

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

1	Definition of Smith normal form in HOL Analysis	1
1.1	Definitions	1
1.2	Basic properties	2
2	Algorithm to transform a diagonal matrix into its Smith normal form	3
2.1	Implementation of the algorithm	4
2.2	Code equations to get an executable version	5
2.3	Examples of execution	6
2.4	Soundness of the algorithm	6
2.5	Implementation and formal proof of the matrices P and Q which transform the input matrix by means of elementary operations.	14
2.6	The final soundness theorem	21
3	A new bridge to convert theorems from JNF to HOL Analysis and vice-versa, based on the <i>mod-type</i> class	21

4	Missing results	29
4.1	Miscellaneous lemmas	30
4.2	Transfer rules for the HMA_Connect file of the Perron-Frobenius development	30
4.3	Lemmas obtained from HOL Analysis using local type definitions	30
4.4	Lemmas about matrices, submatrices and determinants	31
4.5	Lemmas about <i>sorted lists</i> , <i>insort</i> and <i>pick</i>	37
5	The Cauchy–Binet formula	38
5.1	Previous missing results about <i>pick</i> and <i>in sort</i>	39
5.2	Start of the proof	39
5.3	Final theorem	44
6	Definition of Smith normal form in JNF	44
7	Some theorems about rings and ideals	46
7.1	Missing properties on ideals	46
7.2	An equivalent characterization of Bézout rings	50
8	Connection between mod-ring and mod-type	52
9	Generality of the Algorithm to transform from diagonal to Smith normal form	53
9.1	Proof of the \Leftarrow implication in HA.	53
9.2	Trying to prove the \Rightarrow implication in HA.	54
9.3	Proof of the \Rightarrow implication in JNF.	54
9.4	Trying to transfer the \Rightarrow implication to HA.	55
9.5	Transferring the \Leftarrow implication from HA to JNF using transfer rules and local type definitions	56
9.6	Final theorem in JNF	58
10	Uniqueness of the Smith normal form	58
10.1	More specific results about submatrices	59
10.2	On the minors of a diagonal matrix	60
10.3	Relating minors and GCD	61
10.4	Final theorem	62
10.5	Uniqueness fixing a complete set of non-associates	63
11	The Cauchy–Binet formula in HOL Analysis	64
11.1	Definition of submatrices in HOL Analysis	64
11.2	Transferring the proof from JNF to HOL Analysis	65

12 Diagonalizing matrices in JNF and HOL Analysis	65
12.1 Diagonalizing matrices in JNF	65
12.2 Implementation and soundness result moved to HOL Analysis.	66
13 Smith normal form algorithm based on two steps in HOL Analysis	66
13.1 The implementation	66
13.2 Soundness in HOL Analysis	67
14 Algorithm to transform a diagonal matrix into its Smith normal form in JNF	67
14.1 Attempt with the third option: definitions and conditional transfer rules	68
14.2 Attempt with the second option: implementation and soundness in JNF	68
14.3 Applying local type definitions	72
14.4 The final result	74
15 Smith normal form algorithm based on two steps in JNF	75
15.1 Moving the result from HOL Analysis to JNF	75
16 A general algorithm to transform a matrix into its Smith normal form	75
16.1 Previous definitions and lemmas	76
16.2 Previous operations	76
16.3 The implementation	79
16.3.1 Case $1 \times n$	81
16.3.2 Case $n \times 1$	82
16.3.3 Case $2 \times n$	82
16.3.4 Case $n \times 2$	84
16.3.5 Case $m \times n$	84
16.4 Soundness theorem	86
17 The Smith normal form algorithm in HOL Analysis	86
17.1 Transferring the result from JNF to HOL Anaylsis	86
17.2 Soundness in HOL Anaylsis	87
18 Elementary divisor rings	87
18.1 Previous definitions and basic properties of Hermite ring	88
18.2 The class that represents elementary divisor rings	89
18.3 Hermite ring implies Bézout ring	89
18.4 Elementary divisor ring implies Hermite ring	89
18.5 Characterization of Elementary divisor rings	93
18.6 Final theorem	93

19 Executable Smith normal form algorithm over Euclidean domains	94
19.1 Previous code equations	95
19.2 An executable algorithm to transform 2×2 matrices into its Smith normal form in HOL Analysis	95
19.3 An executable algorithm to transform 2×2 matrices into its Smith normal form in JNF	97
19.4 An executable algorithm to transform 1×2 matrices into its Smith normal form	98
19.5 The final executable algorithm to transform any matrix into its Smith normal form	99
20 A certified checker based on an external algorithm to compute Smith normal form	99

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-upk A k = ( $\forall a b. (to\text{-}nat a \neq to\text{-}nat b \wedge (to\text{-}nat a < k \vee to\text{-}nat b < k)) \longrightarrow A \$ a \$ b = 0$ )
definition isDiagonal A = ( $\forall a b. to\text{-}nat a \neq to\text{-}nat b \longrightarrow A \$ a \$ b = 0$ )
```

```
lemma isDiagonal-intro:
fixes A::'a::{zero}^cols::mod-type^rows::mod-type
assumes  $\bigwedge a::'rows. \bigwedge b::'cols. to\text{-}nat a = to\text{-}nat b$ 
shows isDiagonal A
⟨proof⟩
```

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

```

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 = Smith-normal-form-upk A (min (nrows A) (ncols A))
 $\langle \text{proof} \rangle$

lemma Smith-normal-form-upk-0[simp]: Smith-normal-form-upk A 0
 $\langle \text{proof} \rangle$

lemma Smith-normal-form-upk-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-upk A k
 $\langle \text{proof} \rangle$

lemma Smith-normal-form-upk-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-upk A k
shows Smith-normal-form-upk A k
 $\langle \text{proof} \rangle$

lemma Smith-normal-form-upk-condition1:

fixes A::'a::{bezout-ring} ^'cols::mod-type ^'rows::mod-type
assumes Smith-normal-form-upk A k
and to-nat a = to-nat b **and** to-nat a + 1 < k **and** to-nat b + 1 < k
shows A \\$ a \\$ b dvd A \\$ (a+1) \\$ (b+1)
 $\langle \text{proof} \rangle$

lemma Smith-normal-form-upk-condition2:

fixes A::'a::{bezout-ring} ^'cols::mod-type ^'rows::mod-type
assumes Smith-normal-form-upk A k
and to-nat a \neq to-nat b **and** (to-nat a < k \vee to-nat b < k)
shows ((A \\$ a) \\$ b) = 0
 $\langle \text{proof} \rangle$

lemma Smith-normal-form-upk1-intro:

```

fixes A::'a::{bezout-ring} ^'cols::mod-type ^'rows::mod-type
assumes s: Smith-normal-form-upk A k
and cond1: A $ from-nat (k - 1) $ from-nat (k-1) dvd A $ (from-nat k) $
(from-nat k)
and cond2a:  $\forall a. \text{to-nat } a > k \longrightarrow A \$ a \$ \text{from-nat } k = 0$ 
and cond2b:  $\forall b. \text{to-nat } b > k \longrightarrow A \$ \text{from-nat } k \$ b = 0$ 
shows Smith-normal-form-upk A (Suc k)
⟨proof⟩

lemma Smith-normal-form-upk1-intro-diagonal:
fixes A::'a::{bezout-ring} ^'cols::mod-type ^'rows::mod-type
assumes s: Smith-normal-form-upk A k
and d: isDiagonal A
and cond1: A $ from-nat (k - 1) $ from-nat (k-1) dvd A $ (from-nat k) $
(from-nat k)
shows Smith-normal-form-upk A (Suc k)
⟨proof⟩

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))
⟨proof⟩

```

2.1 Implementation of the algorithm

```

type-synonym 'a bezout = 'a ⇒ 'a ⇒ 'a × 'a × 'a × 'a × '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 *from-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. if  $a = \text{from-nat } i \wedge b = \text{from-nat } i$  then  $d$  else
    if  $a = \text{from-nat } j \wedge b = \text{from-nat } j$ 
    then  $v * (A \$ (\text{from-nat } j) \$ (\text{from-nat } j))$  else  $A \$ a \$ b$ )
```

```
fun diagonal-to-Smith-i ::  

  nat list  $\Rightarrow$  'a:{bezout-ring}  $\wedge$ 'cols::mod-type  $\wedge$ 'rows::mod-type  $\Rightarrow$  nat  $\Rightarrow$  ('a bezout)  

   $\Rightarrow$  'a  $\wedge$ 'cols::mod-type  $\wedge$ 'rows::mod-type  

where  

  diagonal-to-Smith-i [] A i bezout = A |  

  diagonal-to-Smith-i (j#xs) A i bezout = (|  

    if  $A \$ (\text{from-nat } i) \$ (\text{from-nat } i)$  dvd  $A \$ (\text{from-nat } j) \$ (\text{from-nat } j)$   

    then diagonal-to-Smith-i xs A i bezout  

    else let (p, q, u, v, d) = bezout ( $A \$ \text{from-nat } i \$ \text{from-nat } i$ ) ( $A \$ \text{from-nat } j \$ \text{from-nat } j$ );  

      A' = diagonal-step A i j d v  

      in diagonal-to-Smith-i xs A' i bezout  

  )
```

```
definition Diagonal-to-Smith-row-i A i bezout  

  = diagonal-to-Smith-i [i+1..<min (nrows A) (ncols A)] A i bezout
```

```
fun diagonal-to-Smith-aux :: 'a:{bezout-ring}  $\wedge$ 'cols::mod-type  $\wedge$ 'rows::mod-type  

   $\Rightarrow$  nat list  $\Rightarrow$  ('a bezout)  $\Rightarrow$  'a  $\wedge$ 'cols::mod-type  $\wedge$ 'rows::mod-type  

where  

  diagonal-to-Smith-aux A [] bezout = A |  

  diagonal-to-Smith-aux A (i#xs) bezout  

  = diagonal-to-Smith-aux (Diagonal-to-Smith-row-i A i bezout) xs 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 bezout
  = diagonal-to-Smith-aux A [0..<min (nrows A) (ncols A) - 1] bezout
```

2.2 Code equations to get an executable version

```
definition diagonal-step-row
  where diagonal-step-row A i j c v a = vec-lambda (%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)
```

```
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)
  ⟨proof⟩
```

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

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))
  ⟨proof⟩
```

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^3^3
  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^4^4
  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^4^5
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*
 $\langle \text{proof} \rangle$

lemma *ncols-diagonal-step*[simp]: *ncols* (*diagonal-step* *A* *i j c v*) = *ncols A*
 $\langle \text{proof} \rangle$

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* =
 $(\text{fst}(\text{bezout } a \ b),$
 $\text{fst} (\text{snd} (\text{bezout } a \ b)),$
 $\text{fst} (\text{snd} (\text{snd} (\text{bezout } a \ b))),$
 $\text{fst} (\text{snd} (\text{snd} (\text{snd} (\text{bezout } a \ b)))),$
 $\text{snd} (\text{snd} (\text{snd} (\text{snd} (\text{bezout } a \ b)))) \langle \text{proof} \rangle$

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} ^'b::mod-type ^'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*: $\forall x. x \in \text{set xs} \longrightarrow x < \min (\text{nrows } A) (\text{ncols } A)$
shows (*diagonal-to-Smith-i xs A i bezout*) \$ *a* \$ *b* = *A \$ a \$ b*
 $\langle \text{proof} \rangle$

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) \Rightarrow 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) \Rightarrow d*

```

shows diagonal-step A i j d v $ from-nat i $ from-nat i dvd A $ from-nat i $
from-nat i
⟨proof⟩

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 $ from-nat i $ from-nat i dvd A $ from-nat j $
from-nat j
  ⟨proof⟩

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::'a::{bezout-ring} ^'b::mod-type ^'c::mod-type
  assumes ib: is-bezout-ext bezout
  shows diagonal-to-Smith-i xs A i bezout $ from-nat i $ from-nat i dvd A $ from-nat i $ from-nat i
  ⟨proof⟩

```

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::'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 elements-xs-range: ∀ x. x ∈ set xs → x < min (nrows A) (ncols A)
  and to-nat a ∈ set xs
  and to-nat a = to-nat b
  and to-nat a ≠ 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
  ⟨proof⟩

```

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
⟨proof⟩

```

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 (nrows A) (ncols A)
shows diagonal-step A i j d v $ from-nat a $ from-nat b = A $ from-nat a $ 
from-nat b
⟨proof⟩

```

```

context GCD-ring
begin

```

lemma gcd-greatest:

```

assumes is-gcd gcd' and c dvd a and c dvd b
shows c dvd gcd' a b
⟨proof⟩

```

end

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

lemma diagonal-to-Smith-i-dvd:

```

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 elements-xs-range: ∀ x. x ∈ set xs → x < min (nrows A) (ncols A)
and ∀ a b. to-nat a ∈ insert i (set xs) ∧ to-nat a = to-nat b →
A $ (from-nat c) $ (from-nat c) dvd A $ a $ b
and c ∉ (set xs) and c < min (nrows A) (ncols A)
and distinct xs
shows A $ (from-nat c) $ (from-nat c) dvd
(diagonal-to-Smith-i xs A i bezout) $ (from-nat i) $ (from-nat i)
⟨proof⟩

```

lemma diagonal-to-Smith-i-dvd2:

```

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 elements-xs-range: ∀ x. x ∈ set xs → x < min (nrows A) (ncols A)
and dvd-condition: ∀ a b. to-nat a ∈ insert i (set xs) ∧ to-nat a = to-nat b →
A $ (from-nat c) $ (from-nat c) dvd A $ a $ b
and c-notin: c ∉ (set xs)
and c: c < min (nrows A) (ncols A)
and distinct: distinct xs
and ab: to-nat a = to-nat b

```

```

and a-in: to-nat a ∈ insert i (set xs)
shows A $ (from-nat c) $ (from-nat c) dvd (diagonal-to-Smith-i xs A i bezout) $
a $ b
⟨proof⟩

```

```

lemma diagonal-to-Smith-i-dvd2-k:
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 elements-xs-range: ∀ x. x ∈ set xs → x < k and k ≤ min (nrows A) (ncols A)
and dvd-condition: ∀ a b. to-nat a ∈ insert i (set xs) and to-nat a = to-nat b →
A $ (from-nat c) $ (from-nat c) dvd A $ a $ b
and c-notin: c ∉ (set xs)
and c: c < min (nrows A) (ncols A)
and distinct: distinct xs
and ab: to-nat a = to-nat b
and a-in: to-nat a ∈ insert i (set xs)
shows A $ (from-nat c) $ (from-nat c) dvd (diagonal-to-Smith-i xs A i bezout) $
a $ b
⟨proof⟩

```

```

lemma diagonal-to-Smith-row-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
shows Diagonal-to-Smith-row-i A i bezout $ a $ b = A $ a $ b
⟨proof⟩

```

```

lemma diagonal-to-Smith-row-i-preserves-previous-diagonal:
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-notin: to-nat a ∉ set [i + 1..<min (nrows A) (ncols A)]
and ab: to-nat a = to-nat b
and ai: to-nat a ≠ i
shows Diagonal-to-Smith-row-i A i bezout $ a $ b = A $ a $ b
⟨proof⟩

```

```

context
fixes bezout::'a::{bezout-ring} ⇒ 'a ⇒ 'a × 'a × 'a × 'a × 'a
assumes ib: is-bezout-ext bezout
begin

```

```

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

```

```

fixes A::'a::{bezout-ring} ^'b::mod-type ^'c::mod-type
assumes to-nat a ∈ {i..and to-nat a = to-nat b
shows (Diagonal-to-Smith-row-i A i bezout) $ (from-nat i) $ (from-nat i)
      dvd (Diagonal-to-Smith-row-i A i bezout) $ a $ b
⟨proof⟩

```

```

lemma diagonal-to-Smith-row-i-dvd-ii:
fixes A::'a::{bezout-ring} ^'b::mod-type ^'c::mod-type
shows Diagonal-to-Smith-row-i A i bezout $ from-nat i $ from-nat i dvd A $
from-nat i $ from-nat i
⟨proof⟩

```

```

lemma diagonal-to-Smith-row-i-dvd-jj':
fixes A::'a::{bezout-ring} ^'b::mod-type ^'c::mod-type
assumes a-in: to-nat a ∈ {i..and ab: to-nat a = to-nat b
and ci: c ≤ i
and dvd-condition: ∀ a b. to-nat a ∈ (set [i..shows (Diagonal-to-Smith-row-i A i bezout) $ (from-nat c) $ (from-nat c)
      dvd (Diagonal-to-Smith-row-i A i bezout) $ a $ b
⟨proof⟩
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
⟨proof⟩

```

```

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
⟨proof⟩

```

```

lemma isDiagonal-eq-upt-k-min:
isDiagonal A = isDiagonal-upt-k A (min (nrows A) (ncols A))
⟨proof⟩

```

```

lemma isDiagonal-eq-upt-k-max:
isDiagonal A = isDiagonal-upt-k A (max (nrows A) (ncols A))
⟨proof⟩

```

```

lemma isDiagonal:
  assumes isDiagonal A
    and to-nat a  $\neq$  to-nat b shows A $ a $ b = 0
   $\langle proof \rangle$ 

lemma nrows-diagonal-to-Smith-aux[simp]:
  shows nrows (diagonal-to-Smith-aux A xs bezout) = nrows A  $\langle proof \rangle$ 

lemma ncols-diagonal-to-Smith-aux[simp]:
  shows ncols (diagonal-to-Smith-aux A xs bezout) = ncols A  $\langle proof \rangle$ 

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)
   $\langle proof \rangle$ 
end

lemma to-nat-less-nrows[simp]:
  fixes A::'a $\wedge$ 'b::mod-type $\wedge$ 'c::mod-type
    and a::'c
  shows to-nat a < nrows A
   $\langle proof \rangle$ 

lemma to-nat-less-ncols[simp]:
  fixes A::'a $\wedge$ 'b::mod-type $\wedge$ 'c::mod-type
    and a::'b
  shows to-nat a < ncols A
   $\langle proof \rangle$ 

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::{bezout-ring} $\wedge$ 'b::mod-type $\wedge$ 'c::mod-type
  assumes ab: to-nat a = to-nat b
  and c: c < k and ca: c  $\leq$  to-nat a and k: k < min (nrows A) (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
   $\langle proof \rangle$ 

```

```

lemma Smith-normal-form-upk-Suc-imp-k:
  fixes A::'a::{bezout-ring} ^'b::mod-type ^'c::mod-type
  assumes s: Smith-normal-form-upk (diagonal-to-Smith-aux A [0..<Suc k] bezout) k
  and k: k < min (nrows A) (ncols A)
  shows Smith-normal-form-upk (diagonal-to-Smith-aux A [0..<k] bezout) k
  ⟨proof⟩

lemma Smith-normal-form-upk-le:
  assumes a ≤ k and Smith-normal-form-upk A k
  shows Smith-normal-form-upk A a ⟨proof⟩

lemma Smith-normal-form-upk-imp-Suc-k:
  assumes s: Smith-normal-form-upk (diagonal-to-Smith-aux A [0..<k] bezout) k
  and k: k < min (nrows A) (ncols A)
  shows Smith-normal-form-upk (diagonal-to-Smith-aux A [0..<Suc k] bezout) k
  ⟨proof⟩

corollary Smith-normal-form-upk-Suc-eq:
  assumes k: k < min (nrows A) (ncols A)
  shows Smith-normal-form-upk (diagonal-to-Smith-aux A [0..<Suc k] bezout) k
  = Smith-normal-form-upk (diagonal-to-Smith-aux A [0..<k] bezout) k
  ⟨proof⟩

end

lemma nrows-diagonal-to-Smith-i[simp]: nrows (diagonal-to-Smith-i xs A i bezout)
= nrows A
⟨proof⟩

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

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

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

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⟩

lemma isDiagonal-diagonal-to-Smith-i:
assumes isDiagonal A
and elements-xs-range: ∀ x. x ∈ set xs → x < min (nrows A) (ncols A)
and i < min (nrows A) (ncols A)
shows isDiagonal (diagonal-to-Smith-i xs A i bezout)
⟨proof⟩

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)
⟨proof⟩

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)
⟨proof⟩

context
fixes bezout:'a:{bezout-ring} ⇒ 'a ⇒ 'a × 'a × 'a × 'a × '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} ^'b::mod-type ^'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
⟨proof⟩

end

lemma nrows-diagonal-to-Smith[simp]: nrows (diagonal-to-Smith A bezout) = nrows
A
⟨proof⟩

lemma ncols-diagonal-to-Smith[simp]: ncols (diagonal-to-Smith A bezout) = ncols
A
⟨proof⟩

lemma isDiagonal-diagonal-to-Smith:
assumes d: isDiagonal A
shows isDiagonal (diagonal-to-Smith A bezout)

```

$\langle proof \rangle$

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⟩
```

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 ⇒ nat
⇒ nat ⇒ 'a bezout ⇒
(
('a::{bezout-ring} ^'rows::mod-type ^'rows::mod-type) ×
('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]]::int^3^3;
      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]]::int^3^3;
      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]]::int^3^3;
      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 $A::'a\sim cols::mod-type\sim rows::mod-type$
assumes $i: i < (nrows A)$ **and** $k: k < (nrows A)$ **and** $ik: i \neq k$
shows $from-nat i \neq (from-nat k::'rows)$
(proof)

lemma *from-nat-neq-cols*:

fixes $A::'a\sim cols::mod-type\sim rows::mod-type$
assumes $i: i < (ncols A)$ **and** $k: k < (ncols A)$ **and** $ik: i \neq k$
shows $from-nat i \neq (from-nat k::'cols)$
(proof)

lemma *diagonal-step-PQ-invertible-P*:

fixes $A::'a::\{bezout-ring\}\sim cols::mod-type\sim rows::mod-type$
assumes $PQ: (P, Q) = diagonal-step-PQ A i k bezout$
and $pquvd: (p, q, u, v, d) = bezout (A \$ from-nat i \$ from-nat i) (A \$ from-nat k \$ from-nat k)$
and $i-not-k: i \neq k$
and $i: i < min (nrows A) (ncols A)$ **and** $k: k < min (nrows A) (ncols A)$
shows *invertible P*
(proof)

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

fixes $A::'a::\{bezout-ring\}\sim cols::mod-type\sim rows::mod-type$
assumes $PQ: (P, Q) = diagonal-step-PQ A i k bezout$
and $pquvd: (p, q, u, v, d) = bezout (A \$ from-nat i \$ from-nat i) (A \$ from-nat k \$ from-nat k)$
and $i-not-k: i \neq k$
and $i: i < min (nrows A) (ncols A)$ **and** $k: k < min (nrows A) (ncols A)$
shows *invertible Q*
(proof)

lemma *mat-q-1[simp]*: $mat q \$ a \$ a = q$ *(proof)*

lemma *mat-q-0[simp]*:

assumes $ab: a \neq b$

shows $\text{mat } q \$ a \$ b = 0 \langle \text{proof} \rangle$

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 pquvd: (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⟩

```

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 pquvd: (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⟩

```

$P^{**}A$ 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  $P^{**}A = \text{row-add} (\text{interchange-rows}$ 
 $(\text{row-add } A (\text{from-nat } k) (\text{from-nat } i) p) (\text{from-nat } i) (\text{from-nat } k)) (\text{from-nat } k)$ 
 $(\text{from-nat } i) (- v)$ 

```

$\langle proof \rangle$

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

fixes $A: 'a::\{bezout-ring\} \rightsquigarrow cols::mod-type \rightsquigarrow rows::mod-type$
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 $P**A**Q = (mult-column (column-add (column-add (P**A) (from-nat i) (from-nat k) q) (from-nat k) (from-nat i) u) (from-nat k) (- 1))$

$\langle proof \rangle$

corollary *diagonal-step-PQ-PAQ*:

fixes $A: 'a::\{bezout-ring\} \rightsquigarrow cols::mod-type \rightsquigarrow rows::mod-type$
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 $P**A**Q = (mult-column (column-add (row-add (interchange-rows (row-add A (from-nat k) (from-nat i) p) (from-nat i) (from-nat k) (from-nat k) (from-nat i) (- v)) (from-nat i) (from-nat k) q) (from-nat k) (from-nat i) u) (from-nat k) (- 1))$

$\langle proof \rangle$

lemma *isDiagonal-imp-0*:

assumes *isDiagonal A*
and *from-nat a ≠ from-nat b*
and *a < min (nrows A) (ncols A)*
and *b < min (nrows A) (ncols A)*
shows *A \\$ from-nat a \\$ from-nat b = 0*
 $\langle proof \rangle$

lemma *diagonal-step-PQ*:

fixes $A: 'a::\{bezout-ring\} \rightsquigarrow cols::mod-type \rightsquigarrow rows::mod-type$
assumes $PQ: (P, Q) = diagonal-step-PQ A i k bezout$
and $b: (p, q, u, v, d) = bezout (A \$ from-nat i \$ from-nat i) (A \$ from-nat k \$ from-nat k)$
and $i: i < min (nrows A) (ncols A)$ **and** $k: k < min (nrows A) (ncols A)$ **and** $ik: i \neq k$
and $ib: is-bezout-ext bezout$ **and** $diag: isDiagonal A$
shows $diagonal-step A i k d v = P**A**Q$
 $\langle proof \rangle$

```

fun diagonal-to-Smith-i-PQ ::  

nat list  $\Rightarrow$  nat  $\Rightarrow$  ('a:{bezout-ring} bezout)  

 $\Rightarrow$  (('a^rows::mod-type ^rows::mod-type)  $\times$  ('a^cols::mod-type ^rows::mod-type)  $\times$   

('a^cols::mod-type ^cols::mod-type))  

 $\Rightarrow$  (('a^rows::mod-type ^rows::mod-type)  $\times$  ('a^cols::mod-type ^rows::mod-type)  

 $\times$  ('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))  

⟨proof⟩

```

```

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⟩

```

```

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:  $P \star\star A \star\star Q = B$ 
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:  $\forall x. x \in set xs \rightarrow x < min (nrows A) (ncols A)$ 
and diag: isDiagonal B and i-notin:  $i \notin set xs$  and i:  $i < min (nrows A) (ncols A)$ 
shows  $B' = diagonal-to-Smith-i xs B i$  bezout
    ⟨proof⟩

```

```

lemma 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:  $\forall x. x \in set xs \rightarrow x < min (nrows A) (ncols A)$ 
and diag: isDiagonal B and i-notin:  $i \notin set xs$  and i:  $i < min (nrows A) (ncols A)$ 
shows  $B' = P' \star\star A \star\star Q' \wedge invertible P' \wedge invertible Q'$ 
    ⟨proof⟩

```

```

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:  $\forall x. x \in set xs \rightarrow x < min (nrows A) (ncols A)$ 
and diag: isDiagonal B and i-notin:  $i \notin set xs$  and i:  $i < min (nrows A) (ncols A)$ 
shows  $B' = P' \star\star A \star\star Q' \wedge invertible P' \wedge invertible Q' \wedge B' = diagonal-to-Smith-i xs B i$  bezout
    ⟨proof⟩

```

```

lemma Diagonal-to-Smith-row-i-PQ-eq:
assumes P'B'Q': ( $P', B', Q'$ ) = Diagonal-to-Smith-row-i-PQ i bezout (P,B,Q)
and diag: isDiagonal B and i:  $i < min (nrows A) (ncols A)$ 
shows  $B' = Diagonal-to-Smith-row-i B i$  bezout
    ⟨proof⟩

```

```

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

```

```

assumes  $P'B'Q': (P',B',Q') = \text{Diagonal-to-Smith-row-}i\text{-}PQ$   $i \text{ bezout } (P,B,Q)$ 
and  $\text{diag: isDiagonal } B$  and  $i: i < \min(\text{nrows } A) (\text{ncols } A)$ 
shows  $B' = P'^{\star\star}A^{\star\star}Q' \wedge \text{invertible } P' \wedge \text{invertible } Q'$ 
⟨proof⟩

lemma  $\text{Diagonal-to-Smith-row-}i\text{-}PQ$ :
assumes  $P'B'Q': (P',B',Q') = \text{Diagonal-to-Smith-row-}i\text{-}PQ$   $i \text{ bezout } (P,B,Q)$ 
and  $\text{diag: isDiagonal } B$  and  $i: i < \min(\text{nrows } A) (\text{ncols } A)$ 
shows  $B' = P'^{\star\star}A^{\star\star}Q' \wedge \text{invertible } P' \wedge \text{invertible } Q' \wedge B' = \text{Diagonal-to-Smith-row-}i\text{-}B$   $i \text{ bezout}$ 
⟨proof⟩

end

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

lemma  $\text{diagonal-to-Smith-aux-PQ}$ :
assumes  $P'B'Q': (P',B',Q') = \text{diagonal-to-Smith-aux-PQ } [0..<k] \text{ bezout } (P,B,Q)$ 
and  $\text{diag: isDiagonal } B$  and  $k:k < \min(\text{nrows } A) (\text{ncols } A)$ 
shows  $B' = P'^{\star\star}A^{\star\star}Q' \wedge \text{invertible } P' \wedge \text{invertible } Q' \wedge B' = \text{diagonal-to-Smith-aux-PQ } [0..<k]$   $\text{bezout}$ 
⟨proof⟩

end

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

declare  $\text{diagonal-to-Smith-PQ.simps}[\text{simp del}]$ 
lemmas  $\text{diagonal-to-Smith-PQ-def} = \text{diagonal-to-Smith-PQ.simps}$ 

lemma  $\text{diagonal-to-Smith-PQ}$ :
fixes  $A::'a::\{\text{bezout-ring}\}^{\sim\text{cols}}::\{\text{mod-type}\}^{\sim\text{rows}}::\{\text{mod-type}\}$ 
assumes  $A: \text{isDiagonal } A$  and  $\text{ib: is-bezout-ext bezout}$ 
assumes  $PBQ: (P,B,Q) = \text{diagonal-to-Smith-PQ } A \text{ bezout}$ 
shows  $B = P^{\star\star}A^{\star\star}Q \wedge \text{invertible } P \wedge \text{invertible } Q \wedge B = \text{diagonal-to-Smith } A$ 
⟨proof⟩

```

```

lemma diagonal-to-Smith-PQ-exists:
  fixes A::'a::{bezout-ring} ^'cols::{mod-type} ^'rows::{mod-type}
  assumes A: isDiagonal A
  shows  $\exists P Q.$ 
    invertible (P::'a ^'rows::{mod-type} ^'rows::{mod-type})
     $\wedge$  invertible (Q::'a ^'cols::{mod-type} ^'cols::{mod-type})
    Smith-normal-form (P**A**Q)
  ⟨proof⟩

```

2.6 The final soundness theorem

```

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,S,Q) = diagonal-to-Smith-PQ A bezout
  shows S = P**A**Q  $\wedge$  invertible P  $\wedge$  invertible Q  $\wedge$  Smith-normal-form S
  ⟨proof⟩

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.to-nat*.

```

lemma inj-to-nat: inj to-nat ⟨proof⟩
lemmas from-nat-inj = from-nat-eq-imp-eq
lemma range-to-nat: range (to-nat :: 'a :: mod-type  $\Rightarrow$  nat) = {0 ..< CARD('a)}
  ⟨proof⟩

```

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  $\wedge$  '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  $\wedge$  'nc :: mod-type  $\wedge$  '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  $\wedge$  'n :: mod-type where
  to-hmav v = ( $\chi$  i. v $v to-nat i)

definition to-hmam :: 'a Matrix.mat  $\Rightarrow$  'a  $\wedge$  'nc :: mod-type  $\wedge$  'nr :: mod-type
where
  to-hmam a = ( $\chi$  i j. a $$ (to-nat i, to-nat j))

lemma to-hma-from-hmav[simp]: to-hmav (from-hmav v) = v
   $\langle$ proof $\rangle$ 

lemma to-hma-from-hmam[simp]: to-hmam (from-hmam v) = v
   $\langle$ proof $\rangle$ 

lemma from-hma-to-hmav[simp]:
  v  $\in$  carrier-vec (CARD('n))  $\implies$  from-hmav (to-hmav v :: 'a  $\wedge$  'n :: mod-type) =
  v
   $\langle$ proof $\rangle$ 

lemma from-hma-to-hmam[simp]:
  A  $\in$  carrier-mat (CARD('nr)) (CARD('nc))  $\implies$  from-hmam (to-hmam A :: 'a  $\wedge$ 
  'nc :: mod-type  $\wedge$  'nr :: mod-type) = A
   $\langle$ proof $\rangle$ 

lemma from-hmav-inj[simp]: from-hmav x = from-hmav y  $\longleftrightarrow$  x = y
   $\langle$ proof $\rangle$ 

lemma from-hmam-inj[simp]: from-hmam x = from-hmam y  $\longleftrightarrow$  x = y
   $\langle$ proof $\rangle$ 

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

definition HMA-M :: 'a Matrix.mat  $\Rightarrow$  'a  $\wedge$  'nc :: mod-type  $\wedge$  'nr :: mod-type  $\Rightarrow$ 
  bool where
  HMA-M = ( $\lambda$  a b. a = from-hmam 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  $\wedge$  'n :: mod-type  $\Rightarrow$  bool) = ( $\lambda$  v. v  $\in$ 
  carrier-vec (CARD('n)))
  ⟨proof⟩

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

lemma Domainp-HMA-I [transfer-domain-rule]:
  Domainp (HMA-I :: nat  $\Rightarrow$  'n :: mod-type  $\Rightarrow$  bool) = ( $\lambda$  i. i < CARD('n)) (is
  ?l = ?r)
  ⟨proof⟩

lemma bi-unique-HMA-V [transfer-rule]: bi-unique HMA-V left-unique HMA-V
right-unique HMA-V
  ⟨proof⟩

lemma bi-unique-HMA-M [transfer-rule]: bi-unique HMA-M left-unique HMA-M
right-unique HMA-M
  ⟨proof⟩

lemma bi-unique-HMA-I [transfer-rule]: bi-unique HMA-I left-unique HMA-I right-unique
HMA-I
  ⟨proof⟩

lemma right-total-HMA-V [transfer-rule]: right-total HMA-V
  ⟨proof⟩

lemma right-total-HMA-M [transfer-rule]: right-total HMA-M
  ⟨proof⟩

lemma right-total-HMA-I [transfer-rule]: right-total HMA-I
  ⟨proof⟩

lemma HMA-V-index [transfer-rule]: (HMA-V ==> HMA-I ==> (=)) ($v)
($h)
  ⟨proof⟩

```

lemma *HMA-M-index* [transfer-rule]:
 $(HMA\text{-}M \implies HMA\text{-}I \implies HMA\text{-}I \implies (=)) (\lambda A i j. A \$\$ (i,j))$
index-hma
 $\langle proof \rangle$

lemma *HMA-V-0* [transfer-rule]: $HMA\text{-}V (0_v CARD('n)) (0 :: 'a :: zero \wedge 'n :: mod\text{-}type)$
 $\langle proof \rangle$

lemma *HMA-M-0* [transfer-rule]:
 $HMA\text{-}M (0_m CARD('nr) CARD('nc)) (0 :: 'a :: zero \wedge 'nc :: mod\text{-}type \wedge 'nr :: mod\text{-}type)$
 $\langle proof \rangle$

lemma *HMA-M-1* [transfer-rule]:
 $HMA\text{-}M (1_m (CARD('n))) (mat 1 :: 'a :: \{zero,one\} \wedge 'n :: mod\text{-}type \wedge 'n :: mod\text{-}type)$
 $\langle proof \rangle$

lemma *from-hma_v-add*: $from\text{-}hma_v v + from\text{-}hma_v w = from\text{-}hma_v (v + w)$
 $\langle proof \rangle$

lemma *HMA-V-add* [transfer-rule]: $(HMA\text{-}V \implies HMA\text{-}V \implies HMA\text{-}V)$
 $(+) (+)$
 $\langle proof \rangle$

lemma *from-hma_v-diff*: $from\text{-}hma_v v - from\text{-}hma_v w = from\text{-}hma_v (v - w)$
 $\langle proof \rangle$

lemma *HMA-V-diff* [transfer-rule]: $(HMA\text{-}V \implies HMA\text{-}V \implies HMA\text{-}V)$
 $(-) (-)$
 $\langle proof \rangle$

lemma *from-hma_m-add*: $from\text{-}hma_m a + from\text{-}hma_m b = from\text{-}hma_m (a + b)$
 $\langle proof \rangle$

lemma *HMA-M-add* [transfer-rule]: $(HMA\text{-}M \implies HMA\text{-}M \implies HMA\text{-}M)$
 $(+) (+)$
 $\langle proof \rangle$

lemma *from-hma_m-diff*: $from\text{-}hma_m a - from\text{-}hma_m b = from\text{-}hma_m (a - b)$
 $\langle proof \rangle$

lemma *HMA-M-diff* [transfer-rule]: $(HMA\text{-}M \implies HMA\text{-}M \implies HMA\text{-}M)$
 $(-) (-)$
 $\langle proof \rangle$

lemma *scalar-product*: **fixes** $v :: 'a :: \text{semiring-1} \wedge 'n :: \text{mod-type}$
shows $\text{scalar-prod} (\text{from-hma}_v v) (\text{from-hma}_v w) = \text{scalar-product} v w$
 $\langle \text{proof} \rangle$

lemma [*simp*]:
 $\text{from-hma}_m (y :: 'a \wedge 'nc :: \text{mod-type} \wedge 'nr :: \text{mod-type}) \in \text{carrier-mat} (\text{CARD}('nr))$
 $(\text{CARD}('nc))$
 $\text{dim-row} (\text{from-hma}_m (y :: 'a \wedge 'nc :: \text{mod-type} \wedge 'nr :: \text{mod-type})) = \text{CARD}('nr)$
 $\text{dim-col} (\text{from-hma}_m (y :: 'a \wedge 'nc :: \text{mod-type} \wedge 'nr :: \text{mod-type})) = \text{CARD}('nc)$
 $\langle \text{proof} \rangle$

lemma [*simp*]:
 $\text{from-hma}_v (y :: 'a \wedge 'n :: \text{mod-type}) \in \text{carrier-vec} (\text{CARD}('n))$
 $\text{dim-vec} (\text{from-hma}_v (y :: 'a \wedge 'n :: \text{mod-type})) = \text{CARD}('n)$
 $\langle \text{proof} \rangle$

lemma *HMA-scalar-prod* [*transfer-rule*]:
 $(\text{HMA-}V \implies \text{HMA-}V \implies (=)) \text{ scalar-prod scalar-product}$
 $\langle \text{proof} \rangle$

lemma *HMA-row* [*transfer-rule*]: $(\text{HMA-}I \implies \text{HMA-}M \implies \text{HMA-}V) (\lambda i a. \text{Matrix.row} a i) \text{ row}$
 $\langle \text{proof} \rangle$

lemma *HMA-col* [*transfer-rule*]: $(\text{HMA-}I \implies \text{HMA-}M \implies \text{HMA-}V) (\lambda i a. \text{col} a i) \text{ column}$
 $\langle \text{proof} \rangle$

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

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

lemma *mat-mult-scalar*: $A ** B = \text{mk-mat} (\lambda i j. \text{scalar-product} (\text{row} i A) (\text{column} j B))$
 $\langle \text{proof} \rangle$

lemma *mult-mat-vec-scalar*: $A *v v = \text{mk-vec} (\lambda i. \text{scalar-product} (\text{row} i A) v)$
 $\langle \text{proof} \rangle$

lemma *dim-row-transfer-rule*:
 $HMA\text{-}M A (A' :: 'a \wedge 'nc:: mod\text{-}type \wedge 'nr:: mod\text{-}type) \implies (=) (dim\text{-}row A)$
 $(CARD('nr))$
 $\langle proof \rangle$

lemma *dim-col-transfer-rule*:
 $HMA\text{-}M A (A' :: 'a \wedge 'nc:: mod\text{-}type \wedge 'nr:: mod\text{-}type) \implies (=) (dim\text{-}col A)$
 $(CARD('nc))$
 $\langle proof \rangle$

lemma *HMA-M-mult [transfer-rule]*: $(HMA\text{-}M \implies HMA\text{-}M \implies HMA\text{-}M)$
 $(*) (**)$
 $\langle proof \rangle$

lemma *HMA-V-smult [transfer-rule]*: $((=) \implies HMA\text{-}V \implies HMA\text{-}V) (\cdot_v)$
 $(*_s)$
 $\langle proof \rangle$

lemma *HMA-M-mult-vec [transfer-rule]*: $(HMA\text{-}M \implies HMA\text{-}V \implies HMA\text{-}V)$
 $(*_v) (*v)$
 $\langle proof \rangle$

lemma *HMA-det [transfer-rule]*: $(HMA\text{-}M \implies (=)) Determinant.det$
 $(det :: 'a :: comm\text{-}ring-1 \wedge 'n :: mod\text{-}type \wedge 'n :: mod\text{-}type \Rightarrow 'a)$
 $\langle proof \rangle$

lemma *HMA-mat[transfer-rule]*: $((=) \implies HMA\text{-}M) (\lambda k. k \cdot_m 1_m CARD('n))$
 $(Finite\text{-}Cartesian\text{-}Product.mat :: 'a::semiring-1 \Rightarrow 'a \wedge 'n :: mod\text{-}type \wedge 'n :: mod\text{-}type)$
 $\langle proof \rangle$

lemma *HMA-mat-minus[transfer-rule]*: $(HMA\text{-}M \implies HMA\text{-}M \implies HMA\text{-}M)$
 $(\lambda A B. A + map\text{-}mat uminus B) ((-) :: 'a :: group\text{-}add \wedge 'nc:: mod\text{-}type \wedge 'nr:: mod\text{-}type$
 $\Rightarrow 'a \wedge 'nc:: mod\text{-}type \wedge 'nr:: mod\text{-}type \Rightarrow 'a \wedge 'nc:: mod\text{-}type \wedge 'nr:: mod\text{-}type)$
 $\langle proof \rangle$

lemma *HMA-transpose-matrix [transfer-rule]*:
 $(HMA\text{-}M \implies HMA\text{-}M) transpose\text{-}mat transpose$
 $\langle proof \rangle$

lemma *HMA-invertible-matrix-mod-type[transfer-rule]*:
 $((Mod\text{-}Type\text{-}Connect.HMA\text{-}M :: - \Rightarrow 'a :: comm\text{-}ring-1 \wedge 'n :: mod\text{-}type \wedge 'n ::$

mod-type
 $\Rightarrow \neg) \implies (=)$) invertible-mat invertible
 $\langle \text{proof} \rangle$

end

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

context

includes lifting-syntax

begin

lemma HMA-swaprows[transfer-rule]:

$((\text{Mod-Type-Connect.HMA-M} :: - \Rightarrow 'a :: \text{comm-ring-1} \wedge 'nc :: \text{mod-type} \wedge 'nr :: \text{mod-type} \Rightarrow \neg) \implies (\text{Mod-Type-Connect.HMA-I} :: - \Rightarrow 'nr :: \text{mod-type} \Rightarrow \neg))$
 $\implies (\text{Mod-Type-Connect.HMA-I} :: - \Rightarrow 'nr :: \text{mod-type} \Rightarrow \neg)$
 $\implies (\text{Mod-Type-Connect.HMA-M})$
 $(\lambda A a b. \text{swaprows } a b A) \text{ interchange-rows}$
 $\langle \text{proof} \rangle$

lemma HMA-swapcols[transfer-rule]:

$((\text{Mod-Type-Connect.HMA-M} :: - \Rightarrow 'a :: \text{comm-ring-1} \wedge 'nc :: \text{mod-type} \wedge 'nr :: \text{mod-type} \Rightarrow \neg) \implies (\text{Mod-Type-Connect.HMA-I} :: - \Rightarrow 'nc :: \text{mod-type} \Rightarrow \neg))$
 $\implies (\text{Mod-Type-Connect.HMA-I} :: - \Rightarrow 'nc :: \text{mod-type} \Rightarrow \neg)$
 $\implies (\text{Mod-Type-Connect.HMA-M})$
 $(\lambda A a b. \text{swapcols } a b A) \text{ interchange-columns}$
 $\langle \text{proof} \rangle$

lemma HMA-addrow[transfer-rule]:

$((\text{Mod-Type-Connect.HMA-M} :: - \Rightarrow 'a :: \text{comm-ring-1} \wedge 'nc :: \text{mod-type} \wedge 'nr :: \text{mod-type} \Rightarrow \neg) \implies (\text{Mod-Type-Connect.HMA-I} :: - \Rightarrow 'nr :: \text{mod-type} \Rightarrow \neg))$
 $\implies (\text{Mod-Type-Connect.HMA-I} :: - \Rightarrow 'nr :: \text{mod-type} \Rightarrow \neg)$
 $\implies (=)$
 $\implies (\text{Mod-Type-Connect.HMA-M})$
 $(\lambda A a b q. \text{addrow } q a b A) \text{ row-add}$
 $\langle \text{proof} \rangle$

lemma HMA-addcol[transfer-rule]:

$((\text{Mod-Type-Connect.HMA-M} :: - \Rightarrow 'a :: \text{comm-ring-1} \wedge 'nc :: \text{mod-type} \wedge 'nr :: \text{mod-type} \Rightarrow \neg) \implies (\text{Mod-Type-Connect.HMA-I} :: - \Rightarrow 'nc :: \text{mod-type} \Rightarrow \neg))$
 $\implies (\text{Mod-Type-Connect.HMA-I} :: - \Rightarrow 'nc :: \text{mod-type} \Rightarrow \neg)$
 $\implies (=)$
 $\implies (\text{Mod-Type-Connect.HMA-M})$
 $(\lambda A a b q. \text{addcol } q a b A) \text{ column-add}$
 $\langle \text{proof} \rangle$

```

lemma HMA-multrow[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-I :: -  $\Rightarrow$  'nr :: mod-type  $\Rightarrow$  - )
   ==> (=)
   ==> Mod-Type-Connect.HMA-M)
  ( $\lambda A i q.$  multrow  $i q A)$  mult-row
  ⟨proof⟩

lemma HMA-multcol[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-I :: -  $\Rightarrow$  'nc :: mod-type  $\Rightarrow$  - )
   ==> (=)
   ==> Mod-Type-Connect.HMA-M)
  ( $\lambda A i q.$  multcol  $i q A)$  mult-column
  ⟨proof⟩

end

fun HMA-M3 where
  HMA-M3 (P,A,Q)
  (P' :: 'a :: comm-ring-1  $\wedge$  'nr :: mod-type  $\wedge$  'nr :: mod-type,
   A' :: 'a  $\wedge$  'nc :: mod-type  $\wedge$  'nr :: mod-type,
   Q' :: 'a  $\wedge$  'nc :: mod-type  $\wedge$  'nc :: mod-type) =
  (Mod-Type-Connect.HMA-M P P'  $\wedge$  Mod-Type-Connect.HMA-M A A'  $\wedge$  Mod-Type-Connect.HMA-M Q Q')

lemma HMA-M3-def:
  HMA-M3 A B = (Mod-Type-Connect.HMA-M (fst A) (fst B)
   $\wedge$  Mod-Type-Connect.HMA-M (fst (snd A)) (fst (snd B))
   $\wedge$  Mod-Type-Connect.HMA-M (snd (snd A)) (snd (snd B)))
  ⟨proof⟩

context
  includes lifting-syntax
begin

lemma Domainp-HMA-M3 [transfer-domain-rule]:
  Domainp (HMA-M3 :: -  $\Rightarrow$  (-  $\times$  ('a::comm-ring-1  $\wedge$  'nc::mod-type  $\wedge$  'nr::mod-type)  $\times$  -)  $\Rightarrow$  -)
  = ( $\lambda (P,A,Q).$  P  $\in$  carrier-mat CARD('nr) CARD('nr)  $\wedge$  A  $\in$  carrier-mat CARD('nr)
  CARD('nc)
   $\wedge$  Q  $\in$  carrier-mat CARD('nc) CARD('nc))
  ⟨proof⟩

lemma bi-unique-HMA-M3 [transfer-rule]: bi-unique HMA-M3 left-unique HMA-M3

```

```

right-unique HMA-M3
 $\langle proof \rangle$ 

lemma right-total-HMA-M3 [transfer-rule]: right-total HMA-M3
 $\langle proof \rangle$ 

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:  $(\sum i = 0..<2. f i) = (\sum i \in \{0,1::nat\}. f i)$ 
 $\langle proof \rangle$ 

```

```

lemma sum-common-left:
  fixes f::'a  $\Rightarrow$  'b::comm-ring-1
  assumes finite A
  shows sum  $(\lambda i. c * f i) A = c * \text{sum } f A$ 
 $\langle proof \rangle$ 

```

```

lemma prod3-intro:
  assumes fst A = a and snd (fst A) = b and snd (snd A) = c
  shows A = (a,b,c)  $\langle proof \rangle$ 

```

4.2 Transfer rules for the HMA_Connect file of the Perron-Frobenius development

```

hide-const (open) HMA-M HMA-I to-hmam from-hmam
hide-fact (open) from-hmam-def from-hma-to-hmam 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]:
   $((HMA\text{-}Connect.HMA\text{-}M :: - \Rightarrow 'a :: comm\text{-}ring\text{-}1 \wedge 'n \wedge 'n \Rightarrow -) \implies (=))$ 
invertible-mat invertible
 $\langle proof \rangle$ 
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 ∈ carrier-mat 0 0
  shows invertible-mat A
 $\langle proof \rangle$ 

lemma invertible-mult-JNF:
  fixes A:'a::comm-ring-1 mat
  assumes A: A ∈ carrier-mat n n and B: B ∈ carrier-mat n n
  and inv-A: invertible-mat A and inv-B: invertible-mat B
  shows invertible-mat (A*B)
 $\langle proof \rangle$ 

lemma invertible-iff-is-unit-JNF:
  assumes A: A ∈ carrier-mat n n
  shows invertible-mat A \leftrightarrow (Determinant.det A) dvd 1
 $\langle proof \rangle$ 

```

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 ∈ carrier-mat n n
  and B: B ∈ carrier-mat n n and AB: A * B = 1m n
  shows B * A = 1m n

```

$\langle proof \rangle$

context comm-ring-1
begin

lemma col-submatrix-UNIV:
assumes $j < card \{i. i < dim\text{-}col A \wedge i \in J\}$
shows $col(\text{submatrix } A \text{ UNIV } J) j = col A (\text{pick } J j)$
 $\langle proof \rangle$

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

lemma submatrix-mult:

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

lemma det-singleton:

assumes $A: A \in \text{carrier-mat } 1 1$
shows $\det A = A \text{ \$\$ } (0,0)$
 $\langle proof \rangle$

lemma submatrix-singleton-index:

assumes $A: A \in \text{carrier-mat } n m$
and $an: a < n$ **and** $bm: b < m$
shows $\text{submatrix } A \{a\} \{b\} \text{ \$\$ } (0,0) = A \text{ \$\$ } (a,b)$
 $\langle proof \rangle$

end

lemma det-not-inj-on:

assumes $\text{not-inj-on}: \neg \text{inj-on } f \{0..<n\}$
shows $\det(\text{mat}_r n n (\lambda i. \text{Matrix.row } B (f i))) = 0$
 $\langle proof \rangle$

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

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** $B \in \text{carrier-mat } n n$
 $\langle proof \rangle$

```

lemma invertible-mat-smult-mat:
  fixes A :: 'a::comm-ring-1 mat
  assumes inv-A: invertible-mat A and k: k dvd 1
  shows invertible-mat (k ·m A)
  ⟨proof⟩

lemma invertible-mat-one[simp]: invertible-mat (1m n)
  ⟨proof⟩

lemma four-block-mat-dim0:
  assumes A: A ∈ carrier-mat n n
  and B: B ∈ carrier-mat n 0
  and C: C ∈ carrier-mat 0 n
  and D: D ∈ carrier-mat 0 0
  shows four-block-mat A B C D = A
  ⟨proof⟩

lemma det-four-block-mat-lower-right-id:
  assumes A: A ∈ carrier-mat m m
  and B: B = 0m m (n-m)
  and C: C = 0m (n-m) m
  and D: D = 1m (n-m)
  and n>m
  shows Determinant.det (four-block-mat A B C D) = Determinant.det A
  ⟨proof⟩

lemma mult-eq-first-row:
  assumes A: A ∈ carrier-mat 1 n
  and B: B ∈ carrier-mat m n
  and m0: m ≠ 0
  and r: Matrix.row A 0 = Matrix.row B 0
  shows Matrix.row (A * V) 0 = Matrix.row (B * V) 0
  ⟨proof⟩

lemma smult-mat-mat-one-element:
  assumes A: A ∈ carrier-mat 1 1 and B: B ∈ carrier-mat 1 n
  shows A * B = A $$ (0,0) ·m B
  ⟨proof⟩

lemma determinant-one-element:
  assumes A: A ∈ carrier-mat 1 1 shows Determinant.det A = A $$ (0,0)
  ⟨proof⟩

```

```

lemma invertible-mat-transpose:
  assumes inv-A: invertible-mat ( $A::'a::comm-ring-1 mat$ )
  shows invertible-mat  $A^T$ 
   $\langle proof \rangle$ 

lemma dvd-elements-mult-matrix-left:
  assumes A: ( $A::'a::comm-ring-1 mat$ )  $\in carrier\text{-}mat m n$ 
  and P:  $P \in carrier\text{-}mat m m$ 
  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))$ 
   $\langle proof \rangle$ 

lemma dvd-elements-mult-matrix-right:
  assumes A: ( $A::'a::comm-ring-1 mat$ )  $\in carrier\text{-}mat m n$ 
  and Q:  $Q \in carrier\text{-}mat n n$ 
  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))$ 
   $\langle proof \rangle$ 

lemma dvd-elements-mult-matrix-left-right:
  assumes A: ( $A::'a::comm-ring-1 mat$ )  $\in carrier\text{-}mat m n$ 
  and P:  $P \in carrier\text{-}mat m m$ 
  and Q:  $Q \in carrier\text{-}mat n n$ 
  and x:  $(\forall i j. i < m \wedge j < n \longrightarrow x \text{ dvd } A\$$(i,j))$ 
  shows  $(\forall i j. i < m \wedge j < n \longrightarrow x \text{ dvd } (P*A*Q)\$$(i,j))$ 
   $\langle proof \rangle$ 

definition append-cols :: ' $a :: zero mat \Rightarrow 'a mat \Rightarrow 'a mat$  (infixr  $\langle @_c \rangle$  65) where
   $A @_c B = four\text{-}block\text{-}mat A B (0_m 0 (dim\text{-}col A)) (0_m 0 (dim\text{-}col B))$ 

lemma append-cols-carrier[simp,intro]:
   $A \in carrier\text{-}mat n a \implies B \in carrier\text{-}mat n b \implies (A @_c B) \in carrier\text{-}mat n (a+b)$ 
   $\langle proof \rangle$ 

lemma append-cols-mult-left:
  assumes A:  $A \in carrier\text{-}mat n a$ 
  and B:  $B \in carrier\text{-}mat n b$ 
  and P:  $P \in carrier\text{-}mat n n$ 
  shows  $P * (A @_c B) = (P*A) @_c (P*B)$ 
   $\langle proof \rangle$ 

lemma append-cols-mult-right-id:
  assumes A: ( $A::'a::semiring-1 mat$ )  $\in carrier\text{-}mat n 1$ 
  and B:  $B \in carrier\text{-}mat n (m-1)$ 
  and C:  $C = four\text{-}block\text{-}mat (1_m 1) (0_m 1 (m-1)) (0_m (m-1) 1) D$ 

```

and $D: D \in \text{carrier-mat } (m-1) \ (m-1)$
shows $(A @_c B) * C = A @_c (B * D)$
 $\langle proof \rangle$

lemma *append-cols-mult-right-id2*:
assumes $A: (A::'a::semiring-1 mat) \in \text{carrier-mat } n \ a$
and $B: B \in \text{carrier-mat } n \ b$
and $C: C = \text{four-block-mat } D \ (0_m \ a \ b) \ (0_m \ b \ a) \ (1_m \ b)$
and $D: D \in \text{carrier-mat } a \ a$
shows $(A @_c B) * C = (A * D) @_c B$
 $\langle proof \rangle$

lemma *append-cols-nth*:
assumes $A: A \in \text{carrier-mat } n \ a$
and $B: B \in \text{carrier-mat } n \ b$
and $i: i < n \ \text{and} \ j: j < a + b$
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** $?lhs = ?rhs$)
 $\langle proof \rangle$

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** $?lhs = ?A1 @_c ?A2$)
 $\langle proof \rangle$

lemma *append-rows-nth*:
assumes $A: A \in \text{carrier-mat } a \ n$
and $B: B \in \text{carrier-mat } b \ n$
and $i: i < a+b \ \text{and} \ j: j < n$
shows $(A @_r B) \$\$ (i, j) = (\text{if } i < \text{dim-row } A \text{ then } A \$\$ (i, j) \text{ else } B \$\$ (i - a, j))$ (**is** $?lhs = ?rhs$)
 $\langle proof \rangle$

lemma *append-rows-split*:
assumes $k: k \leq \text{dim-row } A$
shows $A = (\text{mat-of-rows } (\text{dim-col } A) [\text{Matrix.row } A \ i. \ i \leftarrow [0..<k]]) @_r (\text{mat-of-rows } (\text{dim-col } A) [\text{Matrix.row } A \ i. \ i \leftarrow [k..<\text{dim-row } A]])$ (**is** $?lhs = ?A1 @_r ?A2$)
 $\langle proof \rangle$

lemma *transpose-mat-append-rows*:
assumes $A: A \in \text{carrier-mat } a \ n \ \text{and} \ B: B \in \text{carrier-mat } b \ n$

```

shows  $(A @_r B)^T = A^T @_c B^T$ 
⟨proof⟩

lemma transpose-mat-append-cols:
assumes  $A: A \in \text{carrier-mat } n \text{ } a$  and  $B: B \in \text{carrier-mat } n \text{ } b$ 
shows  $(A @_c B)^T = A^T @_r B^T$ 
⟨proof⟩

lemma append-rows-mult-right:
assumes  $A: (A::'a::comm-semiring-1 mat) \in \text{carrier-mat } a \text{ } n$  and  $B: B \in \text{carrier-mat } b \text{ } n$ 
and  $Q: Q \in \text{carrier-mat } n \text{ } n$ 
shows  $(A @_r B) * Q = (A * Q) @_r (B * Q)$ 
⟨proof⟩

lemma append-rows-mult-left-id:
assumes  $A: (A::'a::comm-semiring-1 mat) \in \text{carrier-mat } 1 \text{ } n$ 
and  $B: B \in \text{carrier-mat } (m-1) \text{ } n$ 
and  $C: C = \text{four-block-mat } (1_m \ 1) (0_m \ 1 \ (m-1)) (0_m \ (m-1) \ 1) \ D$ 
and  $D: D \in \text{carrier-mat } (m-1) \ (m-1)$ 
shows  $C * (A @_r B) = A @_r (D * B)$ 
⟨proof⟩

lemma append-rows-mult-left-id2:
assumes  $A: (A::'a::comm-semiring-1 mat) \in \text{carrier-mat } a \text{ } n$ 
and  $B: B \in \text{carrier-mat } b \text{ } n$ 
and  $C: C = \text{four-block-mat } D (0_m \ a \ b) (0_m \ b \ a) (1_m \ b)$ 
and  $D: D \in \text{carrier-mat } a \text{ } a$ 
shows  $C * (A @_r B) = (D * A) @_r B$ 
⟨proof⟩

lemma four-block-mat-preserves-column:
assumes  $A: (A::'a::semiring-1 mat) \in \text{carrier-mat } n \text{ } m$ 
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⟩

```

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  $A \$\$ (i, j) = 0$ 
⟨proof⟩

```

lemma commute-multiples-identity:

assumes $A: (A::'a::comm-ring-1 mat) \in carrier\text{-}mat n n$
shows $A * (k \cdot_m (1_m n)) = (k \cdot_m (1_m n)) * A$
 $\langle proof \rangle$

lemma *det-2*:

assumes $A: A \in carrier\text{-}mat 2 2$
shows $Determinant.det A = A\$$(0,0) * A \$\$ (1,1) - A\$$(0,1)*A\$$(1,0)$
 $\langle proof \rangle$

lemma *mat-diag-smult*: $mat\text{-}diag n (\lambda x. (k::'a::comm-ring-1)) = (k \cdot_m 1_m n)$
 $\langle proof \rangle$

lemma *invertible-mat-four-block-mat-lower-right*:

assumes $A: (A::'a::comm-ring-1 mat) \in carrier\text{-}mat n n$ **and** $inv\text{-}A: invertible\text{-}mat A$
shows $invertible\text{-}mat (four\text{-}block\text{-}mat (1_m 1) (0_m 1 n) (0_m n 1) A)$
 $\langle proof \rangle$

lemma *invertible-mat-four-block-mat-lower-right-id*:

assumes $A: (A::'a::comm-ring-1 mat) \in carrier\text{-}mat m m$ **and** $B: B = 0_m m$
 $(n-m)$ **and** $C: C = 0_m (n-m) m$
and $D: D = 1_m (n-m)$ **and** $n > m$ **and** $inv\text{-}A: invertible\text{-}mat A$
shows $invertible\text{-}mat (four\text{-}block\text{-}mat A B C D)$
 $\langle proof \rangle$

lemma *split-block4-decreases-dim-row*:

assumes $E: (A,B,C,D) = split\text{-}block E 1 1$
and $E1: dim\text{-}row E > 1$ **and** $E2: dim\text{-}col E > 1$
shows $dim\text{-}row D < dim\text{-}row E$
 $\langle proof \rangle$

lemma *inv-P'PAQQ'*:

assumes $A: A \in carrier\text{-}mat n n$
and $P: P \in carrier\text{-}mat n n$
and $inv\text{-}P: invertible\text{-}mat P' P$
and $inv\text{-}Q: invertible\text{-}mat Q' Q$
and $Q: Q \in carrier\text{-}mat n n$
and $P': P' \in carrier\text{-}mat n n$
and $Q': Q' \in carrier\text{-}mat n n$
shows $(P' * (P * A * Q) * Q') = A$
 $\langle proof \rangle$

lemma

assumes $U \in carrier\text{-}mat 2 2$ **and** $V \in carrier\text{-}mat 2 2$ **and** $A = U * V$
shows $mat\text{-}mult2-00: A \$\$ (0,0) = U \$\$ (0,0) * V \$\$ (0,0) + U \$\$ (0,1) * V \$\$ (1,0)$

```

and mat-mult2-01:  $A \lll (0,1) = U \lll (0,0)*V \lll (0,1) + U \lll (0,1)*V \lll (1,1)$ 
and mat-mult2-10:  $A \lll (1,0) = U \lll (1,0)*V \lll (0,0) + U \lll (1,1)*V \lll (1,0)$ 
and mat-mult2-11:  $A \lll (1,1) = U \lll (1,0)*V \lll (0,1) + U \lll (1,1)*V \lll (1,1)$ 
⟨proof⟩

```

4.5 Lemmas about sorted lists, insort and pick

```

lemma sorted-distinct-imp-sorted-wrt:
assumes sorted xs and distinct xs
shows sorted-wrt (<) xs
⟨proof⟩

```

```

lemma sorted-map-strict:
assumes strict-mono-on {0..<n} g
shows sorted (map g [0..<n])
⟨proof⟩

```

```

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]
⟨proof⟩

```

```

lemma sorted-nth-strict-mono:
sorted xs  $\implies$  distinct xs  $\implies$  i < j  $\implies$  j < length xs  $\implies$  xs!i < xs!j
⟨proof⟩

```

```

lemma sorted-list-of-set-0-LEAST:
assumes finI: finite I and I: I  $\neq \{\}$ 
shows sorted-list-of-set I ! 0 = (LEAST n. n  $\in$  I)
⟨proof⟩

```

```

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⟩

```

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
⟨proof⟩

```

```

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
  ⟨proof⟩

```

```

lemma insort-nth2:
  assumes sorted xs and a ∉ set xs
    and i < length xs and i ≥ index (insort a xs) a
    and xs ≠ []
  shows insort a xs ! (Suc i) = xs ! i
  ⟨proof⟩

```

```

lemma pick-index:
  assumes a: a ∈ I and a'-card: a' < card I
  shows (pick I a' = a) = (index (sorted-list-of-set I) a = a')
  ⟨proof⟩

```

```
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-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⟩

```

```

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 $\text{pick}(\text{insert } a \ I) \ i < \text{pick } I \ i$
 $\langle \text{proof} \rangle$

lemma $\text{pick-insert3}:$
assumes $a\text{-notin-}I: a \notin I$ **and** $i2: i < \text{card } I$
and $a\text{-def:}$ $\text{pick}(\text{insert } a \ I) \ a' = a$
and $ia': i \geq a'$
and $a'\text{-card:}$ $a' < \text{card } I + 1$
shows $\text{pick}(\text{insert } a \ I) (\text{Suc } i) = \text{pick } I \ i$
 $\langle \text{proof} \rangle$

lemma $\text{pick-insert-index}:$
assumes $Ik: \text{card } I = k$
and $a\text{-notin-}I: a \notin I$
and $ik: i < k$
and $a\text{-def:}$ $\text{pick}(\text{insert } a \ I) \ a' = a$
and $a'k: a' < \text{card } I + 1$
shows $\text{pick}(\text{insert } a \ I) (\text{insert-index } a' \ i) = \text{pick } I \ i$
 $\langle \text{proof} \rangle$

5.2 Start of the proof

definition $\text{strict-from-inj } n \ f = (\lambda i. \text{if } i \in \{0..<n\} \text{ then } (\text{sorted-list-of-set } (f \setminus \{0..<n\})) \\ ! \ i \text{ else } i)$

lemma $\text{strict-strict-from-inj}:$
fixes $f::nat \Rightarrow nat$
assumes $\text{inj-on } f \ \{0..<n\}$ **shows** $\text{strict-mono-on } \{0..<n\} (\text{strict-from-inj } n \ f)$
 $\langle \text{proof} \rangle$

lemma $\text{strict-from-inj-image}':$
assumes $f: \text{inj-on } f \ \{0..<n\}$
shows $\text{strict-from-inj } n \ f \setminus \{0..<n\} = f \setminus \{0..<n\}$
 $\langle \text{proof} \rangle$

definition $Z \ (n::nat) \ (m::nat) = \{(f, \pi) | f \ \pi. \ f \in \{0..<n\} \rightarrow \{0..<m\} \\ \wedge (\forall i. i \notin \{0..<n\} \longrightarrow f i = i) \\ \wedge \pi \text{ permutes } \{0..<n\}\}$

lemma $Z\text{-alt-def: } Z \ n \ m = \{f. f \in \{0..<n\} \rightarrow \{0..<m\} \wedge (\forall i. i \notin \{0..<n\} \longrightarrow \\ f i = i)\} \times \{\pi. \pi \text{ permutes } \{0..<n\}\}$
 $\langle \text{proof} \rangle$

```

lemma det-mul-finsum-alt:
  assumes A:  $A \in carrier\text{-}mat\ n\ m$ 
  and B:  $B \in carrier\text{-}mat\ m\ n$ 
  shows  $\det(A * B) = \det(\text{matr}\ n\ n\ (\lambda i. \text{finsum}\text{-}vec\ TYPE('a::comm\text{-}ring\text{-}1)\ n\ (\lambda k. B \$\$ (k, i) \cdot_v \text{Matrix.col}\ A\ k)\ \{0..<m\}))$ 
  ⟨proof⟩

lemma det-cols-mul:
  assumes A:  $A \in carrier\text{-}mat\ n\ m$ 
  and B:  $B \in carrier\text{-}mat\ m\ n$ 
  shows  $\det(A * B) = (\sum f \mid (\forall i \in \{0..<n\}. f\ i \in \{0..<m\}) \wedge (\forall i. i \notin \{0..<n\} \rightarrow f\ i = i)) \cdot (\prod i = 0..<n. B \$\$ (f\ i, i)) * \text{Determinant.det}(\text{matr}\ n\ n\ (\lambda i. \text{col}\ A\ (f\ i)))$ 
  ⟨proof⟩

lemma det-cols-mul':
  assumes A:  $A \in carrier\text{-}mat\ n\ m$ 
  and B:  $B \in carrier\text{-}mat\ m\ n$ 
  shows  $\det(A * B) = (\sum f \mid (\forall i \in \{0..<n\}. f\ i \in \{0..<m\}) \wedge (\forall i. i \notin \{0..<n\} \rightarrow f\ i = i)) \cdot (\prod i = 0..<n. A \$\$ (i, f\ i)) * \det(\text{matr}\ n\ n\ (\lambda i. \text{row}\ B\ (f\ i)))$ 
  ⟨proof⟩

lemma
  assumes F:  $F = \{f. f \in \{0..<n\} \rightarrow \{0..<m\} \wedge (\forall i. i \notin \{0..<n\} \rightarrow f\ i = i)\}$ 
  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⟩

lemma detAB-Znm-aux:
  assumes F:  $F = \{f. f \in \{0..<n\} \rightarrow \{0..<m\} \wedge (\forall i. i \notin \{0..<n\} \rightarrow f\ i = i)\}$ 
  shows  $(\sum \pi \mid \pi \text{ permutes } \{0..<n\}. (\sum f \in F. \text{prod}(\lambda i. B \$\$ (f\ i, i)) \{0..<n\} * (\text{signof}\ \pi * (\prod i = 0..<n. A \$\$ (\pi\ i, f\ i)))))$ 
   $= (\sum \pi \mid \pi \text{ permutes } \{0..<n\}. \sum f \in F. (\prod i = 0..<n. B \$\$ (f\ i, \pi\ i)) * (\text{signof}\ \pi * (\prod i = 0..<n. A \$\$ (i, f\ i))))$ 
  ⟨proof⟩

lemma detAB-Znm:
  assumes A:  $A \in carrier\text{-}mat\ n\ m$ 
  and B:  $B \in carrier\text{-}mat\ m\ n$ 
  shows  $\det(A * B) = (\sum (f, \pi) \in Z\ n\ m. \text{signof}\ \pi * (\prod i = 0..<n. A \$\$ (i, f\ i)) * B \$\$ (f\ i, \pi\ i))$ 
  ⟨proof⟩

```

```

context
  fixes n m and A B::'a::comm-ring-1 mat
  assumes A: A ∈ carrier-mat n m
    and B: B ∈ carrier-mat m n
begin

private definition Z-inj = ({f. f ∈ {0..} → {0..} ∧ (∀ i. i ∉ {0..} → f i = i)} ∧ inj-on f {0..} × {π. π permutes {0..}})

private definition Z-not-inj = ({f. f ∈ {0..} → {0..} ∧ (∀ i. i ∉ {0..} → f i = i)} ∧ ¬ inj-on f {0..} × {π. π permutes {0..}})

private definition Z-strict = ({f. f ∈ {0..} → {0..} ∧ (∀ i. i ∉ {0..} → f i = i)} ∧ strict-mono-on {0..} f) × {π. π permutes {0..}})

private definition Z-not-strict = ({f. f ∈ {0..} → {0..} ∧ (∀ i. i ∉ {0..} → f i = i)} ∧ ¬ strict-mono-on {0..} f) × {π. π permutes {0..}})

private definition weight f π
  = (signof π) * (prod (λi. A$$ (i, f i) * B $$ (f i, π i)) {0..})

private definition Z-good g = ({f. f ∈ {0..} → {0..} ∧ (∀ i. i ∉ {0..} → f i = i)} ∧ inj-on f {0..} ∧ (f {0..} = g {0..})} × {π. π permutes {0..}})

private definition F-strict = {f. f ∈ {0..} → {0..} ∧ (∀ i. i ∉ {0..} → f i = i) ∧ strict-mono-on {0..} f}

private definition F-inj = {f. f ∈ {0..} → {0..} ∧ (∀ i. i ∉ {0..} → f i = i) ∧ inj-on f {0..}}

private definition F-not-inj = {f. f ∈ {0..} → {0..} ∧ (∀ i. i ∉ {0..} → f i = i) ∧ ¬ inj-on f {0..}}

private definition F = {f. f ∈ {0..} → {0..} ∧ (∀ i. i ∉ {0..} → 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 step-weight:

```

fixes  $\varphi \pi$ 
defines  $\sigma \equiv \pi \circ Hilbert\text{-}Choice.inv \varphi$ 
assumes  $f\text{-inj}: f \in F\text{-inj}$  and  $gF: g \in F$  and  $pi: \pi \text{ permutes } \{0..<n\}$ 
and  $\text{phi}: \varphi \text{ permutes } \{0..<n\}$  and  $\text{fg-phi}: \forall x \in \{0..<n\}. f x = g (\varphi x)$ 
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⟩

```

```

lemma  $Z\text{-good-fun-alt-sum}:$ 
fixes  $g$ 
defines  $Z\text{-good-fun} \equiv \{f. f \in \{0..<n\} \rightarrow \{0..<m\} \wedge (\forall i. i \notin \{0..<n\} \rightarrow f i = i) \wedge \text{inj-on } f \{0..<n\} \wedge (f \{0..<n\} = g \{0..<n\})\}$ 
assumes  $g: g \in F\text{-inj}$ 
shows  $(\sum f \in Z\text{-good-fun}. P f) = (\sum \pi \in \{\pi. \pi \text{ permutes } \{0..<n\}\}. P (g \circ \pi))$ 
⟨proof⟩

```

```

lemma  $F\text{-injI}:$ 
assumes  $f \in \{0..<n\} \rightarrow \{0..<m\}$ 
and  $(\forall i. i \notin \{0..<n\} \rightarrow f i = i)$  and  $\text{inj-on } f \{0..<n\}$ 
shows  $f \in F\text{-inj}$  ⟨proof⟩

```

```

lemma  $F\text{-inj-composition-permutation}:$ 
assumes  $\text{phi}: \varphi \text{ permutes } \{0..<n\}$ 
and  $g: g \in F\text{-inj}$ 
shows  $g \circ \varphi \in F\text{-inj}$ 
⟨proof⟩

```

```

lemma  $F\text{-strict-imp-}F\text{-inj}:$ 
assumes  $f: f \in F\text{-strict}$ 
shows  $f \in F\text{-inj}$ 
⟨proof⟩

```

```

lemma  $\text{one-step}:$ 
assumes  $g1: g \in F\text{-strict}$ 
shows  $\det(\text{submatrix } A \text{ UNIV } (g \{0..<n\})) * \det(\text{submatrix } B \text{ (g \{0..<n\}) UNIV})$ 
=  $(\sum (x, y) \in Z\text{-good } g. \text{ weight } x y)$  (is ?lhs = ?rhs)
⟨proof⟩

```

```

lemma  $\text{gather-by-strictness}:$ 
 $\sum (\lambda g. \sum (\lambda(f, \pi). \text{ weight } f \pi) (Z\text{-good } g)) F\text{-strict}$ 
=  $\sum (\lambda g. \det(\text{submatrix } A \text{ UNIV } (g \{0..<n\})) * \det(\text{submatrix } B \text{ (g \{0..<n\}) UNIV})) F\text{-strict}$ 

```

$\langle proof \rangle$

lemma *finite-Z-strict*[simp]: *finite Z-strict*
 $\langle proof \rangle$

lemma *finite-Z-not-strict*[simp]: *finite Z-not-strict*
 $\langle proof \rangle$

lemma *finite-Znm*[simp]: *finite (Z n m)*
 $\langle proof \rangle$

lemma *finite-F-inj*[simp]: *finite F-inj*
 $\langle proof \rangle$

lemma *finite-F-strict*[simp]: *finite F-strict*
 $\langle proof \rangle$

lemma *nth-strict-mono*:
 fixes $f::nat \Rightarrow nat$
 assumes $strictf: strict\text{-mono } f \text{ and } i: i < n$
 shows $f i = (\text{sorted-list-of-set } (f^{\{0..<n\}})) ! i$
 $\langle proof \rangle$

lemma *nth-strict-mono-on*:
 fixes $f::nat \Rightarrow nat$
 assumes $strictf: strict\text{-mono-on } \{0..<n\} f \text{ and } i: i < n$
 shows $f i = (\text{sorted-list-of-set } (f^{\{0..<n\}})) ! i$
 $\langle proof \rangle$

lemma *strict-fun-eq*:
 assumes $f: f \in F\text{-strict} \text{ and } g: g \in F\text{-strict} \text{ and } fg: f^{\{0..<n\}} = g^{\{0..<n\}}$
 shows $f = g$
 $\langle proof \rangle$

lemma *strict-from-inj-preserves-F*:
 assumes $f: f \in F\text{-inj}$
 shows *strict-from-inj* $n f \in F$
 $\langle proof \rangle$

lemma *strict-from-inj-F-strict*: *strict-from-inj n xa \in F-strict*
 if $xa: xa \in F\text{-inj}$ **for** xa
 $\langle proof \rangle$

lemma *strict-from-inj-image*:
 assumes $f: f \in F\text{-inj}$
 shows *strict-from-inj* $n f ` \{0..<n\} = f^{\{0..<n\}}$
 $\langle proof \rangle$

```

lemma Z-good-alt:
  assumes g:  $g \in F\text{-strict}$ 
  shows Z-good  $g = \{x \in F\text{-inj. strict-from-inj } n \mid x = g\} \times \{\pi \in \text{permutations } \{0..n\}\}$ 
   $\langle \text{proof} \rangle$ 

```

```

lemma weight-0:  $(\sum (f, \pi) \in Z\text{-not-inj. weight } f \mid \pi) = 0$ 
   $\langle \text{proof} \rangle$ 

```

5.3 Final theorem

```

lemma Cauchy-Binet1:
  shows  $\det(A * B) =$ 
     $\sum (\lambda f. \det(\text{submatrix } A \text{ UNIV } (f \setminus \{0..n\})) * \det(\text{submatrix } B \text{ (f \setminus \{0..n\}) UNIV}))$ 
     $F\text{-strict}$ 
  (is ?lhs = ?rhs)
   $\langle \text{proof} \rangle$ 

```

```

lemma Cauchy-Binet:
   $\det(A * B) = (\sum I \in \{I \subseteq \{0..n\} \wedge \text{card } I = n\}. \det(\text{submatrix } A \text{ UNIV } I) * \det(\text{submatrix } B \text{ I UNIV}))$ 
   $\langle \text{proof} \rangle$ 
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  $A = (\forall i j. i \neq j \wedge i < \text{dim-row } A \wedge j < \text{dim-col } A \longrightarrow A_{i,j} = 0)$ 

```

```

definition Smith-normal-form-mat  $A =$ 
   $(\forall a. a + 1 < \min(\text{dim-row } A, \text{dim-col } A) \longrightarrow A_{a,a} \text{ divides } A_{a+1,a+1})$ 
   $\wedge \text{isDiagonal-mat } A$ 

```

```

lemma SNF-first-divides:
  assumes SNF-A: Smith-normal-form-mat  $A$  and  $(A :: ('a :: \text{comm-ring-1}) \text{ mat}) \in \text{carrier-mat } n m$ 
  and  $i: i < \min(\text{dim-row } A, \text{dim-col } A)$ 

```

shows $A \in \mathbb{Z}^{(0,0)} \text{ dvd } A \in \mathbb{Z}^{(i,i)}$
 $\langle proof \rangle$

lemma *Smith-normal-form-mat-intro*:

assumes $(\forall a. a + 1 < \min(\dim-row A) (\dim-col A) \rightarrow A \in \mathbb{Z}^{(a,a)} \text{ dvd } A \in \mathbb{Z}^{(a+1,a+1)})$
and *isDiagonal-mat A*
shows *Smith-normal-form-mat A*
 $\langle proof \rangle$

lemma *Smith-normal-form-mat-m0[simp]*:
assumes $A: A \in \text{carrier-mat } m \ 0$
shows *Smith-normal-form-mat A*
 $\langle proof \rangle$

lemma *Smith-normal-form-mat-0m[simp]*:
assumes $A: A \in \text{carrier-mat } 0 \ m$
shows *Smith-normal-form-mat A*
 $\langle proof \rangle$

lemma *S00-dvd-all-A*:
assumes $A: (A::'a::comm-ring-1 mat) \in \text{carrier-mat } m \ n$
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 *S-PAQ: $S = P * A * Q$*
and *SNF-S: Smith-normal-form-mat S*
and $i: i < m$ **and** $j: j < n$
shows $S \in \mathbb{Z}^{(0,0)} \text{ dvd } A \in \mathbb{Z}^{(i,j)}$
 $\langle proof \rangle$

lemma *SNF-first-divides-all*:
assumes $SNF-A: \text{Smith-normal-form-mat } A$ **and** $A: (A::('a::comm-ring-1) mat)$
 $\in \text{carrier-mat } m \ n$
and $i: i < m$ **and** $j: j < n$
shows $A \in \mathbb{Z}^{(0,0)} \text{ dvd } A \in \mathbb{Z}^{(i,j)}$
 $\langle proof \rangle$

lemma *SNF-divides-diagonal*:
fixes $A::'a::comm-ring-1 mat$
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 $A \in \mathbb{Z}^{(i,i)} \text{ dvd } A \in \mathbb{Z}^{(j,j)}$
 $\langle proof \rangle$

```

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$$\langle i,i \rangle = 0
  and j: j < min m n
  and ij: i ≤ j
  shows A$$\langle j,j \rangle = 0
  ⟨proof⟩

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⟩

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 ∀ b ∈ B. b ∈ ideal-generated A
  shows ideal-generated B ⊆ ideal-generated A
  ⟨proof⟩

context comm-ring-1
begin

lemma ideal-explicit: ideal-generated S
  = {y. ∃ f U. finite U ∧ U ⊆ S ∧ (∑ i ∈ U. f i * i) = y}
  ⟨proof⟩
end

lemma ideal-generated-minus:
  assumes a: a ∈ ideal-generated (S - {a})
  shows ideal-generated S = ideal-generated (S - {a})
  ⟨proof⟩

lemma ideal-generated-dvd-eq:
  assumes a-dvd-b: a dvd b

```

```

and  $a: a \in S$ 
and  $a\text{-not-}b: a \neq b$ 
shows ideal-generated  $S = \text{ideal-generated } (S - \{b\})$ 
⟨proof⟩

```

```

lemma ideal-generated-dvd-eq-diff-set:
assumes  $i\text{-in-}I: i \in I$  and  $i\text{-in-}J: i \notin J$  and  $i\text{-dvd-}j: \forall j \in J. i \text{ dvd } j$ 
and  $f: \text{finite } J$ 
shows ideal-generated  $I = \text{ideal-generated } (I - J)$ 
⟨proof⟩

```

```

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 ideal-generated  $\{d\} \subseteq \text{ideal-generated } S$ 
⟨proof⟩

```

```

lemma ideal-generated-singleton-dvd:
assumes  $i: \text{ideal-generated } S = \text{ideal-generated } \{d\}$  and  $x: x \in S$ 
shows  $d \text{ dvd } x$ 
⟨proof⟩

```

```

lemma ideal-generated-UNIV-insert:
assumes ideal-generated  $S = \text{UNIV}$ 
shows ideal-generated  $(\text{insert } a S) = \text{UNIV}$  ⟨proof⟩

```

```

lemma ideal-generated-UNIV-union:
assumes ideal-generated  $S = \text{UNIV}$ 
shows ideal-generated  $(A \cup S) = \text{UNIV}$ 
⟨proof⟩

```

```

lemma ideal-explicit2:
assumes  $\text{finite } S$ 
shows ideal-generated  $S = \{y. \exists f. (\sum i \in S. f i * i) = y\}$ 
⟨proof⟩

```

```

lemma ideal-generated-unit:
assumes  $u: u \text{ dvd } 1$ 
shows ideal-generated  $\{u\} = \text{UNIV}$ 
⟨proof⟩

```

```

lemma ideal-generated-dvd-subset:
assumes  $x: \forall x \in S. d \text{ dvd } x$  and  $S: \text{finite } S$ 
shows ideal-generated  $S \subseteq \text{ideal-generated } \{d\}$ 
⟨proof⟩

```

```

lemma ideal-generated-mult-unit:
  assumes f: finite S and u: u dvd 1
  shows ideal-generated (( $\lambda x. u*x$ )` S) = ideal-generated S
  ⟨proof⟩

corollary ideal-generated-mult-unit2:
  assumes u: u dvd 1
  shows ideal-generated {u*a, u*b} = ideal-generated {a, b}
  ⟨proof⟩

lemma ideal-generated-1[simp]: ideal-generated {1} = UNIV
  ⟨proof⟩

lemma ideal-generated-pair: ideal-generated {a, b} = {p*a + q*b | p q. True}
  ⟨proof⟩

lemma ideal-generated-pair-exists-pq1:
  assumes i: ideal-generated {a, b} = (UNIV::'a set)
  shows  $\exists p q. p*a + q*b = 1$ 
  ⟨proof⟩

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⟩

lemma ideal-generated-pair-exists:
  assumes l: (ideal-generated {a, b} = ideal-generated {d})
  shows ( $\exists p q. p*a + q*b = d$ )
  ⟨proof⟩

lemma obtain-ideal-generated-pair:
  assumes c ∈ ideal-generated {a, b}
  obtains p q where p*a + q*b = c
  ⟨proof⟩

lemma ideal-generated-pair-exists-UNIV:
  shows (ideal-generated {a, b} = ideal-generated {1}) = ( $\exists p q. p*a + q*b = 1$ ) (is
  ?lhs = ?rhs)
  ⟨proof⟩

corollary ideal-generated-UNIV-obtain-pair:
  assumes ideal-generated {a, b} = ideal-generated {1}
  shows ( $\exists p q. p*a + q*b = d$ )
  ⟨proof⟩

```

```

lemma sum-three-elements:
  shows  $\exists x y z::'a. (\sum i \in \{a,b,c\}. f i * i) = x * a + y * b + z * c$ 
   $\langle proof \rangle$ 

lemma sum-three-elements':
  shows  $\exists f::'a \Rightarrow 'a. (\sum i \in \{a,b,c\}. f i * i) = x * a + y * b + z * c$ 
   $\langle proof \rangle$ 

lemma ideal-generated-triple-pair-rewrite:
  assumes i1: ideal-generated {a, b, c} = ideal-generated {d}
  and i2: ideal-generated {a, b} = ideal-generated {d'}
  shows ideal-generated{d',c} = ideal-generated {d}
   $\langle proof \rangle$ 

lemma ideal-generated-dvd:
  assumes i: ideal-generated {a,b::'a} = ideal-generated{d}
  and a: d' dvd a and b: d' dvd b
  shows d' dvd d
   $\langle proof \rangle$ 

lemma ideal-generated-dvd2:
  assumes i: ideal-generated S = ideal-generated{d::'a}
  and finite S
  and x:  $\forall x \in S. d' \text{ dvd } x$ 
  shows d' dvd d
   $\langle proof \rangle$ 

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** $d = p * a + q * b$
 $\langle proof \rangle$

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:  $\forall I::'a$  set. finitely-generated-ideal  $I \rightarrow$  principal-ideal  $I$ 
  shows OFCLASS ('a, bezout-ring-class)
   $\langle proof \rangle$ 
end
```

```
context bezout-ring
begin
```

```
lemma exists-bezout-extended:
  assumes S: finite  $S$  and ne:  $S \neq \{\}$ 
  shows  $\exists f d. (\sum a \in S. f a * a) = d \wedge (\forall a \in S. d \text{ dvd } a) \wedge (\forall d'. (\forall a \in S. d' \text{ dvd } a) \rightarrow d' \text{ dvd } d)$ 
   $\langle proof \rangle$ 
```

end

```
lemma ideal-generated-empty: ideal-generated  $\{\} = \{0\}$ 
   $\langle proof \rangle$ 
```

```
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
   $\langle proof \rangle$ 
```

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 - - - \Rightarrow 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: class.bezout-ring (*) (1::'b::comm-ring-1) (+) 0 (-) uminus
shows  $\forall I::'b$  set. finitely-generated-ideal I  $\longrightarrow$  principal-ideal I
⟨proof⟩
```

The standard library does not connect *OFCLASS* and *class.bezout-ring* in both directions. Here we show that *OFCLASS* \Rightarrow *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
⟨proof⟩
```

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:
 $(\bigwedge I::'a::comm-ring-1 \text{ set. finitely-generated-ideal } I \Rightarrow \text{principal-ideal } I)$ 
 $\equiv \text{OFCASS}('a, \text{bezout-ring-class})$ 
⟨proof⟩
```

Final theorem (with *class.bezout-ring*)

```
lemma bezout-ring-iff-fin-gen-principal-ideal2:
 $(\forall I::'a::comm-ring-1 \text{ set. finitely-generated-ideal } I \rightarrow \text{principal-ideal } I)$ 
 $= (\text{class.bezout-ring } ((*)::'a⇒'a⇒'a) 1 (+) 0 (-) uminus)$ 
⟨proof⟩
```

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 ⇒ 'a mod-ring ⇒ bool
  where less-eq-mod-ring x y = (to-int-mod-ring x ≤ to-int-mod-ring y)

definition less-mod-ring :: 'a mod-ring ⇒ 'a mod-ring ⇒ bool
  where less-mod-ring x y = (to-int-mod-ring x < to-int-mod-ring y)

instance ⟨proof⟩
end

instantiation mod-ring :: (finite) linorder
begin
instance ⟨proof⟩
end

instance mod-ring :: (finite) wellorder
⟨proof⟩

lemma strict-mono-to-int-mod-ring: strict-mono to-int-mod-ring
⟨proof⟩

instantiation mod-ring :: (nontriv) mod-type
begin
definition Rep-mod-ring :: 'a mod-ring ⇒ int
  where Rep-mod-ring x = to-int-mod-ring x

definition Abs-mod-ring :: int ⇒ 'a mod-ring
  where Abs-mod-ring x = of-int-mod-ring x

instance
⟨proof⟩
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 ( $A::'a::\text{comm-ring-1} \wedge n::\{\text{mod-type}\} \wedge n::\{\text{mod-type}\}$ ) =
(isDiagonal A
 → ( $\exists P Q. \text{invertible } ((P::'a::\text{comm-ring-1} \wedge n::\{\text{mod-type}\} \wedge n::\{\text{mod-type}\}))$ )
 ∧  $\text{invertible } (Q::'a::\text{comm-ring-1} \wedge n::\{\text{mod-type}\} \wedge n::\{\text{mod-type}\}) \wedge \text{Smith-normal-form}$ 
( $P**A**Q$ ))

definition admits-SNF-JNF  $A = (\text{square-mat } (A::'a::\text{comm-ring-1 mat}) \wedge \text{isDiag-}$ 
 $\text{onal-mat } A$ 
→ ( $\exists P Q. P \in \text{carrier-mat } (\text{dim-row } A) (\text{dim-row } A) \wedge Q \in \text{carrier-mat}$ 
( $\text{dim-row } A$ ) ( $\text{dim-row } A$ )
 $\wedge \text{invertible-mat } P \wedge \text{invertible-mat } Q \wedge \text{Smith-normal-form-mat } (P*A*Q))$ 

```

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⟩

```

We apply *internalize-sort* to the lemma that we need

```

lemmas diagonal-to-Smith-PQ-exists-internalize-sort
= diagonal-to-Smith-PQ-exists[internalize-sort 'a :: 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  $\forall A::'a \wedge n::\{\text{mod-type}\} \wedge n::\{\text{mod-type}\}. \text{isDiagonal } A$ 
→ ( $\exists P Q.$ 
 $\text{invertible } (P::'a \wedge n::\text{mod-type} \wedge n::\text{mod-type}) \wedge$ 
 $\text{invertible } (Q::'a \wedge n::\text{mod-type} \wedge n::\text{mod-type}) \wedge$ 
 $\text{Smith-normal-form } (P**A**Q))$ 
⟨proof⟩

```

9.2 Trying to prove the \Rightarrow 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  $\forall A::'a::\text{comm-ring-1}^n::\{\text{mod-type}\}^n::\{\text{mod-type}\}$ . admits-SNF-HA
   $A$ 
  shows OFCLASS('a::comm-ring-1, bezout-ring-class) ⟨proof⟩
```

9.3 Proof of the \Rightarrow implication in JNF.

```
lemma exists-f-PAQ-Aii:
  assumes diag-A: isDiagonal-mat (A:'a:: 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  $\exists f. (P * A * Q) \ \$(i, i) = (\sum_{i \in \text{set } (\text{diag-mat } A)} f i * i)$ 
  ⟨proof⟩
```

Proof of the \Rightarrow 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⟩
```

```
corollary diagonal-admits-SNF-imp-bezout-ring-JNF-alt:
  assumes admits-SNF:  $\forall A. \text{square-mat } (A::'a \ \text{mat}) \wedge \text{isDiagonal-mat } A$ 
   $\longrightarrow (\exists P \ Q. P \in \text{carrier-mat } (\text{dim-row } A) (\text{dim-row } A) \wedge Q \in \text{carrier-mat } (\text{dim-row } A) (\text{dim-row } A) \wedge \text{invertible-mat } P \wedge \text{invertible-mat } Q \wedge \text{Smith-normal-form-mat } (P * A * Q))$ 
  shows OFCLASS('a::comm-ring-1, bezout-ring-class)
  ⟨proof⟩
```

9.4 Trying to transfer the \Rightarrow 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
  <proof>

lemma inj-on-Bij-nat-from-nat: inj-on (Bij-Nat.from-nat::nat  $\Rightarrow$  'a) {0..<CARD('a::finite)}
  <proof>

```

This lemma only holds if a and b have the same type. Otherwise, it is possible that $Bij\text{-}Nat.to\text{-}nat a = Bij\text{-}Nat.to\text{-}nat b$

```

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
  <proof>

```

The following proof (a transfer rule for diagonal matrices) is weird, since it does not hold $Bij\text{-}Nat.to\text{-}nat a = \text{mod-type-class.to-nat } a$.

At first, it seems possible to obtain the element a' that satisfies $Bij\text{-}Nat.to\text{-}nat a' = \text{mod-type-class.to-nat } a$ and then continue with the proof, but then we cannot prove *HMA-I* ($Bij\text{-}Nat.to\text{-}nat a'$) 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 ===> (=))
  isDiagonal-mat (isDiagonal::('a::{zero}  $\wedge$  n::{mod-type}  $\wedge$  n::{mod-type})  $\Rightarrow$  bool)
  <proof>

```

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 ===>
  (=))
  isDiagonal-mat (isDiagonal::('a::{zero}  $\wedge$  n::{mod-type}  $\wedge$  m::{mod-type})  $\Rightarrow$  bool)
  <proof>

```

We state the transfer rule using the relations developed in the new bride of the file *Mod-Type-Connect*.

```

lemma HMA-SNF[transfer-rule]: (Mod-Type-Connect.HMA-M ===> (=)) Smith-normal-form-mat

```

$(Smith\text{-}normal\text{-}form::'a::\{comm\text{-}ring\text{-}1\} \wedge 'n::\{mod\text{-}type\} \wedge 'm::\{mod\text{-}type\} \Rightarrow bool)$
 $\langle proof \rangle$

```
lemma HMA-admits-SNF [transfer-rule]:
  ((Mod-Type-Connect.HMA-M :: -  $\Rightarrow$  'a :: comm-ring-1  $\wedge$  'n::{mod-type}  $\wedge$  'n::{mod-type}  $\Rightarrow$  -)  $\equiv\equiv\Rightarrow$  (=))
    admits-SNF-JNF admits-SNF-HA
   $\langle proof \rangle$ 
end
```

Here we have a problem when trying to apply local type definitions

```
lemma diagonal-admits-SNF-imp-bezout-ring:
  assumes admits-SNF:  $\forall A::'a::comm\text{-}ring\text{-}1 \wedge 'n::\{mod\text{-}type\} \wedge 'n::\{mod\text{-}type\}$ . is-Diagonal A
     $\longrightarrow (\exists P Q. invertible (P::'a::comm\text{-}ring\text{-}1 \wedge 'n::\{mod\text{-}type\} \wedge 'n::\{mod\text{-}type\}) \wedge invertible (Q::'a::comm\text{-}ring\text{-}1 \wedge 'n::\{mod\text{-}type\} \wedge 'n::\{mod\text{-}type\}) \wedge Smith\text{-}normal\text{-}form (P**A**Q))$ 
  shows OFCLASS('a::comm-ring-1, bezout-ring-class)
   $\langle proof \rangle$ 
```

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  $\forall A::'a \wedge 'n::nontriv mod\text{-}ring \wedge 'n::nontriv mod\text{-}ring. isDiagonal A \longrightarrow (\exists P Q.$ 
     $invertible (P::'a \wedge 'n::nontriv mod\text{-}ring \wedge 'n::nontriv mod\text{-}ring) \wedge invertible (Q::'a \wedge 'n::nontriv mod\text{-}ring \wedge 'n::nontriv mod\text{-}ring) \wedge Smith\text{-}normal\text{-}form (P**A**Q))$ 
   $\langle proof \rangle$ 
```

```
lemma bezout-ring-imp-diagonal-admits-SNF-mod-ring-admits:
  assumes of: class.bezout-ring (*)
   $(1::'a::comm\text{-}ring\text{-}1) (+) 0 (-) uminus$ 
  shows  $\forall A::'a \wedge 'n::nontriv mod\text{-}ring \wedge 'n::nontriv mod\text{-}ring. admits\text{-}SNF\text{-}HA A$ 
   $\langle proof \rangle$ 
```

I start here to apply local type definitions

```
context
  fixes p::nat
  assumes local-typedef:  $\exists (Rep :: ('b \Rightarrow int)) Abs. type\text{-}definition Rep Abs \{0..<p :: int\}$ 
```

```

and  $p: p > 1$ 
begin

lemma type-to-set:
  shows class.nontriv  $TYPE('b)$  (is  $?a$ ) and  $p = CARD('b)$  (is  $?b$ )
   $\langle proof \rangle$ 

```

I transfer the lemma from HA to JNF, substituting $CARD('n)$ by p . I apply *internalize-sort* to ' n ' and get rid of the *nontriv* restriction.

```

lemma bezout-ring-imp-diagonal-admits-SNF-mod-ring-admits-aux:
  assumes class.bezout-ring (*) (1::' $a$ ::comm-ring-1) (+) 0 (-) uminus
  shows Ball { $A :: 'a :: comm-ring-1$  mat.  $A \in carrier\text{-}mat p p$ } admits-SNF-JNF
   $\langle proof \rangle$ 
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-ring-imp-diagonal-admits-SNF-mod-ring-admits-aux2:
  assumes of: class.bezout-ring (*) (1::' $a$ ::comm-ring-1) (+) 0 (-) uminus
  shows  $\forall (A :: 'a mat) \in carrier\text{-}mat n n$ . admits-SNF-JNF A
   $\langle proof \rangle$ 

```

Alternative statements

```

lemma bezout-ring-imp-diagonal-admits-SNF-JNF:
  assumes of: class.bezout-ring (*) (1::' $a$ ::comm-ring-1) (+) 0 (-) uminus
  shows  $\forall A :: 'a mat$ . admits-SNF-JNF A
   $\langle proof \rangle$ 

```

```

lemma admits-SNF-JNF-alt-def:
   $(\forall A :: 'a :: comm-ring-1$  mat. admits-SNF-JNF A)
   $= (\forall A n. (A :: 'a mat) \in carrier\text{-}mat n n \wedge isDiagonal\text{-}mat A$ 
   $\longrightarrow (\exists P Q. P \in carrier\text{-}mat n n \wedge Q \in carrier\text{-}mat n n \wedge invertible\text{-}mat P \wedge$ 
   $invertible\text{-}mat Q$ 
   $\wedge Smith\text{-}normal\text{-}form\text{-}mat (P * A * Q)))$  (is  $?a = ?b$ )
   $\langle proof \rangle$ 

```

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
   $\longleftrightarrow (\forall A :: 'a mat. admits\text{-}SNF\text{-}JNF A)$  (is  $?a \longleftrightarrow ?b$ )
   $\langle proof \rangle$ 

```

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⟩
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  $\exists u. u \text{ dvd } 1 \wedge a = u * b$ 
  shows  $a \text{ dvd } b \wedge b \text{ dvd } a$ 
  ⟨proof⟩
```

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  $ab: a \text{ dvd } b \text{ and } ba: b \text{ dvd } a \text{ and } a: a \neq 0$ 
  shows  $\exists u. u \text{ dvd } 1 \wedge a = u * b$ 
  ⟨proof⟩
```

```
corollary dvd-associated:
  fixes a:'a::idom
  assumes  $a \neq 0$ 
  shows  $(a \text{ dvd } b \wedge b \text{ dvd } a) = (\exists u. u \text{ dvd } 1 \wedge a = u * b)$ 
  ⟨proof⟩
```

```
lemma exists-inj-ge-index:
  assumes  $S: S \subseteq \{0..\} \text{ and } Sk: \text{card } S = k$ 
  shows  $\exists f. \text{inj-on } f \{0..\} \wedge f\{0..\} = S \wedge (\forall i \in \{0..\}. i \leq f i)$ 
  ⟨proof⟩
```

10.1 More specific results about submatrices

```
lemma diagonal-imp-submatrix0:
  assumes  $dA: \text{diagonal-mat } A \text{ and } A\text{-carrier}: A \in \text{carrier-mat } n m$ 
```

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 $\text{submatrix } A[IJ] \$\$ (a, b) = 0 \vee \text{submatrix } A[IJ] \$\$ (a, b) = A \$\$ (\text{pick } I a, \text{pick } J b)$
 $\langle \text{proof} \rangle$

lemma *diagonal-imp-submatrix-element-not0*:

assumes $dA: \text{diagonal-mat } A$
and $A\text{-carrier}: A \in \text{carrier-mat } n m$
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 $\text{ex-not0}: \exists i. i < k \wedge \text{submatrix } A[IJ] \$\$ (i, b) \neq 0$
shows $\exists !i. i < k \wedge \text{submatrix } A[IJ] \$\$ (i, b) \neq 0$
 $\langle \text{proof} \rangle$

lemma *submatrix-index-exists*:

assumes $A\text{-carrier}: A \in \text{carrier-mat } n m$
and $Ik: \text{card } I = k$ **and** $Jk: \text{card } J = k$
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 $\exists a' b'. a' < k \wedge b' < k \wedge \text{submatrix } A[IJ] \$\$ (a', b') = A \$\$ (a, b)$
 $\wedge a = \text{pick } I a' \wedge b = \text{pick } J b'$
 $\langle \text{proof} \rangle$

lemma *mat-delete-submatrix-insert*:

assumes $A\text{-carrier}: A \in \text{carrier-mat } n m$
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 $a\text{-notin-}I: a \notin I$ **and** $b\text{-notin-}J: b \notin J$
and $a'k: a' < \text{Suc } k$ **and** $b'k: b' < \text{Suc } k$
and $a\text{-def}: \text{pick } (\text{insert } a I) a' = a$
and $b\text{-def}: \text{pick } (\text{insert } b J) b' = b$
shows $\text{mat-delete } (\text{submatrix } A (\text{insert } a I) (\text{insert } b J)) a' b' = \text{submatrix } A[IJ]$
 $(\text{is } ?lhs = ?rhs)$
 $\langle \text{proof} \rangle$

10.2 On the minors of a diagonal matrix

lemma *det-minors-diagonal*:

assumes $dA: \text{diagonal-mat } A$ **and** $A\text{-carrier}: A \in \text{carrier-mat } n m$
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 $\det(\text{submatrix } A I J) = 0$
 $\vee (\exists xs. (\det(\text{submatrix } A I J) = \text{prod-list } xs \vee \det(\text{submatrix } A I J) = -\text{prod-list } xs))$
 $\wedge \text{set } xs \subseteq \{A\$$(i,i)|i. i < \min n m \wedge A\$$(i,i) \neq 0\} \wedge \text{length } xs = k$
 $\langle \text{proof} \rangle$

definition $\text{minors } A k = \{\det(\text{submatrix } A I J) | I J. I \subseteq \{0..<\text{dim-row } A\}$
 $\wedge J \subseteq \{0..<\text{dim-col } A\} \wedge \text{card } I = k \wedge \text{card } J = k\}$

lemma $Gcd\text{-minors-dvd}:$
fixes $A: 'a::\{\text{semiring-Gcd}, \text{comm-ring-1}\} \text{ mat}$
assumes $PAQ\text{-B}: P * A * Q = B$
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 $Ik: \text{card } I = k$ **and** $Jk: \text{card } J = k$
shows $\text{Gcd}(\text{minors } A k) \text{ dvd } \det(\text{submatrix } B I J)$
 $\langle \text{proof} \rangle$

lemma $\det\text{-minors-diagonal2}:$
assumes $dA: \text{diagonal-mat } A$ **and** $A\text{-carrier}: A \in \text{carrier-mat } n m$
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)))$
 $\langle \text{proof} \rangle$

10.3 Relating minors and GCD

lemma $\text{diagonal-dvd-Gcd-minors}:$
fixes $A: 'a::\{\text{semiring-Gcd}, \text{comm-ring-1}\} \text{ mat}$
assumes $A: A \in \text{carrier-mat } n m$
and $SNF\text{-A}: \text{Smith-normal-form-mat } A$
shows $(\prod_{i=0..<k} A \$$(i,i)) \text{ dvd } \text{Gcd}(\text{minors } A k)$
 $\langle \text{proof} \rangle$

lemma $Gcd\text{-minors-dvd-diagonal}:$

```

fixes A::'a::{semiring-Gcd,comm-ring-1} mat
assumes A: A ∈ carrier-mat n m
    and SNF-A: Smith-normal-form-mat A
    and k: k ≤ min n m
shows Gcd (minors A k) dvd (Π i=0... A $$ (i,i))
⟨proof⟩

```

```

lemma Gcd-minors-A-dvd-Gcd-minors-PAQ:
fixes A::'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
shows Gcd (minors A k) dvd Gcd (minors (P*A*Q) k)
⟨proof⟩

```

```

lemma Gcd-minors-PAQ-dvd-Gcd-minors-A:
fixes A::'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 inv-P: invertible-mat P
    and inv-Q: invertible-mat Q
shows Gcd (minors (P*A*Q) k) dvd Gcd (minors A k)
⟨proof⟩

```

```

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 (Π i=0... (P * A * Q) $$ (i,i))
⟨proof⟩

```

```

lemma diag-PAQ-dvd-Gcd-minors:
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 inv-P: invertible-mat P
    and inv-Q: invertible-mat Q
    and SNF: Smith-normal-form-mat (P*A*Q)
shows (Π i=0... (P * A * Q) $$ (i,i)) dvd Gcd (minors A k)
⟨proof⟩

```

```

lemma Smith-prod-zero-imp-last-zero:
  fixes A::'a::{semidom,comm-ring-1} mat
  assumes A: A ∈ carrier-mat m n
    and SNF: Smith-normal-form-mat A
    and prod-0: (Π j=0.. $<$ Suc i. A $$ (j,j)) = 0
    and i: i < min m n
    shows A $$ (i,i) = 0
  ⟨proof⟩

```

10.4 Final theorem

```

lemma Smith-normal-form-uniqueness-aux:
  fixes P A Q::'a::{idom,semiring-Gcd} mat
  assumes A: A ∈ 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 PAQ-B: P*A*Q = B
    and SNF: Smith-normal-form-mat B
    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 P'AQ'-B': P'*A*Q' = B'
    and SNF-B': Smith-normal-form-mat B'
    and k: k < min m n
    shows ∀ i ≤ k. B $$ (i,i) dvd B' $$ (i,i) ∧ B' $$ (i,i) dvd B $$ (i,i)
  ⟨proof⟩

```

```

lemma Smith-normal-form-uniqueness:
  fixes P A Q::'a::{idom,semiring-Gcd} mat
  assumes A: A ∈ 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 PAQ-B: P*A*Q = B
    and SNF: Smith-normal-form-mat B
    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 P'AQ'-B': P'*A*Q' = B'
and SNF-B': Smith-normal-form-mat B'
and i: i < min m n
shows  $\exists u. u \text{ dvd } 1 \wedge B \$\$ (i,i) = u * B' \$\$ (i,i)$ 
(proof)

```

The final theorem, moved to HOL Analysis

```

lemma Smith-normal-form-uniqueness-HOL-Analysis:
fixes A::'a::{'idom,semiring-Gcd}^m::mod-type^n::mod-type
and P P'::'a^n::mod-type^m::mod-type
and Q Q'::'a^m::mod-type^n::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  $\exists u. u \text{ dvd } 1 \wedge B \$h \text{Mod-Type.from-nat } i \$h \text{Mod-Type.from-nat } i$ 
 $= u * B' \$h \text{Mod-Type.from-nat } i \$h \text{Mod-Type.from-nat } i$ 
(proof)

```

10.5 Uniqueness fixing a complete set of non-associates

```

definition Smith-normal-form-wrt A  $\mathcal{Q} = ($ 
 $(\forall a b. \text{Mod-Type.to-nat } a = \text{Mod-Type.to-nat } b \wedge \text{Mod-Type.to-nat } a + 1 <$ 
 $\text{nrows } A \wedge \text{Mod-Type.to-nat } b + 1 < \text{ncols } A \longrightarrow A \$h a \$h b \text{ dvd } A \$h (a+1)$ 
 $\$h (b+1))$ 
 $\wedge \text{isDiagonal } A \wedge \text{Complete-set-non-associates } \mathcal{Q}$ 
 $\wedge (\forall a b. \text{Mod-Type.to-nat } a = \text{Mod-Type.to-nat } b \wedge \text{Mod-Type.to-nat } a < \min$ 
 $(\text{nrows } A) (\text{ncols } A) \wedge \text{Mod-Type.to-nat } b < \min (\text{nrows } A) (\text{ncols } A) \longrightarrow A \$h a \$h b \in \mathcal{Q})$ 
 $)$ 

```

```

lemma Smith-normal-form-wrt-uniqueness-HOL-Analysis:
fixes A::'a::{'idom,semiring-Gcd}^m::mod-type^n::mod-type
and P P'::'a^n::mod-type^m::mod-type
and Q Q'::'a^m::mod-type^n::mod-type
assumes

```

```

P: invertible P
and Q: invertible Q

```

and $PAQ-S$: $P \otimes A \otimes Q = S$
and SNF : *Smith-normal-form-wrt S Q*

and P' : *invertible P'*
and Q' : *invertible Q'*
and $P' A Q' - S'$: $P' \otimes A \otimes Q' = S'$
and $SNF-S'$: *Smith-normal-form-wrt S' Q*
shows $S = S'$
(proof)

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* :: ' $a \wedge' nc \wedge' nr \Rightarrow nat$ set $\Rightarrow nat$ set $\Rightarrow ('a \wedge' nc \wedge' nr2)$
where *submatrix-hma* $A\ I\ J = (\chi\ a\ b.\ A\$h(from-nat(pick\ I\ (to-nat\ a)))\$h(from-nat(pick\ J\ (to-nat\ b))))$

context includes *lifting-syntax*
begin

context

fixes $I::nat\ set$ **and** $J::nat\ set$

assumes I : $\text{card } \{i \mid i < \text{CARD('nr::finite)} \wedge i \in I\} = \text{CARD('nr2::finite)}$
assumes J : $\text{card } \{i \mid i < \text{CARD('nc::finite)} \wedge i \in J\} = \text{CARD('nc2::finite)}$

begin

lemma *HMA-submatrix[transfer-rule]*: (*HMA-M* ==> *HMA-M*) ($\lambda A.$ *submatrix*
 $A\ I\ J$)
 $((\lambda A.$ *submatrix-hma* $A\ I\ J)$): ' $a \wedge' nc \wedge' nr \Rightarrow 'a \wedge' nc2 \wedge' nr2)$
 $\langle proof \rangle$

end

11.2 Transferring the proof from JNF to HOL Analysis

lemma Cauchy-Binet-HOL-Analysis:
fixes $A::'a::comm-ring-1 \wedge m \wedge n$ **and** $B::'a \wedge n \wedge m$

```

shows Determinants.det (A**B) = ( $\sum I \in \{I \mid I \subseteq \{0..<\text{ncols } A\} \wedge \text{card } I = \text{nrows } A\}$ .
Determinants.det ((submatrix-hma A UNIV I)::'a ^'n ^'n) *
Determinants.det ((submatrix-hma B I UNIV)::'a ^'n ^'n))
⟨proof⟩
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 ⇒ 'a bezout ⇒ ('a mat × 'a mat ×
'a mat)
assumes soundness-diagonalize-JNF:
  ∀ A bezout. A ∈ carrier-mat m n ∧ is-bezout-ext bezout →
  (case diagonalize-JNF A bezout of (P,S,Q) ⇒
    P ∈ carrier-mat m m ∧ Q ∈ carrier-mat n n ∧ S ∈ carrier-mat m n
    ∧ invertible-mat P ∧ invertible-mat Q ∧ isDiagonal-mat S ∧ S = P*A*Q)
begin

lemma soundness-diagonalize-JNF':
fixes A::'a mat
assumes is-bezout-ext bezout and A ∈ carrier-mat m n
and diagonalize-JNF A bezout = (P,S,Q)
shows P ∈ carrier-mat m m ∧ Q ∈ carrier-mat n n ∧ S ∈ carrier-mat m n
  ∧ invertible-mat P ∧ invertible-mat Q ∧ isDiagonal-mat S ∧ S = P*A*Q
⟨proof⟩

```

12.2 Implementation and soundness result moved to HOL Analysis.

```

definition diagonalize :: 'a::bezout-ring ^ 'nc :: mod-type ^ 'nr :: mod-type
  ⇒ 'a bezout ⇒
  (('a ^ 'nr :: mod-type ^ 'nr :: mod-type)

```

```

    × ('a ^ 'nc :: mod-type ^ 'nr :: mod-type)
    × ('a ^ 'nc :: mod-type ^ 'nc :: mod-type))
where diagonalize A bezout =
  let (P,S,Q) = diagonalize-JNF (Mod-Type-Connect.from-hmam A) bezout
  in (Mod-Type-Connect.to-hmam P,Mod-Type-Connect.to-hmam S,Mod-Type-Connect.to-hmam
Q)
)

lemma soundness-diagonalize:
assumes b: is-bezout-ext bezout
and d: diagonalize A bezout = (P,S,Q)
shows invertible P ∧ invertible Q ∧ isDiagonal S ∧ S = P**A**Q
⟨proof⟩
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:  $(P, S, Q) = \text{Smith-normal-form-of } A \text{ bezout}$ 
shows  $S = P \star\star A \star\star Q \wedge \text{invertible } P \wedge \text{invertible } Q \wedge \text{Smith-normal-form } S$ 
⟨proof⟩

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_m* and *to-hma_m*. That is, transform the matrix to HOL Analysis, apply the existing algorith 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  $A \in \text{carrier-mat CARD('nr::mod-type) CARD('nc::mod-type)}$ 
begin

private definition diagonal-to-Smith-PQ-JNF' bezout = (
  let  $A' = \text{Mod-Type-Connect.to-hma}_m A : 'a \wedge 'nc::mod-type \wedge 'nr::mod-type;$ 
   $(P, S, Q) = (\text{diagonal-to-Smith-PQ } A' \text{ bezout})$ 
)

```

in (Mod-Type-Connect.from-hma_m P, Mod-Type-Connect.from-hma_m S, 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' TYPE('nc) TYPE('nr) A bezout

And we cannot get rid of such *TYPE('nc)*.

That is, we could get a lemma like:

lemma assumes *A* \in carrier-mat *m n* **and** $(P, S, Q) = \text{diagonal-to-Smith-PQ-JNF}'$
TYPE('nr::mod-type) TYPE('nc::mod-type) A 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* =
 $\text{Matrix.mat}(\text{dim-row } A)(\text{dim-col } A)(\lambda(a, b). \text{if } a = i \wedge b = i \text{ then } d$
else
 $\text{if } a = j \wedge b = j$
 $\text{then } v * (A \$\$ (j, j)) \text{ else } A \$\$ (a, b))$

Conditional transfer rules are required, so I prove them within context with assumptions.

context
includes lifting-syntax
fixes *i* **and** *j::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]:*
 $((\text{Mod-Type-Connect.HMA-M} :: - \Rightarrow 'a :: \text{comm-ring-1} \wedge 'nc :: \text{mod-type} \wedge 'nr :: \text{mod-type} \Rightarrow -)$
 $\implies (\lambda A. \text{diagonal-step-JNF } A \ i \ j) \ (\lambda B. \text{diagonal-step } B \ i \ j)$
 $\langle \text{proof} \rangle$

end

definition *diagonal-step-PQ-JNF ::*
 $'a :: \{\text{bezout-ring}\} \text{ mat} \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a \text{ bezout} \Rightarrow ('a \text{ mat} \times ('a \text{ mat}))$
where *diagonal-step-PQ-JNF A i k bezout* =
 $(\text{let } m = \text{dim-row } A; n = \text{dim-col } A;$
 $(p, q, u, v, d) = \text{bezout} (A \$\$ (i, i)) (A \$\$ (k, k));$

```

 $P = addrow (-v) k i (swaprows i k (addrow p k i (1_m m)));$ 
 $Q = multcol k (-1) (addcol u k i (addcol q i k (1_m n)))$ 
 $in (P, Q)$ 
 $)$ 

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]:
 $((Mod\text{-}Type\text{-}Connect.HMA\text{-}M :: - \Rightarrow 'a :: bezout-ring \wedge 'nc :: mod-type \wedge 'nr :: mod-type \Rightarrow -) \implies (=) \implies rel\text{-}prod Mod\text{-}Type\text{-}Connect.HMA\text{-}M Mod\text{-}Type\text{-}Connect.HMA\text{-}M)$ 
 $(\lambda A \text{ bezout. diagonal-step-PQ-JNF } A i k \text{ bezout}) (\lambda A \text{ bezout. diagonal-step-PQ } A i k \text{ bezout})$ 
 $\langle proof \rangle$ 

end

fun diagonal-to-Smith-i-PQ-JNF :: 
  nat list  $\Rightarrow$  nat  $\Rightarrow$  ('a:{bezout-ring} bezout)
   $\Rightarrow$  ('a mat  $\times$  'a mat  $\times$  'a mat)  $\Rightarrow$  ('a mat  $\times$  'a mat  $\times$  '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  $(P, A, Q) = ($ 
     $\text{if } A \$\$ (i, i) \text{ dvd } A \$\$ (j, j)$ 
     $\text{then diagonal-to-Smith-i-PQ-JNF } xs \ i \text{ bezout } (P, A, Q)$ 
     $\text{else let } (p, q, u, v, d) = \text{bezout } (A \$\$ (i, i)) (A \$\$ (j, j));$ 
     $A' = \text{diagonal-step-JNF } A \ i \ j \ d \ v;$ 
     $(P', Q') = \text{diagonal-step-PQ-JNF } A \ i \ j \ \text{bezout}$ 
     $\text{in diagonal-to-Smith-i-PQ-JNF } xs \ i \ \text{bezout } (P' * P, A', Q * Q')$  — Apply the step
   $)$ 

context
includes lifting-syntax
fixes  $i$  and  $xs$ 
assumes  $i: i < min (CARD('nr::mod-type)) (CARD('nc::mod-type))$ 
and  $xs: \forall j \in set xs. j < min (CARD('nr::mod-type)) (CARD('nc::mod-type))$ 
begin

declare diagonal-step-PQ.simps[simp del]

lemma HMA-diagonal-to-Smith-i-PQ-aux: HMA-M3  $(P, A, Q)$ 

```

```

( $P' :: 'a :: \text{bezout-ring} \wedge 'nr :: \text{mod-type} \wedge 'nr :: \text{mod-type}$ ,
 $A' :: 'a :: \text{bezout-ring} \wedge 'nc :: \text{mod-type} \wedge 'nr :: \text{mod-type}$ ,
 $Q' :: 'a :: \text{bezout-ring} \wedge 'nc :: \text{mod-type} \wedge 'nc :: \text{mod-type}$ )
 $\implies \text{HMA-M3 } (\text{diagonal-to-Smith-i-PQ-JNF } xs \ i \ \text{bezout } (P, A, Q))$ 
 $\quad (\text{diagonal-to-Smith-i-PQ } xs \ i \ \text{bezout } (P', A', Q'))$ 
 $\langle \text{proof} \rangle$ 

lemma HMA-diagonal-to-Smith-i-PQ[transfer-rule]:
  ((=)
  ===> (HMA-M3 :: (-  $\Rightarrow$  (-  $\times$  ('a :: bezout-ring  $\wedge$  'nc :: mod-type  $\wedge$  'nr :: mod-type)
   $\times$  -)  $\Rightarrow$  -))
  ===> HMA-M3) (diagonal-to-Smith-i-PQ-JNF xs i) (diagonal-to-Smith-i-PQ
  xs i)
  ⟨proof⟩

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 :: (-  $\Rightarrow$  (-  $\times$  ('a::bezout-ring  $\wedge$  'nc::mod-type  $\wedge$  'nr::mod-type)
   $\times$  -)  $\Rightarrow$  -)) ===> HMA-M3)
  (Diagonal-to-Smith-row-i-PQ-JNF i) (Diagonal-to-Smith-row-i-PQ i)
  ⟨proof⟩

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:  $\forall j \in \text{set } xs. j < \min (\text{CARD}('nr::mod-type)) (\text{CARD}('nc::mod-type))$ 
begin

lemma HMA-diagonal-to-Smith-aux-PQ-JNF[transfer-rule]:
 $((=) ==> (\text{HMA-M3} :: (- \Rightarrow (- \times ('a::bezout-ring \wedge 'nc::mod-type \wedge 'nr::mod-type) \times -) \Rightarrow -)) ==> \text{HMA-M3})$ 
 $(\text{diagonal-to-Smith-aux-PQ-JNF } xs) (\text{diagonal-to-Smith-aux-PQ } xs)$ 
⟨proof⟩

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 (1_m (\dim-row A), A, 1_m (\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
  ⟨proof⟩

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
  ⟨proof⟩

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 ∈ carrier-mat m m ∧ A ∈ carrier-mat m n ∧ Q ∈ carrier-mat n n
  ⟨proof⟩

lemma Diagonal-to-Smith-row-i-PQ-JNF-dim:
  assumes P' ∈ carrier-mat m m ∧ A' ∈ carrier-mat m n ∧ Q' ∈ carrier-mat n n
  and Diagonal-to-Smith-row-i-PQ-JNF i bezout (P', A', Q') = (P, A, Q)
  shows P ∈ carrier-mat m m ∧ A ∈ carrier-mat m n ∧ Q ∈ carrier-mat n n
  ⟨proof⟩

lemma diagonal-to-Smith-aux-PQ-JNF-dim:
  assumes P' ∈ carrier-mat m m ∧ A' ∈ carrier-mat m n ∧ Q' ∈ carrier-mat n n
  and diagonal-to-Smith-aux-PQ-JNF xs bezout (P', A', Q') = (P, A, Q)
  shows P ∈ carrier-mat m m ∧ A ∈ carrier-mat m n ∧ Q ∈ carrier-mat n n
  ⟨proof⟩

```

```

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
  ⟨proof⟩

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⟩

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 ∈ carrier-mat CARD('nr::mod-type) CARD('nc::mod-type)
  and S: S ∈ carrier-mat CARD('nr) CARD('nc)
  and P: P ∈ carrier-mat CARD('nr) CARD('nr)
  and Q: Q ∈ carrier-mat CARD('nc) CARD('nc)
  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
  ⟨proof⟩ 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
  ⟨proof⟩

```

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
⟨proof⟩ lemma diagonal-to-Smith-PQ-JNF-internalized:
```

```

class.nontriv TYPE('c::type) ==>
class.nontriv TYPE('a::type) ==>
A ∈ carrier-mat CARD('a) CARD('c) ==>
S ∈ carrier-mat CARD('a) CARD('c) ==>
P ∈ carrier-mat CARD('a) CARD('a) ==>
Q ∈ carrier-mat CARD('c) CARD('c) ==>
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
⟨proof⟩
```

context

```

fixes m::nat and n::nat
assumes local-typedef1: ∃(Rep :: ('b ⇒ int)) Abs. type-definition Rep Abs {0..<m
:: int}
assumes local-typedef2: ∃(Rep :: ('c ⇒ 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=CARD('b) (is ?b)
⟨proof⟩
```

lemma *type-to-set2*:

```

shows class.nontriv TYPE('c) (is ?a) and n=CARD('c) (is ?b)
⟨proof⟩
```

lemma *diagonal-to-Smith-PQ-JNF-local-typedef*:

```

assumes A: isDiagonal-mat A and ib: is-bezout-ext bezout
and A-dim: A ∈ carrier-mat m n
assumes PSQ: (P,S,Q) = diagonal-to-Smith-PQ-JNF A bezout
```

```

shows  $S = P * A * Q \wedge \text{invertible-mat } P \wedge \text{invertible-mat } Q \wedge \text{Smith-normal-form-mat}$ 
 $S$ 
 $\wedge P \in \text{carrier-mat } m \ m \wedge S \in \text{carrier-mat } m \ n \wedge Q \in \text{carrier-mat } n \ n$ 
 $\langle \text{proof} \rangle$ 
end
end

context
begin
private lemma diagonal-to-Smith-PQ-JNF-canceled-first:
 $\exists \text{Rep Abs. type-definition Rep Abs } \{0..<\text{int } n\} \implies \{0..<\text{int } m\} \neq \{\} \implies$ 
 $1 < m \implies 1 < n \implies \text{isDiagonal-mat } A \implies \text{is-bezout-ext bezout} \implies$ 
 $A \in \text{carrier-mat } m \ n \implies (P, S, Q) = \text{diagonal-to-Smith-PQ-JNF } A \text{ bezout} \implies$ 
 $S = P * A * Q \wedge \text{invertible-mat } P \wedge \text{invertible-mat } Q \wedge \text{Smith-normal-form-mat}$ 
 $S$ 
 $\wedge P \in \text{carrier-mat } m \ m \wedge S \in \text{carrier-mat } m \ n \wedge Q \in \text{carrier-mat } n \ n$ 
 $\langle \text{proof} \rangle$  lemma diagonal-to-Smith-PQ-JNF-canceled-both:
 $\{0..<\text{int } n\} \neq \{\} \implies \{0..<\text{int } m\} \neq \{\} \implies 1 < m \implies 1 < n \implies$ 
 $\text{isDiagonal-mat } A \implies \text{is-bezout-ext bezout} \implies A \in \text{carrier-mat } m \ n \implies$ 
 $(P, S, Q) = \text{diagonal-to-Smith-PQ-JNF } A \text{ bezout} \implies S = P * A * Q \wedge$ 
 $\text{invertible-mat } P \wedge \text{invertible-mat } Q \wedge \text{Smith-normal-form-mat } S$ 
 $\wedge P \in \text{carrier-mat } m \ m \wedge S \in \text{carrier-mat } m \ n \wedge Q \in \text{carrier-mat } n \ n$ 
 $\langle \text{proof} \rangle$ 

```

14.4 The final result

```

lemma diagonal-to-Smith-PQ-JNF:
assumes  $A: \text{isDiagonal-mat } A$  and  $ib: \text{is-bezout-ext bezout}$ 
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  $S = P * A * Q \wedge \text{invertible-mat } P \wedge \text{invertible-mat } Q \wedge \text{Smith-normal-form-mat}$ 
 $S$ 
 $\wedge P \in \text{carrier-mat } m \ m \wedge S \in \text{carrier-mat } m \ n \wedge Q \in \text{carrier-mat } n \ n$ 
 $\langle \text{proof} \rangle$ 
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:  $A \in \text{carrier-mat } m \ n$ 
  and n:  $1 < n$  and m:  $1 < m$ 
  and PSQ: Smith-normal-form-of-JNF A bezout = (P,S,Q)
  shows  $S = P * A * Q \wedge \text{invertible-mat } P \wedge \text{invertible-mat } Q \wedge \text{Smith-normal-form-mat } S$ 
   $\wedge P \in \text{carrier-mat } m \ m \wedge S \in \text{carrier-mat } m \ n \wedge Q \in \text{carrier-mat } n \ n$ 
  ⟨proof⟩

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) =>
   $P \in \text{carrier-mat} (\dim\text{-row } A) (\dim\text{-row } A) \wedge$ 
   $Q \in \text{carrier-mat} (\dim\text{-col } A) (\dim\text{-col } A)$ 
   $\wedge \text{invertible-mat } P \wedge \text{invertible-mat } Q$ 
   $\wedge \text{Smith-normal-form-mat } S \wedge S = P * A * Q)$ 

```

```

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) $\langle proof \rangle$

lemma *Smith-1xn-two-matrices*:
fixes $A :: 'a::comm-ring-1 mat$
assumes $A : carrier\text{-}mat 1 n$
and $PSQ : (P, S, Q) = (Smith\text{-}1xn A)$
and *is-SNF*: *is-SNF* A (*Smith-1xn* A)
shows $\exists Smith\text{-}1xn'. is\text{-}SNF A (1_m 1, (Smith\text{-}1xn' A))$
 $\langle proof \rangle$

lemma *Smith-1xn-two-matrices-all*:
assumes *is-SNF*: $\forall (A :: 'a::comm-ring-1 mat) \in carrier\text{-}mat 1 n. is\text{-}SNF A (Smith\text{-}1xn A)$
shows $\exists Smith\text{-}1xn'. \forall (A :: 'a::comm-ring-1 mat) \in carrier\text{-}mat 1 n. is\text{-}SNF A (1_m 1, (Smith\text{-}1xn' A))$
 $\langle proof \rangle$

16.2 Previous operations

context
assumes *SORT-CONSTRAINT* (' $a::comm-ring-1$)
begin

definition *is-div-op* :: (' $a \Rightarrow 'a \Rightarrow 'a$) \Rightarrow bool
where *is-div-op* $div\text{-}op = (\forall a b. b \text{ dvd } a \longrightarrow div\text{-}op a b * b = a)$

lemma *div-op-SOME*: *is-div-op* $(\lambda a b. (SOME k. k * b = a))$
 $\langle proof \rangle$

fun *reduce-column-aux* :: (' $a \Rightarrow 'a \Rightarrow 'a$) \Rightarrow nat list \Rightarrow ' a mat \Rightarrow (' a mat \times ' a mat)
 \Rightarrow (' a mat \times ' a mat)
where *reduce-column-aux* $div\text{-}op [] H (P, K) = (P, K)$
 $| reduce\text{-}column\text{-}aux div\text{-}op (i \# xs) H (P, K) = ($
 — Reduce the i -th row
 $let k = div\text{-}op (H \$\$ (i, 0)) (H \$\$ (0, 0));$
 $P' = addrow\text{-}mat (dim\text{-}row H) (-k) i 0;$
 $K' = addrow (-k) i 0 K$
 $in reduce\text{-}column\text{-}aux div\text{-}op xs H (P' * P, K')$
 $)$

definition *reduce-column div-op* $H = reduce\text{-}column\text{-}aux div\text{-}op [2.. < dim\text{-}row H]$
 $H (1_m (dim\text{-}row H), H)$

lemma *reduce-column-aux*:

assumes $H: H \in \text{carrier-mat } m \ n$

and $P\text{-init}: P\text{-init} \in \text{carrier-mat } m \ m$

and $K\text{-init}: K\text{-init} \in \text{carrier-mat } m \ n$

and $P\text{-init}\text{-}H\text{-}K\text{-init}: P\text{-init} * H = K\text{-init}$

and $PK\text{-}H: (P, K) = \text{reduce-column-aux div-op } xs \ H \ (P\text{-init}, K\text{-init})$

and $m: 0 < m$

and $\text{inv-}P: \text{invertible-mat } P\text{-init}$

and $xs: 0 \notin \text{set } xs$

shows $P \in \text{carrier-mat } m \ m \wedge K \in \text{carrier-mat } m \ n \wedge P * H = K \wedge \text{invertible-mat } P$

$\langle \text{proof} \rangle$

lemma *reduce-column-aux-preserves*:

assumes $H: H \in \text{carrier-mat } m \ n$

and $P\text{-init}: P\text{-init} \in \text{carrier-mat } m \ m$

and $K\text{-init}: K\text{-init} \in \text{carrier-mat } m \ n$

and $P\text{-init}\text{-}H\text{-}K\text{-init}: P\text{-init} * H = K\text{-init}$

and $PK\text{-}H: (P, K) = \text{reduce-column-aux div-op } xs \ H \ (P\text{-init}, K\text{-init})$

and $m: 0 < m$

and $\text{inv-}P: \text{invertible-mat } P\text{-init}$

and $xs: 0 \notin \text{set } xs \text{ and } i: i \notin \text{set } xs \text{ and } im: i < m$

shows $\text{Matrix.row } K \ i = \text{Matrix.row } K\text{-init} \ i$

$\langle \text{proof} \rangle$

lemma *reduce-column-aux-index'*:

assumes $H: H \in \text{carrier-mat } m \ n$

and $P\text{-init}: P\text{-init} \in \text{carrier-mat } m \ m$

and $K\text{-init}: K\text{-init} \in \text{carrier-mat } m \ n$

and $P\text{-init}\text{-}H\text{-}K\text{-init}: P\text{-init} * H = K\text{-init}$

and $PK\text{-}H: (P, K) = \text{reduce-column-aux div-op } xs \ H \ (P\text{-init}, K\text{-init})$

and $m: 0 < m$

and $\text{inv-}P: \text{invertible-mat } P\text{-init}$

and $xs: 0 \notin \text{set } xs$

and $\forall x \in \text{set } xs. \ x < m$

and $\text{distinct } xs$

shows $(\forall i \in \text{set } xs. \ \text{Matrix.row } K \ i =$

$\text{Matrix.row} (\text{addrow} (-(\text{div-op} (H \$\$ (i, 0)) (H \$\$ (0, 0)))) \ i \ 0 \ K\text{-init}) \ i)$

$\langle \text{proof} \rangle$

corollary *reduce-column-aux-index*:

assumes $H: H \in \text{carrier-mat } m \ n$

and $P\text{-init}: P\text{-init} \in \text{carrier-mat } m \ m$

and $K\text{-init}: K\text{-init} \in \text{carrier-mat } m \ n$

and $P\text{-init}\text{-}H\text{-}K\text{-init}: P\text{-init} * H = K\text{-init}$

and $PK\text{-}H: (P, K) = \text{reduce-column-aux div-op } xs \ H \ (P\text{-init}, K\text{-init})$

and $m: 0 < m$

and $\text{inv-}P: \text{invertible-mat } P\text{-init}$

and $xs: 0 \notin set xs$
and $\forall x \in set xs. x < m$
and $distinct xs$
and $i \in set xs$
shows $Matrix.row K i =$
 $Matrix.row (addrow (-(div-op (H \$\$ (i, 0)) (H \$\$ (0, 0)))) i 0 K\text{-init}) i$
 $\langle proof \rangle$

corollary *reduce-column-aux-works*:
assumes $H: H \in carrier\text{-}mat m n$
and $PK\text{-}H: (P, K) = reduce\text{-}column\text{-}aux div\text{-}op xs H (1_m (dim\text{-}row H), H)$
and $m: 0 < m$
and $xs: 0 \notin set xs$
and $xm: \forall x \in set xs. x < m$
and $d\text{-}xs: distinct xs$
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 $id: is\text{-}div\text{-}op div\text{-}op$
shows $K \$\$ (i, 0) = 0$ **and** $K\$$(i, j) = H \$\$ (i, j)$
 $\langle proof \rangle$

lemma *reduce-column*:
assumes $H: H \in carrier\text{-}mat m n$
and $PK\text{-}H: (P, K) = reduce\text{-}column div\text{-}op H$
and $m: 0 < m$
shows $P \in carrier\text{-}mat m m \wedge K \in carrier\text{-}mat m n \wedge P * H = K \wedge invertible\text{-}mat P$
 $\langle proof \rangle$

lemma *reduce-column-preserves*:
assumes $H: H \in carrier\text{-}mat m n$
and $PK\text{-}H: (P, K) = reduce\text{-}column div\text{-}op H$
and $m: 0 < m$
and $i \in \{0, 1\}$
and $i < m$
shows $Matrix.row K i = Matrix.row H i$
 $\langle proof \rangle$

lemma *reduce-column-preserves2*:
assumes $H: H \in carrier\text{-}mat m n$
and $PK\text{-}H: (P, K) = reduce\text{-}column div\text{-}op H$
and $m: 0 < m$ **and** $i: i \in \{0, 1\}$ **and** $im: i < m$ **and** $j: j < n$
shows $K \$\$ (i, j) = H \$\$ (i, j)$
 $\langle proof \rangle$

```

corollary reduce-column-works:
  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 dvd:  $H \$\$ (0, 0) \text{ dvd } H \$\$ (i, 0)$ 
  and j0:  $\forall j \in \{1..n\}. H \$\$ (0, j) = 0$ 
  and j1n:  $j \in \{1..n\}$ 
  and n:  $0 < n$ 
  and i<m:  $i \in \{2..m\}$ 
  and id:  $\text{is-div-op div-op}$ 
shows  $K \$\$ (i, 0) = 0$  and  $K \$\$ (i, j) = H \$\$ (i, j)$ 
  ⟨proof⟩

```

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 1×2 matrix into its Smith normal form
2. an operation to transform any 2×2 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 2×1 matrices via the 1×2 operation.

```

locale Smith-Impl =
  fixes Smith-1x2 ::  $('a::\text{comm-ring-1}) \text{ mat} \Rightarrow ('a \text{ mat} \times 'a \text{ mat})$ 
  and Smith-2x2 ::  $'a \text{ mat} \Rightarrow ('a \text{ mat} \times 'a \text{ mat} \times 'a \text{ mat})$ 
  and div-op ::  $'a \Rightarrow 'a \Rightarrow 'a$ 
  assumes SNF-1x2-works:  $\forall (A::'a \text{ mat}) \in \text{carrier-mat } 1 \ 2. \text{ is-SNF } A \ (1_m \ 1, (Smith-1x2 \ A))$ 
  and SNF-2x2-works:  $\forall (A::'a \text{ mat}) \in \text{carrier-mat } 2 \ 2. \text{ is-SNF } A \ (Smith-2x2 \ A)$ 
  and id:  $\text{is-div-op div-op}$ 
begin

```

From a 2×2 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) = ( $\text{Matrix.mat } n \ n \ (\lambda(i, j). \text{ if } i = 0 \wedge j = 0 \text{ then } B \$\$ (0, 0) \text{ else }$ 
 $\text{ if } i = 0 \wedge j = k \text{ then } B \$\$ (0, 1) \text{ else if } i = k \wedge j = 0 \text{ then } B \$\$ (1, 0) \text{ else if } i = k \wedge j = k \text{ then } B \$\$ (1, 1)$ )

```

```

else if  $i=j$  then 1 else 0))

lemma make-mat-carrier[simp]:
  shows make-mat  $n k B \in carrier\text{-}mat n n$ 
  ⟨proof⟩

lemma upper-triangular-mat-delete-make-mat:
  shows upper-triangular (mat-delete (make-mat  $n k B$ ) 0 0)
  ⟨proof⟩

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⟩

corollary det-mat-delete-make-mat:
  assumes  $kn: k < n$ 
  shows Determinant.det (mat-delete (mat-delete (make-mat  $n k B$ ) 0  $k$ ) ( $k - 1$ ) 0) = 1
  ⟨proof⟩

lemma swaprows-make-mat:
  assumes  $B: B \in carrier\text{-}mat 2 2$  and  $k0: k \neq 0$  and  $k: k < n$ 
  shows swaprows  $k 0$  (make-mat  $n k B$ ) = make-mat  $n k$  (swaprows 1 0  $B$ ) (is
  ?lhs = ?rhs)
  ⟨proof⟩

lemma cofactor-make-mat-00:
  assumes  $k: k < n$  and  $k0: k \neq 0$ 
  shows cofactor (make-mat  $n k B$ ) 0 0 =  $B \$\$ (1,1)$ 
  ⟨proof⟩

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⟩

lemma invertible-make-mat:
  assumes  $inv\text{-}B: invertible\text{-}mat B$  and  $B: B \in carrier\text{-}mat 2 2$ 
  and  $kn: k < n$  and  $k0: k \neq 0$ 
  shows invertible-mat (make-mat  $n k B$ )
  ⟨proof⟩

lemma make-mat-index:

```

assumes $i: i < n$ **and** $j: j < n$
shows $\text{make-mat } n \ k \ B \ \$\$ (i,j) = (\text{if } i = 0 \ \wedge \ j = 0 \ \text{then } B\$(0,0) \ \text{else}$
 $\quad \text{if } i = 0 \ \wedge \ j = k \ \text{then } B\$(0,1) \ \text{else if } i=k \ \wedge \ j = 0$
 $\quad \text{then } B\$(1,0) \ \text{else if } i=k \ \wedge \ j=k \ \text{then } B\$(1,1)$
 $\quad \text{else if } i=j \ \text{then } 1 \ \text{else } 0)$
 $\langle \text{proof} \rangle$

lemma $\text{make-mat-works}:$

assumes $A: A \in \text{carrier-mat } m \ n$ **and** $\text{Suc-}i\text{-less-}n: \text{Suc } i < n$
and $Q\text{-step-def: } Q\text{-step} = (\text{make-mat } n \ (\text{Suc } i) \ (\text{snd } (\text{Smith-}1x2$
 $\quad (\text{Matrix.mat } 1 \ 2 \ (\lambda(a,b). \ \text{if } b = 0 \ \text{then } A \ \$\$ (0,0) \ \text{else } A \ \$\$ (0,\text{Suc } i))))))$
shows $A \ \$\$ (0,0) * Q\text{-step} \$\$ (0,(\text{Suc } i)) + A \ \$\$ (0, \text{Suc } i) * Q\text{-step} \$\$ (\text{Suc } i,$
 $\text{Suc } i) = 0$
 $\langle \text{proof} \rangle$

16.3.1 Case $1 \times n$

fun $\text{Smith-}1xn\text{-aux} :: \text{nat} \Rightarrow 'a \text{ mat} \Rightarrow ('a \text{ mat} \times 'a \text{ mat}) \Rightarrow ('a \text{ mat} \times 'a \text{ mat})$
where

$\text{Smith-}1xn\text{-aux } 0 \ A \ (S,Q) = (S,Q) \mid$
 $\text{Smith-}1xn\text{-aux } (\text{Suc } i) \ A \ (S,Q) = (\text{let}$
 $\quad A\text{-step-}1x2 = (\text{Matrix.mat } 1 \ 2 \ (\lambda(a,b). \ \text{if } b = 0 \ \text{then } S \ \$\$ (0,0) \ \text{else } S$
 $\quad \$\$ (0,\text{Suc } i)));$
 $\quad (S\text{-step-}1x2, Q\text{-step-}1x2) = \text{Smith-}1x2 \ A\text{-step-}1x2;$
 $\quad Q\text{-step} = \text{make-mat } (\text{dim-col } A) \ (\text{Suc } i) \ Q\text{-step-}1x2;$
 $\quad S' = S * Q\text{-step}$
 $\quad \text{in } \text{Smith-}1xn\text{-aux } i \ A \ (S', Q * Q\text{-step}))$

definition $\text{Smith-}1xn \ A = (\text{if } \text{dim-col } A = 0 \ \text{then } (A, 1_m \ (\text{dim-col } A))$
 $\quad \text{else } \text{Smith-}1xn\text{-aux } (\text{dim-col } A - 1) \ A \ (A, 1_m \ (\text{dim-col } A)))$

lemma $\text{Smith-}1xn\text{-aux-}Q\text{-carrier}:$

assumes $r: (S', Q') = (\text{Smith-}1xn\text{-aux } i \ A \ (S, Q))$
assumes $A: A \in \text{carrier-mat } 1 \ n$ **and** $Q: Q \in \text{carrier-mat } n \ n$
shows $Q' \in \text{carrier-mat } n \ n$
 $\langle \text{proof} \rangle$

lemma $\text{Smith-}1xn\text{-aux-invertible-}Q:$

assumes $r: (S', Q') = (\text{Smith-}1xn\text{-aux } i \ A \ (S, Q))$
assumes $A: A \in \text{carrier-mat } 1 \ n$ **and** $Q: Q \in \text{carrier-mat } n \ n$
and $i: i < n$ **and** $\text{inv-}Q: \text{invertible-mat } Q$
shows $\text{invertible-mat } Q'$
 $\langle \text{proof} \rangle$

lemma $\text{Smith-}1xn\text{-aux-}S'\text{-}AQ':$

assumes $r: (S', Q') = (\text{Smith-}1xn\text{-aux } i \ A \ (S, Q))$
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 $S \cdot A \cdot Q : S = A * Q$ **and** $i : i < n$
shows $S' = A * Q'$
 $\langle proof \rangle$

lemma *Smith-1xn-aux-S'-works:*

assumes $r : (S', Q') = (\text{Smith-1xn-aux } i \text{ } A \text{ } (S, Q))$
assumes $A : A \in \text{carrier-mat } 1 \text{ } n$ **and** $S : S \in \text{carrier-mat } 1 \text{ } n$ **and** $Q : Q \in \text{carrier-mat } n \text{ } n$
and $S \cdot A \cdot Q : S = A * Q$ **and** $i : i < n$ **and** $j_0 : 0 < j$ **and** $j_n : j < n$
and *all-j-zero*: $\forall j \in \{i+1..n\}. S \text{ } \$\$ (0, j) = 0$
shows $S' \text{ } \$\$ (0, j) = 0$
 $\langle proof \rangle$

lemma *Smith-1xn-works:*

assumes $A : A \in \text{carrier-mat } 1 \text{ } n$
and $SQ : (S, Q) = \text{Smith-1xn } A$
shows *is-SNF A* ($1_m \text{ } 1$, S, Q)
 $\langle proof \rangle$

16.3.2 Case $n \times 1$

definition *Smith-nx1 A* =
 $(\text{let } (S, P) = (\text{Smith-1xn-aux } (\text{dim-row } A - 1) \text{ } (\text{transpose-mat } A) \text{ } (\text{transpose-mat } A, 1_m \text{ } (\text{dim-row } A)))$
 $\text{in } (\text{transpose-mat } P, \text{ transpose-mat } S))$

lemma *Smith-nx1-works:*

assumes $A : A \in \text{carrier-mat } n \text{ } 1$
and $SQ : (P, S) = \text{Smith-nx1 } A$
shows *is-SNF A* ($P, S, 1_m \text{ } 1$)
 $\langle proof \rangle$

16.3.3 Case $2 \times n$

function *Smith-2xn* :: ' a mat \Rightarrow (' a mat \times ' a mat \times ' a mat)
where
 $\text{Smith-2xn } A =$
if dim-col A = 0 then ($1_m \text{ } (\text{dim-row } A), A, 1_m \text{ } 0$) *else*
if dim-col A = 1 then let (P, S) = *Smith-nx1 A* *in* ($P, S, 1_m \text{ } (\text{dim-col } A)$) *else*
if dim-col A = 2 then *Smith-2x2 A*
else
let $A1 = \text{mat-of-cols } (\text{dim-row } A) [\text{col } A \text{ } 0];$
 $A2 = \text{mat-of-cols } (\text{dim-row } A) [\text{col } A \text{ } i. i \leftarrow [1..<\text{dim-col } A]];$
 $(P1, D1, Q1) = \text{Smith-2xn } A2;$
 $C = (P1 * A1) @_c (P1 * A2 * Q1);$
 $D = \text{mat-of-cols } (\text{dim-row } A) [\text{col } C \text{ } 0, \text{ col } C \text{ } 1];$
 $E = \text{mat-of-cols } (\text{dim-row } A) [\text{col } C \text{ } i. i \leftarrow [2..<\text{dim-col } A]];$
 $(P2, D2, Q2) = \text{Smith-2x2 } D;$

```

 $H = (P2*D*Q2) @_c (P2 * E);$ 
 $k = (\text{div-op } (H \$\$ (0,2)) (H \$\$ (0,0)));$ 
 $H2 = \text{addcol } (-k) 2 0 H;$ 
 $(-, -, H2-DR) = \text{split-block } H2 1 1;$ 
 $(H-1xn, Q3) = \text{Smith-1xn } H2-DR;$ 
 $S = \text{four-block-mat } (\text{Matrix.mat } 1 1 (\lambda(a,b). H\$\$(0,0))) (0_m 1 (\text{dim-col } A - 1)) (0_m 1 1) H-1xn;$ 
 $Q1' = \text{four-block-mat } (1_m 1) (0_m 1 (\text{dim-col } A - 1)) (0_m (\text{dim-col } A - 1) 1) Q1;$ 
 $Q2' = \text{four-block-mat } Q2 (0_m 2 (\text{dim-col } A - 2)) (0_m (\text{dim-col } A - 2) 2) (1_m (\text{dim-col } A - 2));$ 
 $Q\text{-div-}k = \text{addrrow-mat } (\text{dim-col } A) (-k) 0 2;$ 
 $Q3' = \text{four-block-mat } (1_m 1) (0_m 1 (\text{dim-col } A - 1)) (0_m (\text{dim-col } A - 1) 1) Q3$ 
 $\quad \text{in } (P2 * P1, S, Q1' * Q2' * Q\text{-div-}k * Q3')$ 
 $\langle \text{proof} \rangle$ 

```

termination $\langle \text{proof} \rangle$

lemma *Smith-2xn-0*:

assumes $A: A \in \text{carrier-mat } 2 0$

shows *is-SNF A (Smith-2xn A)*

 $\langle \text{proof} \rangle$

lemma *Smith-2xn-1*:

assumes $A: A \in \text{carrier-mat } 2 1$

shows *is-SNF A (Smith-2xn A)*

 $\langle \text{proof} \rangle$

lemma *Smith-2xn-2*:

assumes $A: A \in \text{carrier-mat } 2 2$

shows *is-SNF A (Smith-2xn A)*

 $\langle \text{proof} \rangle$

lemma *is-SNF-Smith-2xn-n-ge-2*:

assumes $A: A \in \text{carrier-mat } 2 n \text{ and } n: n > 2$

shows *is-SNF A (Smith-2xn A)*

 $\langle \text{proof} \rangle$

lemma *is-SNF-Smith-2xn*:

assumes $A: A \in \text{carrier-mat } 2 n$

shows *is-SNF A (Smith-2xn A)*

 $\langle \text{proof} \rangle$

16.3.4 Case $n \times 2$

definition *Smith-nx2 A = (let (P,S,Q) = 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 ( $\text{Smith-nx2 } A$ )
  ⟨proof⟩

```

16.3.5 Case $m \times n$

```
declare Smith-2xn.simps[simp del]
```

```

function (domintros) Smith-mxn :: ' $a \text{ mat} \Rightarrow ('a \text{ mat} \times 'a \text{ mat} \times 'a \text{ mat})$ '
  where
    Smith-mxn A = (
      if dim-row A = 0 ∨ dim-col A = 0 then (1m (dim-row A), A, 1m (dim-col A))
      else if dim-row A = 1 then (1m 1, Smith-1xn A)
      else if dim-row A = 2 then Smith-2xn A
      else if dim-col A = 1 then let (P,S) = Smith-nx1 A in (P,S,1m 1)
      else if dim-col A = 2 then Smith-nx2 A
      else
        let A1 = mat-of-row (Matrix.row A 0);
        A2 = mat-of-rows (dim-col A) [Matrix.row A i. i ← [1..<dim-row A]];
        (P1,D1,Q1) = Smith-mxn A2;
        C = (A1 * Q1) @r (P1 * A2 * Q1);
        D = mat-of-rows (dim-col A) [Matrix.row C 0, Matrix.row C 1];
        E = mat-of-rows (dim-col A) [Matrix.row C i. i ← [2..<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 = four-block-mat (Matrix.mat 1 1 (λ(a, b). H $$ (0, 0))) (0m 1 (dim-col A - 1)) (0m (dim-row A - 1) 1) S';
        P1' = four-block-mat (1m 1) (0m 1 (dim-row A - 1)) (0m (dim-row A - 1) 1) P1;
        P2' = four-block-mat P2 (0m 2 (dim-row A - 2)) (0m (dim-row A - 2) 2) (1m (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')
    )
  ⟨proof⟩

```

```
declare Smith-2xn.simps[simp]
```

lemma *Smith-mxn-dom-nm-less-2*:
assumes $A: A \in \text{carrier-mat } m \ n$ **and** $mn: n \leq 2 \vee m \leq 2$
shows *Smith-mxn-dom A*
(proof)

lemma *Smith-mxn-pinduct-carrier-less-2*:
assumes $A: A \in \text{carrier-mat } m \ n$ **and** $mn: n \leq 2 \vee m \leq 2$
shows $\text{fst}(\text{Smith-mxn } A) \in \text{carrier-mat } m \ m$
 $\wedge \text{fst}(\text{snd}(\text{Smith-mxn } A)) \in \text{carrier-mat } m \ n$
 $\wedge \text{snd}(\text{snd}(\text{Smith-mxn } A)) \in \text{carrier-mat } n \ n$
(proof)

lemma *Smith-mxn-pinduct-carrier-ge-2*: $\llbracket \text{Smith-mxn-dom } A; A \in \text{carrier-mat } m \ n; m > 2; n > 2 \rrbracket \implies$
 $\text{fst}(\text{Smith-mxn } A) \in \text{carrier-mat } m \ m$
 $\wedge \text{fst}(\text{snd}(\text{Smith-mxn } A)) \in \text{carrier-mat } m \ n$
 $\wedge \text{snd}(\text{snd}(\text{Smith-mxn } A)) \in \text{carrier-mat } n \ n$
(proof)

corollary *Smith-mxn-pinduct-carrier*: $\llbracket \text{Smith-mxn-dom } A; A \in \text{carrier-mat } m \ n \rrbracket \implies$
 $\text{fst}(\text{Smith-mxn } A) \in \text{carrier-mat } m \ m$
 $\wedge \text{fst}(\text{snd}(\text{Smith-mxn } A)) \in \text{carrier-mat } m \ n$
 $\wedge \text{snd}(\text{snd}(\text{Smith-mxn } A)) \in \text{carrier-mat } n \ n$
(proof)

termination *(proof)*

lemma *is-SNF-Smith-mxn-less-2*:
assumes $A: A \in \text{carrier-mat } m \ n$ **and** $mn: n \leq 2 \vee m \leq 2$
shows *is-SNF A (Smith-mxn A)*
(proof)

lemma *is-SNF-Smith-mxn-ge-2*:
assumes $A: A \in \text{carrier-mat } m \ n$ **and** $m: m > 2$ **and** $n: n > 2$
shows *is-SNF A (Smith-mxn A)*
(proof)

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)*
(proof)

```

declare Smith-mxn.simps[code]
end

declare Smith-Impl.Smith-mxn.simps[code-unfold]

definition T-spec :: ('a:{comm-ring-1}  $\Rightarrow$  'a  $\Rightarrow$  ('a  $\times$  'a  $\times$  'a))  $\Rightarrow$  bool
where T-spec T = ( $\forall$  a b:'a. let (a1,b1,d) = T a b in
           a = a1*d  $\wedge$  b = b1*d  $\wedge$  ideal-generated {a1,b1} = ideal-generated
           {1})

definition D'-spec :: ('a:{comm-ring-1}  $\Rightarrow$  'a  $\Rightarrow$  'a  $\Rightarrow$  ('a  $\times$  'a))  $\Rightarrow$  bool
where D'-spec D' = ( $\forall$  a b c:'a. let (p,q) = D' a b c in
           ideal-generated{a,b,c} = ideal-generated{1}
            $\longrightarrow$  ideal-generated {p*a,p*b+q*c} = 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 Anaylsis

```

definition Smith-mxn-HMA :: (('a:{comm-ring-1}^2)  $\Rightarrow$  (('a^2)  $\times$  ('a^2^2)))  $\Rightarrow$ 
   $\Rightarrow$  (('a^2^2)  $\Rightarrow$  (('a^2^2)  $\times$  ('a^2^2)  $\times$  ('a^2^2)))  $\Rightarrow$  ('a  $\Rightarrow$  'a  $\Rightarrow$  'a)  $\Rightarrow$  ('a'^n::mod-type ^'m::mod-type)
   $\Rightarrow$  (('a'^m::mod-type ^'m::mod-type)  $\times$  ('a'^n::mod-type ^'m::mod-type)  $\times$  ('a'^n::mod-type ^'n::mod-type))
where
  Smith-mxn-HMA Smith-1x2 Smith-2x2 div-op A =
  (let Smith-1x2-JNF = ( $\lambda$ A'. 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'));
  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'));
  (P,S,Q) = Smith-Impl.Smith-mxn Smith-1x2-JNF Smith-2x2-JNF div-op
  (Mod-Type-Connect.from-hmam A)
  in (Mod-Type-Connect.to-hmam P, Mod-Type-Connect.to-hmam S, Mod-Type-Connect.to-hmam
  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 Anaylsis

```

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⟩
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::comm-ring-1 mat. U ∈ carrier-mat (dim-col A) (dim-col A)
  & invertible-mat U & lower-triangular (A * U))

```

```

class Hermite-ring =
  assumes ∀ (A::'a::comm-ring-1 mat). admits-triangular-reduction A

```

```

lemma admits-triangular-reduction-intro:
  assumes invertible-mat (U::'a::comm-ring-1 mat)
  and U ∈ carrier-mat (dim-col A) (dim-col A)
  and lower-triangular (A * U)

```

shows admits-triangular-reduction A
 $\langle proof \rangle$

lemma OFCLASS-Hermite-ring-def:
 $OFCLASS('a::comm-ring-1, Hermite-ring-class)$
 $\equiv (\bigwedge (A:'a::comm-ring-1 mat). admits-triangular-reduction A)$
 $\langle proof \rangle$

definition admits-diagonal-reduction::'a::comm-ring-1 mat \Rightarrow bool
where admits-diagonal-reduction $A = (\exists P Q. P \in carrier\text{-}mat (dim\text{-}row A) \wedge (dim\text{-}row A) \wedge Q \in carrier\text{-}mat (dim\text{-}col A) (dim\text{-}col A) \wedge invertible\text{-}mat P \wedge invertible\text{-}mat Q \wedge Smith\text{-}normal\text{-}form\text{-}mat (P * A * Q))$

lemma admits-diagonal-reduction-intro:
assumes $P \in carrier\text{-}mat (dim\text{-}row A) (dim\text{-}row A)$
and $Q \in carrier\text{-}mat (dim\text{-}col A) (dim\text{-}col A)$
and invertible-mat P **and** invertible-mat Q
and Smith-normal-form-mat $(P * A * Q)$
shows admits-diagonal-reduction A $\langle proof \rangle$

lemma admits-diagonal-reduction-imp-exists-algorithm-is-SNF:
assumes $A \in carrier\text{-}mat m n$
and admits-diagonal-reduction A
shows $\exists algorithm. is\text{-}SNF A (algorithm A)$
 $\langle proof \rangle$

lemma exists-algorithm-is-SNF-imp-admits-diagonal-reduction:
assumes $A \in carrier\text{-}mat m n$
and $\exists algorithm. is\text{-}SNF A (algorithm A)$
shows admits-diagonal-reduction A
 $\langle proof \rangle$

lemma admits-diagonal-reduction-eq-exists-algorithm-is-SNF:
assumes $A: A \in carrier\text{-}mat m n$
shows admits-diagonal-reduction $A = (\exists algorithm. is\text{-}SNF A (algorithm A))$
 $\langle proof \rangle$

lemma admits-diagonal-reduction-imp-exists-algorithm-is-SNF-all:
assumes $(\forall (A:'a::comm-ring-1 mat) \in carrier\text{-}mat m n. admits\text{-}diagonal\text{-}reduction A)$
shows $(\exists algorithm. \forall (A:'a mat) \in carrier\text{-}mat m n. is\text{-}SNF A (algorithm A))$
 $\langle proof \rangle$

```

lemma exists-algorithm-is-SNF-imp-admits-diagonal-reduction-all:
  assumes ( $\exists$  algorithm.  $\forall (A::'a mat) \in \text{carrier-mat } m n.$  is-SNF A (algorithm A))
  shows ( $\forall (A::'a::\text{comm-ring-1} mat) \in \text{carrier-mat } m n.$  admits-diagonal-reduction
A)
   $\langle proof \rangle$ 

```

```

lemma admits-diagonal-reduction-eq-exists-algorithm-is-SNF-all:
  shows ( $\forall (A::'a::\text{comm-ring-1} mat) \in \text{carrier-mat } m n.$  admits-diagonal-reduction
A)
   $= (\exists \text{algorithm. } \forall (A::'a mat) \in \text{carrier-mat } m n.$  is-SNF A (algorithm A))
   $\langle proof \rangle$ 

```

18.2 The class that represents elementary divisor rings

```

class elementary-divisor-ring =
  assumes  $\forall (A::'a::\text{comm-ring-1} mat).$  admits-diagonal-reduction A

```

```

lemma dim-row-mat-diag[simp]: dim-row (mat-diag n f) = n and
  dim-col-mat-diag[simp]: dim-col (mat-diag n f) = n
   $\langle proof \rangle$ 

```

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: OFCLASS('a::comm-ring-1, Hermite-ring-class)
  shows OFCLASS('a::comm-ring-1, bezout-ring-class)
   $\langle proof \rangle$ 

```

18.4 Elementary divisor ring implies Hermite ring

```

context
  assumes SORT-CONSTRAINT('a::comm-ring-1)
begin

```

```

lemma triangularizable-m0:
  assumes A:  $A \in \text{carrier-mat } m 0$ 
  shows  $\exists U. U \in \text{carrier-mat } 0 0 \wedge \text{invertible-mat } U \wedge \text{lower-triangular } (A * U)$ 
   $\langle proof \rangle$ 

```

```

lemma triangularizable-0n:
  assumes A:  $A \in \text{carrier-mat } 0 n$ 
  shows  $\exists U. U \in \text{carrier-mat } n n \wedge \text{invertible-mat } U \wedge \text{lower-triangular } (A * U)$ 
   $\langle proof \rangle$ 

```

lemma *diagonal-imp-triangular-1x2*:

assumes $A: A \in \text{carrier-mat } 1 \ 2$ **and** $d: \text{admits-diagonal-reduction } (A::'a \text{ mat})$

shows *admits-triangular-reduction A*

$\langle proof \rangle$

lemma *triangular-imp-diagonal-1x2*:

assumes $A: A \in \text{carrier-mat } 1 \ 2$ **and** $t: \text{admits-triangular-reduction } (A::'a \text{ mat})$

shows *admits-diagonal-reduction A*

$\langle proof \rangle$

lemma *triangular-eq-diagonal-1x2*:

$(\forall A \in \text{carrier-mat } 1 \ 2. \text{ admits-triangular-reduction } (A::'a \text{ mat}))$
 $= (\forall A \in \text{carrier-mat } 1 \ 2. \text{ admits-diagonal-reduction } (A::'a \text{ mat}))$

$\langle proof \rangle$

lemma *admits-triangular-mat-1x1*:

assumes $A: A \in \text{carrier-mat } 1 \ 1$

shows *admits-triangular-reduction (A::'a mat)*

$\langle proof \rangle$

lemma *admits-diagonal-mat-1x1*:

assumes $A: A \in \text{carrier-mat } 1 \ 1$

shows *admits-diagonal-reduction (A::'a mat)*

$\langle proof \rangle$

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 $\forall A \in \text{carrier-mat } 1 \ n. \text{ admits-triangular-reduction } (A::'a \text{ mat})$

$\langle proof \rangle$

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 $\forall A. \text{ admits-triangular-reduction } (A::'a \text{ mat})$

$\langle proof \rangle$

corollary *admits-diagonal-imp-admits-triangular'*:

assumes $a: \forall A. \text{ admits-diagonal-reduction } (A::'a \text{ mat})$

shows $\forall A. \text{ admits-triangular-reduction } (A::'a \text{ mat})$

$\langle proof \rangle$

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 $\forall C::'a \text{ mat. admits-triangular-reduction } C$
 $\langle proof \rangle$

lemma *Hermite-ring-OFCLASS*:

assumes $\forall A \in \text{carrier-mat } 1 \text{ 2. admits-triangular-reduction } (A::'a \text{ mat})$
shows *OFCLASS('a, Hermite-ring-class)*
 $\langle proof \rangle$

lemma *Hermite-ring-OFCLASS'*:

assumes $\forall A \in \text{carrier-mat } 1 \text{ 2.admits-diagonal-reduction } (A::'a \text{ mat})$
shows *OFCLASS('a, Hermite-ring-class)*
 $\langle proof \rangle$

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 $\forall A::'a \text{ mat. admits-triangular-reduction } A$
 $\langle proof \rangle$

lemma *theorem3-part2*:

assumes $1: \forall A::'a \text{ mat. admits-triangular-reduction } A$
shows $\forall a b::'a. \exists a1 b1 d. a = a1*d \wedge b = b1*d \wedge \text{ideal-generated } \{a1,b1\} =$
 $\text{ideal-generated } \{1\}$
 $\langle proof \rangle$

theorem *theorem3*:

shows $(\forall A::'a \text{ mat. admits-triangular-reduction } A)$
 $= (\forall a b::'a. \exists a1 b1 d. a = a1*d \wedge b = b1*d \wedge \text{ideal-generated } \{a1,b1\} =$
 $\text{ideal-generated } \{1\})$
 $\langle proof \rangle$

end

context *comm-ring-1*
begin

lemma *lemma4-prev*:

assumes $a: a = a1*d \text{ and } b: b = b1*d$
and $i: \text{ideal-generated } \{a1,b1\} = \text{ideal-generated } \{1\}$
shows $\text{ideal-generated } \{a,b\} = \text{ideal-generated } \{d\}$
 $\langle proof \rangle$

```

lemma lemma4:
  assumes a:  $a = a1 * d$  and b:  $b = b1 * d$ 
  and i1: ideal-generated {a1,b1} = ideal-generated {1}
  and i2: ideal-generated {a,b} = ideal-generated {d'}
  shows  $\exists a1' b1'. a = a1' * d' \wedge b = b1' * d'$ 
         $\wedge$  ideal-generated {a1',b1'} = ideal-generated {1}
  ⟨proof⟩

```

```

lemma corollary5:
  assumes T:  $\forall a b. \exists a1 b1 d. a = a1 * d \wedge b = b1 * d$ 
         $\wedge$  ideal-generated {a1, b1} = ideal-generated {1::'a}
  and i2: ideal-generated {a,b,c} = ideal-generated {d}
  shows  $\exists a1 b1 c1. a = a1 * d \wedge b = b1 * d \wedge c = c1 * d$ 
         $\wedge$  ideal-generated {a1,b1,c1} = ideal-generated {1}
  ⟨proof⟩

```

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)
  ⟨proof⟩

```

```

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⟩

```

```

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)
  ⟨proof⟩

```

18.5 Characterization of Elementary divisor rings

```

lemma necessity-D':
  assumes edr:  $(\forall (A::'a mat). admits-diagonal-reduction A)$ 
  shows  $\forall a b c::'a. \text{ideal-generated } \{a,b,c\} = \text{ideal-generated } \{1\}$ 
         $\longrightarrow (\exists p q. \text{ideal-generated } \{p*a, p*b + q*c\} = \text{ideal-generated } \{1\})$ 

```

$\langle proof \rangle$

lemma necessity:

assumes $(\forall (A::'a mat). admits-diagonal-reduction A)$
shows $(\forall (A::'a mat). admits-triangular-reduction A)$
and $\forall a b c::'a. ideal-generated\{a,b,c\} = ideal-generated\{1\}$
 $\longrightarrow (\exists p q. ideal-generated\{p*a,p*b+q*c\} = ideal-generated\{1\})$
 $\langle proof \rangle$

In the article, the authors change the notation and assume $(a, b, c) = (1)$. However, we have to provide here the complete prove. To do 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*: $(\forall (A::'a mat). admits-triangular-reduction A)$
and $D': \forall a b c::'a. ideal-generated\{a,b,c\} = ideal-generated\{1\}$
 $\longrightarrow (\exists p q. ideal-generated\{p*a,p*b+q*c\} = ideal-generated\{1\})$
shows $(\forall (A::'a mat). admits-diagonal-reduction A)$
 $\langle proof \rangle$

18.6 Final theorem

theorem edr-characterization:

$(\forall (A::'a mat). admits-diagonal-reduction A) = ((\forall (A::'a mat). admits-triangular-reduction A)$
 $\wedge (\forall a b c::'a. ideal-generated\{a,b,c\} = ideal-generated\{1\})$
 $\longrightarrow (\exists p q. ideal-generated\{p*a,p*b+q*c\} = ideal-generated\{1\}))$
 $\langle proof \rangle$

corollary OFCLASS-edr-characterization:

$OFCLASS('a, elementary-divisor-ring-class) \equiv (OFCLASS('a, Hermite-ring-class))$

$\&\&\& (\forall a b c::'a. ideal-generated\{a,b,c\} = ideal-generated\{1\})$
 $\longrightarrow (\exists p q. ideal-generated\{p*a,p*b+q*c\} = ideal-generated\{1\}))$ (**is** $?lhs \equiv ?rhs$)
 $\langle proof \rangle$

corollary edr-characterization-class:

$class.elementary-divisor-ring TYPE('a)$
 $= (class.Hermite-ring TYPE('a))$
 $\wedge (\forall a b c::'a. ideal-generated\{a,b,c\} = ideal-generated\{1\})$
 $\longrightarrow (\exists p q. ideal-generated\{p*a,p*b+q*c\} = ideal-generated\{1\}))$ (**is** $?lhs = (?H \wedge ?D')$)
 $\langle proof \rangle$

```

corollary edr-iff-T-D':
  shows class.elementary-divisor-ring TYPE('a) = (
    ( $\forall a b::'a. \exists a1 b1 d. a = a1*d \wedge b = b1*d \wedge \text{ideal-generated } \{a1, b1\} = \text{ideal-generated } \{1\}$ )
     $\wedge (\forall a b c::'a. \text{ideal-generated}\{a, b, c\} = \text{ideal-generated}\{1\}$ 
       $\longrightarrow (\exists p q. \text{ideal-generated } \{p*a, p*b + q*c\} = \text{ideal-generated } \{1\}))$ 
    ) (is ?lhs = (?T  $\wedge$  ?D'))
   $\langle proof \rangle$ 

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
 $\langle proof \rangle$ 

```

19.1 Previous code equations

```

definition to-hmam-row A i
  = (vec-lambda (λj. A $$ (Mod-Type.to-nat i, Mod-Type.to-nat j)))

```

```

lemma bezout-matrix-row-code [code abstract]:
  vec-nth (to-hmam-row A i) =
    (λj. A $$ (Mod-Type.to-nat i, Mod-Type.to-nat j))
 $\langle proof \rangle$ 

```

```

lemma [code abstract]: vec-nth (Mod-Type-Connect.to-hmam A) = to-hmam-row A
 $\langle proof \rangle$ 

```

19.2 An executable algorithm to transform 2×2 matrices into its Smith normal form in HOL Analysis

subclass (*in euclidean-ring-gcd*) *bezout-ring-div*
(proof)

context

fixes *bezout*::(*'a::euclidean-ring-gcd* \Rightarrow *'a* \Rightarrow (*'a* \times *'a* \times *'a* \times *'a*))

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

begin

lemma *normalize-bezout-gcd*:

assumes *b*: (*p,q,u,v,d*) = *bezout a b*

shows *normalize d* = *gcd a b*

(proof)

end

lemma *bezout-matrix-works-transpose1*:

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

and *a-not-b*: *a* \neq *b*

shows (*A**transpose (bezout-matrix (transpose A) a b i bezout)*) $\$ i \$ a$

$= \text{snd}(\text{snd}(\text{snd}(\text{bezout}(A \$ i \$ a)(A \$ i \$ b))))$

(proof)

lemma *invertible-bezout-matrix-transpose*:

fixes *A*::*'a::{bezout-ring-div}* \sim *cols*::{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 (transpose (bezout-matrix (transpose A) a b i bezout))*

(proof)

function *diagonalize-2x2-aux* :: ((*'a::euclidean-ring-gcd* \wedge *'a* \wedge *'a*) \times (*'a* \wedge *'a*) \times (*'a* \wedge *'a*)) \Rightarrow ((*'a* \wedge *'a*) \times (*'a* \wedge *'a*) \times (*'a* \wedge *'a*))
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 \neq 0 *and 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')
) ⟨proof⟩

```

termination

⟨proof⟩

```

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
⟨proof⟩

```

```

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⟩

```

```

definition diagonalize-2x2-JNF (A::'a::euclidean-ring-gcd mat)
= (let (P, D, Q) = diagonalize-2x2 (Mod-Type-Connect.to-hmam A::'a22) in
(Mod-Type-Connect.from-hmam P, Mod-Type-Connect.from-hmam D, Mod-Type-Connect.from-hmam

```

$Q))$

lemma *diagonalize-2x2-JNF-works*:
 assumes $A: A \in \text{carrier-mat } 2 \ 2$
 and $PDQ: (P, D, Q) = \text{diagonalize-2x2-JNF } A$
 shows $D = P * A * Q \wedge \text{invertible-mat } P \wedge \text{invertible-mat } Q \wedge \text{isDiagonal-mat } D \wedge P \in \text{carrier-mat } 2 \ 2$
 $\wedge Q \in \text{carrier-mat } 2 \ 2 \wedge D \in \text{carrier-mat } 2 \ 2$
 ⟨proof⟩

definition *Smith-2x2-eucl* $A = ($
 let $(P, D, Q) = \text{diagonalize-2x2 } A;$
 $(P', S, Q') = \text{diagonal-to-Smith-PQ } D \text{ euclid-ext2}$
 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⟩

19.3 An executable algorithm to transform 2×2 matrices into its Smith normal form in JNF

definition *Smith-2x2-JNF-eucl* $A = ($
 let $(P, D, Q) = \text{diagonalize-2x2-JNF } A;$
 $(P', S, Q') = \text{diagonal-to-Smith-PQ-JNF } D \text{ euclid-ext2}$
 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⟩

19.4 An executable algorithm to transform 1×2 matrices into its Smith normal form

definition *Smith-1x2-eucl* $(A::'a::\text{euclidean-ring-gcd}^2)^1 = ($
 if $A \$h 0 \$h 0 = 0 \wedge A \$h 0 \$h 1 \neq 0$ **then**
 let $Q = \text{interchange-columns } (\text{Finite-Cartesian-Product.mat } 1) 0 1;$
 $A' = \text{interchange-columns } A 0 1 \text{ in } (A', Q)$
 else
 if $A \$h 0 \$h 0 \neq 0 \wedge A \$h 0 \$h 1 \neq 0$ **then**
 let $\text{bezout-matrix-right} = \text{transpose } (\text{bezout-matrix } (\text{transpose } A) 0 1 0 \text{ eu-}$

```


$$clid-ext2) \\
in (A ** bezout-matrix-right, bezout-matrix-right) \\
else (A, Finite-Cartesian-Product.mat 1) \\
)$$


```

lemma *Smith-1x2-eucl-works*:
assumes $SQ: (S, Q) = \text{Smith-1x2-eucl } A$
shows $S = A ** Q \wedge \text{invertible } Q \wedge S \$h 0 \$h 1 = 0$
(proof)

definition *bezout-matrix-JNF* :: ' $a::\text{comm-ring-1}$ 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
 $\text{bezout-matrix-JNF } A \ a \ b \ j \ \text{bezout} = \text{Matrix.mat} (\dim-row A) (\dim-row A) (\lambda(x,y).$

```

(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* ($A::'a::\text{euclidean-ring-gcd}$ mat) = (
if $A \$\$ (0, 0) = 0 \wedge A \$\$ (0, 1) \neq 0$ *then*
let $Q = \text{swaprows-mat } 2 \ 0 \ 1;$
 $A' = \text{swapcols } 0 \ 1 \ A$
in (A', Q)
else
if $A \$\$ (0, 0) \neq 0 \wedge A \$\$ (0, 1) \neq 0$ *then*
let $\text{bezout-matrix-right} = \text{transpose-mat} (\text{bezout-matrix-JNF} (\text{transpose-mat}$
 $A) 0 \ 1 \ 0 \ \text{euclid-ext2})$
in $(A * \text{bezout-matrix-right}, \text{bezout-matrix-right})$
else $(A, 1_m \ 2)$
)

lemma *Smith-1x2-eucl-JNF-works*:
assumes $A: A \in \text{carrier-mat } 1 \ 2$
and $SQ: (S, Q) = \text{Smith-1x2-eucl-JNF } A$
shows $\text{is-SNF } A (1_m \ 1, (\text{Smith-1x2-eucl-JNF } A))$
(proof)

19.5 The final executable algorithm to transform any matrix into its Smith normal form

global-interpreter *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⟩
```

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 ⇒ int list list × int list list × int list list × int list list × 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' = mat-to-list A; m = dim-row A; n = dim-col A in
  case external-SNF A' of (P-ext,S-ext,Q-ext,P'-ext,Q'-ext) ⇒ let
```

```

 $P = \text{mat-of-rows-list } m \text{ } P\text{-ext};$ 
 $S = \text{mat-of-rows-list } m \text{ } S\text{-ext};$ 
 $Q = \text{mat-of-rows-list } m \text{ } Q\text{-ext};$ 
 $P' = \text{mat-of-rows-list } m \text{ } P'\text{-ext};$ 
 $Q' = \text{mat-of-rows-list } m \text{ } Q'\text{-ext} \text{ in}$ 
 $(\text{if } \dim\text{-row } P = m \wedge \dim\text{-col } P = m$ 
 $\wedge \dim\text{-row } S = m \wedge \dim\text{-col } S = n$ 
 $\wedge \dim\text{-row } Q = n \wedge \dim\text{-col } Q = n$ 
 $\wedge \dim\text{-row } P' = m \wedge \dim\text{-col } P' = m$ 
 $\wedge \dim\text{-row } Q' = n \wedge \dim\text{-col } Q' = n$ 
 $\wedge P * P' = 1_m \text{ } m \wedge Q * Q' = 1_m \text{ } n$ 
 $\wedge \text{Smith-normal-form-mat } S \wedge (S = P * A * Q) \text{ then}$ 
 $(P, S, Q) \text{ else Code.abort (STR "Certification failed") } (\lambda \_. \text{Smith-ED-mxn } A))$ 
)

```

theorem checker-SNF-soudness:

```

assumes  $A: A \in \text{carrier-mat } m \text{ } n$ 
and  $c: \text{checker-SNF } A = (P, S, Q)$ 
shows  $\text{is-SNF } A \text{ } (P, S, Q)$ 
⟨proof⟩

```

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) ==> (A)

lemma diagonal-2x2-admits-SNF-imp-bezout-ring-JNF:

```

assumes admits-SNF:  $\forall A. (A::'a \text{ mat}) \in \text{carrier-mat } 2 \text{ } 2 \wedge \text{isDiagonal-mat } A$ 
 $\longrightarrow (\exists P Q. P \in \text{carrier-mat } 2 \text{ } 2 \wedge Q \in \text{carrier-mat } 2 \text{ } 2 \wedge \text{invertible-mat } P \wedge$ 
 $\text{invertible-mat } Q$ 
 $\wedge \text{Smith-normal-form-mat } (P * A * Q))$ 
shows OFCLASS('a::comm-ring-1, bezout-ring-class)
⟨proof⟩

```

Theorem 2: (A) ==> (C)

lemma bezout-ring-imp-diagonal-2x2-admits-SNF-JNF:

```

assumes c: OFCLASS('a::comm-ring-1, bezout-ring-class)
shows  $\forall A. (A::'a \text{ mat}) \in \text{carrier-mat } 2 \text{ } 2 \wedge \text{isDiagonal-mat } A$ 
 $\longrightarrow (\exists P Q. P \in \text{carrier-mat } 2 \text{ } 2 \wedge Q \in \text{carrier-mat } 2 \text{ } 2$ 
 $\wedge \text{invertible-mat } P \wedge \text{invertible-mat } Q \wedge \text{Smith-normal-form-mat } (P * A * Q))$ 
⟨proof⟩

```

Theorem 2: (A) <=> (C)

lemma diagonal-2x2-admits-SNF-iff-bezout-ring:

```

shows OFCLASS('a::comm-ring-1, bezout-ring-class)
 $\equiv (\bigwedge A::'a \text{ mat}. A \in \text{carrier-mat } 2 \text{ } 2 \longrightarrow \text{admits-SNF-JNF } A) \text{ } (\text{is } ?lhs \equiv ?rhs)$ 
⟨proof⟩

```

Theorem 2: (B) \iff (C)

lemma *diagonal-2x2-admits-SNF-iff-diagonal-admits-SNF*:
shows $(\forall (A::'a::comm-ring-1 mat). admits-SNF-JNF A) = (\forall (A::'a mat) \in carrier-mat 2 2. admits-SNF-JNF A)$
(proof)

Theorem 2: final statements

theorem *Theorem2-final*:
shows *A-imp-B*: $OFCLASS('a::comm-ring-1, bezout-ring-class)$
 $\implies (\forall A::'a mat. admits-SNF-JNF A)$
and *B-imp-C*: $(\forall A::'a mat. admits-SNF-JNF A) \implies (\forall (A::'a mat) \in carrier-mat 2 2. admits-SNF-JNF A)$
and *C-imp-A*: $(\forall (A::'a mat) \in carrier-mat 2 2. admits-SNF-JNF A) \implies OFCLASS('a::comm-ring-1, bezout-ring-class)$
(proof)

theorem *Theorem2-final'*:
shows *A-eq-B*: $OFCLASS('a::comm-ring-1, bezout-ring-class) \equiv (\bigwedge A::'a mat. admits-SNF-JNF A)$
and *A-eq-C*: $OFCLASS('a::comm-ring-1, bezout-ring-class) \equiv (\bigwedge (A::'a mat). A \in carrier-mat 2 2 \rightarrow admits-SNF-JNF A)$
and *B-eq-C*: $(\forall (A::'a::comm-ring-1 mat). admits-SNF-JNF A) = (\forall (A::'a mat) \in carrier-mat 2 2. admits-SNF-JNF A)$
(proof)

Theorem 2: final statement in HA. (A) \iff (C).

theorem *Theorem2-A-eq-C-HA*:
 $OFCLASS('a::comm-ring-1, bezout-ring-class) \equiv (\bigwedge (A::'a \hat{2} \hat{2}). admits-SNF-HA A)$
(proof)

Hermite implies Bezout

Theorem 3, proof for 1x2 matrices

lemma *theorem3-restricted-12-part1*:
assumes *T*: $(\forall a b::'a::comm-ring-1. \exists a1 b1 d. a = a1*d \wedge b = b1*d \wedge ideal-generated \{a1, b1\} = ideal-generated \{1\})$
shows $\forall (A::'a mat) \in carrier-mat 1 2. admits-triangular-reduction A$
(proof)

lemma *theorem3-restricted-12-part2*:
assumes *1*: $\forall (A::'a::comm-ring-1 mat) \in carrier-mat 1 2. admits-triangular-reduction A$
shows $\forall a b::'a. \exists a1 b1 d. a = a1*d \wedge b = b1*d \wedge ideal-generated \{a1, b1\} = ideal-generated \{1\}$
(proof)

```
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)  
end
```