = (four\text{-}block\text{-}mat\ (1_m\ 1)\ (0_m\ 1\ (n-1))\ (0_m\ (n-1)$ 
1) V) *
      ( $four\text{-}block\text{-}mat\ W\ (0_m\ 2\ (n-2))\ (0_m\ (n-2)\ 2)\ (1_m$ 
( $n-2$ )))
    let ?U1 =  $four\text{-}block\text{-}mat\ (1_m\ 1)\ (0_m\ 1\ (n-1))\ (0_m\ (n-1)\ 1)\ V$ 
    let ?U2 =  $four\text{-}block\text{-}mat\ W\ (0_m\ 2\ (n-2))\ (0_m\ (n-2)\ 2)\ (1_m\ (n-2))$ 
    have U1[simp]:  $?U1 \in carrier\text{-}mat\ n\ n$  using four-block-carrier-mat[OF -
V] nm-not-0
    by fastforce
    have U2[simp]:  $?U2 \in carrier\text{-}mat\ n\ n$  using four-block-carrier-mat[OF W
id-n2-carrier]
    by (metis True add-diff-inverse-nat less-imp-add-positive not-add-less1)
    have U[simp]:  $U \in carrier\text{-}mat\ n\ n$  unfolding U-def using U1 U2 by auto
    moreover have inv-U: invertible-mat U
    proof -
      have invertible-mat ?U1
      by (metis U1 V det-four-block-mat-lower-left-zero-col det-one inv-V
invertible-iff-is-unit-JNF more-arith-simps(5) one-carrier-mat
zero-carrier-mat)
      moreover have invertible-mat ?U2
      proof -
      have Determinant.det ?U2 = Determinant.det W
      by (rule det-four-block-mat-lower-right-id, insert less.prems W n-ge-2,
auto)
      also have ... dvd 1
      using W inv-W invertible-iff-is-unit-JNF by auto
      finally show ?thesis using invertible-iff-is-unit-JNF[OF U2] by auto
      qed
      ultimately show ?thesis
      using U1 U2 U-def invertible-mult-JNF by blast
      qed
    moreover have lower-triangular (A*U)
    proof -
      let ?A =  $Matrix.mat\ 1\ n\ (\lambda(i,j). \text{if } j = 0 \text{ then } a \text{ else if } j=1 \text{ then } b \text{ else } 0)$ 
      let ?T =  $Matrix.mat\ 1\ n\ (\lambda(i,j). \text{if } j = 0 \text{ then } (ab*W) \$\$ (0,0) \text{ else } 0)$ 
      have  $A*?U1 = ?A$ 

```

```

proof (rule eq-matI)
  fix  $i\ j$  assume  $i: i < \dim\text{-row } ?A$  and  $j: j < \dim\text{-col } ?A$ 
  have  $i0: i=0$  using  $i$  by auto
  let  $?f = \lambda i. A \ \$\$ (0, i) *$ 
  (if  $i = 0$  then if  $j < 1$  then  $1_m (1) \ \$\$ (i, j)$  else  $0_m (1) (n - 1) \ \$\$ (i, j$ 
- 1)
    else if  $j < 1$  then  $0_m (n - 1) (1) \ \$\$ (i - 1, j)$  else  $V \ \$\$ (i - 1, j -$ 
1))
  have  $(A * ?U1) \ \$\$ (i, j) = \text{Matrix.row } A\ i \cdot \text{col } ?U1\ j$ 
    by (rule index-mult-mat, insert  $i\ j\ A\ V$ , auto)
  also have  $\dots = (\sum_{i=0..<n.} ?f\ i)$ 
    using  $i\ j\ A\ V$  unfolding scalar-prod-def
    by auto (unfold index-one-mat, insert One-nat-def, presburger)
  also have  $\dots = ?A \ \$\$ (i, j)$ 
  proof (cases  $j=0$ )
    case True
      have  $rw0: \text{sum } ?f \{1..<n\} = 0$  by (rule sum.neutral, insert True,
auto)
      have  $\text{set-rw}: \{0..<n\} = \text{insert } 0 \{1..<n\}$  using n-ge-2 by auto
      hence  $\text{sum } ?f \{0..<n\} = ?f\ 0 + \text{sum } ?f \{1..<n\}$  by auto
      also have  $\dots = ?f\ 0$  unfolding rw0 by simp
      also have  $\dots = a$  using True unfolding a-def by simp
      also have  $\dots = ?A \ \$\$ (i, j)$  using True  $i\ j$  by auto
      finally show ?thesis .
    next
      case False note  $j\text{-not-0} = \text{False}$ 
      have  $rw\text{-simp}: \text{Matrix.row } (\text{mat-of-row } (\text{vec-last } (\text{Matrix.row } A\ 0) (n$ 
- 1))))  $0$ 
        =  $(\text{vec-last } (\text{Matrix.row } A\ 0) (n - 1))$  unfolding Matrix.row-def
      by auto
      let  $?g = \lambda i. A \ \$\$ (0, i) * V \ \$\$ (i - 1, j - 1)$ 
      let  $?h = \lambda i. A \ \$\$ (0, i+1) * V \ \$\$ (i, j - 1)$ 
      have  $f0: ?f\ 0 = 0$  using  $j\text{-not-0}\ j$  by auto
      have  $\text{set-rw2}: (\lambda i. i+1) \{0..<n-1\} = \{1..<n\}$ 
        unfolding image-def using Suc-le-D by fastforce
      have  $\text{set-rw}: \{0..<n\} = \text{insert } 0 \{1..<n\}$  using n-ge-2 by auto
      hence  $\text{sum } ?f \{0..<n\} = ?f\ 0 + \text{sum } ?f \{1..<n\}$  by auto
      also have  $\dots = \text{sum } ?f \{1..<n\}$  using f0 by simp
      also have  $\dots = \text{sum } ?g \{1..<n\}$  by (rule sum.cong, insert  $j\text{-not-0}$ ,
auto)
      also have  $\dots = \text{sum } ?g ((\lambda i. i+1) \{0..<n-1\})$  using set-rw2 by simp
      also have  $\dots = \text{sum } (?g \circ (\lambda i. i+1)) \{0..<n-1\}$ 
        by (rule sum.reindex, unfold inj-on-def, auto)
      also have  $\dots = \text{sum } ?h \{0..<n-1\}$  by (rule sum.cong, auto)
      also have  $\dots = \text{Matrix.row } ?B\ 0 \cdot \text{col } V (j-1)$  unfolding scalar-prod-def

  proof (rule sum.cong)
    fix  $x$  assume  $x: x \in \{0..<\dim\text{-vec } (\text{col } V (j - 1))\}$ 
    have  $\text{Matrix.row } ?B\ 0 \ \$v\ x = ?B \ \$\$ (0, x)$  by (rule index-row, insert

```

$x \ V, \text{ auto}$   
**also have** ... = (vec-last (Matrix.row A 0) (n - 1)) \$v x  
**by** (rule mat-of-row-index, insert x V, auto)  
**also have** ... = A \$\$ (0, x + 1)  
**by** (smt (verit) Suc-less-eq V add.right-neutral add-Suc-right  
add-diff-cancel-right'  
add-diff-inverse-nat atLeastLessThan-iff carrier-matD(1)  
carrier-matD(2)  
dim-col index-row(1) index-row(2) index-vec less.premis less-Suc0  
n-not-0  
plus-1-eq-Suc vec-last-def x)  
**finally have** Matrix.row ?B 0 \$v x = A \$\$ (0, x + 1) .  
**moreover have** col V (j - 1) \$v x = V \$\$ (x, j - 1) **using** V j x  
**by** auto  
**ultimately show** A \$\$ (0, x + 1) \* V \$\$ (x, j - 1)  
= Matrix.row ?B 0 \$v x \* col V (j - 1) \$v x **by** simp  
**qed** (insert V j-not-0, auto)  
**also have** ... = (?B\*V) \$\$ (0,j-1)  
**by** (rule index-mult-mat[symmetric], insert V j False, auto)  
**also have** ... = ?A \$\$ (i, j)  
**by** (cases j=1, insert False V j i0 BV-0 b-def, auto simp add: Suc-leI)  
  
**finally show** ?thesis .  
**qed**  
**finally show** (A\*?U1) \$\$ (i,j) = ?A \$\$ (i,j) .  
**next**  
**show** dim-row (A\*?U1) = dim-row ?A **using** A **by** auto  
**show** dim-col (A\*?U1) = dim-col ?A **using** U1 **by** auto  
**qed**  
**also have** ... \* ?U2 = ?T  
**proof** -  
**let** ?A1.0 = ab  
**let** ?B1.0 = Matrix.mat 1 (n-2) ( $\lambda(i,j). 0$ )  
**let** ?C1.0 = Matrix.mat 0 2 ( $\lambda(i,j). 0$ )  
**let** ?D1.0 = Matrix.mat 0 (n-2) ( $\lambda(i,j). 0$ )  
**let** ?B2.0 = ( $0_m \ 2 \ (n - 2)$ )  
**let** ?C2.0 = ( $0_m \ (n - 2) \ 2$ )  
**let** ?D2.0 = ( $1_m \ (n - 2)$ )  
**have** A-eq: ?A = four-block-mat ?A1.0 ?B1.0 ?C1.0 ?D1.0  
**by** (rule eq-matI, insert ab-def n-ge-2, auto)  
**hence** ?A \* ?U2 = four-block-mat ?A1.0 ?B1.0 ?C1.0 ?D1.0 \* ?U2 **by**  
simp  
**also have** ... = four-block-mat (?A1.0 \* W + ?B1.0 \* ?C2.0)  
(?A1.0 \* ?B2.0 + ?B1.0 \* ?D2.0) (?C1.0 \* W + ?D1.0 \* ?C2.0)  
(?C1.0 \* ?B2.0 + ?D1.0 \* ?D2.0)  
**by** (rule mult-four-block-mat, auto simp add: W ab-def)  
**also have** ... = four-block-mat (?A1.0 \* W) (?B1.0) (?C1.0) (?D1.0)  
**by** (rule cong-four-block-mat, insert W ab-def, auto)  
**also have** ... = ?T



```

      by (rule eq-matI, insert W n-ge-2 ab-def ab-W, auto simp add:
lower-triangular-def)
      finally show ?thesis .
    qed
    finally have A * U = ?T
      using assoc-mult-mat[OF - U1 U2] less.prem1 unfolding U-def by auto
    moreover have lower-triangular ?T unfolding lower-triangular-def by
simp
    ultimately show ?thesis by simp
  qed
  ultimately show ?thesis using A U by blast
  qed
  qed
  qed
  from this show admits-triangular-reduction A unfolding admits-triangular-reduction-def
by simp
qed

lemma admits-diagonal-imp-admits-triangular:
  assumes a:  $\forall A \in \text{carrier-mat } 1 \ 2. \text{ admits-diagonal-reduction } (A::'a \text{ mat})$ 
  shows  $\forall A. \text{ admits-triangular-reduction } (A::'a \text{ mat})$ 
proof
  fix A::'a mat
  obtain m n where A:  $A \in \text{carrier-mat } m \ n$  by auto
  have  $\exists U. U \in \text{carrier-mat } n \ n \wedge \text{invertible-mat } U \wedge \text{lower-triangular } (A * U)$ 
  using A
  proof (induct n arbitrary: m A rule: less-induct)
    case (less n)
    note A = less.prem1(1)
    show ?case
    proof (cases n=0  $\vee$  m=0)
      case True
      then show ?thesis using triangularizable-m0 triangularizable-0n less.prem1
by auto
    next
      case False note nm-not-0 = False
      from this have m-not-0:  $m \neq 0$  and n-not-0:  $n \neq 0$  by auto
      show ?thesis
      proof (cases m = 1)
        case True note m1 = True
        show ?thesis using admits-diagonal-imp-admits-triangular-1xn A m1 a
unfolding admits-triangular-reduction-def by blast
      next
        case False note m-not-1 = False

      show ?thesis
      proof (cases n=1)
        case True
        thus ?thesis using invertible-mat-zero lower-triangular-def

```

**by** (*metis carrier-matD(2) det-one gr-implies-not0 invertible-iff-is-unit-JNF*  
*less(2)*  
*less-one one-carrier-mat right-mult-one-mat')*  
**next**  
**case** *False* **note** *n-not-1 = False*  
**let** *?first-row = mat-of-row (Matrix.row A 0)*  
**have** *first-row: ?first-row ∈ carrier-mat 1 n* **using** *less.prem*s **by** *auto*  
**have** *m1: m > 1* **using** *m-not-1 m-not-0* **by** *linarith*  
**have** *n1: n > 1* **using** *n-not-1 n-not-0* **by** *linarith*  
**obtain** *V* **where** *lt-first-row-V: lower-triangular (?first-row \* V)*  
**and** *inv-V: invertible-mat V* **and** *V: V ∈ carrier-mat n n*  
  
**using** *admits-diagonal-imp-admits-triangular-1xn a first-row*  
**unfolding** *admits-triangular-reduction-def* **by** *blast*  
**have** *AV: A \* V ∈ carrier-mat m n* **using** *V less* **by** *auto*  
**have** *dim-row-AV: dim-row (A \* V) = 1 + (m - 1)* **using** *m1 AV* **by** *auto*  
**have** *dim-col-AV: dim-col (A \* V) = 1 + (n - 1)* **using** *n1 AV* **by** *fastforce*  
**have** *reduced-first-row: Matrix.row (?first-row \* V) 0 = Matrix.row (A \**  
*V) 0*  
**by** (*rule mult-eq-first-row, insert first-row m1 less.prem*s, *auto*)  
**obtain** *a zero B C* **where** *split: split-block (A \* V) 1 1 = (a, zero, B, C)*  
  
**using** *prod-cases4* **by** *blast*  
**have** *a: a ∈ carrier-mat 1 1* **and** *zero: zero ∈ carrier-mat 1 (n - 1)* **and**  
*B: B ∈ carrier-mat (m - 1) 1* **and** *C: C ∈ carrier-mat (m - 1) (n - 1)*  
**by** (*rule split-block[OF split dim-row-AV dim-col-AV]*)+  
**have** *AV-block: A \* V = four-block-mat a zero B C*  
**by** (*rule split-block[OF split dim-row-AV dim-col-AV]*)  
**have**  $\exists W. W \in \text{carrier-mat } (n - 1) (n - 1) \wedge \text{invertible-mat } W \wedge \text{lower-triangular}$   
*(C \* W)*  
**by** (*rule less.hyps, insert n1 C, auto*)  
**from this** **obtain** *W* **where** *inv-W: invertible-mat W* **and** *lt-CW:*  
*lower-triangular (C \* W)*  
**and** *W: W ∈ carrier-mat (n - 1) (n - 1)* **by** *blast*  
**let** *?W2 = four-block-mat (1<sub>m</sub> 1) (0<sub>m</sub> 1 (n - 1)) (0<sub>m</sub> (n - 1) 1) W*  
**have** *W2: ?W2 ∈ carrier-mat n n* **using** *V W dim-col-AV* **by** *auto*  
**have** *Determinant.det ?W2 = Determinant.det (1<sub>m</sub> 1) \* Determinant.det*  
*W*  
**by** (*rule det-four-block-mat-lower-left-zero-col[OF - - - W]*, *auto*)  
**hence** *det-W2: Determinant.det ?W2 = Determinant.det W* **by** *auto*  
**hence** *inv-W2: invertible-mat ?W2*  
**by** (*metis W four-block-carrier-mat inv-W invertible-iff-is-unit-JNF*  
*one-carrier-mat*)  
**have** *inv-V-W2: invertible-mat (V \* ?W2)* **using** *inv-W2 inv-V V W2*  
*invertible-mult-JNF* **by** *blast*  
**have** *lower-triangular (A \* V \* ?W2)*  
**proof** –  
**let** *?T = (four-block-mat a (0<sub>m</sub> 1 (n - 1)) B (C \* W))*  
**have** *zero-eq: zero = 0<sub>m</sub> 1 (n - 1)*

```

proof (rule eq-matI)
  show 1: dim-row zero = dim-row (0m 1 (n - 1)) and 2: dim-col zero
= dim-col (0m 1 (n - 1))
  using zero by auto
  fix i j assume i: i < dim-row (0m 1 (n - 1)) and j: j < dim-col (0m
1 (n - 1))
  have i0: i=0 using i by auto
  have 0 = Matrix.row (?first-row * V) 0 $v (j+1)
  using lt-first-row-V j unfolding lower-triangular-def
  by (metis Suc-eq-plus1 carrier-matD(2) index-mult-mat(2,3) in-
dex-row(1) less-diff-conv
mat-of-row-dim(1) zero zero-less-Suc zero-less-one-class.zero-less-one
V 2)
  also have ... = Matrix.row (A*V) 0 $v (j+1) by (simp add:
reduced-first-row)
  also have ... = (A*V) $$ (i, j+1) using V dim-row-AV i0 j by auto
  also have ... = four-block-mat a zero B C $$ (i, j+1) by (simp add:
AV-block)
  also have ... = (if i < dim-row a then if (j+1) < dim-col a
then a $$ (i, (j+1)) else zero $$ (i, (j+1) - dim-col a) else if (j+1)
< dim-col a
then B $$ (i - dim-row a, (j+1)) else C $$ (i - dim-row a, (j+1) -
dim-col a))
  by (rule index-mat-four-block, insert a zero i j C, auto)
  also have ... = zero $$ (i, (j+1) - dim-col a) using a zero i j C by
auto
  also have ... = zero $$ (i, j) using a i by auto
  finally show zero $$ (i, j) = 0m 1 (n - 1) $$ (i, j) using i j by auto
qed
have rw1: a * (1m 1) + zero * (0m (n-1) 1) = a using a zero by auto
have rw2: a * (0m 1 (n-1)) + zero * W = 0m 1 (n-1) using a zero
zero-eq W by auto
have rw3: B * (1m 1) + C * (0m (n-1) 1) = B using B C by auto
have rw4: B * (0m 1 (n-1)) + C * W = C * W using B C W by
auto
have A*V = four-block-mat a zero B C by (rule AV-block)
also have ... * ?W2 = four-block-mat (a * (1m 1) + zero * (0m (n-1)
1))
(a * (0m 1 (n-1)) + zero * W) (B * (1m 1) + C * (0m (n-1) 1))
(B * (0m 1 (n-1)) + C * W) by (rule mult-four-block-mat[OF a zero B
C], insert W, auto)
also have ... = ?T using rw1 rw2 rw3 rw4 by simp
finally have AVW2: A*V * ?W2 = ... .
moreover have lower-triangular ?T
using lt-CW unfolding lower-triangular-def using a zero B C W
by (auto, metis (full-types) Suc-less-eq Suc-pred basic-trans-rules(19))
ultimately show ?thesis by simp
qed
then show ?thesis using inv-V-W2 V W2 less.prem

```

```

      by (smt (verit) assoc-mult-mat mult-carrier-mat)
    qed
  qed
  qed
  qed
  thus admits-triangular-reduction A using A unfolding admits-triangular-reduction-def
  by simp
  qed

```

```

corollary admits-diagonal-imp-admits-triangular':
  assumes a:  $\forall A. \text{admits-diagonal-reduction } (A::'a \text{ mat})$ 
  shows  $\forall A. \text{admits-triangular-reduction } (A::'a \text{ mat})$ 
  using admits-diagonal-imp-admits-triangular assms by blast

```

```

lemma admits-triangular-reduction-1x2:
  assumes  $\forall A::'a \text{ mat}. A \in \text{carrier-mat } 1 \ 2 \longrightarrow \text{admits-triangular-reduction } A$ 
  shows  $\forall C::'a \text{ mat}. \text{admits-triangular-reduction } C$ 
  using admits-diagonal-imp-admits-triangular assms triangular-eq-diagonal-1x2
  by auto

```

```

lemma Hermite-ring-OFCLASS:
  assumes  $\forall A \in \text{carrier-mat } 1 \ 2. \text{admits-triangular-reduction } (A::'a \text{ mat})$ 
  shows OFCLASS('a, Hermite-ring-class)
proof
  show  $\forall A::'a \text{ mat}. \text{admits-triangular-reduction } A$ 
  by (rule admits-diagonal-imp-admits-triangular[OF assms[unfolded triangular-eq-diagonal-1x2]])
qed

```

```

lemma Hermite-ring-OFCLASS':
  assumes  $\forall A \in \text{carrier-mat } 1 \ 2. \text{admits-diagonal-reduction } (A::'a \text{ mat})$ 
  shows OFCLASS('a, Hermite-ring-class)
proof
  show  $\forall A::'a \text{ mat}. \text{admits-triangular-reduction } A$ 
  by (rule admits-diagonal-imp-admits-triangular[OF assms])
qed

```

```

lemma theorem3-part1:
  assumes T:  $(\forall a b::'a. \exists a1 b1 d. a = a1*d \wedge b = b1*d$ 
     $\wedge \text{ideal-generated } \{a1, b1\} = \text{ideal-generated } \{1\})$ 
  shows  $\forall A::'a \text{ mat}. \text{admits-triangular-reduction } A$ 
proof (rule admits-triangular-reduction-1x2, rule allI, rule impI)
  fix A::'a mat
  assume A:  $A \in \text{carrier-mat } 1 \ 2$ 
  let ?a = A $$ (0,0)
  let ?b = A $$ (0,1)
  obtain a1 b1 d where a:  $?a = a1*d$  and b:  $?b = b1*d$ 

```

**and**  $i$ : *ideal-generated*  $\{a1, b1\} = \text{ideal-generated } \{1\}$   
**using**  $T$  **by** *blast*  
**obtain**  $s\ t$  **where**  $sa1tb1:s*a1+t*b1=1$  **using** *ideal-generated-pair-exists-pq1* [*OF*  
 $i$  [*simplified*]] **by** *blast*  
**let**  $?Q = \text{Matrix.mat } 2\ 2$  ( $\lambda(i,j)$ . *if*  $i = 0 \wedge j = 0$  *then*  $s$  *else*  
*if*  $i = 0 \wedge j = 1$  *then*  $-b1$  *else*  
*if*  $i = 1 \wedge j = 0$  *then*  $t$  *else*  $a1$ )  
**have**  $Q$ :  $?Q \in \text{carrier-mat } 2\ 2$  **by** *auto*  
**have**  $\text{det-}Q$ : *Determinant.det*  $?Q = 1$  **unfolding**  $\text{det-}2$  [*OF*  $Q$ ]  
**using**  $sa1tb1$  **by** (*simp add: mult.commute*)  
**hence**  $\text{inv-}Q$ : *invertible-mat*  $?Q$  **using** *invertible-iff-is-unit-JNF* [*OF*  $Q$ ] **by** *auto*  
**have**  $\text{lower-}AQ$ : *lower-triangular* ( $A*?Q$ )  
**proof** –  
**have**  $\text{Matrix.row } A\ 0\ \$v\ \text{Suc } 0 * a1 = \text{Matrix.row } A\ 0\ \$v\ 0 * b1$  **if**  $j2: j < 2$   
**and**  $j0: 0 < j$  **for**  $j$   
**by** (*metis A One-nat-def a b carrier-matD(1) carrier-matD(2) index-row(1)*  
*lessI*  
*more-arith-simps(11) mult.commute numeral-2-eq-2 pos2*)  
**thus**  $?thesis$  **unfolding** *lower-triangular-def* **using**  $A$   
**by** (*auto simp add: scalar-prod-def sum-two-rw*)  
**qed**  
**show** *admits-triangular-reduction*  $A$   
**unfolding** *admits-triangular-reduction-def* **using**  $\text{lower-}AQ\ \text{inv-}Q\ Q\ A$  **by** *force*  
**qed**

**lemma** *theorem3-part2*:

**assumes**  $1: \forall A::'a\ \text{mat. admits-triangular-reduction } A$   
**shows**  $\forall a\ b::'a. \exists a1\ b1\ d. a = a1*d \wedge b = b1*d \wedge \text{ideal-generated } \{a1, b1\} =$   
*ideal-generated*  $\{1\}$   
**proof** (*rule allI*)  
**fix**  $a\ b::'a$   
**let**  $?A = \text{Matrix.mat } 1\ 2$  ( $\lambda(i,j)$ . *if*  $i = 0 \wedge j = 0$  *then*  $a$  *else*  $b$ )  
**obtain**  $Q$  **where**  $AQ$ : *lower-triangular* ( $?A*Q$ ) **and**  $\text{inv-}Q$ : *invertible-mat*  $Q$   
**and**  $Q$ :  $Q \in \text{carrier-mat } 2\ 2$   
**using**  $1$  **unfolding** *admits-triangular-reduction-def* **by** *fastforce*  
**hence** [*simp*]: *dim-col*  $Q = 2$  **and** [*simp*]: *dim-row*  $Q = 2$  **by** *auto*  
**let**  $?s = Q\ \$\$ (0,0)$   
**let**  $?t = Q\ \$\$ (1,0)$   
**let**  $?a1 = Q\ \$\$ (1,1)$   
**let**  $?b1 = -(Q\ \$\$ (0,1))$   
**let**  $?d = (?A*Q)\ \$\$ (0,0)$   
**have**  $ab1-ba1: a*?b1 = b*?a1$   
**proof** –  
**have**  $(?A*Q)\ \$\$ (0,1) = (\sum i = 0..<2. (\text{if } i = 0 \text{ then } a \text{ else } b) * Q\ \$\$ (i,$   
 $\text{Suc } 0))$   
**unfolding** *times-mat-def col-def scalar-prod-def* **by** *auto*  
**also have**  $\dots = (\sum i \in \{0,1\}. (\text{if } i = 0 \text{ then } a \text{ else } b) * Q\ \$\$ (i, \text{Suc } 0))$

by (rule sum.cong, auto)  
 also have ... = - a\*?b1 + b\*?a1 by auto  
 finally have (?A\*Q) \$\$ (0,1) = - a\*?b1 + b\*?a1 by simp  
 moreover have (?A\*Q) \$\$ (0,1) = 0 using AQ unfolding lower-triangular-def  
 by auto  
 ultimately show ?thesis  
 by (metis add-left-cancel more-arith-simps(3) more-arith-simps(7))  
 qed  
 have sa-tb-d: ?s\*a+?t\*b = ?d  
 proof -  
 have ?d = ( $\sum i = 0..<2$ . (if i = 0 then a else b) \* Q) \$\$ (i, 0)  
 unfolding times-mat-def col-def scalar-prod-def by auto  
 also have ... = ( $\sum i \in \{0,1\}$ . (if i = 0 then a else b) \* Q) \$\$ (i, 0) by (rule  
 sum.cong, auto)  
 also have ... = ?s\*a+?t\*b by auto  
 finally show ?thesis by simp  
 qed  
 have det-Q-dvd-1: (Determinant.det Q dvd 1)  
 using invertible-iff-is-unit-JNF[OF Q] inv-Q by auto  
 moreover have det-Q-eq: Determinant.det Q = ?s\*?a1 + ?t\*?b1 unfolding  
 det-2[OF Q] by simp  
 ultimately have ?s\*?a1 + ?t\*?b1 dvd 1 by auto  
 from this obtain u where u-eq: ?s\*?a1 + ?t\*?b1 = u and u: u dvd 1 by auto  
 hence eq1: ?s\*?a1\*a + ?t\*?b1\*a = u\*a  
 by (metis ring-class.ring-distrib(2))  
 hence ?s\*?a1\*a + ?t\*?a1\*b = u\*a  
 by (metis (no-types, lifting) ab1-ba1 mult.assoc mult.commute)  
 hence a1d-ua: ?a1\*?d=u\*a  
 by (smt (verit) Groups.mult-ac(2) distrib-left more-arith-simps(11) sa-tb-d)  
 hence b1d-ub: ?b1\*?d=u\*b  
 by (smt (verit) Groups.mult-ac(2) Groups.mult-ac(3) ab1-ba1 distrib-right  
 sa-tb-d u-eq)  
 obtain inv-u where inv-u: inv-u \* u = 1 using u unfolding dvd-def  
 by (metis mult.commute)  
 hence inv-u-dvd-1: inv-u dvd 1 unfolding dvd-def by auto  
 have cond1: (inv-u\*?b1)\*?d = b using b1d-ub inv-u  
 by (metis (no-types, lifting) Groups.mult-ac(3) more-arith-simps(11) more-arith-simps(6))  
 have cond2: (inv-u\*?a1)\*?d = a using a1d-ua inv-u  
 by (metis (no-types, lifting) Groups.mult-ac(3) more-arith-simps(11) more-arith-simps(6))  
 have ideal-generated {inv-u\*?a1, inv-u\*?b1} = ideal-generated {?a1,?b1}  
 by (rule ideal-generated-mult-unit2[OF inv-u-dvd-1])  
 also have ... = UNIV using ideal-generated-pair-UNIV[OF u-eq u] by simp  
 finally have cond3: ideal-generated {inv-u\*?a1, inv-u\*?b1} = ideal-generated  
 {1} by auto  
 show  $\exists a1\ b1\ d. a = a1 * d \wedge b = b1 * d \wedge$  ideal-generated {a1, b1} =  
 ideal-generated {1}  
 by (rule exI[of - inv-u\*?a1], rule exI[of - inv-u\*?b1], rule exI[of - ?d],  
 insert cond1 cond2 cond3, auto)  
 qed

```

theorem theorem3:
  shows ( $\forall A::'a \text{ mat. admits-triangular-reduction } A$ )
    = ( $\forall a b::'a. \exists a1 b1 d. a = a1*d \wedge b = b1*d \wedge \text{ideal-generated } \{a1,b1\} =$ 
ideal-generated  $\{1\}$ )
  using theorem3-part1 theorem3-part2 by auto

end

context comm-ring-1
begin

lemma lemma4-prev:
  assumes a:  $a = a1*d$  and b:  $b = b1*d$ 
  and i: ideal-generated  $\{a1,b1\} = \text{ideal-generated } \{1\}$ 
  shows ideal-generated  $\{a,b\} = \text{ideal-generated } \{d\}$ 
  proof -
  have  $1: \exists k. p * (a1 * d) + q * (b1 * d) = k * d$  for  $p \ q$ 
    by (metis (full-types) local.distrib-right local.mult.semigroup-axioms semigroup.assoc)

  have ideal-generated  $\{a,b\} \subseteq \text{ideal-generated } \{d\}$ 
  proof -
    have ideal-generated  $\{a,b\} = \{p*a+q*b \mid p \ q. \text{True}\}$  using ideal-generated-pair
  by auto
    also have  $\dots = \{p*(a1*d)+q*(b1*d) \mid p \ q. \text{True}\}$  using a b by auto
    also have  $\dots \subseteq \{k*d \mid k. \text{True}\}$  using  $1$  by auto
    finally show ?thesis
    by (simp add: a b local.dvd-ideal-generated-singleton' local.ideal-generated-subset2)
  qed
  moreover have ideal-generated $\{d\} \subseteq \text{ideal-generated } \{a,b\}$ 
  proof (rule ideal-generated-singleton-subset)
    obtain  $p \ q$  where  $p*a1+q*b1 = 1$  using ideal-generated-pair-exists-UNIV i
  by auto
    hence  $d = p * (a1 * d) + q * (b1 * d)$ 
    by (metis local.mult-ac(3) local.ring-distrib(1) local.semiring-normalization-rules(12))
    also have  $\dots \in \{p*(a1*d)+q*(b1*d) \mid p \ q. \text{True}\}$  by auto
    also have  $\dots = \text{ideal-generated } \{a,b\}$  unfolding ideal-generated-pair a b by
auto
    finally show  $d \in \text{ideal-generated } \{a,b\}$  by simp
  qed (simp)
  ultimately show ?thesis by simp
qed

```

```

lemma lemma4:

```

```

assumes  $a: a = a1*d$  and  $b: b = b1*d$ 
  and  $i: \text{ideal-generated } \{a1,b1\} = \text{ideal-generated } \{1\}$ 
  and  $i2: \text{ideal-generated } \{a,b\} = \text{ideal-generated } \{d'\}$ 
shows  $\exists a1' b1'. a = a1' * d' \wedge b = b1' * d'$ 
   $\wedge \text{ideal-generated } \{a1',b1'\} = \text{ideal-generated } \{1\}$ 
proof –
  have  $i3: \text{ideal-generated } \{a,b\} = \text{ideal-generated } \{d\}$  using lemma4-prev assms
by auto
  have  $d\text{-dvd-}d': d \text{ dvd } d'$ 
    by (metis a b i2 dvd-ideal-generated-singleton dvd-ideal-generated-singleton'
      dvd-triv-right ideal-generated-subset2)
  have  $d'\text{-dvd-}d: d' \text{ dvd } d$ 
    using  $i3 i2 \text{local.dvd-ideal-generated-singleton}$  by auto
  obtain  $k$  and  $l$  where  $d: d = k*d'$  and  $d': d' = l*d$ 
    using  $d\text{-dvd-}d' d'\text{-dvd-}d \text{mult-ac unfolding dvd-def}$  by auto
  obtain  $s t$  where  $sa1\text{-}tb1: s*a1 + t*b1 = 1$ 
    using  $i \text{ideal-generated-pair-exists-UNIV}[of a1 b1]$  by auto
  let  $?a1' = k * l * t - t + a1 * k$ 
  let  $?b1' = s - k * l * s + b1 * k$ 
  have  $1: ?a1'*d'=a$ 
    by (metis a d d' add-ac(2) add-diff-cancel add-diff-eq mult-ac(2) ring-distrib(1,4)
      semiring-normalization-rules(18))
  have  $2: ?b1'*d' = b$ 
    by (metis (no-types, opaque-lifting) b d d' add-ac(2) add-diff-cancel add-diff-eq
      mult-ac(2) mult-ac(3) ring-distrib(2,4) semiring-normalization-rules(18))
  have  $(s*l-b1)*?a1' + (t*l+a1)*?b1' = 1$ 
proof –
  have  $aux\text{-}rw1: s * l * k * l * t = t * l * k * l * s$  and  $aux\text{-}rw2: s * l * t = t * l * s$ 
  and  $aux\text{-}rw3: b1 * a1 * k = a1 * b1 * k$  and  $aux\text{-}rw4: t * l * b1 * k = b1 * k * l * t$ 
  and  $aux\text{-}rw5: s * l * a1 * k = a1 * k * l * s$ 
    using mult.commute mult.assoc by auto
  note  $aux\text{-}rw = aux\text{-}rw1 aux\text{-}rw2 aux\text{-}rw3 aux\text{-}rw4 aux\text{-}rw5$ 
  have  $(s*l-b1)*?a1' + (t*l+a1)*?b1' = s*l*?a1' - b1*?a1' + t*l*?b1' + a1*?b1'$ 
    using local.add-ac(1) local.left-diff-distrib' local.ring-distrib(2) by auto
  also have  $\dots = s * l * k * l * t - s * l * t + s * l * a1 * k - b1 * k * l * t +$ 
 $b1 * t - b1 * a1 * k$ 
   $+ t * l * s - t * l * k * l * s + t * l * b1 * k + a1 * s - a1 * k * l * s + a1$ 
 $* b1 * k$ 
    by (smt (verit) local.add-diff-eq local.diff-add-eq local.diff-diff-eq2 local.mult-ac(1)
      local.ring-distrib(4))
  also have  $\dots = a1 * s + b1 * t$  unfolding  $aux\text{-}rw$ 
    by (smt (verit, ccfv-SIG) local.add-diff-cancel-left' local.diff-add-eq local.eq-diff-eq)
  also have  $\dots = 1$  using  $sa1\text{-}tb1 \text{mult.commute}$  by auto
  finally show  $?thesis$  by simp
qed

```



hence  $\text{ideal-generated } \{?a1', ?b1'\} = \text{ideal-generated } \{1\}$   
 using  $\text{ideal-generated-pair-exists-UNIV}[of ?a1' ?b1']$  by *auto*  
 thus *?thesis* using 1 2 by *auto*  
 qed

lemma *corollary5*:

assumes  $T: \forall a b. \exists a1 b1 d. a = a1 * d \wedge b = b1 * d$   
 $\wedge \text{ideal-generated } \{a1, b1\} = \text{ideal-generated } \{1::'a\}$   
 and  $i2: \text{ideal-generated } \{a, b, c\} = \text{ideal-generated } \{d\}$   
 shows  $\exists a1 b1 c1. a = a1 * d \wedge b = b1 * d \wedge c = c1 * d$   
 $\wedge \text{ideal-generated } \{a1, b1, c1\} = \text{ideal-generated } \{1\}$   
 proof –  
 have  $da: d \text{ dvd } a$  using  $\text{ideal-generated-singleton-dvd}[OF i2]$  by *auto*  
 have  $db: d \text{ dvd } b$  using  $\text{ideal-generated-singleton-dvd}[OF i2]$  by *auto*  
 have  $dc: d \text{ dvd } c$  using  $\text{ideal-generated-singleton-dvd}[OF i2]$  by *auto*  
 from *this* obtain  $c1'$  where  $c: c = c1' * d$  using  $\text{dvd-def mult-ac}(2)$  by *auto*  
 obtain  $a1 b1 d'$  where  $a: a = a1 * d'$  and  $b: b = b1 * d'$   
 and  $i: \text{ideal-generated } \{a1, b1\} = \text{ideal-generated } \{1::'a\}$  using  $T$  by *blast*  
 have  $i\text{-ab-}d': \text{ideal-generated } \{a, b\} = \text{ideal-generated } \{d'\}$   
 by (*simp add: a b i lemma4-prev*)  
 have  $i2: \text{ideal-generated } \{d', c\} = \text{ideal-generated } \{d\}$   
 by (*rule ideal-generated-triple-pair-rewrite[OF i2 i-ab-d']*)  
 obtain  $u v dp$  where  $d'1: d' = u * dp$  and  $d'2: c = v * dp$   
 and  $xy: \text{ideal-generated}\{u, v\} = \text{ideal-generated}\{1\}$  using  $T$  by *blast*  
 have  $\exists a1' b1'. d' = a1' * d \wedge c = b1' * d \wedge \text{ideal-generated } \{a1', b1'\} =$   
 $\text{ideal-generated } \{1\}$   
 by (*rule lemma4[OF d'1 d'2 xy i2]*)  
 from *this* obtain  $a1' c1$  where  $d'\text{-}a1: d' = a1' * d$  and  $c: c = c1 * d$   
 and  $i3: \text{ideal-generated } \{a1', c1\} = \text{ideal-generated } \{1\}$  by *blast*  
 have  $r1: a = a1 * a1' * d$  by (*simp add: d'\text{-}a1 a local.semiring-normalization-rules(18)*)  
 have  $r2: b = b1 * a1' * d$  by (*simp add: d'\text{-}a1 b local.semiring-normalization-rules(18)*)  
 have  $i4: \text{ideal-generated } \{a1 * a1', b1 * a1', c1\} = \text{ideal-generated } \{1\}$   
 proof –  
 obtain  $p q$  where  $1: p * a1' + q * c1 = 1$   
 using  $i3$  unfolding  $\text{ideal-generated-pair-exists-UNIV}$  by *auto*  
 obtain  $x y$  where  $2: x * a1 + y * b1 = p$  using  $\text{ideal-generated-UNIV-obtain-pair}[OF$   
 $i]$  by *blast*  
 have  $1 = (x * a1 + y * b1) * a1' + q * c1$  using 1 2 by *auto*  
 also have  $\dots = x * a1 * a1' + y * b1 * a1' + q * c1$  by (*simp add: local.ring-distrib(2)*)  
 finally have  $1 = x * a1 * a1' + y * b1 * a1' + q * c1$ .  
 hence  $1 \in \text{ideal-generated } \{a1 * a1', b1 * a1', c1\}$   
 using  $\text{ideal-explicit2}[of \{a1 * a1', b1 * a1', c1\}]$  *sum-three-elements'*  
 by (*simp add: mult-assoc*)  
 hence  $\text{ideal-generated } \{1\} \subseteq \text{ideal-generated } \{a1 * a1', b1 * a1', c1\}$   
 by (*rule ideal-generated-singleton-subset, auto*)  
 thus *?thesis* by *auto*  
 qed

```

  show ?thesis using r1 r2 i4 c by auto
qed

end

context
  assumes SORT-CONSTRAINT('a::comm-ring-1)
begin

lemma OFCLASS-elementary-divisor-ring-imp-class:
  assumes OFCLASS('a::comm-ring-1, elementary-divisor-ring-class)
  shows class.elementary-divisor-ring TYPE('a)
  by (rule conjunctionD2[OF assms[unfolded elementary-divisor-ring-class-def]])

corollary Elementary-divisor-ring-imp-Hermite-ring:
  assumes OFCLASS('a::comm-ring-1, elementary-divisor-ring-class)
  shows OFCLASS('a::comm-ring-1, Hermite-ring-class)
proof
  have  $\forall A::'a \text{ mat. admits-diagonal-reduction } A$ 
    using OFCLASS-elementary-divisor-ring-imp-class[OF assms]
    unfolding class.elementary-divisor-ring-def by auto
  thus  $\forall A::'a \text{ mat. admits-triangular-reduction } A$ 
    using admits-diagonal-imp-admits-triangular by auto
qed

corollary Elementary-divisor-ring-imp-Bezout-ring:
  assumes OFCLASS('a::comm-ring-1, elementary-divisor-ring-class)
  shows OFCLASS('a::comm-ring-1, bezout-ring-class)
  by (rule Hermite-ring-imp-Bezout-ring, rule Elementary-divisor-ring-imp-Hermite-ring[OF
  assms])

```

## 18.5 Characterization of Elementary divisor rings

```

lemma necessity-D':
  assumes edr: ( $\forall (A::'a \text{ mat}). \text{ admits-diagonal-reduction } A$ )
  shows  $\forall a \ b \ c::'a. \text{ ideal-generated } \{a,b,c\} = \text{ ideal-generated}\{1\}$ 
   $\longrightarrow (\exists p \ q. \text{ ideal-generated } \{p*a,p*b+q*c\} = \text{ ideal-generated } \{1\})$ 
proof ((rule allI)+, rule impI)
  fix a b c::'a
  assume i: ideal-generated {a,b,c} = ideal-generated{1}
  define A where A = Matrix.mat 2 2 ( $\lambda(i,j). \text{ if } i = 0 \wedge j = 0 \text{ then } a \text{ else}$ 
    if  $i = 0 \wedge j = 1$  then b else
    if  $i = 1 \wedge j = 0$  then 0 else c)
  have A:  $A \in \text{ carrier-mat } 2 \ 2$  unfolding A-def by auto
  obtain P Q where P:  $P \in \text{ carrier-mat } (\text{dim-row } A) (\text{dim-row } A)$ 

```

```

    and Q: Q ∈ carrier-mat (dim-col A) (dim-col A)
    and inv-P: invertible-mat P and inv-Q: invertible-mat Q
    and SNF-PAQ: Smith-normal-form-mat (P * A * Q)
    using edr unfolding admits-diagonal-reduction-def by blast
    have [simp]: dim-row P = 2 and [simp]: dim-col P = 2 and [simp]: dim-row Q
= 2
    and [simp]: dim-col Q = 2 and [simp]: dim-col A = 2 and [simp]: dim-row A
= 2
    using A P Q by auto
    define u where u = (P*A*Q) $$ (0,0)
    define p where p = P $$ (0,0)
    define q where q = P $$ (0,1)
    define x where x = Q $$ (0,0)
    define y where y = Q $$ (1,0)
    have eq: p*a*x + p*b*y + q*c*y = u
    proof -
      have rw1: (∑ ia = 0..<2. P $$ (0, ia) * A $$ (ia, x)) * Q $$ (x, 0)
= (∑ ia∈{0,1}. P $$ (0, ia) * A $$ (ia, x)) * Q $$ (x, 0)
      for x by (unfold sum-distrib-right, rule sum.cong, auto)
      have u = (∑ i = 0..<2. (∑ ia = 0..<2. P $$ (0, ia) * A $$ (ia, i)) * Q $$
(i, 0))
      unfolding u-def p-def q-def x-def y-def
      unfolding times-mat-def scalar-prod-def by auto
      also have ... = (∑ i ∈{0,1}. (∑ ia ∈ {0,1}. P $$ (0, ia) * A $$ (ia, i)) * Q
$$ (i, 0))
      by (rule sum.cong[OF - rw1], auto)
      also have ... = p*a*x + p*b*y+q*c*y
      unfolding u-def p-def q-def x-def y-def A-def
      using ring-class.ring-distrib(2) by auto
      finally show ?thesis ..
    qed
    have u-dvd-1: u dvd 1

    proof (rule ideal-generated-dvd2[OF i])
      define D where D = (P*A*Q)
      obtain P' where P'[simp]: P' ∈ carrier-mat 2 2 and inv-P: inverts-mat P'
P
      using inv-P obtain-inverse-matrix[OF P inv-P]
      by (metis ‹dim-row A = 2›)
      obtain Q' where [simp]: Q' ∈ carrier-mat 2 2 and inv-Q: inverts-mat Q Q'
      using inv-Q obtain-inverse-matrix[OF Q inv-Q]
      by (metis ‹dim-col A = 2›)
      have D[simp]: D ∈ carrier-mat 2 2 unfolding D-def by auto
      have e: P' * D * Q' = A unfolding D-def by (rule inv-P'PAQQ'[OF - - inv-P
inv-Q], auto)
      have [simp]: (P' * D) ∈ carrier-mat 2 2 using D P' mult-carrier-mat by blast
      have D-01: D $$ (0, 1) = 0
      using D-def SNF-PAQ unfolding Smith-normal-form-mat-def isDiagonal-mat-def
      by force

```

**have**  $D-10: D \text{ \textit{\$} } (1, 0) = 0$   
**using**  $D\text{-def SNF-PAQ unfolding Smith-normal-form-mat-def isDiagonal-mat-def}$   
**by** *force*  
**have**  $D \text{ \textit{\$} } (0,0) \text{ dvd } D \text{ \textit{\$} } (1, 1)$   
**using**  $D\text{-def SNF-PAQ unfolding Smith-normal-form-mat-def by auto}$   
**from this obtain**  $k$  **where**  $D11: D \text{ \textit{\$} } (1, 1) = D \text{ \textit{\$} } (0,0) * k$  **unfolding**  
*dvd-def by blast*  
**have**  $P'D-00: (P' * D) \text{ \textit{\$} } (0, 0) = P' \text{ \textit{\$} } (0, 0) * D \text{ \textit{\$} } (0, 0)$   
**using**  $mat\text{-mult2-00}[of P' D] D-10$  **by** *auto*  
**have**  $P'D-01: (P' * D) \text{ \textit{\$} } (0, 1) = P' \text{ \textit{\$} } (0, 1) * D \text{ \textit{\$} } (1, 1)$   
**using**  $mat\text{-mult2-01}[of P' D] D-01$  **by** *auto*  
**have**  $P'D-10: (P' * D) \text{ \textit{\$} } (1, 0) = P' \text{ \textit{\$} } (1, 0) * D \text{ \textit{\$} } (0, 0)$   
**using**  $mat\text{-mult2-10}[of P' D] D-10$  **by** *auto*  
**have**  $P'D-11: (P' * D) \text{ \textit{\$} } (1, 1) = P' \text{ \textit{\$} } (1, 1) * D \text{ \textit{\$} } (1, 1)$   
**using**  $mat\text{-mult2-11}[of P' D] D-01$  **by** *auto*  
**have**  $a = (P' * D * Q') \text{ \textit{\$} } (0,0)$  **using**  $e A\text{-def}$  **by** *auto*  
**also have**  $\dots = (P' * D) \text{ \textit{\$} } (0, 0) * Q' \text{ \textit{\$} } (0, 0) + (P' * D) \text{ \textit{\$} } (0, 1) * Q'$   
 $\text{ \textit{\$} } (1, 0)$   
**by**  $(rule\ mat\text{-mult2-00}, auto)$   
**also have**  $\dots = P' \text{ \textit{\$} } (0, 0) * D \text{ \textit{\$} } (0, 0) * Q' \text{ \textit{\$} } (0, 0)$   
 $+ P' \text{ \textit{\$} } (0, 1) * (D \text{ \textit{\$} } (0, 0) * k) * Q' \text{ \textit{\$} } (1, 0)$  **unfolding**  $P'D-00 P'D-01$   
 $D11 ..$   
**also have**  $\dots = D \text{ \textit{\$} } (0, 0) * (P' \text{ \textit{\$} } (0, 0) * Q' \text{ \textit{\$} } (0, 0)$   
 $+ P' \text{ \textit{\$} } (0, 1) * k * Q' \text{ \textit{\$} } (1, 0))$  **by**  $(simp\ add: distrib\text{-left})$   
**finally have**  $u\text{-dvd}\text{-}a: u \text{ dvd } a$  **unfolding**  $u\text{-def } D\text{-def } dvd\text{-def}$  **by** *auto*  
**have**  $b = (P' * D * Q') \text{ \textit{\$} } (0,1)$  **using**  $e A\text{-def}$  **by** *auto*  
**also have**  $\dots = (P' * D) \text{ \textit{\$} } (0, 0) * Q' \text{ \textit{\$} } (0, 1) + (P' * D) \text{ \textit{\$} } (0, 1) * Q'$   
 $\text{ \textit{\$} } (1, 1)$   
**by**  $(rule\ mat\text{-mult2-01}, auto)$   
**also have**  $\dots = P' \text{ \textit{\$} } (0, 0) * D \text{ \textit{\$} } (0, 0) * Q' \text{ \textit{\$} } (0, 1) +$   
 $P' \text{ \textit{\$} } (0, 1) * (D \text{ \textit{\$} } (0, 0) * k) * Q' \text{ \textit{\$} } (1, 1)$   
**unfolding**  $P'D-00 P'D-01 D11 ..$   
**also have**  $\dots = D \text{ \textit{\$} } (0, 0) * (P' \text{ \textit{\$} } (0, 0) * Q' \text{ \textit{\$} } (0, 1) +$   
 $P' \text{ \textit{\$} } (0, 1) * k * Q' \text{ \textit{\$} } (1, 1))$  **by**  $(simp\ add: distrib\text{-left})$   
**finally have**  $u\text{-dvd}\text{-}b: u \text{ dvd } b$  **unfolding**  $u\text{-def } D\text{-def } dvd\text{-def}$  **by** *auto*  
**have**  $c = (P' * D * Q') \text{ \textit{\$} } (1,1)$  **using**  $e A\text{-def}$  **by** *auto*  
**also have**  $\dots = (P' * D) \text{ \textit{\$} } (1, 0) * Q' \text{ \textit{\$} } (0, 1) + (P' * D) \text{ \textit{\$} } (1, 1) * Q'$   
 $\text{ \textit{\$} } (1, 1)$   
**by**  $(rule\ mat\text{-mult2-11}, auto)$   
**also have**  $\dots = P' \text{ \textit{\$} } (1, 0) * D \text{ \textit{\$} } (0, 0) * Q' \text{ \textit{\$} } (0, 1)$   
 $+ P' \text{ \textit{\$} } (1, 1) * (D \text{ \textit{\$} } (0, 0) * k) * Q' \text{ \textit{\$} } (1, 1)$  **unfolding**  $P'D-11 P'D-10$   
 $D11 ..$   
**also have**  $\dots = D \text{ \textit{\$} } (0, 0) * (P' \text{ \textit{\$} } (1, 0) * Q' \text{ \textit{\$} } (0, 1)$   
 $+ P' \text{ \textit{\$} } (1, 1) * k * Q' \text{ \textit{\$} } (1, 1))$  **by**  $(simp\ add: distrib\text{-left})$   
**finally have**  $u\text{-dvd}\text{-}c: u \text{ dvd } c$  **unfolding**  $u\text{-def } D\text{-def } dvd\text{-def}$  **by** *auto*  
**show**  $\forall x \in \{a,b,c\}. u \text{ dvd } x$  **using**  $u\text{-dvd}\text{-}a\ u\text{-dvd}\text{-}b\ u\text{-dvd}\text{-}c$  **by** *auto*  
**qed**  $(simp)$   
**have**  $ideal\text{-generated } \{p*a, p*b+q*c\} = ideal\text{-generated } \{1\}$   
**by**  $(metis\ (no\text{-types}, lifting)\ eq\ add.\text{assoc}\ ideal\text{-generated}\text{-}1\ ideal\text{-generated}\text{-}pair\text{-UNIV}$

*mult.commute semiring-normalization-rules(34) u-dvd-1*  
**from this show**  $\exists p q. \text{ideal-generated } \{p * a, p * b + q * c\} = \text{ideal-generated } \{1\}$   
**by auto**  
**qed**

**lemma necessity:**

**assumes**  $(\forall (A::'a \text{ mat}). \text{admits-diagonal-reduction } A)$   
**shows**  $(\forall (A::'a \text{ mat}). \text{admits-triangular-reduction } A)$   
**and**  $\forall a b c::'a. \text{ideal-generated}\{a,b,c\} = \text{ideal-generated}\{1\}$   
 $\rightarrow (\exists p q. \text{ideal-generated } \{p*a,p*b+q*c\} = \text{ideal-generated } \{1\})$   
**using** *necessity-D' admits-diagonal-imp-admits-triangular assms*  
**by** *blast+*

In the article, the authors change the notation and assume  $(a, b, c) = (1)$ . However, we have to provide here the complete prove. To to this, I obtained a  $D$  matrix such that  $A' = A * D$  and  $D$  is a diagonal matrix with  $d$  in the diagonal. Proving that  $D$  is left and right commutative, I can follow the reasoning in the article

**lemma sufficiency:**

**assumes** *hermite-ring*:  $(\forall (A::'a \text{ mat}). \text{admits-triangular-reduction } A)$   
**and**  $D'$ :  $\forall a b c::'a. \text{ideal-generated}\{a,b,c\} = \text{ideal-generated}\{1\}$   
 $\rightarrow (\exists p q. \text{ideal-generated } \{p*a,p*b+q*c\} = \text{ideal-generated } \{1\})$   
**shows**  $(\forall (A::'a \text{ mat}). \text{admits-diagonal-reduction } A)$

**proof** –

**have** *admits-1x2*:  $\forall (A::'a \text{ mat}) \in \text{carrier-mat } 1 \ 2. \text{admits-diagonal-reduction } A$   
**using** *hermite-ring triangular-eq-diagonal-1x2* **by** *blast*

**have** *admits-2x2*:  $\forall (A::'a \text{ mat}) \in \text{carrier-mat } 2 \ 2. \text{admits-diagonal-reduction } A$

**proof**

**fix**  $B::'a \text{ mat}$  **assume**  $B: B \in \text{carrier-mat } 2 \ 2$

**obtain**  $U$  **where**  $BU$ : *lower-triangular*  $(B*U)$  **and**  $inv-U$ : *invertible-mat*  $U$

**and**  $U$ :  $U \in \text{carrier-mat } 2 \ 2$

**using** *hermite-ring unfolding admits-triangular-reduction-def* **using**  $B$  **by** *fastforce*

**define**  $A$  **where**  $A = B*U$

**define**  $a$  **where**  $a = A \ \$\$ (0,0)$

**define**  $b$  **where**  $b = A \ \$\$ (1,0)$

**define**  $c$  **where**  $c = A \ \$\$ (1,1)$

**have**  $A$ :  $A \in \text{carrier-mat } 2 \ 2$  **using**  $U B A\text{-def}$  **by** *auto*

**have**  $A-01$ :  $A \ \$\$ (0,1) = 0$  **using**  $BU U B$  **unfolding** *lower-triangular-def A-def*

**by** *auto*

**obtain**  $d::'a$  **where**  $i$ :  $\text{ideal-generated } \{a,b,c\} = \text{ideal-generated } \{d\}$

**proof** –

**have** *OFCLASS*( $'a$ , *bezout-ring-class*) **by** (*rule Hermite-ring-imp-Bezout-ring*,  
*insert OFCLASS-Hermite-ring-def*[**where**  $?'a='a$ ] *hermite-ring*, *auto*)

**hence** *class.bezout-ring*  $(*) (1::'a) (+) 0 (-) \text{uminus}$

**using** *OFCLASS-bezout-ring-imp-class-bezout-ring*[**where**  $?'a = 'a$ ] **by** *auto*  
**hence**  $(\forall I::'a::\text{comm-ring-1 set. finitely-generated-ideal } I \longrightarrow \text{principal-ideal } I)$

**using** *bezout-ring-iff-fin-gen-principal-ideal2* **by** *auto*  
**moreover have** *finitely-generated-ideal (ideal-generated {a,b,c})*  
**unfolding** *finitely-generated-ideal-def*  
**using** *ideal-ideal-generated* **by** *force*  
**ultimately have** *principal-ideal (ideal-generated {a,b,c})* **by** *auto*  
**thus** *?thesis using that unfolding principal-ideal-def* **by** *auto*  
**qed**  
**have** *d-dvd-a: d dvd a and d-dvd-b: d dvd b and d-dvd-c: d dvd c*  
**using** *i ideal-generated-singleton-dvd* **by** *blast+*  
**obtain** *a1 b1 c1 where a1: a = a1 \* d and b1: b = b1 \* d and c1: c = c1*  
*\* d*  
**and** *i2: ideal-generated {a1,b1,c1} = ideal-generated {1}*  
**proof** –  
**have** *T:  $\forall a b. \exists a1 b1 d. a = a1 * d \wedge b = b1 * d$*   
 $\wedge \text{ideal-generated } \{a1, b1\} = \text{ideal-generated } \{1::'a\}$   
**by** *(rule theorem3-part2[OF hermite-ring])*  
**from this obtain** *a1' b1' d' where 1: a = a1' \* d' and 2: b = b1' \* d'*  
**and** *3: ideal-generated {a1', b1'} = ideal-generated {1::'a}* **by** *blast*  
**have**  $\exists a1 b1 c1. a = a1 * d \wedge b = b1 * d \wedge c = c1 * d$   
 $\wedge \text{ideal-generated } \{a1, b1, c1\} = \text{ideal-generated } \{1\}$   
**by** *(rule corollary5[OF T i])*  
**from this show** *?thesis using that* **by** *auto*  
**qed**

**define** *D where*  $D = d \cdot_m (1_m \ 2)$   
**define** *A' where*  $A' = \text{Matrix.mat } 2 \ 2 (\lambda(i,j). \text{if } i = 0 \wedge j = 0 \text{ then } a1 \text{ else}$   
 $\text{if } i = 1 \wedge j = 0 \text{ then } b1 \text{ else}$   
 $\text{if } i = 0 \wedge j = 1 \text{ then } 0 \text{ else } c1)$   
**have** *D: D  $\in$  carrier-mat 2 2 and A': A'  $\in$  carrier-mat 2 2* **unfolding** *A'-def*  
*D-def* **by** *auto*  
**have** *A-A'D: A = A' \* D*  
**by** *(rule eq-matI, insert D A' A a1 b1 c1 A-01 sum-two-rw a-def b-def c-def,*  
*unfold scalar-prod-def Matrix.row-def col-def D-def A'-def,*  
*auto simp add: sum-two-rw less-Suc-eq numerals(2))*  
**have**  $1 \in \text{ideal-generated}\{a1, b1, c1\}$  **using** *i2* **by** *(simp add: ideal-generated-in)*

**from this obtain** *f where*  $d: (\sum_{i \in \{a1, b1, c1\}} f i * i) = 1$   
**using** *ideal-explicit2[of {a1,b1,c1}]* **by** *auto*  
**from this obtain** *x y z where*  $x*a1+y*b1+z*c1 = 1$   
**using** *sum-three-elements[of - a1 b1 c1]* **by** *metis*  
**hence** *xa1-yb1-zc1-dvd-1: x \* a1 + y \* b1 + z \* c1 dvd 1* **by** *auto*  
**obtain** *p q where*  $i3: \text{ideal-generated } \{p*a1, p*b1+q*c1\} = \text{ideal-generated}$   
 $\{1\}$   
**using** *D' i2* **by** *blast*  
**have** *ideal-generated {p,q} = UNIV*  
**proof** –

```

obtain  $X\ Y$  where  $e: X*p*a1 + Y*(p*b1+q*c1) = 1$ 
  by (metis i3 ideal-generated-1 ideal-generated-pair-exists-UNIV mult.assoc)
have  $X*p*a1 + Y*(p*b1+q*c1) = X*p*a1 + Y*p*b1+Y*q*c1$ 
  by (simp add: add.assoc mult.assoc semiring-normalization-rules(34))
also have  $\dots = (X*a1+Y*b1) * p + (Y * c1) * q$ 
  by (simp add: mult.commute ring-class.ring-distrib)
finally have  $(X*a1+Y*b1) * p + Y * c1 * q = 1$  using  $e$  by simp
from this show ?thesis by (rule ideal-generated-pair-UNIV, simp)
qed
from this obtain  $u\ v$  where  $pu-qv-1: p*u - q * v = 1$ 
  by (metis Groups.mult-ac(2) diff-minus-eq-add ideal-generated-1
    ideal-generated-pair-exists-UNIV mult-minus-left)
let  $?P = \text{Matrix.mat } 2\ 2\ (\lambda(i,j). \text{if } i = 0 \wedge j = 0 \text{ then } p \text{ else}$ 
   $\text{if } i = 1 \wedge j = 0 \text{ then } q \text{ else}$ 
   $\text{if } i = 0 \wedge j = 1 \text{ then } v \text{ else } u)$ 
have  $P: ?P \in \text{carrier-mat } 2\ 2$  by auto
have Determinant.det  $?P = 1$  using  $pu-qv-1$  unfolding  $\text{det-2}[OF\ P]$  by (simp
add: mult.commute)
hence  $\text{inv-}P: \text{invertible-mat } ?P$ 
  by (metis (no-types, lifting) P dvd-refl invertible-iff-is-unit-JNF)
define  $S1$  where  $S1 = A'*?P$ 
have  $S1: S1 \in \text{carrier-mat } 2\ 2$  using  $A'\ P\ S1\text{-def}$  mult-carrier-mat by blast
have  $S1-00: S1\ \$\$(0,0) = p*a1$  and  $S1-01: S1\ \$\$(1,0) = p*b1+q*c1$ 
  unfolding  $S1\text{-def}$   $\text{times-mat-def}$   $\text{scalar-prod-def}$  using  $A'\ P\ BU\ UB$ 
  unfolding  $A'\text{-def}$   $\text{upper-triangular-def}$ 
  by (auto, unfold sum-two-rw, auto simp add: A'-def a-def b-def c-def)
obtain  $q00$  and  $q01$  where  $q00-q01: p*a1*q00 + (p*b1+q*c1)*q01 = 1$ 
using i3
  by (metis ideal-generated-1 ideal-generated-pair-exists-pq1 mult.commute)
define  $q10$  where  $q10 = -(p*b1+q*c1)$ 
define  $q11$  where  $q11 = p*a1$ 
have  $q10-q11: p*a1*q10 + (p*b1+q*c1)*q11 = 0$  unfolding  $q10\text{-def}$   $q11\text{-def}$ 
  by (auto simp add: Rings.ring-distrib(1) Rings.ring-distrib(4) semiring-normalization-rules(7))

let  $?Q = \text{Matrix.mat } 2\ 2\ (\lambda(i,j). \text{if } i = 0 \wedge j = 0 \text{ then } q00 \text{ else}$ 
   $\text{if } i = 1 \wedge j = 0 \text{ then } q10 \text{ else}$ 
   $\text{if } i = 0 \wedge j = 1 \text{ then } q01 \text{ else } q11)$ 
have  $Q: ?Q \in \text{carrier-mat } 2\ 2$  by auto
have Determinant.det  $?Q = 1$  using  $q00-q01$  unfolding  $\text{det-2}[OF\ Q]$  unfolding
 $q10\text{-def}$   $q11\text{-def}$ 
  by (auto, metis (no-types, lifting) add-uminus-conv-diff diff-minus-eq-add
more-arith-simps(7)
  more-arith-simps(9) mult.commute)
hence  $\text{inv-}Q: \text{invertible-mat } ?Q$  by (smt (verit) Q dvd-refl invertible-iff-is-unit-JNF)
define  $S2$  where  $S2 = ?Q * S1$ 
have  $S2: S2 \in \text{carrier-mat } 2\ 2$  using  $A'\ P\ S2\text{-def}$   $S1\ Q$  mult-carrier-mat by
blast
have  $S2-00: S2\ \$\$(0,0) = 1$  unfolding  $\text{mat-mult2-00}[OF\ Q\ S1\ S2\text{-def}]$  using
 $q00-q01$ 

```

**unfolding**  $S1-00$   $S1-01$  **by** (*simp add: mult.commute*)  
**have**  $S2-10$ :  $S2 \ \$\$ (1,0) = 0$  **unfolding**  $mat-mult2-10$ [*OF Q S1 S2-def*]  
**using**  $q10-q11$  **unfolding**  $S1-00$   $S1-01$  **by** (*simp add: Groups.mult-ac(2)*)

**let**  $?P1 = (addrow-mat \ 2 \ (- \ (S2 \ \$\$ (0,1))) \ 0 \ 1)$   
**have**  $P1$ :  $?P1 \in carrier-mat \ 2 \ 2$  **by** *auto*  
**have**  $inv-P1$ : *invertible-mat*  $?P1$   
**by** (*metis addrow-mat-carrier arithmetic-simps(78) det-addrow-mat dvd-def invertible-iff-is-unit-JNF numeral-One zero-neq-numeral*)  
**define**  $S3$  **where**  $S3 = S2 * ?P1$   
**have**  $P1-P-A'$ :  $A' * ?P * ?P1 \in carrier-mat \ 2 \ 2$  **using**  $P1$   $P$   $A'$  *mult-carrier-mat*  
**by** *auto*  
**have**  $S3$ :  $S3 \in carrier-mat \ 2 \ 2$  **using**  $P1$   $S2$   $S3-def$  *mult-carrier-mat* **by** *blast*  
**have**  $S3-00$ :  $S3 \ \$\$ (0,0) = 1$  **using**  $S2-00$  **unfolding**  $mat-mult2-00$ [*OF S2 P1 S3-def*] **by** *auto*  
**moreover** **have**  $S3-01$ :  $S3 \ \$\$ (0,1) = 0$  **using**  $S2-00$  **unfolding**  $mat-mult2-01$ [*OF S2 P1 S3-def*] **by** *auto*  
**moreover** **have**  $S3-10$ :  $S3 \ \$\$ (1,0) = 0$  **using**  $S2-10$  **unfolding**  $mat-mult2-10$ [*OF S2 P1 S3-def*] **by** *auto*  
**ultimately** **have**  $SNF-S3$ : *Smith-normal-form-mat*  $S3$   
**using**  $S3$  **unfolding** *Smith-normal-form-mat-def isDiagonal-mat-def*  
**using** *less-2-cases* **by** *auto*  
**hence**  $SNF-S3-D$ : *Smith-normal-form-mat*  $(S3 * D)$   
**using**  $D-def$   $S3$  *SNF-preserved-multiples-identity* **by** *blast*  
**have**  $S3 * D = ?Q * A' * ?P * ?P1 * D$  **using**  $S1-def$   $S2-def$   $S3-def$   
**by** (*smt (verit) A' P Q S1 addrow-mat-carrier assoc-mult-mat*)  
**also** **have**  $... = ?Q * A' * ?P * (?P1 * D)$   
**by** (*meson A' D addrow-mat-carrier assoc-mult-mat mat-carrier mult-carrier-mat*)  
**also** **have**  $... = ?Q * A' * ?P * (D * ?P1)$   
**using** *commute-multiples-identity*[*OF P1*] **unfolding**  $D-def$  **by** *auto*  
**also** **have**  $... = ?Q * A' * (?P * (D * ?P1))$   
**by** (*smt (verit) A' D assoc-mult-mat carrier-matD(1) carrier-matD(2) mat-carrier times-mat-def*)  
**also** **have**  $... = ?Q * A' * (D * (?P * ?P1))$   
**by** (*smt (verit) D D-def P P1 assoc-mult-mat commute-multiples-identity*)  
**also** **have**  $... = ?Q * (A' * D) * (?P * ?P1)$   
**by** (*smt (verit) A' D assoc-mult-mat carrier-matD(1) carrier-matD(2) mat-carrier times-mat-def*)  
**also** **have**  $... = ?Q * A * (?P * ?P1)$  **unfolding**  $A-A'D$  **by** *auto*  
**also** **have**  $... = ?Q * B * (U * (?P * ?P1))$  **unfolding**  $A-def$   
**by** (*smt (verit) B U assoc-mult-mat carrier-matD(1) carrier-matD(2) mat-carrier times-mat-def*)  
**finally** **have**  $S3-D-rw$ :  $S3 * D = ?Q * B * (U * (?P * ?P1))$  .  
**show** *admits-diagonal-reduction*  $B$   
**proof** (*rule admits-diagonal-reduction-intro*[*OF - - inv-Q*])  
**show**  $(U * (?P * ?P1)) \in carrier-mat \ (dim-col \ B) \ (dim-col \ B)$  **using**  $B$   $U$  **by**  
*auto*  
**show**  $?Q \in carrier-mat \ (dim-row \ B) \ (dim-row \ B)$  **using**  $Q$   $B$  **by** *auto*  
**show** *invertible-mat*  $(U * (?P * ?P1))$



**by** (*metis (no-types, lifting) P1 U carrier-matD(1) carrier-matD(2) inv-P inv-P1 inv-U*  
*invertible-mult-JNF mat-carrier times-mat-def*)  
**show** *Smith-normal-form-mat (?Q \* B \*(U\* (?P \* ?P1)))* **using** *SNF-S3-D S3-D-rw* **by** *simp*  
**qed**  
**qed**  
**obtain** *Smith-1x2* **where** *Smith-1x2:  $\forall (A::'a \text{ mat}) \in \text{carrier-mat } 1 \ 2. \text{ is-SNF } A$*   
*(Smith-1x2 A)*  
**using** *admits-diagonal-reduction-imp-exists-algorithm-is-SNF-all[OF admits-1x2]*  
**by** *auto*  
**from** *this* **obtain** *Smith-1x2'*  
**where** *Smith-1x2':  $\forall (A::'a \text{ mat}) \in \text{carrier-mat } 1 \ 2. \text{ is-SNF } A$*   
*(1<sub>m</sub> 1, Smith-1x2' A)*  
**using** *Smith-1xn-two-matrices-all[OF Smith-1x2]* **by** *auto*  
**obtain** *Smith-2x2* **where** *Smith-2x2:  $\forall (A::'a \text{ mat}) \in \text{carrier-mat } 2 \ 2. \text{ is-SNF } A$*   
*(Smith-2x2 A)*  
**using** *admits-diagonal-reduction-imp-exists-algorithm-is-SNF-all[OF admits-2x2]*  
**by** *auto*  
**have** *d: is-div-op ( $\lambda a b. (\text{SOME } k. k * b = a)$ )* **using** *div-op-SOME* **by** *auto*  
**interpret** *Smith-Impl Smith-1x2' Smith-2x2* ( $\lambda a b. (\text{SOME } k. k * b = a)$ )  
**using** *Smith-1x2' Smith-2x2 d* **by** (*unfold-locales, auto*)  
**show** *?thesis* **using** *is-SNF-Smith-mxn*  
**by** (*meson admits-diagonal-reduction-eq-exists-algorithm-is-SNF carrier-mat-triv*)  
**qed**

## 18.6 Final theorem

**theorem** *edr-characterization:*

$(\forall (A::'a \text{ mat}). \text{ admits-diagonal-reduction } A) = ((\forall (A::'a \text{ mat}). \text{ admits-triangular-reduction } A)$   
 $\wedge (\forall a b c::'a. \text{ ideal-generated}\{a,b,c\} = \text{ ideal-generated}\{1\}$   
 $\longrightarrow (\exists p q. \text{ ideal-generated } \{p*a,p*b+q*c\} = \text{ ideal-generated } \{1\})))$   
**using** *necessity sufficiency* **by** *blast*

**corollary** *OFCLASS-edr-characterization:*

*OFCLASS('a, elementary-divisor-ring-class)  $\equiv$  (OFCLASS('a, Hermite-ring-class)*

$\&\&\& (\forall a b c::'a. \text{ ideal-generated}\{a,b,c\} = \text{ ideal-generated}\{1\}$   
 $\longrightarrow (\exists p q. \text{ ideal-generated } \{p*a,p*b+q*c\} = \text{ ideal-generated } \{1\})))$  (**is** *?lhs  $\equiv$*   
*?rhs*)

**proof**

**assume** *1: OFCLASS('a, elementary-divisor-ring-class)*

**hence** *admits-diagonal:  $\forall A::'a \text{ mat}. \text{ admits-diagonal-reduction } A$*

**using** *conjunctionD2[OF 1[unfolded elementary-divisor-ring-class-def]]*

**unfolding** *class.elementary-divisor-ring-def* **by** *auto*

**have**  $\forall A::'a \text{ mat}. \text{ admits-triangular-reduction } A$  **by** (*simp add: admits-diagonal*)

*necessity(1)*  
**hence** *OFCLASS-Hermite*: *OFCLASS('a, Hermite-ring-class)* **by** (*intro-classes, simp*)  
**moreover have**  $\forall a b c::'a. \text{ideal-generated } \{a, b, c\} = \text{ideal-generated } \{1\}$   
 $\longrightarrow (\exists p q. \text{ideal-generated } \{p * a, p * b + q * c\} = \text{ideal-generated } \{1\})$   
**using** *admits-diagonal necessity(2)* **by** *blast*  
**ultimately show** *OFCLASS('a, Hermite-ring-class)* &&&  
 $\forall a b c::'a. \text{ideal-generated } \{a, b, c\} = \text{ideal-generated } \{1\}$   
 $\longrightarrow (\exists p q. \text{ideal-generated } \{p * a, p * b + q * c\} = \text{ideal-generated } \{1\})$   
**by** *auto*  
**next**  
**assume** *1*: *OFCLASS('a, Hermite-ring-class)* &&&  
 $\forall a b c::'a. \text{ideal-generated } \{a, b, c\} = \text{ideal-generated } \{1\} \longrightarrow$   
 $(\exists p q. \text{ideal-generated } \{p * a, p * b + q * c\} = \text{ideal-generated } \{1\})$   
**have** *H*: *OFCLASS('a, Hermite-ring-class)*  
**and** *2*:  $\forall a b c::'a. \text{ideal-generated } \{a, b, c\} = \text{ideal-generated } \{1\} \longrightarrow$   
 $(\exists p q. \text{ideal-generated } \{p * a, p * b + q * c\} = \text{ideal-generated } \{1\})$   
**using** *conjunctionD1[OF 1]* *conjunctionD2[OF 1]* **by** *auto*  
**have**  $\forall A::'a \text{ mat. admits-triangular-reduction } A$   
**using** *H unfolding OFCLASS-Hermite-ring-def* **by** *auto*  
**hence** *a*:  $\forall A::'a \text{ mat. admits-diagonal-reduction } A$  **using** *2 sufficiency* **by** *blast*  
**show** *OFCLASS('a, elementary-divisor-ring-class)* **by** (*intro-classes, simp add*:  
*a*)  
**qed**

**corollary** *edr-characterization-class*:  
*class.elementary-divisor-ring TYPE('a)*  
 $= (\text{class.Hermite-ring } \text{TYPE('a)})$   
 $\wedge (\forall a b c::'a. \text{ideal-generated } \{a, b, c\} = \text{ideal-generated } \{1\})$   
 $\longrightarrow (\exists p q. \text{ideal-generated } \{p * a, p * b + q * c\} = \text{ideal-generated } \{1\}))$  (**is** *?lhs = (?H*  
 $\wedge ?D')$ )  
**proof**  
**assume** *1*: *?lhs*  
**hence** *admits-diagonal*:  $\forall A::'a \text{ mat. admits-diagonal-reduction } A$   
**unfolding** *class.elementary-divisor-ring-def* .  
**have** *admits-triangular*:  $\forall A::'a \text{ mat. admits-triangular-reduction } A$   
**using** *1 necessity(1) unfolding class.elementary-divisor-ring-def* **by** *blast*  
**hence** *?H unfolding class.Hermite-ring-def* **by** *auto*  
**moreover have** *?D'* **using** *admits-diagonal necessity(2)* **by** *blast*  
**ultimately show** (*?H*  $\wedge$  *?D'*) **by** *simp*  
**next**  
**assume** *HD'*: (*?H*  $\wedge$  *?D'*)  
**hence** *admits-triangular*:  $\forall A::'a \text{ mat. admits-triangular-reduction } A$   
**unfolding** *class.Hermite-ring-def* **by** *auto*  
**hence** *admits-diagonal*:  $\forall A::'a \text{ mat. admits-diagonal-reduction } A$   
**using** *edr-characterization HD'* **by** *auto*  
**thus** *?lhs* **unfolding** *class.elementary-divisor-ring-def* **by** *auto*  
**qed**

**corollary** *edr-iff-T-D'*:

**shows** *class.elementary-divisor-ring*  $TYPE('a) = ($   
     $(\forall a\ b::'a. \exists a1\ b1\ d. a = a1*d \wedge b = b1*d \wedge \text{ideal-generated } \{a1,b1\} =$   
*ideal-generated*  $\{1\})$   
     $\wedge (\forall a\ b\ c::'a. \text{ideal-generated}\{a,b,c\} = \text{ideal-generated}\{1\}$   
         $\longrightarrow (\exists p\ q. \text{ideal-generated } \{p*a,p*b+q*c\} = \text{ideal-generated } \{1\}))$   
     $)$  (**is**  $?lhs = (?T \wedge ?D')$ )

**proof**

**assume**  $1: ?lhs$

**hence**  $\forall A::'a\ \text{mat. admits-triangular-reduction } A$

**unfolding** *class.elementary-divisor-ring-def* **using** *necessity(1)* **by** *blast*

**hence**  $?T$  **using** *theorem3-part2* **by** *simp*

**moreover have**  $?D'$  **using**  $1$  **unfolding** *edr-characterization-class* **by** *auto*

**ultimately show**  $(?T \wedge ?D')$  **by** *simp*

**next**

**assume**  $TD': (?T \wedge ?D')$

**hence** *class.Hermite-ring*  $TYPE('a)$

**unfolding** *class.Hermite-ring-def* **using** *theorem3-part1*  $TD'$  **by** *auto*

**thus**  $?lhs$  **using** *edr-characterization-class*  $TD'$  **by** *auto*

**qed**

**end**

**end**

## 19 Executable Smith normal form algorithm over Euclidean domains

**theory** *SNF-Algorithm-Euclidean-Domain*

**imports**

*Diagonal-To-Smith*

*Echelon-Form.Examples-Echelon-Form-Abstract*

*Elementary-Divisor-Rings*

*Diagonal-To-Smith-JNF*

*Mod-Type-Connect*

*Show.Show-Instances*

*Jordan-Normal-Form.Show-Matrix*

*Show.Show-Poly*

**begin**

This provides an executable implementation of the verified general algorithm, providing executable operations over a Euclidean domain.

**lemma** *zero-less-one-type2*:  $(0::2) < 1$

**proof** –

**have** *Mod-Type.from-nat*  $0 = (0::2)$  **by** (*simp add: from-nat-0*)

**moreover have**  $Mod\text{-}Type.from\text{-}nat\ 1 = (1::2)$  **using**  $from\text{-}nat\text{-}1$  **by**  $blast$   
**moreover have**  $(Mod\text{-}Type.from\text{-}nat\ 0::2) < Mod\text{-}Type.from\text{-}nat\ 1$  **by**  $(rule\ from\text{-}nat\text{-}mono, auto)$   
**ultimately show**  $?thesis$  **by**  $simp$   
**qed**

## 19.1 Previous code equations

**definition**  $to\text{-}hma_m\text{-}row\ A\ i$   
 $= (vec\text{-}lambda\ (\lambda j. A\ \$\$ (Mod\text{-}Type.to\text{-}nat\ i, Mod\text{-}Type.to\text{-}nat\ j)))$

**lemma**  $bezout\text{-}matrix\text{-}row\text{-}code$   $[code\ abstract]:$   
 $vec\text{-}nth\ (to\text{-}hma_m\text{-}row\ A\ i) =$   
 $(\lambda j. A\ \$\$ (Mod\text{-}Type.to\text{-}nat\ i, Mod\text{-}Type.to\text{-}nat\ j))$   
**unfolding**  $to\text{-}hma_m\text{-}row\text{-}def$  **by**  $auto$

**lemma**  $[code\ abstract]:\ vec\text{-}nth\ (Mod\text{-}Type\text{-}Connect.to\text{-}hma_m\ A) = to\text{-}hma_m\text{-}row\ A$   
**unfolding**  $Mod\text{-}Type\text{-}Connect.to\text{-}hma_m\text{-}def$  **unfolding**  $to\text{-}hma_m\text{-}row\text{-}def$   $[abs\text{-}def]$   
**by**  $auto$

## 19.2 An executable algorithm to transform $2 \times 2$ matrices into its Smith normal form in HOL Analysis

**subclass**  $(in\ euclidean\text{-}ring\text{-}gcd)$   $bezout\text{-}ring\text{-}div$   
**proof** **qed**

**context**  
**fixes**  $bezout::('a::euclidean\text{-}ring\text{-}gcd \Rightarrow 'a \Rightarrow ('a \times 'a \times 'a \times 'a))$   
**assumes**  $ib: is\text{-}bezout\text{-}ext\ bezout$   
**begin**

**lemma**  $normalize\text{-}bezout\text{-}gcd:$   
**assumes**  $b: (p,q,u,v,d) = bezout\ a\ b$   
**shows**  $normalize\ d = gcd\ a\ b$   
**proof**  $-$   
**let**  $?gcd = (\lambda a\ b. case\ bezout\ a\ b\ of\ (x, xa, u, v, gcd') \Rightarrow gcd')$   
**have**  $is\text{-}gcd: is\text{-}gcd\ ?gcd$  **by**  $(simp\ add: ib\ is\text{-}gcd\text{-}is\text{-}bezout\text{-}ext)$   
**have**  $(?gcd\ a\ b) = d$  **using**  $b$  **by**  $(metis\ case\text{-}prod\text{-}conv)$   
**moreover have**  $normalize\ (?gcd\ a\ b) = normalize\ (gcd\ a\ b)$   
**proof**  $(rule\ associatedI)$   
**show**  $(?gcd\ a\ b)\ dvd\ (gcd\ a\ b)$  **using**  $is\text{-}gcd\ is\text{-}gcd\text{-}def$  **by**  $fastforce$   
**show**  $(gcd\ a\ b)\ dvd\ (?gcd\ a\ b)$  **by**  $(metis\ (no\text{-}types)\ gcd\text{-}dvd1\ gcd\text{-}dvd2\ is\text{-}gcd\ is\text{-}gcd\text{-}def)$   
**qed**  
**ultimately show**  $?thesis$  **by**  $auto$   
**qed**

**end**

**lemma** *bezout-matrix-works-transpose1*:

**assumes** *ib*: *is-bezout-ext bezout*

**and** *a-not-b*:  $a \neq b$

**shows**  $(A^{**}\text{transpose } (\text{bezout-matrix } (\text{transpose } A) a b i \text{ bezout})) \$ i \$ a$   
 $= \text{snd } (\text{snd } (\text{snd } (\text{snd } (\text{bezout } (A \$ i \$ a) (A \$ i \$ b))))))$

**proof** –

**have**  $(A^{**}\text{transpose } (\text{bezout-matrix } (\text{transpose } A) a b i \text{ bezout})) \$h i \$h a$   
 $= \text{transpose } (A^{**}\text{transpose } (\text{bezout-matrix } (\text{transpose } A) a b i \text{ bezout})) \$h a \$h$

*i*

**by** (*simp add: transpose-code transpose-row-code*)

**also have**  $\dots = ((\text{bezout-matrix } (\text{transpose } A) a b i \text{ bezout}) ** (\text{transpose } A)) \$h$   
 $a \$h i$

**by** (*simp add: matrix-transpose-mul*)

**also have**  $\dots = \text{snd } (\text{snd } (\text{snd } (\text{snd } (\text{bezout } ((\text{transpose } A) \$ a \$ i) ((\text{transpose } A) \$ b \$ i))))))$

**by** (*rule bezout-matrix-works1 [OF ib a-not-b]*)

**also have**  $\dots = \text{snd } (\text{snd } (\text{snd } (\text{snd } (\text{bezout } (A \$ i \$ a) (A \$ i \$ b))))))$

**by** (*simp add: transpose-code transpose-row-code*)

**finally show** *?thesis* .

**qed**

**lemma** *invertible-bezout-matrix-transpose*:

**fixes** *A*::*'a*:: $\{\text{bezout-ring-div}\}^{\sim}$ *cols*:: $\{\text{finite,wellorder}\}^{\sim}$ *rows*

**assumes** *ib*: *is-bezout-ext bezout*

**and** *a-less-b*:  $a < b$

**and** *aj*:  $A \$h i \$h a \neq 0$

**shows** *invertible*  $(\text{transpose } (\text{bezout-matrix } (\text{transpose } A) a b i \text{ bezout}))$

**proof** –

**have** *Determinants.det*  $(\text{bezout-matrix } (\text{transpose } A) a b i \text{ bezout}) = 1$

**by** (*rule det-bezout-matrix [OF ib a-less-b], insert aj, auto simp add: transpose-def*)

**hence** *Determinants.det*  $(\text{transpose } (\text{bezout-matrix } (\text{transpose } A) a b i \text{ bezout}))$   
 $= 1$  **by** *simp*

**thus** *?thesis* **by** (*simp add: invertible-iff-is-unit*)

**qed**

**function** *diagonalize-2x2-aux* ::  $(('a::\text{euclidean-ring-gcd}^{\sim 2}) \times ('a^{\sim 2}) \times ('a^{\sim 2}))$   
 $\Rightarrow$

$((('a^{\sim 2}) \times ('a^{\sim 2}) \times ('a^{\sim 2}))$

**where** *diagonalize-2x2-aux*  $(P, A, Q) =$

(

*let*

$a = A \$h 0 \$h 0;$

```

    b = A $h 0 $h 1;
    c = A $h 1 $h 0;
    d = A $h 1 $h 1 in
  if a ≠ 0 ∧ ¬ a dvd b then let bezout-mat = transpose (bezout-matrix (transpose
A) 0 1 0 euclid-ext2) in
    diagonalize-2x2-aux (P, A**bezout-mat,Q**bezout-mat) else
    if a ≠ 0 ∧ ¬ a dvd c then let bezout-mat = bezout-matrix A 0 1 0 euclid-ext2
    in diagonalize-2x2-aux (bezout-mat**P,bezout-mat**A,Q) else — We can
divide an get zeros
    let Q' = column-add (Finite-Cartesian-Product.mat 1) 1 0 (− (b div a));
    P' = row-add (Finite-Cartesian-Product.mat 1) 1 0 (− (c div a)) in
    (P'**P,P'**A**Q',Q**Q')
) by auto

```

**termination**

**proof** –

```

  have ib: is-bezout-ext euclid-ext2 by (simp add: is-bezout-ext-euclid-ext2)
  have euclidean-size ((bezout-matrix A 0 1 0 euclid-ext2 ** A) $h 0 $h 0) <
euclidean-size (A $h 0 $h 0)
  if a-not-dvd-c: ¬ A $h 0 $h 0 dvd A $h 1 $h 0 and a-not0: A $h 0 $h 0 ≠ 0
for A::'a2

```

**proof** –

```

  let ?a = (A $h 0 $h 0) let ?c = (A $h 1 $h 0)
  obtain p q u v d where pqvd: (p,q,u,v,d) = euclid-ext2 ?a ?c by (metis
prod-cases5)
  have (bezout-matrix A 0 1 0 euclid-ext2 ** A) $h 0 $h 0 = d
  by (metis bezout-matrix-works1 ib one-neq-zero pqvd prod.sel(2))
  hence normalize ((bezout-matrix A 0 1 0 euclid-ext2 ** A) $h 0 $h 0) =
normalize d by auto
  also have ... = gcd ?a ?c by (rule normalize-bezout-gcd[OF ib pqvd])
  finally have euclidean-size ((bezout-matrix A 0 1 0 euclid-ext2 ** A) $h 0 $h
0)
  = euclidean-size (gcd ?a ?c) by (metis euclidean-size-normalize)
  also have ... < euclidean-size ?a by (rule euclidean-size-gcd-less1[OF a-not0
a-not-dvd-c])
  finally show ?thesis .

```

**qed**

```

  moreover have euclidean-size ((A ** transpose (bezout-matrix (transpose A) 0
1 0 euclid-ext2)) $h 0 $h 0)
  < euclidean-size (A $h 0 $h 0)
  if a-not-dvd-b: ¬ A $h 0 $h 0 dvd A $h 0 $h 1 and a-not0: A $h 0 $h 0 ≠ 0
for A::'a2

```

**proof** –

```

  let ?a = (A $h 0 $h 0) let ?b = (A $h 0 $h 1)
  obtain p q u v d where pqvd: (p,q,u,v,d) = euclid-ext2 ?a ?b by (metis
prod-cases5)
  have (A ** transpose (bezout-matrix (transpose A) 0 1 0 euclid-ext2)) $h 0 $h
0 = d

```

by (metis bezout-matrix-works-transpose1 ib pqvd prod.sel(2) zero-neq-one)  
 hence normalize ((A \*\* transpose (bezout-matrix (transpose A) 0 1 0 euclid-ext2)) \$h 0 \$h 0) = normalize d by auto  
 also have ... = gcd ?a ?b by (rule normalize-bezout-gcd[OF ib pqvd])  
 finally have euclidean-size ((A \*\* transpose (bezout-matrix (transpose A) 0 1 0 euclid-ext2)) \$h 0 \$h 0)  
 = euclidean-size (gcd ?a ?b) by (metis euclidean-size-normalize)  
 also have ... < euclidean-size ?a by (rule euclidean-size-gcd-less1[OF a-not0 a-not-dvd-b])  
 finally show ?thesis .  
 qed  
 ultimately show ?thesis  
 by (relation Wellfounded.measure (λ(P,A,Q). euclidean-size (A \$h 0 \$h 0)), auto)  
 qed

lemma diagonalize-2x2-aux-works:

assumes A = P \*\* A-input \*\* Q  
 and invertible P and invertible Q  
 and (P',D,Q') = diagonalize-2x2-aux (P,A,Q)  
 and A \$h 0 \$h 0 ≠ 0  
 shows D = P' \*\* A-input \*\* Q' ∧ invertible P' ∧ invertible Q' ∧ isDiagonal D  
 using assms  
 proof (induct (P,A,Q) arbitrary: P A Q rule: diagonalize-2x2-aux.induct)  
 case (1 P A Q)  
 let ?a = A \$h 0 \$h 0  
 let ?b = A \$h 0 \$h 1  
 let ?c = A \$h 1 \$h 0  
 let ?d = A \$h 1 \$h 1  
 have a-not-0: ?a ≠ 0 using 1.prem1 by blast  
 have ib: is-bezout-ext euclid-ext2 by (simp add: is-bezout-ext-euclid-ext2)  
 have one-not-zero: 1 ≠ (0::2) by auto  
 show ?case  
 proof (cases ¬ ?a dvd ?b)  
 case True  
 let ?bezout-mat-right = transpose (bezout-matrix (transpose A) 0 1 0 euclid-ext2)  
 have (P', D, Q') = diagonalize-2x2-aux (P, A, Q) using 1.prem1 by blast  
 also have ... = diagonalize-2x2-aux (P, A\*\* ?bezout-mat-right, Q \*\* ?bezout-mat-right)  
 using True a-not-0 by (auto simp add: Let-def)  
 finally have eq: (P',D,Q') = ... .  
 show ?thesis  
 proof (rule 1.hyps(1)[OF ----- eq])  
 have invertible ?bezout-mat-right  
 by (rule invertible-bezout-matrix-transpose[OF ib zero-less-one-type2 a-not-0])  
 thus invertible (Q \*\* ?bezout-mat-right)  
 using 1.prem1 invertible-mult by blast  
 show A \*\* ?bezout-mat-right = P \*\* A-input \*\* (Q \*\* ?bezout-mat-right)  
 by (simp add: 1.prem1 matrix-mul-assoc)
 end
 case False
 end
end

```

    show (A ** ?bezout-mat-right) $h 0 $h 0 ≠ 0
      by (metis (no-types, lifting) a-not-0 bezout-matrix-works-transpose1 be-
zout-matrix-not-zero
        bezout-matrix-works1 is-bezout-ext-euclid-ext2 one-neq-zero transpose-code
transpose-row-code)
    qed (insert True a-not-0 1.premis, blast+)
next
case False note a-dvd-b = False
show ?thesis
proof (cases ¬ ?a dvd ?c)
  case True
  let ?bezout-mat = (bezout-matrix A 0 1 0 euclid-ext2)
  have (P', D, Q') = diagonalize-2x2-aux (P, A, Q) using 1.premis by blast
  also have ... = diagonalize-2x2-aux (?bezout-mat**P, ?bezout-mat ** A, Q)
  using True a-dvd-b a-not-0 by (auto simp add: Let-def)
  finally have eq: (P',D,Q') = ... .
  show ?thesis
  proof (rule 1.hyps(2)[OF - - - - - eq])
    have invertible ?bezout-mat
    by (rule invertible-bezout-matrix[OF ib zero-less-one-type2 a-not-0])
    thus invertible (?bezout-mat ** P)
    using 1.premis invertible-mult by blast
    show ?bezout-mat ** A = (?bezout-mat ** P) ** A-input ** Q
    by (simp add: 1.premis matrix-mul-assoc)
    show (?bezout-mat ** A) $h 0 $h 0 ≠ 0
    by (simp add: a-not-0 bezout-matrix-not-zero is-bezout-ext-euclid-ext2)
  qed (insert True a-not-0 a-dvd-b 1.premis, blast+)
next
case False
hence a-dvd-c: ?a dvd ?c by simp
  let ?Q' = column-add (Finite-Cartesian-Product.mat 1) 1 0 (- (?b div
?a))::'a^2^2
  let ?P' = (row-add (Finite-Cartesian-Product.mat 1) 1 0 (- (?c div ?a))::'a^2^2
  have eq: (P', D, Q') = (?P'**P, ?P'**A**?Q', Q**?Q')
  using 1.premis a-dvd-b a-dvd-c a-not-0 by (auto simp add: Let-def)
  have d: isDiagonal (?P'**A**?Q')
  proof -
    {
      fix a b::2 assume a-not-b: a ≠ b
      have (?P' ** A ** ?Q') $h a $h b = 0
      proof (cases (a,b) = (0,1))
        case True
        hence a0: a = 0 and b1: b = 1 by auto
        have (?P' ** A ** ?Q') $h a $h b = (?P' ** (A ** ?Q')) $h a $h b
        by (simp add: matrix-mul-assoc)
        also have ... = (A**?Q') $h a $h b
        by (simp add: row-add-mat-1 a0 row-add-code row-add-code-nth)
        also have ... = 0 unfolding column-add-mat-1 a0 b1
        by (smt (verit, ccfv-threshold) a-dvd-b column-add-code column-add-code-nth

```



```

      dvd-mult-div-cancel more-arith-simps(4) more-arith-simps(8))
    finally show ?thesis .
  next
  case False
  hence a1: a = 1 and b0: b = 0
  by (metis (no-types, opaque-lifting) False a-not-b exhaust-2 zero-neq-one)+
  have (?P' ** A ** ?Q') $h a $h b = (?P' ** A) $h a $h b
    unfolding a1 b0 column-add-mat-1
    by (simp add: column-add-code-nth column-add-row-def)
  also have ... = 0 unfolding row-add-mat-1 a1 b0
    by (simp add: a-dvd-c row-add-def)
  finally show ?thesis .
qed}
thus ?thesis unfolding isDiagonal-def by auto
qed
have inv-P': invertible ?P' by (rule invertible-row-add[OF one-not-zero])
have inv-Q': invertible ?Q' by (rule invertible-column-add[OF one-not-zero])
have invertible (?P' ** P) using 1.premis(2) inv-P' invertible-mult by blast
moreover have invertible (Q ** ?Q') using 1.premis(3) inv-Q' invertible-mult
by blast
moreover have D = P' ** A-input ** Q'
  by (metis (no-types, lifting) 1.premis(1) Pair-inject eq matrix-mul-assoc)
ultimately show ?thesis using eq d by auto
qed
qed
qed

```

**definition** *diagonalize-2x2* A =  
 (if A \$h 0 \$h 0 = 0 then  
   if A \$h 0 \$h 1 ≠ 0 then  
     let A' = interchange-columns A 0 1;  
     Q' = interchange-columns (Finite-Cartesian-Product.mat 1) 0 1 in  
     diagonalize-2x2-aux (Finite-Cartesian-Product.mat 1, A', Q')  
   else  
     if A \$h 1 \$h 0 ≠ 0 then  
       let A' = interchange-rows A 0 1;  
       P' = interchange-rows (Finite-Cartesian-Product.mat 1) 0 1 in  
       diagonalize-2x2-aux (P', A', Finite-Cartesian-Product.mat 1)  
     else (Finite-Cartesian-Product.mat 1, A, Finite-Cartesian-Product.mat 1)  
 else diagonalize-2x2-aux (Finite-Cartesian-Product.mat 1, A, Finite-Cartesian-Product.mat 1)  
 )  
 )

**lemma** *diagonalize-2x2-works*:  
 assumes PDQ: (P, D, Q) = diagonalize-2x2 A  
 shows D = P \*\* A \*\* Q ∧ invertible P ∧ invertible Q ∧ isDiagonal D  
**proof** –

```

let ?a = A $h 0 $h 0
let ?b = A $h 0 $h 1
let ?c = A $h 1 $h 0
let ?d = A $h 1 $h 1
show ?thesis
proof (cases ?a = 0)
  case False
  hence eq: (P,D,Q) = diagonalize-2x2-aux (Finite-Cartesian-Product.mat 1,A,Finite-Cartesian-Product.mat
1)
    using PDQ unfolding diagonalize-2x2-def by auto
  show ?thesis
    by (rule diagonalize-2x2-aux-works[OF - - - eq False], auto simp add: invert-
ible-mat-1)
  next
  case True note a0 = True
  show ?thesis
  proof (cases ?b ≠ 0)
    case True
    let ?A' = interchange-columns A 0 1
    let ?Q' = (interchange-columns (Finite-Cartesian-Product.mat 1) 0 1)::'a22
    have eq: (P,D,Q) = diagonalize-2x2-aux (Finite-Cartesian-Product.mat 1,
?A', ?Q')
      using PDQ a0 True unfolding diagonalize-2x2-def by (auto simp add:
Let-def)
    show ?thesis
    proof (rule diagonalize-2x2-aux-works[OF - - - eq -])
      show ?A' $h 0 $h 0 ≠ 0
      by (simp add: True interchange-columns-code interchange-columns-code-nth)
      show invertible ?Q' by (simp add: invertible-interchange-columns)
      show ?A' = Finite-Cartesian-Product.mat 1 ** A ** ?Q'
      by (simp add: interchange-columns-mat-1)
    qed (auto simp add: invertible-mat-1)
  next
  case False note b0 = False
  show ?thesis
  proof (cases ?c ≠ 0)
    case True
    let ?A' = interchange-rows A 0 1
    let ?P' = (interchange-rows (Finite-Cartesian-Product.mat 1) 0 1)::'a22
    have eq: (P,D,Q) = diagonalize-2x2-aux (?P', ?A',Finite-Cartesian-Product.mat
1)
      using PDQ a0 b0 True unfolding diagonalize-2x2-def by (auto simp add:
Let-def)
    show ?thesis
    proof (rule diagonalize-2x2-aux-works[OF - - - eq -])
      show ?A' $h 0 $h 0 ≠ 0
      by (simp add: True interchange-columns-code interchange-columns-code-nth)
      show invertible ?P' by (simp add: invertible-interchange-rows)
      show ?A' = ?P' ** A ** Finite-Cartesian-Product.mat 1

```

```

      by (simp add: interchange-rows-mat-1)
    qed (auto simp add: invertible-mat-1)
  next
    case False
  have eq: (P,D,Q) = (Finite-Cartesian-Product.mat 1, A,Finite-Cartesian-Product.mat
1)
    using PDQ a0 b0 True False unfolding diagonalize-2x2-def by (auto simp
add: Let-def)
    have isDiagonal A unfolding isDiagonal-def using a0 b0 True False
      by (metis (full-types) exhaust-2 one-neq-zero)
    thus ?thesis using invertible-mat-1 eq by auto
  qed
qed
qed
qed

```

**definition** *diagonalize-2x2-JNF* (A::'a::euclidean-ring-gcd mat)  
= (let (P,D,Q) = diagonalize-2x2 (Mod-Type-Connect.to-hma<sub>m</sub> A::'a<sup>2</sup><sup>2</sup>) in  
(Mod-Type-Connect.from-hma<sub>m</sub> P,Mod-Type-Connect.from-hma<sub>m</sub> D,Mod-Type-Connect.from-hma<sub>m</sub>  
Q))

**lemma** *diagonalize-2x2-JNF-works:*

```

  assumes A: A ∈ carrier-mat 2 2
  and PDQ: (P,D,Q) = diagonalize-2x2-JNF A
  shows D = P * A * Q ∧ invertible-mat P ∧ invertible-mat Q ∧ isDiagonal-mat
D ∧ P ∈ carrier-mat 2 2
  ∧ Q ∈ carrier-mat 2 2 ∧ D ∈ carrier-mat 2 2
proof –
  let ?A = (Mod-Type-Connect.to-hmam A::'a22)
  have A[transfer-rule]: Mod-Type-Connect.HMA-M A ?A
    using A unfolding Mod-Type-Connect.HMA-M-def by auto
  obtain P-HMA D-HMA Q-HMA where PDQ-HMA: (P-HMA,D-HMA,Q-HMA)
= diagonalize-2x2 ?A
    by (metis prod-cases3)

```

have P: P = Mod-Type-Connect.from-hma<sub>m</sub> P-HMA and Q: Q = Mod-Type-Connect.from-hma<sub>m</sub>  
Q-HMA

```

  and D: D = Mod-Type-Connect.from-hmam D-HMA
  using PDQ-HMA PDQ unfolding diagonalize-2x2-JNF-def
  by (metis prod.simps(1) split-conv)+
  have [transfer-rule]: Mod-Type-Connect.HMA-M P P-HMA
  unfolding Mod-Type-Connect.HMA-M-def using P by auto
  have [transfer-rule]: Mod-Type-Connect.HMA-M Q Q-HMA
  unfolding Mod-Type-Connect.HMA-M-def using Q by auto
  have [transfer-rule]: Mod-Type-Connect.HMA-M D D-HMA
  unfolding Mod-Type-Connect.HMA-M-def using D by auto

```

**have**  $r$ :  $D\text{-HMA} = P\text{-HMA} ** ?A ** Q\text{-HMA} \wedge \text{invertible } P\text{-HMA} \wedge \text{invertible } Q\text{-HMA} \wedge \text{isDiagonal } D\text{-HMA}$   
**by** (*rule diagonalize-2x2-works[OF PDQ-HMA]*)  
**have**  $D = P * A * Q \wedge \text{invertible-mat } P \wedge \text{invertible-mat } Q \wedge \text{isDiagonal-mat } D$   
**using**  $r$  **by** (*transfer, rule*)  
**thus** *?thesis* **using**  $P$   $Q$   $D$  **by** *auto*  
**qed**

**definition** *Smith-2x2-eucl*  $A = ($   
 $\text{let } (P, D, Q) = \text{diagonalize-2x2 } A;$   
 $(P', S, Q') = \text{diagonal-to-Smith-PQ } D \text{ euclid-ext2}$   
 $\text{in } (P' ** P, S, Q ** Q')$

**lemma** *Smith-2x2-eucl-works*:

**assumes**  $PBQ$ :  $(P, S, Q) = \text{Smith-2x2-eucl } A$   
**shows**  $S = P ** A ** Q \wedge \text{invertible } P \wedge \text{invertible } Q \wedge \text{Smith-normal-form } S$   
**proof** –  
**have**  $ib$ : *is-bezout-ext euclid-ext2* **by** (*simp add: is-bezout-ext-euclid-ext2*)  
**obtain**  $P1$   $D$   $Q1$  **where**  $P1DQ1$ :  $(P1, D, Q1) = \text{diagonalize-2x2 } A$  **by** (*metis prod-cases3*)  
**obtain**  $P2$   $S'$   $Q2$  **where**  $P2SQ2$ :  $(P2, S', Q2) = \text{diagonal-to-Smith-PQ } D \text{ euclid-ext2}$   
**by** (*metis prod-cases3*)  
**have**  $P$ :  $P = P2 ** P1$  **and**  $S$ :  $S = S'$  **and**  $Q$ :  $Q = Q1 ** Q2$   
**by** (*metis (mono-tags, lifting) PBQ Pair-inject Smith-2x2-eucl-def P1DQ1 P2SQ2 old.prod.case*)  
**have**  $1$ :  $D = P1 ** A ** Q1 \wedge \text{invertible } P1 \wedge \text{invertible } Q1 \wedge \text{isDiagonal } D$   
**by** (*rule diagonalize-2x2-works[OF P1DQ1]*)  
**have**  $2$ :  $S' = P2 ** D ** Q2 \wedge \text{invertible } P2 \wedge \text{invertible } Q2 \wedge \text{Smith-normal-form } S'$   
**by** (*rule diagonal-to-Smith-PQ'[OF - ib P2SQ2], insert 1, auto*)  
**show** *?thesis* **using**  $1$   $2$   $P$   $S$   $Q$  **by** (*simp add: 2 invertible-mult matrix-mul-assoc*)  
**qed**

### 19.3 An executable algorithm to transform $2 \times 2$ matrices into its Smith normal form in JNF

**definition** *Smith-2x2-JNF-eucl*  $A = ($   
 $\text{let } (P, D, Q) = \text{diagonalize-2x2-JNF } A;$   
 $(P', S, Q') = \text{diagonal-to-Smith-PQ-JNF } D \text{ euclid-ext2}$   
 $\text{in } (P' * P, S, Q * Q')$

**lemma** *Smith-2x2-JNF-eucl-works*:

**assumes**  $A$ :  $A \in \text{carrier-mat } 2$   $2$   
**and**  $PBQ$ :  $(P, S, Q) = \text{Smith-2x2-JNF-eucl } A$

**shows** *is-SNF*  $A (P, S, Q)$   
**proof** –  
**have** *ib: is-bezout-ext euclid-ext2* **by** (*simp add: is-bezout-ext-euclid-ext2*)  
**obtain**  $P1\ D\ Q1$  **where**  $P1DQ1: (P1, D, Q1) = \text{diagonalize-}2x2\text{-JNF } A$  **by** (*metis prod-cases3*)  
**obtain**  $P2\ S'\ Q2$  **where**  $P2SQ2: (P2, S', Q2) = \text{diagonal-to-Smith-PQ-JNF } D$   
*euclid-ext2*  
**by** (*metis prod-cases3*)  
**have**  $P: P = P2 * P1$  **and**  $S: S = S'$  **and**  $Q: Q = Q1 * Q2$   
**by** (*metis (mono-tags, lifting) PBQ Pair-inject Smith-2x2-JNF-eucl-def P1DQ1 P2SQ2 old.prod.case*)  
**have**  $1: D = P1 * A * Q1 \wedge \text{invertible-mat } P1 \wedge \text{invertible-mat } Q1 \wedge \text{isDiagonal-mat } D$   
 $\wedge P1 \in \text{carrier-mat } 2\ 2 \wedge Q1 \in \text{carrier-mat } 2\ 2 \wedge D \in \text{carrier-mat } 2\ 2$   
**by** (*rule diagonalize-2x2-JNF-works[OF A P1DQ1]*)  
**have**  $2: S' = P2 * D * Q2 \wedge \text{invertible-mat } P2 \wedge \text{invertible-mat } Q2 \wedge \text{Smith-normal-form-mat } S'$   
 $\wedge P2 \in \text{carrier-mat } 2\ 2 \wedge S' \in \text{carrier-mat } 2\ 2 \wedge Q2 \in \text{carrier-mat } 2\ 2$   
**by** (*rule diagonal-to-Smith-PQ-JNF[OF - ib - P2SQ2], insert 1, auto*)  
**show** *?thesis*  
**proof** (*rule is-SNF-intro*)  
**have** *dim-Q: Q*  $\in \text{carrier-mat } 2\ 2$  **using**  $Q\ 1\ 2$  **by** *auto*  
**have**  $P1AQ1: (P1 * A * Q1) \in \text{carrier-mat } 2\ 2$  **using**  $1\ 2\ A$  **by** *auto*  
**have**  $rw1: (P1 * A * Q1) * Q2 = (P1 * A * (Q1 * Q2))$   
**by** (*meson 1 2 A assoc-mult-mat mult-carrier-mat*)  
**have**  $rw2: (P1 * A * Q) = P1 * (A * Q)$  **by** (*rule assoc-mult-mat[OF - A dim-Q], insert 1, auto*)  
**show** *invertible-mat Q* **using**  $1\ 2\ Q$  *invertible-mult-JNF* **by** *blast*  
**show** *invertible-mat P* **using**  $1\ 2\ P$  *invertible-mult-JNF* **by** *blast*  
**have**  $P2 * D * Q2 = P2 * (P1 * A * Q1) * Q2$  **using**  $1\ 2$  **by** *auto*  
**also have**  $\dots = P2 * ((P1 * A * Q1) * Q2)$  **using**  $1\ 2$  **by** *auto*  
**also have**  $\dots = P2 * (P1 * A * (Q1 * Q2))$  **unfolding**  $rw1$  **by** *simp*  
**also have**  $\dots = P2 * (P1 * A * Q)$  **using**  $Q$  **by** *auto*  
**also have**  $\dots = P2 * (P1 * (A * Q))$  **unfolding**  $rw2$  **by** *simp*  
**also have**  $\dots = P2 * P1 * (A * Q)$  **by** (*rule assoc-mult-mat[symmetric], insert 1 2 A Q, auto*)  
**also have**  $\dots = P * (A * Q)$  **unfolding**  $P$  **by** *simp*  
**also have**  $\dots = P * A * Q$   
**by** (*smt (verit, ccfv-SIG) 1 2 A P assoc-mult-mat dim-Q mult-carrier-mat*)  
**finally show**  $S = P * A * Q$  **using**  $1\ 2\ S$  **by** *auto*  
**qed** (*insert 1 2 P Q A S, auto*)  
**qed**

## 19.4 An executable algorithm to transform $1 \times 2$ matrices into its Smith normal form

**definition** *Smith-1x2-eucl* ( $A::'a::\text{euclidean-ring-gcd}^{\sim}2^{\sim}1$ ) = (  
*if*  $A\ \$h\ 0\ \$h\ 0 = 0 \wedge A\ \$h\ 0\ \$h\ 1 \neq 0$  *then*  
*let*  $Q = \text{interchange-columns } (\text{Finite-Cartesian-Product.mat } 1)\ 0\ 1;$

```

    A' = interchange-columns A 0 1 in (A',Q)
  else
    if A $h 0 $h 0 ≠ 0 ∧ A $h 0 $h 1 ≠ 0 then
      let bezout-matrix-right = transpose (bezout-matrix (transpose A) 0 1 0 euclid-ext2)
      in (A ** bezout-matrix-right, bezout-matrix-right)
    else (A, Finite-Cartesian-Product.mat 1)
  )

```

**lemma** *Smith-1x2-eucl-works:*

```

  assumes SQ: (S,Q) = Smith-1x2-eucl A
  shows S = A ** Q ∧ invertible Q ∧ S $h 0 $h 1 = 0
proof (cases A $h 0 $h 0 = 0 ∧ A $h 0 $h 1 ≠ 0)
  case True
  have Q: Q = interchange-columns (Finite-Cartesian-Product.mat 1) 0 1
  and S: S = interchange-columns A 0 1
  using SQ True unfolding Smith-1x2-eucl-def by (auto simp add: Let-def)
  have S $h 0 $h 1 = 0 by (simp add: S True interchange-columns-code interchange-columns-code-nth)
  moreover have invertible Q using Q invertible-interchange-columns by blast
  moreover have S = A ** Q by (simp add: Q S interchange-columns-mat-1)
  ultimately show ?thesis by simp
next
  case False note A00-A01 = False
  show ?thesis
  proof (cases A $h 0 $h 0 ≠ 0 ∧ A $h 0 $h 1 ≠ 0)
  case True
  have ib: is-bezout-ext euclid-ext2 by (simp add: is-bezout-ext-euclid-ext2)
  let ?bezout-matrix-right = transpose (bezout-matrix (transpose A) 0 1 0 euclid-ext2)
  have Q: Q = ?bezout-matrix-right and S: S = A**?bezout-matrix-right
  using SQ True A00-A01 unfolding Smith-1x2-eucl-def by (auto simp add: Let-def)
  have invertible Q unfolding Q
  by (rule invertible-bezout-matrix-transpose[OF ib zero-less-one-type2], insert True, auto)
  moreover have S $h 0 $h 1 = 0
  by (smt (verit) Finite-Cartesian-Product.transpose-transpose S True bezout-matrix-works2 ib
    matrix-transpose-mul rel-simps(92) transpose-code transpose-row-code)
  moreover have S = A**Q unfolding S Q by simp
  ultimately show ?thesis by simp
  next
  case False
  have Q: Q = (Finite-Cartesian-Product.mat 1) and S: S = A
  using SQ False A00-A01 unfolding Smith-1x2-eucl-def by (auto simp add: Let-def)
  show ?thesis using False A00-A01 S Q invertible-mat-1 by auto

```

qed  
qed

**definition** *bezout-matrix-JNF* :: 'a::comm-ring-1 mat  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  nat  
 $\Rightarrow$  ('a  $\Rightarrow$  'a  $\Rightarrow$  ('a  $\times$  'a  $\times$  'a  $\times$  'a  $\times$  'a))  $\Rightarrow$  'a mat  
**where**  
*bezout-matrix-JNF* A a b j bezout = Matrix.mat (dim-row A) (dim-row A) ( $\lambda(x,y).$

(let  
 (p, q, u, v, d) = bezout (A \$\$ (a, j)) (A \$\$ (b, j))  
 in  
 if x = a  $\wedge$  y = a then p else  
 if x = a  $\wedge$  y = b then q else  
 if x = b  $\wedge$  y = a then u else  
 if x = b  $\wedge$  y = b then v else  
 if x = y then 1 else 0))

**definition** *Smith-1x2-eucl-JNF* (A:'a::euclidean-ring-gcd mat) = (  
 if A \$\$ (0, 0) = 0  $\wedge$  A \$\$ (0, 1)  $\neq$  0 then  
 let Q = swaprows-mat 2 0 1;  
 A' = swapcols 0 1 A  
 in (A',Q)  
 else  
 if A \$\$ (0, 0)  $\neq$  0  $\wedge$  A \$\$ (0, 1)  $\neq$  0 then  
 let bezout-matrix-right = transpose-mat (bezout-matrix-JNF (transpose-mat  
 A) 0 1 0 euclid-ext2)  
 in (A \* bezout-matrix-right, bezout-matrix-right)  
 else (A, 1\_m 2)  
 )

**lemma** *Smith-1x2-eucl-JNF-works*:

**assumes** A: A  $\in$  carrier-mat 1 2

**and** SQ: (S,Q) = *Smith-1x2-eucl-JNF* A

**shows** is-SNF A (1\_m 1, (Smith-1x2-eucl-JNF A))

**proof** –

**have** i: 0 < dim-row A **and** j: 1 < dim-col A **using** A **by** auto

**have** ib: is-bezout-ext euclid-ext2 **by** (simp add: is-bezout-ext-euclid-ext2)

**show** ?thesis

**proof** (cases A \$\$ (0, 0) = 0  $\wedge$  A \$\$ (0, 1)  $\neq$  0)

**case** True

**have** Q: Q = swaprows-mat 2 0 1

**and** S: S = swapcols 0 1 A

**using** SQ True **unfolding** *Smith-1x2-eucl-JNF-def* **by** (auto simp add:

*Let-def*)

**have** S01: S \$\$ (0,1) = 0 **unfolding** S **using** index-mat-swapcols j i True **by**

```

simp
  have dim-S:  $S \in \text{carrier-mat } 1 \ 2$  using  $S \ A$  by auto
  moreover have dim-Q:  $Q \in \text{carrier-mat } 2 \ 2$  using  $S \ Q$  by auto
  moreover have invertible-mat  $Q$ 
  proof -
    have Determinant.det (swaprows-mat 2 0 1) = -1 by (rule det-swaprows-mat,
    auto)
    also have ... dvd 1 by simp
    finally show ?thesis using  $Q \ \text{dim-Q} \ \text{invertible-iff-is-unit-JNF}$  by blast
  qed
  moreover have  $S = A * Q$  unfolding  $S \ Q$  using  $A$  by (simp add: swap-
  cols-mat)
  moreover have Smith-normal-form-mat  $S$  unfolding Smith-normal-form-mat-def
  isDiagonal-mat-def
    using  $S01 \ \text{dim-S} \ \text{less-2-cases}$  by fastforce
  ultimately show ?thesis using  $SQ \ S \ Q \ A$  unfolding is-SNF-def by auto
next
case False note  $A00-A01 = \text{False}$ 
show ?thesis
proof (cases  $A \ \$\$ (0,0) \neq 0 \wedge A \ \$\$ (0,1) \neq 0$ )
  case True
    have ib: is-bezout-ext euclid-ext2 by (simp add: is-bezout-ext-euclid-ext2)
    let ?BM = (bezout-matrix-JNF  $A^T \ 0 \ 1 \ 0 \ \text{euclid-ext2}$ )T
    have Q:  $Q = ?BM$  and S:  $S = A * ?BM$ 
      using  $SQ \ \text{True} \ A00-A01$  unfolding Smith-1x2-eucl-JNF-def by (auto simp
    add: Let-def)
    let ?a =  $A \ \$\$ (0, 0)$  let ?b =  $A \ \$\$ (0, \text{Suc } 0)$ 
      obtain  $p \ q \ u \ v \ d$  where  $pquvd: (p,q,u,v,d) = \text{euclid-ext2 } ?a \ ?b$  by (metis
    prod-cases5)
      have  $d: p * ?a + q * ?b = d$  and  $u: u = - ?b \ \text{div } d$  and  $v: v = ?a \ \text{div } d$ 
        using  $pquvd$  unfolding euclid-ext2-def using bezout-coefficients-fst-snd by
    blast+
      have  $da: d \ \text{dvd } ?a$  and  $db: d \ \text{dvd } ?b$  and  $\text{gcd-ab}: d = \text{gcd } ?a \ ?b$ 
        by (metis euclid-ext2-def gcd-dvd1 gcd-dvd2  $pquvd \ \text{prod.sel}(2)$ )
      have dim-S:  $S \in \text{carrier-mat } 1 \ 2$  using  $S \ A$  by (simp add: bezout-matrix-JNF-def)
      moreover have dim-Q:  $Q \in \text{carrier-mat } 2 \ 2$  using  $A \ Q$  by (simp add:
    bezout-matrix-JNF-def)
      have invertible-mat  $Q$ 
      proof -
        have Determinant.det ?BM = ?BM  $\ \$\$ (0, 0) * ?BM \ \$\$ (1, 1) - ?BM \ \$\$$ 
        ( $0, 1$ ) * ?BM  $\ \$\$ (1, 0)$ 
          by (rule det-2, insert  $A$ , auto simp add: bezout-matrix-JNF-def)
        also have ... =  $p * v - u * q$ 
          by (insert  $i \ j \ pquvd$ , auto simp add: bezout-matrix-JNF-def, metis split-conv)
        also have ... =  $(p * ?a) \ \text{div } d - (q * (-?b)) \ \text{div } d$  unfolding  $v \ u$ 
          by (simp add:  $da \ db \ \text{div-mult-swap} \ \text{mult.commute}$ )
        also have ... =  $(p * ?a + q * ?b) \ \text{div } d$ 
          by (metis (no-types, lifting)  $da \ db \ \text{diff-minus-eq-add} \ \text{div-diff} \ \text{dvd-minus-iff}$ 
    dvd-trans)

```



```

      dvd-triv-right more-arith-simps(8))
    also have ... = 1 unfolding d using True da by fastforce
    finally show ?thesis unfolding Q
  by (metis (full-types) Determinant.det-def Q carrier-matI invertible-iff-is-unit-JNF

      not-is-unit-0 one-dvd)
qed
moreover have S-AQ: S = A*Q unfolding S Q by simp
moreover have S01: S $$ (0,1) = 0
proof -
  have Q01: Q $$ (0, 1) = u
  proof -
    have ?BM $$ (0,1) = (bezout-matrix-JNF AT 0 1 0 euclid-ext2) $$ (1, 0)
      using Q dim-Q by auto
    also have ... = (λ(x::nat, y::nat).
      let (p, q, u, v, d) = euclid-ext2 (AT $$ (0, 0)) (AT $$ (1, 0)) in if x = 0
    ∧ y = 0 then p
      else if x = 0 ∧ y = 1 then q else if x = 1 ∧ y = 0 then u else if x = 1
    ∧ y = 1 then v
      else if x = y then 1 else 0) (1, 0)
    unfolding bezout-matrix-JNF-def by (rule index-mat(1), insert A, auto)
    also have ... = u using pqvd unfolding split-beta Let-def
  by (auto, metis A One-nat-def carrier-matD(2) fst-conv i index-transpose-mat(1)

      j rel-simps(51) snd-conv)
    finally show ?thesis unfolding Q by auto
qed
have Q11: Q $$ (1, 1) = v
proof -
  have ?BM $$ (1,1) = (bezout-matrix-JNF AT 0 1 0 euclid-ext2) $$ (1, 1)
    using Q dim-Q by auto
  also have ... = (λ(x::nat, y::nat).
    let (p, q, u, v, d) = euclid-ext2 (AT $$ (0, 0)) (AT $$ (1, 0)) in if x = 0
  ∧ y = 0 then p
    else if x = 0 ∧ y = 1 then q else if x = 1 ∧ y = 0 then u else if x = 1
  ∧ y = 1 then v
    else if x = y then 1 else 0) (1, 1)
  unfolding bezout-matrix-JNF-def by (rule index-mat(1), insert A, auto)
  also have ... = v using pqvd unfolding split-beta Let-def
  by (auto, metis A One-nat-def carrier-matD(2) fst-conv i index-transpose-mat(1)

      j rel-simps(51) snd-conv)
    finally show ?thesis unfolding Q by auto
qed
have S $$ (0,1) = Matrix.row A 0 · col Q 1 using index-mult-mat Q S
dim-S i by auto
also have ... = (∑ i = 0..<2. Matrix.row A 0 $v i * Q $$ (i, 1))
  unfolding scalar-prod-def using dim-S dim-Q by auto
also have ... = (∑ i ∈ {0,1}. Matrix.row A 0 $v i * Q $$ (i, 1)) by (rule

```

```

sum.cong, auto)
  also have ... = Matrix.row A 0 $v 0 * Q $$ (0, 1) + Matrix.row A 0 $v 1
* Q $$ (1, 1)
  using sum-two-elements by auto
  also have ... = ?a*u + ?b * v unfolding Q01 Q11 using i index-row(1)
j A by auto
  also have ... = 0 unfolding u v
  by (smt (verit) Groups.mult-ac(2) Groups.mult-ac(3) add.right-inverse
add-uminus-conv-diff da db
diff-minus-eq-add dvd-div-mult-self dvd-neg-div minus-mult-left)
  finally show ?thesis .
qed
moreover have Smith-normal-form-mat S
  using less-2-cases S01 dim-S unfolding Smith-normal-form-mat-def isDi-
agonal-mat-def
  by fastforce
  ultimately show ?thesis using S Q A SQ unfolding is-SNF-def be-
zout-matrix-JNF-def by force
next
case False
have Q: Q = 1m 2 and S: S = A
  using SQ False A00-A01 unfolding Smith-1x2-eucl-JNF-def by (auto simp
add: Let-def)
have is-SNF A (1m 1, A, 1m 2)
  by (rule is-SNF-intro, insert A False A00-A01 S Q A less-2-cases,
unfold Smith-normal-form-mat-def isDiagonal-mat-def, fastforce+)
thus ?thesis using SQ S Q by auto
qed
qed
qed

```

## 19.5 The final executable algorithm to transform any matrix into its Smith normal form

**global-interpretation** *Smith-ED: Smith-Impl Smith-1x2-eucl-JNF Smith-2x2-JNF-eucl (div)*

```

defines Smith-ED-1xn-aux = Smith-ED.Smith-1xn-aux
  and Smith-ED-nx1 = Smith-ED.Smith-nx1
and Smith-ED-1xn = Smith-ED.Smith-1xn
and Smith-ED-2xn = Smith-ED.Smith-2xn
and Smith-ED-nx2 = Smith-ED.Smith-nx2
and Smith-ED-mxn = Smith-ED.Smith-mxn
proof
show  $\forall (A::'a \text{ mat}) \in \text{carrier-mat } 1 \ 2. \text{ is-SNF } A (1_m \ 1, \text{ Smith-1x2-eucl-JNF } A)$ 
  using Smith-1x2-eucl-JNF-works prod.collapse by blast
show  $\forall A \in \text{carrier-mat } 2 \ 2. \text{ is-SNF } A (\text{Smith-2x2-JNF-eucl } A)$ 
  by (simp add: Smith-2x2-JNF-eucl-def Smith-2x2-JNF-eucl-works split-beta)
show is-div-op ((div)::'a  $\Rightarrow$  'a  $\Rightarrow$  'a::euclidean-ring-gcd)
  by (unfold is-div-op-def, simp)

```

qed

end

## 20 A certified checker based on an external algorithm to compute Smith normal form

```
theory Smith-Certified
  imports
    SNF-Algorithm-Euclidean-Domain
begin
```

This (unspecified) function takes as input the matrix  $A$  and returns five matrices  $(P, S, Q, P', Q')$ , which must satisfy  $S = PAQ$ ,  $S$  is in Smith normal form,  $P'$  and  $Q'$  are the inverse matrices of  $P$  and  $Q$  respectively

The matrices are given in terms of lists for the sake of simplicity when connecting the function to external solvers, like Mathematica or Sage.

```
consts external-SNF ::
  int list list  $\Rightarrow$  int list list  $\times$  int list list  $\times$  int list list  $\times$  int list list  $\times$  int list list
```

We implement the checker by means of the following definition. The checker is implemented in the JNF representation of matrices to make use of the Strassen matrix multiplication algorithm. In case that the certification fails, then the verified Smith normal form algorithm is executed. Thus, we will always get a verified result.

```
definition checker-SNF  $A = ($ 
  let  $A' = \text{mat-to-list } A; m = \text{dim-row } A; n = \text{dim-col } A$  in
  case external-SNF  $A'$  of  $(P\text{-ext}, S\text{-ext}, Q\text{-ext}, P'\text{-ext}, Q'\text{-ext}) \Rightarrow$  let
     $P = \text{mat-of-rows-list } m$   $P\text{-ext};$ 
     $S = \text{mat-of-rows-list } m$   $S\text{-ext};$ 
     $Q = \text{mat-of-rows-list } m$   $Q\text{-ext};$ 
     $P' = \text{mat-of-rows-list } m$   $P'\text{-ext};$ 
     $Q' = \text{mat-of-rows-list } m$   $Q'\text{-ext}$  in
     $(\text{if } \text{dim-row } P = m \wedge \text{dim-col } P = m$ 
       $\wedge \text{dim-row } S = m \wedge \text{dim-col } S = n$ 
       $\wedge \text{dim-row } Q = n \wedge \text{dim-col } Q = n$ 
       $\wedge \text{dim-row } P' = m \wedge \text{dim-col } P' = m$ 
       $\wedge \text{dim-row } Q' = n \wedge \text{dim-col } Q' = n$ 
       $\wedge P * P' = 1_m \wedge Q * Q' = 1_n$ 
```

$\wedge$  *Smith-normal-form-mat*  $S \wedge (S = P * A * Q)$  then  
 $(P, S, Q)$  else *Code.abort* (*STR* "Certification failed") ( $\lambda$  -. *Smith-ED-mxn*  $A$ )  
)

**theorem** *checker-SNF-soudness*:

**assumes**  $A: A \in$  *carrier-mat*  $m$   $n$   
**and**  $c:$  *checker-SNF*  $A = (P, S, Q)$   
**shows** *is-SNF*  $A (P, S, Q)$

**proof** –

**let**  $?ext =$  *external-SNF* (*mat-to-list*  $A$ )

**obtain**  $P$ -*ext*  $S$ -*ext*  $Q$ -*ext*  $P'$ -*ext*  $Q'$ -*ext* **where**  $ext:$   $?ext = (P$ -*ext*,  $S$ -*ext*,  $Q$ -*ext*,  $P'$ -*ext*,  $Q'$ -*ext*)  
**by** (*cases*  $?ext$ , *auto*)

**let**  $?case$ -*external* = *let*

$P =$  *mat-of-rows-list*  $m$   $P$ -*ext*;  
 $S =$  *mat-of-rows-list*  $m$   $S$ -*ext*;  
 $Q =$  *mat-of-rows-list*  $n$   $Q$ -*ext*;  
 $P' =$  *mat-of-rows-list*  $m$   $P'$ -*ext*;  
 $Q' =$  *mat-of-rows-list*  $n$   $Q'$ -*ext* *in*  
 $(dim$ -*row*  $P = m \wedge dim$ -*col*  $P = m$   
 $\wedge dim$ -*row*  $S = m \wedge dim$ -*col*  $S = n$   
 $\wedge dim$ -*row*  $Q = n \wedge dim$ -*col*  $Q = n$   
 $\wedge dim$ -*row*  $P' = m \wedge dim$ -*col*  $P' = m$   
 $\wedge dim$ -*row*  $Q' = n \wedge dim$ -*col*  $Q' = n$   
 $\wedge P * P' = 1_m$   $m \wedge Q * Q' = 1_m$   $n$   
 $\wedge$  *Smith-normal-form-mat*  $S \wedge (S = P * A * Q))$

**show**  $?thesis$

**proof** (*cases*  $?case$ -*external*)

**case** *True*

**define**  $P'$  **where**  $P' =$  *mat-of-rows-list*  $m$   $P'$ -*ext*

**define**  $Q'$  **where**  $Q' =$  *mat-of-rows-list*  $m$   $Q'$ -*ext*

**have**  $S$ -*PAQ*:  $S = P * A * Q$

**and**  $SNF$ - $S$ : *Smith-normal-form-mat*  $S$  **and**  $PP'$ - $1$ :  $P * P' = 1_m$   $m$  **and**

$QQ'$ - $1$ :  $Q * Q' = 1_m$   $n$

**and**  $sm$ - $P$ : *square-mat*  $P$  **and**  $sm$ - $Q$ : *square-mat*  $Q$

**using**  $ext$  *True*  $c$   $A$

**unfolding** *checker-SNF-def* *Let-def* *mat-of-rows-list-def*  $P'$ -*def*  $Q'$ -*def*

**by** (*auto split: if-splits*)

**have**  $inv$ - $P$ : *invertible-mat*  $P$

**proof** (*unfold invertible-mat-def*, *rule conjI*, *rule sm-P*,

*unfold inverts-mat-def*, *rule exI*[*of - P'*], *rule conjI*)

**show**  $*$ :  $P * P' = 1_m$  (*dim-row*  $P$ )

**by** (*metis*  $PP'$ - $1$  *True* *index-mult-mat*(2))

**show**  $P' * P = 1_m$  (*dim-row*  $P'$ )

**proof** (*rule mat-mult-left-right-inverse*)

**show**  $P \in$  *carrier-mat* (*dim-row*  $P'$ ) (*dim-row*  $P'$ )

**by** (*metis*  $*$   $P'$ -*def*  $PP'$ - $1$  *True* *carrier-mat-triv* *index-one-mat*(2)  $sm$ - $P$

*square-mat.elims*(2))

**show**  $P' \in$  *carrier-mat* (*dim-row*  $P'$ ) (*dim-row*  $P'$ )

**by** (*metis*  $P'$ -*def* *True* *carrier-mat-triv*)

```

    show  $P * P' = 1_m$  (dim-row  $P'$ )
      by (metis  $P'$ -def  $PP'-1$  True)
  qed
qed
have  $inv-Q$ : invertible-mat  $Q$ 
proof (unfold invertible-mat-def, rule conjI, rule sm- $Q$ ,
  unfold inverts-mat-def, rule exI[of -  $Q'$ ], rule conjI)
  show *:  $Q * Q' = 1_m$  (dim-row  $Q$ )
    by (metis  $QQ'-1$  True index-mult-mat(2))
  show  $Q' * Q = 1_m$  (dim-row  $Q'$ )
  proof (rule mat-mult-left-right-inverse)
    show 1:  $Q \in$  carrier-mat (dim-row  $Q'$ ) (dim-row  $Q'$ )
    by (metis  $Q'$ -def  $QQ'-1$  True carrier-mat-triv dim-row-mat(1) index-mult-mat(2)
      mat-of-rows-list-def sm- $Q$  square-mat.simps)
    thus  $Q' \in$  carrier-mat (dim-row  $Q'$ ) (dim-row  $Q'$ )
      by (metis * carrier-matD(1) carrier-mat-triv index-mult-mat(3) in-
        dex-one-mat(3))
    show  $Q * Q' = 1_m$  (dim-row  $Q'$ ) using * 1 by auto
  qed
qed
have  $P \in$  carrier-mat  $m$   $m$ 
  by (metis  $PP'-1$  True carrier-matI index-mult-mat(2) sm- $P$  square-mat.simps)
moreover have  $Q \in$  carrier-mat  $n$   $n$ 
  by (metis  $QQ'-1$  True carrier-matI index-mult-mat(2) sm- $Q$  square-mat.simps)
ultimately show ?thesis unfolding is-SNF-def using  $inv-P$   $inv-Q$  SNF- $S$ 
 $S-PAQ$   $A$  by auto
next
case False
hence checker-SNF  $A =$  Smith-ED- $m \times n$   $A$ 
  using ext False  $c$   $A$ 
  unfolding checker-SNF-def Let-def Code.abort-def
  by (smt (verit) carrier-matD case-prod-conv dim-col-mat(1) mat-of-rows-list-def)
  then show ?thesis using Smith-ED.is-SNF-Smith- $m \times n$ [OF  $A$ ]  $c$  unfolding
is-SNF-def
  by auto
qed
qed
end
theory Alternative-Proofs
  imports Smith-Normal-Form.Admits-SNF-From-Diagonal-Iff-Bezout-Ring
    Smith-Normal-Form.Elementary-Divisor-Rings
begin

```

Theorem 2: (C)  $\implies$  (A)

**lemma** *diagonal-2x2-admits-SNF-imp-bezout-ring-JNF*:

```

  assumes admits-SNF:  $\forall A. (A::'a$  mat)  $\in$  carrier-mat 2 2  $\wedge$  isDiagonal-mat  $A$ 
     $\longrightarrow$  ( $\exists P$   $Q. P \in$  carrier-mat 2 2  $\wedge Q \in$  carrier-mat 2 2  $\wedge$  invertible-mat  $P \wedge$ 
invertible-mat  $Q$ 

```

```

     $\wedge$  Smith-normal-form-mat ( $P * A * Q$ )
  shows OFCLASS('a::comm-ring-1, bezout-ring-class)
proof (intro-classes)
  fix a b::'a
  show  $\exists p q d. p * a + q * b = d \wedge d \text{ dvd } a \wedge d \text{ dvd } b \wedge (\forall d'. d' \text{ dvd } a \wedge d' \text{ dvd } b \longrightarrow d' \text{ dvd } d)$ 
  proof (cases a=b)
    case True
      show ?thesis
      by (metis True add.right-neutral comm-semiring-class.distrib dvd-refl mult-1)
    next
      case False note a-not-b = False
      let  $?A = \text{Matrix.mat } 2 \ 2 \ (\lambda(i,j). \text{ if } i = 0 \wedge j = 0 \text{ then } a \text{ else if } i = 1 \wedge j = 1 \text{ then } b \text{ else } 0)$ 
      have A-carrier:  $?A \in \text{carrier-mat } 2 \ 2$  by auto
      moreover have diag-A: isDiagonal-mat  $?A$  by (simp add: isDiagonal-mat-def)
      ultimately obtain P Q where P:  $P \in \text{carrier-mat } 2 \ 2$ 
        and Q:  $Q \in \text{carrier-mat } 2 \ 2$  and inv-P: invertible-mat P and inv-Q:
invertible-mat Q
        and SNF-PAQ: Smith-normal-form-mat ( $P * ?A * Q$ )
      using admits-SNF by blast
      let  $?p = P \ \$\$ (0,0) * Q \ \$\$ (0,0)$ 
      let  $?q = P \ \$\$ (0,1) * Q \ \$\$ (1,0)$ 
      let  $?d = (P * ?A * Q) \ \$\$ (0,0)$ 
      let  $?d' = (P * ?A * Q) \ \$\$ (1,1)$ 
      have d-dvd-d':  $?d \text{ dvd } ?d'$ 
      by (metis (no-types, lifting) A-carrier One-nat-def P Q SNF-PAQ SNF-first-divides-all bot-nat-0.not-eq-extremum less-Suc-numeral mult-carrier-mat pred-numeral-simps(2) zero-neq-numeral)
      have pa-qb-d:  $?p * a + ?q * b = ?d$ 
      proof -
        let  $?U = P * ?A$ 
        have  $?U \ \$\$ (0, 0) = P \ \$\$ (0,0) * ?A \ \$\$ (0,0) + P \ \$\$ (0,1) * ?A \ \$\$ (1,0)$ 
          by (rule mat-mult2-00, insert P, auto)
        also have  $\dots = P \ \$\$ (0,0) * a$  by auto
        finally have 1:  $(P * ?A) \ \$\$ (0, 0) = P \ \$\$ (0,0) * a$  .
        have  $?U \ \$\$ (0, 1) = P \ \$\$ (0,0) * ?A \ \$\$ (0,1) + P \ \$\$ (0,1) * ?A \ \$\$ (1,1)$ 
          by (rule mat-mult2-01, insert P, auto)
        hence 2:  $(P * ?A) \ \$\$ (0, 1) = P \ \$\$ (0,1) * b$  by auto
        have  $?d = ?U \ \$\$ (0, 0) * Q \ \$\$ (0, 0) + ?U \ \$\$ (0, 1) * Q \ \$\$ (1, 0)$ 
          by (rule mat-mult2-00, insert Q P, auto)
        also have  $\dots = ?p * a + ?q * b$  unfolding 1 unfolding 2 by auto
        finally show ?thesis ..
      qed
    have i: ideal-generated  $\{a, b\} = \text{ideal-generated } \{?d\}$ 
    proof
      show ideal-generated  $\{?d\} \subseteq \text{ideal-generated } \{a, b\}$ 
      proof (rule ideal-generated-subset2, rule ballI, simp)
        fix x

```

```

let ?f = λx. if x = a then ?p else ?q
show ?d ∈ ideal-generated {a, b}
  unfolding ideal-explicit
  by simp (rule exI[of - ?f], rule exI[of - {a,b}],
    insert a-not-b One-nat-def pa-qb-d, auto)
qed
show ideal-generated {a, b} ⊆ ideal-generated {?d}
proof -
  obtain P' where inverts-mat-P': inverts-mat P P' ∧ inverts-mat P' P
    using inv-P unfolding invertible-mat-def by auto
  have P': P' ∈ carrier-mat 2 2
    using inverts-mat-P'
    unfolding carrier-mat-def inverts-mat-def
    by (auto,metis P carrier-matD index-mult-mat(3) one-carrier-mat)+
  obtain Q' where inverts-mat-Q': inverts-mat Q Q' ∧ inverts-mat Q' Q
    using inv-Q unfolding invertible-mat-def by auto
  have Q': Q' ∈ carrier-mat 2 2
    using inverts-mat-Q'
    unfolding carrier-mat-def inverts-mat-def
    by (auto,metis Q carrier-matD index-mult-mat(3) one-carrier-mat)+
  have rw-PAQ: (P'*(P*?A*Q)*Q') $$ (i, i) = ?A $$ (i,i) for i
    using inv-P'PAQQ'[OF A-carrier P - - Q P' Q'] inverts-mat-P' in-
verts-mat-Q' by auto
  have diag-PAQ: isDiagonal-mat (P*?A*Q)
    using SNF-PAQ unfolding Smith-normal-form-mat-def by auto
  have PAQ-carrier: (P*?A*Q) ∈ carrier-mat 2 2 using P Q by auto
  have z1: 0 < (2::nat) and z2: 1 < (2::nat) by auto
  obtain f where f: (P'*(P*?A*Q)*Q') $$ (0, 0) = (∑ i ∈ set (diag-mat
(P*?A*Q)). f i * i)
    using exists-f-PAQ-Aii[OF diag-PAQ P' PAQ-carrier Q' z1] by blast
  obtain g where g: (P'*(P*?A*Q)*Q') $$ (1, 1) = (∑ i ∈ set (diag-mat
(P*?A*Q)). g i * i)
    using exists-f-PAQ-Aii[OF diag-PAQ P' PAQ-carrier Q' z2] by blast
  have A00: ?A $$ (0, 0) = (∑ i ∈ set (diag-mat (P*?A*Q)). f i * i)
    using rw-PAQ[of 0] using f by presburger
  have A11: ?A $$ (1, 1) = (∑ i ∈ set (diag-mat (P*?A*Q)). g i * i)
    using rw-PAQ[of 1] using g by presburger
  have d-dvd-a: ?d dvd a using A00 d-dvd-d'
    by (auto, smt (verit, best) A00 A-carrier P Q S00-dvd-all-A SNF-PAQ
inv-P inv-Q
  numeral-2-eq-2 zero-less-Suc)
  have d-dvd-b: ?d dvd b using A11 d-dvd-d'
    by (smt (verit, ccfv-threshold) A-carrier One-nat-def P Q S00-dvd-all-A
SNF-PAQ
  index-mat(1) inv-P inv-Q lessI nat.simps(3) numeral-2-eq-2 split-conv)
  have 1: a ∈ ideal-generated {?d} and 2: b ∈ ideal-generated {?d}
  using d-dvd-a d-dvd-b dvd-ideal-generated-singleton' ideal-generated-subset-generator
  by blast+
  show ?thesis by (rule ideal-generated-subset2, insert 1 2, auto)

```

```

    qed
  qed
  have  $\exists p q. p * a + q * b = ?d$  by (rule ideal-generated-pair-exists[OF i])
  moreover have  $d \text{ dvd } a$ :  $?d \text{ dvd } a$  and  $d \text{ dvd } b$ :  $?d \text{ dvd } b$ 
    using i ideal-generated-singleton-dvd by blast+
  moreover have  $(\forall d'. d' \text{ dvd } a \wedge d' \text{ dvd } b \longrightarrow d' \text{ dvd } ?d)$  using ideal-generated-dvd[OF i] by auto
  ultimately show ?thesis
    by blast
  qed
  qed

```

Theorem 2: (A)  $\implies$  (C)

```

lemma bezout-ring-imp-diagonal-2x2-admits-SNF-JNF:
  assumes c: OFCLASS('a::comm-ring-1, bezout-ring-class)
  shows  $\forall A. (A::'a \text{ mat}) \in \text{carrier-mat } 2 \ 2 \wedge \text{isDiagonal-mat } A$ 
 $\longrightarrow (\exists P Q. P \in \text{carrier-mat } 2 \ 2 \wedge Q \in \text{carrier-mat } 2 \ 2$ 
 $\wedge \text{invertible-mat } P \wedge \text{invertible-mat } Q \wedge \text{Smith-normal-form-mat } (P * A * Q))$ 
  using bezout-ring-imp-diagonal-admits-SNF-JNF
    [OF OFCLASS-bezout-ring-imp-class-bezout-ring[where ?'a='a, OF c]]
  unfolding admits-SNF-JNF-def
  using  $\langle \forall A. \text{admits-SNF-JNF } A \rangle$  admits-SNF-JNF-alt-def by blast

```

Theorem 2: (A)  $\iff$  (C)

```

lemma diagonal-2x2-admits-SNF-iff-bezout-ring:
  shows OFCLASS('a::comm-ring-1, bezout-ring-class)
 $\equiv (\bigwedge A::'a \text{ mat}. A \in \text{carrier-mat } 2 \ 2 \longrightarrow \text{admits-SNF-JNF } A)$  (is ?lhs  $\equiv$  ?rhs)
proof
  fix A::'a mat
  assume c: OFCLASS('a, bezout-ring-class)
  show  $A \in \text{carrier-mat } 2 \ 2 \longrightarrow \text{admits-SNF-JNF } A$ 
    using bezout-ring-imp-diagonal-admits-SNF-JNF
      [OF OFCLASS-bezout-ring-imp-class-bezout-ring[where ?'a='a, OF c]]
    unfolding admits-SNF-JNF-def by blast
next
  assume rhs:  $(\bigwedge A::'a \text{ mat}. A \in \text{carrier-mat } 2 \ 2 \longrightarrow \text{admits-SNF-JNF } A)$ 
  show OFCLASS('a::comm-ring-1, bezout-ring-class)
    by (rule diagonal-2x2-admits-SNF-imp-bezout-ring-JNF, insert rhs, simp add:
admits-SNF-JNF-def)
qed

```

Theorem 2: (B)  $\iff$  (C)

```

lemma diagonal-2x2-admits-SNF-iff-diagonal-admits-SNF:
  shows  $(\forall (A::'a::\text{comm-ring-1 mat}). \text{admits-SNF-JNF } A) =$ 
 $(\forall (A::'a \text{ mat}) \in \text{carrier-mat } 2 \ 2. \text{admits-SNF-JNF } A)$ 
proof
  assume  $\forall A::'a \text{ mat}. \text{admits-SNF-JNF } A$ 
  thus  $\forall (A::'a \text{ mat}) \in \text{carrier-mat } 2 \ 2. \text{admits-SNF-JNF } A$ 
    by (insert admits-SNF-JNF-alt-def, blast)

```



**next**  
**assume**  $\forall A::'a \text{ mat} \in \text{carrier-mat } 2 \ 2. \text{ admits-SNF-JNF } A$   
**hence**  $H: \text{ OFCLASS}('a, \text{ bezout-ring-class})$   
**using** *diagonal-2x2-admits-SNF-iff-bezout-ring*[**where**  $?'a = 'a$ ] **by** *auto*  
**show**  $\forall A::'a \text{ mat}. \text{ admits-SNF-JNF } A$   
**using** *bezout-ring-imp-diagonal-admits-SNF-JNF*  
 $[\text{ OF OFCLASS-bezout-ring-imp-class-bezout-ring}[\text{where } ?'a='a, \text{ OF } H]]$   
**by** *simp*  
**qed**

Theorem 2: final statements

**theorem** *Theorem2-final*:

**shows**  $A\text{-imp-}B: \text{ OFCLASS}('a::\text{comm-ring-1}, \text{ bezout-ring-class})$   
 $\implies (\forall A::'a \text{ mat}. \text{ admits-SNF-JNF } A)$   
**and**  $B\text{-imp-}C: (\forall A::'a \text{ mat}. \text{ admits-SNF-JNF } A) \implies$   
 $(\forall (A::'a \text{ mat}) \in \text{carrier-mat } 2 \ 2. \text{ admits-SNF-JNF } A)$   
**and**  $C\text{-imp-}A: (\forall (A::'a \text{ mat}) \in \text{carrier-mat } 2 \ 2. \text{ admits-SNF-JNF } A)$   
 $\implies \text{ OFCLASS}('a::\text{comm-ring-1}, \text{ bezout-ring-class})$

**proof**

**fix**  $A::'a \text{ mat}$   
**assume**  $H: \text{ OFCLASS}('a, \text{ bezout-ring-class})$   
**show**  $\text{ admits-SNF-JNF } A$   
**using** *bezout-ring-imp-diagonal-admits-SNF-JNF*[*OF OFCLASS-bezout-ring-imp-class-bezout-ring*[**where**  
 $?'a='a, \text{ OF } H$ ]]  
**by** *simp*  
**next**  
**assume**  $\forall A::'a \text{ mat}. \text{ admits-SNF-JNF } A$   
**thus**  $\forall (A::'a \text{ mat}) \in \text{carrier-mat } 2 \ 2. \text{ admits-SNF-JNF } A$   
**by** (*insert admits-SNF-JNF-alt-def, blast*)  
**next**  
**assume**  $\forall (A::'a \text{ mat}) \in \text{carrier-mat } 2 \ 2. \text{ admits-SNF-JNF } A$   
**thus**  $\text{ OFCLASS}('a, \text{ bezout-ring-class})$   
**using** *diagonal-2x2-admits-SNF-iff-bezout-ring*[**where**  $?'a = 'a$ ] **by** *auto*  
**qed**

**theorem** *Theorem2-final'*:

**shows**  $A\text{-eq-}B: \text{ OFCLASS}('a::\text{comm-ring-1}, \text{ bezout-ring-class}) \equiv (\bigwedge A::'a \text{ mat}. \text{ admits-SNF-JNF } A)$   
**and**  $A\text{-eq-}C: \text{ OFCLASS}('a::\text{comm-ring-1}, \text{ bezout-ring-class}) \equiv$   
 $(\bigwedge (A::'a \text{ mat}). A \in \text{carrier-mat } 2 \ 2 \longrightarrow \text{ admits-SNF-JNF } A)$   
**and**  $B\text{-eq-}C: (\forall (A::'a::\text{comm-ring-1 mat}). \text{ admits-SNF-JNF } A) =$   
 $(\forall (A::'a \text{ mat}) \in \text{carrier-mat } 2 \ 2. \text{ admits-SNF-JNF } A)$   
**using** *diagonal-admits-SNF-iff-bezout-ring'*  
**using** *diagonal-2x2-admits-SNF-iff-bezout-ring*  
**using** *diagonal-2x2-admits-SNF-iff-diagonal-admits-SNF* **by** *auto*

Theorem 2: final statement in HA. (A)  $\iff$  (C).

**theorem** *Theorem2-A-eq-C-HA*:

```

  OFCLASS('a::comm-ring-1, bezout-ring-class) ≡ (∧(A::'a^2^2). admits-SNF-HA
A)
proof
  fix A::'a^2^2
  assume H: OFCLASS('a, bezout-ring-class)
  let ?A = Mod-Type-Connect.from-hma_m A
  have A: ?A ∈ carrier-mat 2 2 by auto
  have [transfer-rule]: Mod-Type-Connect.HMA-M ?A A
    unfolding Mod-Type-Connect.HMA-M-def A by auto
  have admits-SNF-JNF ?A using A-imp-B[OF H] by auto
  thus admits-SNF-HA A by transfer'
next
  assume a: (∧A::'a^2^2. admits-SNF-HA A)
  have [transfer-rule]: Mod-Type-Connect.HMA-M (Mod-Type-Connect.from-hma_m
A) A for A::'a^2^2
    unfolding Mod-Type-Connect.HMA-M-def by auto
  have a': (∧A::'a^2^2. admits-SNF-JNF (Mod-Type-Connect.from-hma_m A))
  proof –
    fix A::'a^2^2
    have ad: admits-SNF-HA A using a by simp
    let ?A = Mod-Type-Connect.from-hma_m A
    have A: ?A ∈ carrier-mat 2 2 by auto
    have [transfer-rule]: Mod-Type-Connect.HMA-M ?A A
      unfolding Mod-Type-Connect.HMA-M-def A by auto
    show admits-SNF-JNF (Mod-Type-Connect.from-hma_m A) using ad by trans-
fer'
  qed
  have (∀ A::'a^2^2. admits-SNF-JNF (Mod-Type-Connect.from-hma_m A))
    = (∀ (A::'a mat) ∈ carrier-mat 2 2. admits-SNF-JNF A)
  proof (auto)
  fix A::'a mat assume a1: ∀ A::'a^2^2. admits-SNF-JNF (Mod-Type-Connect.from-hma_m
A)
    and A ∈ carrier-mat 2 2
    thus admits-SNF-JNF A by (metis Mod-Type-Connect.from-hma-to-hma_m
One-nat-def UNIV-1 a1
card.empty card.insert card-bit0 empty-iff finite mult.right-neutral)
  next
  fix A::'a^2^2 assume ∀ A ∈ carrier-mat 2 2. admits-SNF-JNF A
  have ad: admits-SNF-HA A using a by simp
  let ?A = Mod-Type-Connect.from-hma_m A
  have A: ?A ∈ carrier-mat 2 2 by auto
  have [transfer-rule]: Mod-Type-Connect.HMA-M ?A A
    unfolding Mod-Type-Connect.HMA-M-def A by auto
  show admits-SNF-JNF (Mod-Type-Connect.from-hma_m A) using ad by trans-
fer'
  qed
  hence (∧A::'a mat. A ∈ carrier-mat 2 2 → admits-SNF-JNF A) using a' by
auto
  thus OFCLASS('a, bezout-ring-class) using Theorem2-final'[where ?'a='a] by

```

*auto*  
**qed**

Hermite implies Bezout

Theorem 3, proof for 1x2 matrices

**lemma** *theorem3-restricted-12-part1*:

**assumes**  $T: (\forall a b::'a::\text{comm-ring-1}. \exists a1 b1 d. a = a1*d \wedge b = b1*d$   
 $\wedge \text{ideal-generated } \{a1, b1\} = \text{ideal-generated } \{1\})$

**shows**  $\forall (A::'a \text{ mat}) \in \text{carrier-mat } 1 \ 2. \text{ admits-triangular-reduction } A$

**proof** (*rule*)

**fix**  $A::'a \text{ mat}$

**assume**  $A: A \in \text{carrier-mat } 1 \ 2$

**let**  $?a = A \ \$\$ (0,0)$

**let**  $?b = A \ \$\$ (0,1)$

**obtain**  $a1 b1 d$  **where**  $a: ?a = a1*d$  **and**  $b: ?b = b1*d$

**and**  $i: \text{ideal-generated } \{a1, b1\} = \text{ideal-generated } \{1\}$

**using**  $T$  **by** *blast*

**obtain**  $s t$  **where**  $sa1tb1: s*a1 + t*b1 = 1$  **using** *ideal-generated-pair-exists-pq1* [*OF*  
 $i[\text{simplified}]$ ] **by** *blast*

**let**  $?Q = \text{Matrix.mat } 2 \ 2 (\lambda(i,j). \text{ if } i = 0 \wedge j = 0 \text{ then } s \text{ else}$   
 $\text{ if } i = 0 \wedge j = 1 \text{ then } -b1 \text{ else}$   
 $\text{ if } i = 1 \wedge j = 0 \text{ then } t \text{ else } a1)$

**have**  $Q: ?Q \in \text{carrier-mat } 2 \ 2$  **by** *auto*

**have**  $\text{det-}Q: \text{Determinant.det } ?Q = 1$  **unfolding** *det-2* [*OF*  $Q$ ]

**using**  $sa1tb1$  **by** (*simp add: mult commute*)

**hence**  $\text{inv-}Q: \text{invertible-mat } ?Q$  **using** *invertible-iff-is-unit-JNF* [*OF*  $Q$ ] **by** *auto*

**have**  $\text{lower-}AQ: \text{lower-triangular } (A * ?Q)$

**proof** –

**have**  $\text{Matrix.row } A \ 0 \ \$v \ \text{Suc } 0 * a1 = \text{Matrix.row } A \ 0 \ \$v \ 0 * b1$  **if**  $j2: j < 2$   
**and**  $j0: 0 < j$  **for**  $j$

**by** (*metis*  $A \ \text{One-nat-def } a \ b \ \text{carrier-matD}(1) \ \text{carrier-matD}(2) \ \text{index-row}(1)$   
 $\text{lessI}$

$\text{more-arith-simps}(11) \ \text{mult commute numeral-2-eq-2 pos2}$ )

**thus**  $?thesis$  **unfolding** *lower-triangular-def* **using**  $A$

**by** (*auto simp add: scalar-prod-def sum-two-rw*)

**qed**

**show** *admits-triangular-reduction*  $A$

**unfolding** *admits-triangular-reduction-def* **using**  $\text{lower-}AQ \ \text{inv-}Q \ Q \ A$  **by** *force*

**qed**

**lemma** *theorem3-restricted-12-part2*:

**assumes**  $1: \forall (A::'a::\text{comm-ring-1 mat}) \in \text{carrier-mat } 1 \ 2. \text{ admits-triangular-reduction } A$

**shows**  $\forall a b::'a. \exists a1 b1 d. a = a1*d \wedge b = b1*d \wedge \text{ideal-generated } \{a1, b1\} =$   
 $\text{ideal-generated } \{1\}$

**proof** (*rule allI*)+

**fix**  $a b::'a$

**let**  $?A = \text{Matrix.mat } 1 \ 2 \ (\lambda(i,j). \text{ if } i = 0 \wedge j = 0 \text{ then } a \text{ else } b)$   
**obtain**  $Q$  **where**  $AQ$ : lower-triangular ( $?A*Q$ ) **and**  $\text{inv-}Q$ : invertible-mat  $Q$   
**and**  $Q$ :  $Q \in \text{carrier-mat } 2 \ 2$   
**using** 1 **unfolding** admits-triangular-reduction-def **by** fastforce  
**hence** [simp]:  $\text{dim-col } Q = 2$  **and** [simp]:  $\text{dim-row } Q = 2$  **by** auto  
**let**  $?s = Q \ \$\$ (0,0)$   
**let**  $?t = Q \ \$\$ (1,0)$   
**let**  $?a1 = Q \ \$\$ (1,1)$   
**let**  $?b1 = -(Q \ \$\$ (0,1))$   
**let**  $?d = (?A*Q) \ \$\$ (0,0)$   
**have**  $\text{ab1-ba1}: a*?b1 = b*?a1$   
**proof** –  
**have**  $(?A*Q) \ \$\$ (0,1) = (\sum i = 0..<2. (\text{if } i = 0 \text{ then } a \text{ else } b) * Q \ \$\$ (i, \text{Suc } 0))$   
**unfolding** times-mat-def col-def scalar-prod-def **by** auto  
**also have**  $\dots = (\sum i \in \{0,1\}. (\text{if } i = 0 \text{ then } a \text{ else } b) * Q \ \$\$ (i, \text{Suc } 0))$   
**by** (rule sum.cong, auto)  
**also have**  $\dots = - a*?b1 + b*?a1$  **by** auto  
**finally have**  $(?A*Q) \ \$\$ (0,1) = - a*?b1 + b*?a1$  **by** simp  
**moreover have**  $(?A*Q) \ \$\$ (0,1) = 0$  **using**  $AQ$  **unfolding** lower-triangular-def  
**by** auto  
**ultimately show** ?thesis  
**by** (metis add-left-cancel more-arith-simps(3) more-arith-simps(7))  
**qed**  
**have**  $\text{sa-tb-d}: ?s*a + ?t*b = ?d$   
**proof** –  
**have**  $?d = (\sum i = 0..<2. (\text{if } i = 0 \text{ then } a \text{ else } b) * Q \ \$\$ (i, 0))$   
**unfolding** times-mat-def col-def scalar-prod-def **by** auto  
**also have**  $\dots = (\sum i \in \{0,1\}. (\text{if } i = 0 \text{ then } a \text{ else } b) * Q \ \$\$ (i, 0))$  **by** (rule sum.cong, auto)  
**also have**  $\dots = ?s*a + ?t*b$  **by** auto  
**finally show** ?thesis **by** simp  
**qed**  
**have**  $\text{det-Q-dvd-1}: (\text{Determinant.det } Q \ \text{dvd } 1)$   
**using** invertible-iff-is-unit-JNF[OF  $Q$ ]  $\text{inv-}Q$  **by** auto  
**moreover have**  $\text{det-Q-eq}: \text{Determinant.det } Q = ?s*?a1 + ?t*?b1$  **unfolding**  $\text{det-2}[OF \ Q]$  **by** simp  
**ultimately have**  $?s*?a1 + ?t*?b1 \ \text{dvd } 1$  **by** auto  
**from this obtain**  $u$  **where**  $u\text{-eq}: ?s*?a1 + ?t*?b1 = u$  **and**  $u: u \ \text{dvd } 1$  **by** auto  
**hence**  $\text{eq1}: ?s*?a1*a + ?t*?b1*a = u*a$   
**by** (metis ring-class.ring-distrib(2))  
**hence**  $?s*?a1*a + ?t*?a1*b = u*a$   
**by** (metis (no-types, lifting) ab1-ba1 mult.assoc mult.commute)  
**hence**  $a1d\text{-ua}: ?a1*?d = u*a$   
**by** (smt (verit) Groups.mult-ac(2) distrib-left more-arith-simps(11) sa-tb-d)  
**hence**  $b1d\text{-ub}: ?b1*?d = u*b$   
**by** (smt (verit, ccfv-threshold) Groups.mult-ac(2) Groups.mult-ac(3) ab1-ba1 distrib-right sa-tb-d u-eq)  
**obtain**  $\text{inv-}u$  **where**  $\text{inv-}u: \text{inv-}u * u = 1$  **using**  $u$  **unfolding** dvd-def

```

    by (metis mult.commute)
  hence inv-u-dvd-1: inv-u dvd 1 unfolding dvd-def by auto
  have cond1: (inv-u*?b1)*?d = b using b1d-ub inv-u
  by (metis (no-types, lifting) Groups.mult-ac(3) more-arith-simps(11) more-arith-simps(6))
  have cond2: (inv-u*?a1)*?d = a using a1d-ua inv-u
  by (metis (no-types, lifting) Groups.mult-ac(3) more-arith-simps(11) more-arith-simps(6))
  have ideal-generated {inv-u*?a1, inv-u*?b1} = ideal-generated {?a1,?b1}
  by (rule ideal-generated-mult-unit2[OF inv-u-dvd-1])
  also have ... = UNIV using ideal-generated-pair-UNIV[OF u-eq u] by simp
  finally have cond3: ideal-generated {inv-u*?a1, inv-u*?b1} = ideal-generated
  {1} by auto
  show  $\exists a1\ b1\ d. a = a1 * d \wedge b = b1 * d \wedge \text{ideal-generated } \{a1, b1\} =$ 
  ideal-generated {1}
  by (rule exI[of - inv-u*?a1], rule exI[of - inv-u*?b1], rule exI[of - ?d],
  insert cond1 cond2 cond3, auto)
qed

```

lemma *Hermite-ring-imp-Bezout-ring*:

```

  assumes H: OFCLASS('a::comm-ring-1, Hermite-ring-class)
  shows OFCLASS('a::comm-ring-1, bezout-ring-class)
proof (intro-classes)
  fix a b::'a
  let ?A = Matrix.mat 1 2 ( $\lambda(i,j). \text{if } i = 0 \wedge j = 0 \text{ then } a \text{ else } b$ )
  have *: ( $\bigwedge(A::'a::comm-ring-1\ \text{mat}). \text{admits-triangular-reduction } A$ )
  using OFCLASS-Hermite-ring-def[where ?'a='a] H by auto
  have admits-triangular-reduction ?A
  using H unfolding OFCLASS-Hermite-ring-def by auto
  have  $\exists a1\ b1\ d. a = a1*d \wedge b = b1*d \wedge \text{ideal-generated } \{a1,b1\} = \text{ideal-generated}$ 
  {1}
  using theorem3-restricted-12-part2 * by auto
  from this obtain a1 b1 d where a-a'd:  $a = a1*d$  and b-b'd:  $b = b1*d$ 
  and a'b'-1: ideal-generated {a1,b1} = ideal-generated {1}
  by blast
  obtain p q where  $p * a1 + q * b1 = 1$  using a'b'-1
  using ideal-generated-pair-exists-UNIV by blast
  hence pa-qb-d:  $p * a + q * b = d$  unfolding a-a'd b-b'd
  by (metis mult.assoc mult-1 ring-class.ring-distrib(2))
  moreover have d-dvd-a:  $d\ \text{dvd}\ a$  using a-a'd by auto
  moreover have d-dvd-b:  $d\ \text{dvd}\ b$  using b-b'd by auto
  moreover have ( $\forall d'. d'\ \text{dvd}\ a \wedge d'\ \text{dvd}\ b \longrightarrow d'\ \text{dvd}\ d$ ) using pa-qb-d by force
  ultimately show  $\exists p\ q\ d. p * a + q * b = d \wedge d\ \text{dvd}\ a \wedge d\ \text{dvd}\ b$ 
   $\wedge (\forall d'. d'\ \text{dvd}\ a \wedge d'\ \text{dvd}\ b \longrightarrow d'\ \text{dvd}\ d)$  by blast
qed

```

end