

# Two algorithms based on modular arithmetic: lattice basis reduction and Hermite normal form computation\*

Ralph Bottesch      Jose Divasón      René Thiemann

February 6, 2026

## Abstract

We verify two algorithms for which modular arithmetic plays an essential role: Storjohann’s variant of the LLL lattice basis reduction algorithm and Kopparty’s algorithm for computing the Hermite normal form of a matrix. To do this, we also formalize some facts about the modulo operation with symmetric range. Our implementations are based on the original papers, but are otherwise efficient. For basis reduction we formalize two versions: one that includes all of the optimizations/heuristics from Storjohann’s paper, and one excluding a heuristic that we observed to often decrease efficiency. We also provide a fast, self-contained certifier for basis reduction, based on the efficient Hermite normal form algorithm.

## Contents

<b>1</b>	<b>Signed Modulo Operation</b>	<b>1</b>
<b>2</b>	<b>Storjohann’s Lemma 13</b>	<b>3</b>
<b>3</b>	<b>Storjohann’s basis reduction algorithm (abstract version)</b>	<b>7</b>
3.1	Definition of algorithm . . . . .	7
3.2	Towards soundness of Storjohann’s algorithm . . . . .	11
3.3	Soundness of Storjohann’s algorithm . . . . .	21
<b>4</b>	<b>Storjohann’s basis reduction algorithm (concrete implementation)</b>	<b>22</b>
4.1	Implementation . . . . .	22
4.2	Towards soundness proof of implementation . . . . .	29
4.3	Soundness of implementation . . . . .	32

---

\*Supported by FWF (Austrian Science Fund) project Y757 and by project MTM2017-88804-P (Spanish Ministry of Science and Innovation).

<b>5</b>	<b>Generalization of the statement about the uniqueness of the Hermite normal form</b>	<b>32</b>
<b>6</b>	<b>Uniqueness of Hermite normal form in JNF</b>	<b>34</b>
<b>7</b>	<b>Formalization of an efficient Hermite normal form algorithm</b>	<b>40</b>
7.1	Implementation of the algorithm using generic modulo operation . . . . .	40
7.1.1	Echelon form algorithm . . . . .	40
7.1.2	From echelon form to Hermite normal form . . . . .	44
7.1.3	Some examples of execution . . . . .	45
7.2	Soundness of the algorithm . . . . .	45
7.2.1	Results connecting lattices and Hermite normal form . . . . .	46
7.2.2	Missing results . . . . .	48
7.2.3	The algorithm is sound . . . . .	54
7.3	Instantiation of the HNF-algorithm with modulo-operation . . . . .	98
<b>8</b>	<b>LLL certification via Hermite normal forms</b>	<b>100</b>

## 1 Signed Modulo Operation

**theory** *Signed-Modulo*

**imports**

*Berlekamp-Zassenhaus.Poly-Mod*

*Sqrt-Babylonian.Sqrt-Babylonian-Auxiliary*

**begin**

The upcoming definition of symmetric modulo is different to the HOL-Library-Signed\_Division.sm<sub>od</sub>, since here the modulus will be in range  $\{-m/2, \dots, m/2\}$ , whereas there  $-1 \text{ symmod } m = m - 1$ .

The advantage of have range  $\{-m/2, \dots, m/2\}$  is that small negative numbers are represented by small numbers.

One limitation is that the symmetric modulo is only working properly, if the modulus is a positive number.

**definition** *sym-mod* :: *int*  $\Rightarrow$  *int*  $\Rightarrow$  *int* (**infixl**  $\langle \text{symmod} \rangle$  70) **where**  
*sym-mod* *x y* = *poly-mod.inv-M y (x mod y)*

**lemma** *sym-mod-code*[*code*]: *sym-mod* *x y* = (*let* *m* = *x mod y*  
*in if* *m* + *m*  $\leq$  *y* *then* *m* *else* *m* - *y*)  
*<proof>*

**lemma** *sym-mod-zero*[*simp*]: *n symmod 0* = *n* *n* > 0  $\implies$  *0 symmod n* = 0  
*<proof>*

**lemma** *sym-mod-range*:

$\langle x \text{ symmod } y \in \{ -((y - 1) \text{ div } 2) .. y \text{ div } 2 \} \rangle$  **if**  $\langle y > 0 \rangle$

*<proof>*

The range is optimal in the sense that exactly  $y$  elements can be represented.

**lemma** *card-sym-mod-range*:  $y > 0 \implies \text{card } \{-((y - 1) \text{ div } 2) .. y \text{ div } 2\} = y$   
*<proof>*

**lemma** *sym-mod-abs*:  $y > 0 \implies |x \text{ symmod } y| < y$   
 $y \geq 1 \implies |x \text{ symmod } y| \leq y \text{ div } 2$   
*<proof>*

**lemma** *sym-mod-sym-mod[simp]*:  $x \text{ symmod } y \text{ symmod } y = x \text{ symmod } (y :: \text{int})$   
*<proof>*

**lemma** *sym-mod-diff-eq*:  $(a \text{ symmod } c - b \text{ symmod } c) \text{ symmod } c = (a - b) \text{ symmod } c$   
*<proof>*

**lemma** *sym-mod-sym-mod-cancel*:  $c \text{ dvd } b \implies a \text{ symmod } b \text{ symmod } c = a \text{ symmod } c$   
*<proof>*

**lemma** *sym-mod-diff-right-eq*:  $(a - b \text{ symmod } c) \text{ symmod } c = (a - b) \text{ symmod } c$   
*<proof>*

**lemma** *sym-mod-mult-right-eq*:  $a * (b \text{ symmod } c) \text{ symmod } c = a * b \text{ symmod } c$   
*<proof>*

**lemma** *dvd-imp-sym-mod-0 [simp]*:  
 $b \text{ symmod } a = 0$  **if**  $a > 0$   $a \text{ dvd } b$   
*<proof>*

**lemma** *sym-mod-0-imp-dvd [dest!]*:  
 $b \text{ dvd } a$  **if**  $a \text{ symmod } b = 0$   
*<proof>*

**definition** *sym-div*  $:: \text{int} \Rightarrow \text{int} \Rightarrow \text{int}$  (**infixl** *<symdiv>* 70) **where**  
 $\text{sym-div } x \ y = (\text{let } d = x \text{ div } y; m = x \text{ mod } y \text{ in}$   
 $\text{if } m + m \leq y \text{ then } d \text{ else } d + 1)$

**lemma** *of-int-mod-integer*:  $(\text{of-int } (x \text{ mod } y) :: \text{integer}) = (\text{of-int } x :: \text{integer}) \text{ mod } (\text{of-int } y)$   
*<proof>*

**lemma** *sym-div-code[code]*:  
 $\text{sym-div } x \ y = (\text{let } yy = \text{integer-of-int } y \text{ in}$   
 $\text{case divmod-integer } (\text{integer-of-int } x) \ yy$   
 $\text{of } (d, m) \Rightarrow \text{if } m + m \leq yy \text{ then int-of-integer } d \text{ else } (\text{int-of-integer } (d + 1))))$   
*<proof>*

**lemma** *sym-mod-sym-div*: **assumes**  $y: y > 0$  **shows**  $x \text{ symmod } y = x - \text{sym-div } x \ y * y$   
 <proof>

**lemma** *dvd-sym-div-mult-right* [*simp*]:  
 $(a \ \text{symdiv} \ b) * b = a$  **if**  $b > 0$   $b \ \text{dvd} \ a$   
 <proof>

**lemma** *dvd-sym-div-mult-left* [*simp*]:  
 $b * (a \ \text{symdiv} \ b) = a$  **if**  $b > 0$   $b \ \text{dvd} \ a$   
 <proof>

**end**

## 2 Storjohann's Lemma 13

This theory contains the result that one can always perform a mod-operation on the entries of the  $d\mu$ -matrix.

**theory** *Storjohann-Mod-Operation*  
**imports**  
   *LLL-Basis-Reduction.LLL-Certification*  
   *Signed-Modulo*  
**begin**

**lemma** *map-vec-map-vec*:  $\text{map-vec } f \ (\text{map-vec } g \ v) = \text{map-vec } (f \ o \ g) \ v$   
 <proof>

**context** *semiring-hom*  
**begin**

**lemma** *mat-hom-add*: **assumes**  $A: A \in \text{carrier-mat } nr \ nc$  **and**  $B: B \in \text{carrier-mat } nr \ nc$   
 $\text{shows } \text{mat}_h \ (A + B) = \text{mat}_h \ A + \text{mat}_h \ B$   
 <proof>  
**end**

We now start to prove lemma 13 of Storjohann's paper.

**context**  
**fixes**  $A \ I :: 'a :: \text{field mat}$  **and**  $n :: \text{nat}$   
**assumes**  $A: A \in \text{carrier-mat } n \ n$   
**and**  $\text{det}: \text{det } A \neq 0$   
**and**  $I: I = \text{the } (\text{mat-inverse } A)$   
**begin**  
**lemma** *inverse-via-det*:  $I * A = 1_m \ n \ A * I = 1_m \ n \ I \in \text{carrier-mat } n \ n$   
 $I = \text{mat } n \ n \ (\lambda \ (i,j). \text{det } (\text{replace-col } A \ (\text{unit-vec } n \ j) \ i) / \text{det } A)$   
 <proof>

**lemma** *matrix-for-singleton-entry*: **assumes**  $i: i < n$  **and**  
 $j: j < n$   
**and**  $Rdef: R = mat\ n\ n\ (\lambda\ ij.\ if\ ij = (i,j)\ then\ c :: 'a\ else\ 0)$   
**shows**  $mat\ n\ n\ (\lambda(i', j').\ if\ i' = i\ then\ c * det\ (replace-col\ A\ (unit-vec\ n\ j')\ j) / det\ A\ else\ 0) * A = R$   
 $\langle proof \rangle$   
**end**

**lemma** (**in** *gram-schmidt-fs-Rn*) *det-M-1*:  $det\ (M\ m) = 1$   
 $\langle proof \rangle$

**context** *gram-schmidt-fs-int*

**begin**

**lemma** **assumes**  $IM: IM = the\ (mat-inverse\ (M\ m))$   
**shows** *inv-mu-lower-triangular*:  $\bigwedge k\ i.\ k < i \implies i < m \implies IM\ \$\$ (k, i) = 0$   
**and** *inv-mu-diag*:  $\bigwedge k.\ k < m \implies IM\ \$\$ (k, k) = 1$   
**and** *d-inv-mu-integer*:  $\bigwedge i\ j.\ i < m \implies j < m \implies d\ i * IM\ \$\$ (i, j) \in \mathbb{Z}$   
**and** *inv-mu-inverse*:  $IM * M\ m = 1_m\ m\ M\ m * IM = 1_m\ m\ IM \in carrier-mat\ m\ m$   
 $\langle proof \rangle$

**definition** *inv-mu-ij-mat* ::  $nat \Rightarrow nat \Rightarrow int \Rightarrow int\ mat$  **where**

*inv-mu-ij-mat*  $i\ j\ c = (let$   
 $B = mat\ m\ m\ (\lambda\ ij.\ if\ ij = (i, j)\ then\ c\ else\ 0);$   
 $C = mat\ m\ m\ (\lambda\ (i, j).\ the-inv\ (of-int\ :: - \Rightarrow 'a)\ (d\ i * the\ (mat-inverse\ (M\ m))\ \$\$ (i, j)))$   
**in**  $B * C + 1_m\ m)$

**lemma** *inv-mu-ij-mat*: **assumes**  $i: i < m$  **and**  $ji: j < i$

**shows**

$map-mat\ of-int\ (inv-mu-ij-mat\ i\ j\ c) * M\ m =$   
 $mat\ m\ m\ (\lambda ij.\ if\ ij = (i, j)\ then\ of-int\ c * d\ j\ else\ 0) + M\ m$

$A \in carrier-mat\ m\ n \implies c\ mod\ p = 0 \implies map-mat\ (\lambda x.\ x\ mod\ p)\ (inv-mu-ij-mat\ i\ j\ c * A) =$   
 $(map-mat\ (\lambda x.\ x\ mod\ p)\ A)$

$inv-mu-ij-mat\ i\ j\ c \in carrier-mat\ m\ m$   
 $i' < j' \implies j' < m \implies inv-mu-ij-mat\ i\ j\ c\ \$\$ (i', j') = 0$   
 $k < m \implies inv-mu-ij-mat\ i\ j\ c\ \$\$ (k, k) = 1$

$\langle proof \rangle$

**end**

**lemma** *Gramian-determinant-of-int*: **assumes**  $fs: set\ fs \subseteq carrier-vec\ n$

**and**  $j: j \leq length\ fs$

**shows**  $of-int\ (gram-schmidt.Gramian-determinant\ n\ fs\ j)$

= *gram-schmidt.Gramian-determinant*  $n$  (*map* (*map-vec rat-of-int*)  $fs$ )  $j$   
 ⟨*proof*⟩

**context** *LLL*  
**begin**

**lemma** *multiply-invertible-mat*: **assumes** *lin*: *lin-indep*  $fs$   
**and** *len*: *length*  $fs = m$   
**and**  $A$ :  $A \in \text{carrier-mat } m \ m$   
**and** *A-invertible*:  $\exists B. B \in \text{carrier-mat } m \ m \wedge B * A = 1_m \ m$   
**and** *fs'-prod*:  $fs' = \text{Matrix.rows } (A * \text{mat-of-rows } n \ fs)$   
**shows** *lattice-of*  $fs' = \text{lattice-of } fs$   
*lin-indep*  $fs'$   
*length*  $fs' = m$   
 ⟨*proof*⟩

This is the key lemma.

**lemma** *change-single-element*: **assumes** *lin*: *lin-indep*  $fs$   
**and** *len*: *length*  $fs = m$   
**and**  $i$ :  $i < m$  **and**  $ji$ :  $j < i$   
**and**  $A$ :  $A = \text{gram-schmidt-fs-int.inv-mu-ij-mat } n \ (RAT \ fs)$  — the transformation matrix  $A$   
**and** *fs'-prod*:  $fs' = \text{Matrix.rows } (A \ i \ j \ c * \text{mat-of-rows } n \ fs)$  —  $fs'$  is the new basis  
**and** *latt*: *lattice-of*  $fs = L$   
**shows** *lattice-of*  $fs' = L$   
 $c \bmod p = 0 \implies \text{map } (\text{map-vec } (\lambda x. x \bmod p)) \ fs' = \text{map } (\text{map-vec } (\lambda x. x \bmod p)) \ fs$   
*lin-indep*  $fs'$   
*length*  $fs' = m$   
 $\bigwedge k. k < m \implies \text{gso } fs' \ k = \text{gso } fs \ k$   
 $\bigwedge k. k \leq m \implies \text{d } fs' \ k = \text{d } fs \ k$   
 $i' < m \implies j' < m \implies$   
 $\mu \ fs' \ i' \ j' = (\text{if } (i',j') = (i,j) \text{ then } \text{rat-of-int } (c * \text{d } fs \ j) + \mu \ fs \ i' \ j' \text{ else } \mu \ fs \ i' \ j')$   
 $i' < m \implies j' < m \implies$   
 $d\mu \ fs' \ i' \ j' = (\text{if } (i',j') = (i,j) \text{ then } c * \text{d } fs \ j * \text{d } fs \ (\text{Suc } j) + d\mu \ fs \ i' \ j' \text{ else } d\mu \ fs \ i' \ j')$   
 ⟨*proof*⟩

Eventually: Lemma 13 of Storjohann's paper.

**lemma** *mod-single-element*: **assumes** *lin*: *lin-indep*  $fs$   
**and** *len*: *length*  $fs = m$   
**and**  $i$ :  $i < m$  **and**  $ji$ :  $j < i$   
**and** *latt*: *lattice-of*  $fs = L$   
**and** *pgtz*:  $p > 0$   
**shows**  $\exists fs'. \text{lattice-of } fs' = L \wedge$   
 $\text{map } (\text{map-vec } (\lambda x. x \bmod p)) \ fs' = \text{map } (\text{map-vec } (\lambda x. x \bmod p)) \ fs \wedge$   
 $\text{map } (\text{map-vec } (\lambda x. x \bmod p)) \ fs' = \text{map } (\text{map-vec } (\lambda x. x \bmod p)) \ fs \wedge$

```

lin-indep fs' ∧
length fs' = m ∧
(∀ k < m. gso fs' k = gso fs k) ∧
(∀ k ≤ m. d fs' k = d fs k) ∧
(∀ i' < m. ∀ j' < m. dμ fs' i' j' = (if (i',j') = (i,j) then dμ fs i j symmod (p
* d fs j' * d fs (Suc j')) else dμ fs i' j'))
⟨proof⟩

```

A slight generalization to perform modulo on arbitrary set of indices  $I$ .

```

lemma mod-finite-set: assumes lin: lin-indep fs
and len: length fs = m
and I: I ⊆ {(i,j). i < m ∧ j < i}
and latt: lattice-of fs = L
and pgtz: p > 0
shows ∃ fs'. lattice-of fs' = L ∧
map (map-vec (λ x. x mod p)) fs' = map (map-vec (λ x. x mod p)) fs ∧
map (map-vec (λ x. x symmod p)) fs' = map (map-vec (λ x. x symmod p)) fs ∧
lin-indep fs' ∧
length fs' = m ∧
(∀ k < m. gso fs' k = gso fs k) ∧
(∀ k ≤ m. d fs' k = d fs k) ∧
(∀ i' < m. ∀ j' < m. dμ fs' i' j' =
(if (i',j') ∈ I then dμ fs i' j' symmod (p * d fs j' * d fs (Suc j')) else dμ fs i'
j'))
⟨proof⟩

```

**end**

**end**

### 3 Storjohann's basis reduction algorithm (abstract version)

This theory contains the soundness proofs of Storjohann's basis reduction algorithms, both for the normal and the improved-swap-order variant.

The implementation of Storjohann's version of LLL uses modular operations throughout. It is an abstract implementation that is already quite close to what the actual implementation will be. In particular, the swap operation here is derived from the computation lemma for the swap operation in the old, integer-only formalization of LLL.

```

theory Storjohann
imports
  Storjohann-Mod-Operation
  LLL-Basis-Reduction.LLL-Number-Bounds
  Sqrt-Babylonian.NthRoot-Impl
begin

```

### 3.1 Definition of algorithm

In the definition of the algorithm, the first-flag determines, whether only the first vector of the reduced basis should be computed, i.e., a short vector. Then the modulus can be slightly decreased in comparison to the required modulus for computing the whole reduced matrix.

```
fun max-list-rats-with-index :: (int * int * nat) list  $\Rightarrow$  (int * int * nat) where
  max-list-rats-with-index [x] = x |
  max-list-rats-with-index ((n1,d1,i1) # (n2,d2,i2) # xs)
    = max-list-rats-with-index ((if n1 * d2  $\leq$  n2 * d1 then (n2,d2,i2) else
(n1,d1,i1)) # xs)
```

```
context LLL
begin
```

```
definition log-base = (10 :: int)
```

```
definition bound-number :: bool  $\Rightarrow$  nat where
  bound-number first = (if first  $\wedge$  m  $\neq$  0 then 1 else m)
```

```
definition compute-mod-of-max-gso-norm :: bool  $\Rightarrow$  rat  $\Rightarrow$  int where
  compute-mod-of-max-gso-norm first mn = log-base ^ (log-ceiling log-base (max 2
(
  root-rat-ceiling 2 (mn * (rat-of-nat (bound-number first) + 3)) + 1)))
```

```
definition g-bnd-mode :: bool  $\Rightarrow$  rat  $\Rightarrow$  int vec list  $\Rightarrow$  bool where
  g-bnd-mode first b fs = (if first  $\wedge$  m  $\neq$  0 then sq-norm (gso fs 0)  $\leq$  b else g-bnd
b fs)
```

```
definition d-of where d-of dmi = (if i = 0 then 1 :: int else dmi $$ (i - 1, i
- 1))
```

```
definition compute-max-gso-norm :: bool  $\Rightarrow$  int mat  $\Rightarrow$  rat  $\times$  nat where
  compute-max-gso-norm first dmi = (if m = 0 then (0,0) else
  case max-list-rats-with-index (map ( $\lambda$  i. (d-of dmi (Suc i), d-of dmi i, i)) [0
.. $\leq$  (if first then 1 else m)])
  of (num, denom, i)  $\Rightarrow$  (of-int num / of-int denom, i))
```

```
context
```

```
  fixes p :: int — the modulus
```

```
  and first :: bool — only compute first vector of reduced basis
```

```
begin
```

```
definition basis-reduction-mod-add-row ::
  int vec list  $\Rightarrow$  int mat  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  (int vec list  $\times$  int mat) where
  basis-reduction-mod-add-row mfs dmi j =
    (let c = round-num-denom (dmi $$ (i,j)) (d-of dmi (Suc j)) in
    (if c = 0 then (mfs, dmi))
```

```

else (mfs[ i := (map-vec (λ x. x symmod p)) (mfs ! i - c ·v mfs ! j)],
      mat m m (λ(i',j'). (if (i' = i ∧ j' ≤ j)
                             then (if j'=j then (dmu $$ (i,j') - c * dmu $$ (j,j'))
                                     else (dmu $$ (i,j') - c * dmu $$ (j,j'))
                             symmod (p * d-of dmu j' * d-of dmu (Suc j'))
                             else (dmu $$ (i',j'))))))))

```

**fun** *basis-reduction-mod-add-rows-loop* **where**

```

basis-reduction-mod-add-rows-loop mfs dmu i 0 = (mfs, dmu)
| basis-reduction-mod-add-rows-loop mfs dmu i (Suc j) = (
  let (mfs', dmu') = basis-reduction-mod-add-row mfs dmu i j
  in basis-reduction-mod-add-rows-loop mfs' dmu' i j)

```

**definition** *basis-reduction-mod-swap-dmu-mod* :: *int mat* ⇒ *nat* ⇒ *int mat* **where**

```

basis-reduction-mod-swap-dmu-mod dmu k = mat m m (λ(i, j). (
  if j < i ∧ (j = k ∨ j = k - 1) then
    dmu $$ (i, j) symmod (p * d-of dmu j * d-of dmu (Suc j))
  else dmu $$ (i, j)))

```

**definition** *basis-reduction-mod-swap* **where**

```

basis-reduction-mod-swap mfs dmu k =
(mfs[k := mfs ! (k - 1), k - 1 := mfs ! k],
 basis-reduction-mod-swap-dmu-mod (mat m m (λ(i, j). (
  if j < i then
    if i = k - 1 then
      dmu $$ (k, j)
    else if i = k ∧ j ≠ k - 1 then
      dmu $$ (k - 1, j)
    else if i > k ∧ j = k then
      ((d-of dmu (Suc k)) * dmu $$ (i, k - 1) - dmu $$ (k, k - 1) * dmu $$
(i, j))
      div (d-of dmu k)
    else if i > k ∧ j = k - 1 then
      (dmu $$ (k, k - 1) * dmu $$ (i, j) + dmu $$ (i, k) * (d-of dmu (k-1)))
      div (d-of dmu k)
    else dmu $$ (i, j)
  else if i = j then
    if i = k - 1 then
      ((d-of dmu (Suc k)) * (d-of dmu (k-1)) + dmu $$ (k, k - 1) * dmu $$
(k, k - 1))
      div (d-of dmu k)
    else (d-of dmu (Suc i))
  else dmu $$ (i, j)
))) k)

```

**fun** *basis-reduction-adjust-mod* **where**

```

basis-reduction-adjust-mod mfs dmu =
(let (b,g-idx) = compute-max-gso-norm first dmu;
    p' = compute-mod-of-max-gso-norm first b

```

*in if*  $p' < p$  *then*  
 let  $mfs' = \text{map } (\text{map-vec } (\lambda x. x \text{ symmod } p')) \text{ mfs};$   
 $d\text{-vec} = \text{vec } (\text{Suc } m) (\lambda i. d\text{-of } dm\mu \ i);$   
 $dm\mu' = \text{mat } m \ m (\lambda (i,j). \text{if } j < i \text{ then } dm\mu \ \$\$ (i,j)$   
 $\text{symmod } (p' * d\text{-vec } \$ j * d\text{-vec } \$ (\text{Suc } j)) \text{ else}$   
 $dm\mu \ \$\$ (i,j))$   
*in*  $(p', mfs', dm\mu', g\text{-idx})$   
*else*  $(p, mfs, dm\mu, g\text{-idx})$

**definition** *basis-reduction-adjust-swap-add-step* **where**

*basis-reduction-adjust-swap-add-step*  $mfs \ dm\mu \ g\text{-idx} \ i =$   
 let  $i1 = i - 1;$   
 $(mfs1, dm\mu1) = \text{basis-reduction-mod-add-row } mfs \ dm\mu \ i \ i1;$   
 $(mfs2, dm\mu2) = \text{basis-reduction-mod-swap } mfs1 \ dm\mu1 \ i$   
*in if*  $i1 = g\text{-idx}$  *then* *basis-reduction-adjust-mod*  $mfs2 \ dm\mu2$   
*else*  $(p, mfs2, dm\mu2, g\text{-idx})$

**definition** *basis-reduction-mod-step* **where**

*basis-reduction-mod-step*  $mfs \ dm\mu \ g\text{-idx} \ i \ (j :: \text{int}) =$  (*if*  $i = 0$  *then*  $(p, mfs, dm\mu,$   
 $g\text{-idx}, \text{Suc } i, j)$   
*else* let  $di = d\text{-of } dm\mu \ i;$   
 $(num, denom) = \text{quotient-of } \alpha$   
*in if*  $di * di * denom \leq num * d\text{-of } dm\mu \ (i - 1) * d\text{-of } dm\mu \ (\text{Suc } i)$  *then*  
 $(p, mfs, dm\mu, g\text{-idx}, \text{Suc } i, j)$   
*else* let  $(p', mfs', dm\mu', g\text{-idx}') = \text{basis-reduction-adjust-swap-add-step } mfs$   
 $dm\mu \ g\text{-idx} \ i$   
*in*  $(p', mfs', dm\mu', g\text{-idx}', i - 1, j + 1))$

**primrec** *basis-reduction-mod-add-rows-outer-loop* **where**

*basis-reduction-mod-add-rows-outer-loop*  $mfs \ dm\mu \ 0 = (mfs, dm\mu) |$   
*basis-reduction-mod-add-rows-outer-loop*  $mfs \ dm\mu \ (\text{Suc } i) =$   
 $(\text{let } (mfs', dm\mu') = \text{basis-reduction-mod-add-rows-outer-loop } mfs \ dm\mu \ i \ \text{in}$   
 $\text{basis-reduction-mod-add-rows-loop } mfs' \ dm\mu' \ (\text{Suc } i) \ (\text{Suc } i))$

**end**

the main loop of the normal Storjohann algorithm

**partial-function** (*tailrec*) *basis-reduction-mod-main* **where**

*basis-reduction-mod-main*  $p \ \text{first} \ mfs \ dm\mu \ g\text{-idx} \ i \ (j :: \text{int}) =$   
 $(\text{if } i < m$   
*then*  
 $\text{case } \text{basis-reduction-mod-step } p \ \text{first} \ mfs \ dm\mu \ g\text{-idx} \ i \ j$   
 $\text{of } (p', mfs', dm\mu', g\text{-idx}', i', j') \Rightarrow$   
 $\text{basis-reduction-mod-main } p' \ \text{first} \ mfs' \ dm\mu' \ g\text{-idx}' \ i' \ j'$   
*else*  
 $(p, mfs, dm\mu))$

**definition** *compute-max-gso-quot*::  $\text{int } \text{mat} \Rightarrow (\text{int} * \text{int} * \text{nat})$  **where**

*compute-max-gso-quot*  $dm\mu = \text{max-list-rats-with-index}$

$(\text{map } (\lambda i. ((d\text{-of } dm\mu (i+1)) * (d\text{-of } dm\mu (i+1)), (d\text{-of } dm\mu (i+2)) * (d\text{-of } dm\mu i), \text{Suc } i)) [0..<(m-1)])$

the main loop of Storjohann's algorithm with improved swap order

**partial-function** (*tailrec*) *basis-reduction-iso-main* **where**

*basis-reduction-iso-main*  $p$  *first*  $mfs$   $dm\mu$   $g\text{-idx}$  ( $j :: \text{int}$ ) = (  
 (if  $m > 1$  then  
 (let ( $max\text{-gso}\text{-num}$ ,  $max\text{-gso}\text{-denum}$ ,  $indx$ ) = *compute-max-gso-quot*  $dm\mu$ ;  
 ( $num$ ,  $denum$ ) = *quotient-of*  $\alpha$  *in*  
 (if ( $max\text{-gso}\text{-num} * denum > num * max\text{-gso}\text{-denum}$ ) then  
   *case basis-reduction-adjust-swap-add-step*  $p$  *first*  $mfs$   $dm\mu$   $g\text{-idx}$   $indx$  of  
     ( $p'$ ,  $mfs'$ ,  $dm\mu'$ ,  $g\text{-idx}'$ )  $\Rightarrow$   
     *basis-reduction-iso-main*  $p'$  *first*  $mfs'$   $dm\mu'$   $g\text{-idx}'$  ( $j + 1$ )  
   else  
     ( $p$ ,  $mfs$ ,  $dm\mu$ ))  
 else ( $p$ ,  $mfs$ ,  $dm\mu$ ))

**definition** *compute-initial-mfs* **where**

*compute-initial-mfs*  $p$  = *map-vec* ( $\lambda x. x$  *symmod*  $p$ ) *fs-init*

**definition** *compute-initial-dm\mu* **where**

*compute-initial-dm\mu*  $p$   $dm\mu$  = *mat*  $m$   $m$  ( $\lambda(i',j').$  if  $j' < i'$   
 then  $dm\mu$  \$\$ ( $i'$ ,  $j'$ ) *symmod* ( $p * d\text{-of } dm\mu j' * d\text{-of } dm\mu (\text{Suc } j')$ )  
 else  $dm\mu$  \$\$ ( $i'$ ,  $j'$ ))

**definition** *dm\mu-initial* = (let  $dm\mu$  =  $d\mu\text{-impl}$  *fs-init*

*in* *mat*  $m$   $m$  ( $\lambda (i,j).$   
 if  $j \leq i$  then  $d\mu\text{-impl}$  *fs-init* !!  $i$  !!  $j$  else 0))

**definition** *compute-initial-state* *first* =

(let  $dm\mu$  =  $dm\mu\text{-initial}$ ;  
 ( $b$ ,  $g\text{-idx}$ ) = *compute-max-gso-norm* *first*  $dm\mu$ ;  
 $p$  = *compute-mod-of-max-gso-norm* *first*  $b$   
*in* ( $p$ , *compute-initial-mfs*  $p$ , *compute-initial-dm\mu*  $p$   $dm\mu$ ,  $g\text{-idx}$ ))

Storjohann's algorithm

**definition** *reduce-basis-mod* :: *int* *vec* *list* **where**

*reduce-basis-mod* = (  
 let *first* = *False*;  
 ( $p0$ ,  $mfs0$ ,  $dm\mu0$ ,  $g\text{-idx}$ ) = *compute-initial-state* *first*;  
 ( $p'$ ,  $mfs'$ ,  $dm\mu'$ ) = *basis-reduction-mod-main*  $p0$  *first*  $mfs0$   $dm\mu0$   $g\text{-idx}$  0 0;  
 ( $mfs''$ ,  $dm\mu''$ ) = *basis-reduction-mod-add-rows-outer-loop*  $p'$   $mfs'$   $dm\mu'$   
 ( $m-1$ )  
*in*  $mfs''$ )

Storjohann's algorithm with improved swap order

**definition** *reduce-basis-iso* :: *int* *vec* *list* **where**

*reduce-basis-iso* = (  
 let *first* = *False*;

```

      (p0, mfs0, dm0, g-idx) = compute-initial-state first;
      (p', mfs', dm0') = basis-reduction-iso-main p0 first mfs0 dm0 g-idx 0;
      (mfs'', dm0'') = basis-reduction-mod-add-rows-outer-loop p' mfs' dm0'
(m-1)
      in mfs'')

```

Storjohann's algorithm for computing a short vector

**definition**

```

short-vector-mod = (
  let first = True;
    (p0, mfs0, dm0, g-idx) = compute-initial-state first;
    (p', mfs', dm0') = basis-reduction-mod-main p0 first mfs0 dm0 g-idx 0 0
  in hd mfs')

```

Storjohann's algorithm (iso-variant) for computing a short vector

**definition**

```

short-vector-iso = (
  let first = True;
    (p0, mfs0, dm0, g-idx) = compute-initial-state first;
    (p', mfs', dm0') = basis-reduction-iso-main p0 first mfs0 dm0 g-idx 0
  in hd mfs')

```

**end**

### 3.2 Towards soundness of Storjohann's algorithm

**lemma** *max-list-rats-with-index-in-set*:

```

assumes max: max-list-rats-with-index xs = (nm, dm, im)
and len: length xs ≥ 1
shows (nm, dm, im) ∈ set xs
⟨proof⟩

```

**lemma** *max-list-rats-with-index*: **assumes**  $\bigwedge n d i. (n,d,i) \in \text{set } xs \implies d > 0$

```

and max: max-list-rats-with-index xs = (nm, dm, im)
and (n,d,i) ∈ set xs
shows rat-of-int n / of-int d ≤ of-int nm / of-int dm
⟨proof⟩

```

**context** *LLL*

**begin**

**lemma** *log-base*:  $\log\text{-base} \geq 2$  ⟨proof⟩

**definition** *LLL-invariant-weak'* ::  $\text{nat} \Rightarrow \text{int vec list} \Rightarrow \text{bool}$  **where**

```

LLL-invariant-weak' i fs = (
  gs.lin-indpt-list (RAT fs) ∧
  lattice-of fs = L ∧
  weakly-reduced fs i ∧
  i ≤ m ∧
  length fs = m

```

)

**lemma** *LLL-invD-weak*: **assumes** *LLL-invariant-weak' i fs*

**shows**

*lin-indep fs*

*length (RAT fs) = m*

*set fs ⊆ carrier-vec n*

$\bigwedge i. i < m \implies fs ! i \in carrier-vec n$

$\bigwedge i. i < m \implies gso fs i \in carrier-vec n$

*length fs = m*

*lattice-of fs = L*

*weakly-reduced fs i*

*i ≤ m*

*<proof>*

**lemma** *LLL-invI-weak*: **assumes**

*set fs ⊆ carrier-vec n*

*length fs = m*

*lattice-of fs = L*

*i ≤ m*

*lin-indep fs*

*weakly-reduced fs i*

**shows** *LLL-invariant-weak' i fs*

*<proof>*

**lemma** *LLL-invw'-imp-w*: *LLL-invariant-weak' i fs ⟹ LLL-invariant-weak fs*

*<proof>*

**lemma** *basis-reduction-add-row-weak*:

**assumes** *Linvw*: *LLL-invariant-weak' i fs*

**and** *i*: *i < m* **and** *j*: *j < i*

**and** *fs'*: *fs' = fs[ i := fs ! i - c · v fs ! j ]*

**shows** *LLL-invariant-weak' i fs'*

*g-bnd B fs ⟹ g-bnd B fs'*

*<proof>*

**lemma** *LLL-inv-weak-m-impl-i*:

**assumes** *inv*: *LLL-invariant-weak' m fs*

**and** *i*: *i ≤ m*

**shows** *LLL-invariant-weak' i fs*

*<proof>*

**definition** *mod-invariant where*

*mod-invariant b p first = (b ≤ rat-of-int (p - 1)<sup>2</sup> / (rat-of-nat (bound-number first) + 3))*

$\wedge (\exists e. p = \log\text{-base } e)$

**lemma** *compute-mod-of-max-gso-norm*: **assumes** *mn*: *mn ≥ 0*

**and** *m*: *m = 0 ⟹ mn = 0*

**and**  $p: p = \text{compute-mod-of-max-gso-norm first mn}$   
**shows**  
 $p > 1$   
 $\text{mod-invariant mn } p \text{ first}$   
 $\langle \text{proof} \rangle$

**lemma**  $g\text{-bnd-mode-cong}$ : **assumes**  $\bigwedge i. i < m \implies \text{gso fs } i = \text{gso fs}' i$   
**shows**  $g\text{-bnd-mode first } b \text{ fs} = g\text{-bnd-mode first } b \text{ fs}'$   
 $\langle \text{proof} \rangle$

**definition**  $LLL\text{-invariant-mod} :: \text{int vec list} \Rightarrow \text{int vec list} \Rightarrow \text{int mat} \Rightarrow \text{int} \Rightarrow \text{bool} \Rightarrow \text{rat} \Rightarrow \text{nat} \Rightarrow \text{bool}$  **where**  
 $LLL\text{-invariant-mod fs mfs dm u } p \text{ first } b \text{ } i = ($   
 $\text{length fs} = m \wedge$   
 $\text{length mfs} = m \wedge$   
 $i \leq m \wedge$   
 $\text{lattice-of fs} = L \wedge$   
 $\text{gs.lin-indpt-list (RAT fs)} \wedge$   
 $\text{weakly-reduced fs } i \wedge$   
 $(\text{map (map-vec } (\lambda x. x \text{ symmod } p)) \text{ fs} = \text{mfs}) \wedge$   
 $(\forall i' < m. \forall j' < i'. |d\mu \text{ fs } i' j'| < p * d \text{ fs } j' * d \text{ fs (Suc } j')) \wedge$   
 $(\forall i' < m. \forall j' < m. d\mu \text{ fs } i' j' = \text{dmu } \$\$ (i',j')) \wedge$   
 $p > 1 \wedge$   
 $g\text{-bnd-mode first } b \text{ fs} \wedge$   
 $\text{mod-invariant } b \text{ } p \text{ first}$   
 $)$

**lemma**  $LLL\text{-invD-mod}$ : **assumes**  $LLL\text{-invariant-mod fs mfs dm u } p \text{ first } b \text{ } i$   
**shows**  
 $\text{length mfs} = m$   
 $i \leq m$   
 $\text{length fs} = m$   
 $\text{lattice-of fs} = L$   
 $\text{gs.lin-indpt-list (RAT fs)}$   
 $\text{weakly-reduced fs } i$   
 $(\text{map (map-vec } (\lambda x. x \text{ symmod } p)) \text{ fs} = \text{mfs})$   
 $(\forall i' < m. \forall j' < i'. |d\mu \text{ fs } i' j'| < p * d \text{ fs } j' * d \text{ fs (Suc } j'))$   
 $(\forall i' < m. \forall j' < m. d\mu \text{ fs } i' j' = \text{dmu } \$\$ (i',j'))$   
 $\bigwedge i. i < m \implies \text{fs } ! i \in \text{carrier-vec } n$   
 $\text{set fs} \subseteq \text{carrier-vec } n$   
 $\bigwedge i. i < m \implies \text{gso fs } i \in \text{carrier-vec } n$   
 $\bigwedge i. i < m \implies \text{mfs } ! i \in \text{carrier-vec } n$   
 $\text{set mfs} \subseteq \text{carrier-vec } n$   
 $p > 1$   
 $g\text{-bnd-mode first } b \text{ fs}$   
 $\text{mod-invariant } b \text{ } p \text{ first}$   
 $\langle \text{proof} \rangle$

**lemma**  $LLL\text{-invI-mod}$ : **assumes**

$length\ mfs = m$   
 $i \leq m$   
 $length\ fs = m$   
 $lattice-of\ fs = L$   
 $gs.lin-indpt-list\ (RAT\ fs)$   
 $weakly-reduced\ fs\ i$   
 $map\ (map-vec\ (\lambda x. x\ symmod\ p))\ fs = mfs$   
 $(\forall i' < m. \forall j' < i'. |d\mu\ fs\ i'\ j'| < p * d\ fs\ j' * d\ fs\ (Suc\ j'))$   
 $(\forall i' < m. \forall j' < m. d\mu\ fs\ i'\ j' = dm\mu\ \$\$ (i',j'))$   
 $p > 1$   
 $g-bnd-mode\ first\ b\ fs$   
 $mod-invariant\ b\ p\ first$   
**shows**  $LLL-invariant-mod\ fs\ mfs\ dm\mu\ p\ first\ b\ i$   
 $\langle proof \rangle$

**definition**  $LLL-invariant-mod-weak :: int\ vec\ list \Rightarrow int\ vec\ list \Rightarrow int\ mat \Rightarrow int$   
 $\Rightarrow bool \Rightarrow rat \Rightarrow bool$  **where**  
 $LLL-invariant-mod-weak\ fs\ mfs\ dm\mu\ p\ first\ b = ($   
 $length\ fs = m \wedge$   
 $length\ mfs = m \wedge$   
 $lattice-of\ fs = L \wedge$   
 $gs.lin-indpt-list\ (RAT\ fs) \wedge$   
 $(map\ (map-vec\ (\lambda x. x\ symmod\ p))\ fs = mfs) \wedge$   
 $(\forall i' < m. \forall j' < i'. |d\mu\ fs\ i'\ j'| < p * d\ fs\ j' * d\ fs\ (Suc\ j')) \wedge$   
 $(\forall i' < m. \forall j' < m. d\mu\ fs\ i'\ j' = dm\mu\ \$\$ (i',j')) \wedge$   
 $p > 1 \wedge$   
 $g-bnd-mode\ first\ b\ fs \wedge$   
 $mod-invariant\ b\ p\ first$   
 $)$

**lemma**  $LLL-invD-modw$ : **assumes**  $LLL-invariant-mod-weak\ fs\ mfs\ dm\mu\ p\ first\ b$   
**shows**  
 $length\ mfs = m$   
 $length\ fs = m$   
 $lattice-of\ fs = L$   
 $gs.lin-indpt-list\ (RAT\ fs)$   
 $(map\ (map-vec\ (\lambda x. x\ symmod\ p))\ fs = mfs)$   
 $(\forall i' < m. \forall j' < i'. |d\mu\ fs\ i'\ j'| < p * d\ fs\ j' * d\ fs\ (Suc\ j'))$   
 $(\forall i' < m. \forall j' < m. d\mu\ fs\ i'\ j' = dm\mu\ \$\$ (i',j'))$   
 $\bigwedge i. i < m \implies fs\ !\ i \in carrier-vec\ n$   
 $set\ fs \subseteq carrier-vec\ n$   
 $\bigwedge i. i < m \implies gso\ fs\ i \in carrier-vec\ n$   
 $\bigwedge i. i < m \implies mfs\ !\ i \in carrier-vec\ n$   
 $set\ mfs \subseteq carrier-vec\ n$   
 $p > 1$   
 $g-bnd-mode\ first\ b\ fs$   
 $mod-invariant\ b\ p\ first$   
 $\langle proof \rangle$

**lemma** *LLL-invI-modw: assumes*  
*length mfs = m*  
*length fs = m*  
*lattice-of fs = L*  
*gs.lin-indpt-list (RAT fs)*  
*map (map-vec ( $\lambda x. x \text{ symmod } p$ )) fs = mfs*  
 $(\forall i' < m. \forall j' < i'. |d\mu \text{ fs } i' j'| < p * d \text{ fs } j' * d \text{ fs } (\text{Suc } j'))$   
 $(\forall i' < m. \forall j' < m. d\mu \text{ fs } i' j' = dmu \text{ \$\$ } (i', j'))$   
*p > 1*  
*g-bnd-mode first b fs*  
*mod-invariant b p first*  
**shows** *LLL-invariant-mod-weak fs mfs dmu p first b*  
*<proof>*

**lemma** *dd $\mu$ :*  
**assumes** *i: i < m*  
**shows** *d fs (Suc i) = d $\mu$  fs i i*  
*<proof>*

**lemma** *d-of-main: assumes*  $(\forall i' < m. d\mu \text{ fs } i' i' = dmu \text{ \$\$ } (i', i'))$   
**and** *i  $\leq$  m*  
**shows** *d-of dmu i = d fs i*  
*<proof>*

**lemma** *d-of: assumes inv: LLL-invariant-mod fs mfs dmu p b first j*  
**and** *i  $\leq$  m*  
**shows** *d-of dmu i = d fs i*  
*<proof>*

**lemma** *d-of-weak: assumes inv: LLL-invariant-mod-weak fs mfs dmu p first b*  
**and** *i  $\leq$  m*  
**shows** *d-of dmu i = d fs i*  
*<proof>*

**lemma** *compute-max-gso-norm: assumes dmu:  $(\forall i' < m. d\mu \text{ fs } i' i' = dmu \text{ \$\$ } (i', i'))$*   
**and** *Liniv: LLL-invariant-weak fs*  
**shows** *g-bnd-mode first (fst (compute-max-gso-norm first dmu)) fs*  
 $\text{fst (compute-max-gso-norm first dmu)} \geq 0$   
 $m = 0 \implies \text{fst (compute-max-gso-norm first dmu)} = 0$   
*<proof>*

**lemma** *increase-i-mod:*  
**assumes** *Liniv: LLL-invariant-mod fs mfs dmu p first b i*  
**and** *i: i < m*  
**and** *red-i: i  $\neq$  0  $\implies$  sq-norm (gso fs (i - 1))  $\leq$   $\alpha$  \* sq-norm (gso fs i)*  
**shows** *LLL-invariant-mod fs mfs dmu p first b (Suc i) LLL-measure i fs > LLL-measure (Suc i) fs*

*<proof>*

**lemma** *basis-reduction-mod-add-row-main:*

**assumes** *Linvmw: LLL-invariant-mod-weak fs mfs dm u p first b*  
**and** *i: i < m and j: j < i*  
**and** *c: c = round (μ fs i j)*  
**and** *mfs': mfs' = mfs[ i := (map-vec (λ x. x symmod p)) (mfs ! i - c ·<sub>v</sub> mfs ! j)]*  
**and** *dmu': dmu' = mat m m (λ(i',j'). (if (i' = i ∧ j' ≤ j)  
then (if j'=j then (dmu \$\$ (i,j') - c \* dmu \$\$ (j,j'))  
else (dmu \$\$ (i,j') - c \* dmu \$\$ (j,j'))  
symmod (p \* (d-of dmu j) \* (d-of dmu (Suc j'))))  
else (dmu \$\$ (i',j'))))*

**shows**  $(\exists fs'. \text{LLL-invariant-mod-weak } fs' \text{ mfs}' \text{ dm u}' \text{ p first } b \wedge$

$\text{LLL-measure } i \text{ fs}' = \text{LLL-measure } i \text{ fs}$

$\wedge (\mu\text{-small-row } i \text{ fs}' (\text{Suc } j) \longrightarrow \mu\text{-small-row } i \text{ fs}' j)$

$\wedge (\forall k < m. \text{gso } fs' k = \text{gso } fs k)$

$\wedge (\forall ii \leq m. \text{d } fs' ii = \text{d } fs ii)$

$\wedge |\mu \text{ fs}' i j| \leq 1 / 2$

$\wedge (\forall i' j'. i' < i \longrightarrow j' \leq i' \longrightarrow \mu \text{ fs}' i' j' = \mu \text{ fs } i' j')$

$\wedge (\text{LLL-invariant-mod } fs \text{ mfs } dm u \text{ p first } b \text{ i} \longrightarrow \text{LLL-invariant-mod } fs' \text{ mfs}'$

$dmu' \text{ p first } b \text{ i}))$

*<proof>*

**definition** *D-mod :: int mat ⇒ nat where D-mod dm u = nat (∏ i < m. d-of dm u i)*

**definition** *logD-mod :: int mat ⇒ nat*

**where** *logD-mod dm u = (if α = 4/3 then (D-mod dm u) else nat (floor (log (1 / of-rat reduction) (D-mod dm u))))*

**end**

**locale** *fs-int'-mod =*

**fixes** *n m fs-init α i fs mfs dm u p first b*

**assumes** *LLL-inv-mod: LLL.LLL-invariant-mod n m fs-init α fs mfs dm u p first b i*

**context** *LLL-with-assms*

**begin**

**lemma** *basis-reduction-swap-weak': assumes Linvw: LLL-invariant-weak' i fs*

**and** *i: i < m*

**and** *i0: i ≠ 0*

**and** *mu-F1-i: |μ fs i (i-1)| ≤ 1 / 2*

**and** *norm-ineq: sq-norm (gso fs (i - 1)) > α \* sq-norm (gso fs i)*

**and** *fs'-def: fs' = fs[i := fs ! (i - 1), i - 1 := fs ! i]*

**shows** *LLL-invariant-weak' (i - 1) fs'*

*<proof>*

**lemma** *basis-reduction-add-row-done-weak:*

**assumes** *Lin*v: *LLL-invariant-weak'* *i fs*  
**and** *i*:  $i < m$   
**and** *mu-small*:  $\mu$ -small-row *i fs 0*  
**shows**  $\mu$ -small *fs i*  
 ⟨*proof*⟩

**lemma** *LLL-invariant-mod-to-weak-m-to-i*: **assumes**  
*inv*: *LLL-invariant-mod fs mfs dm**u p first b m*  
**and** *i*:  $i \leq m$   
**shows** *LLL-invariant-mod fs mfs dm**u p first b i*  
*LLL-invariant-weak'* *m fs*  
*LLL-invariant-weak'* *i fs*  
 ⟨*proof*⟩

**lemma** *basis-reduction-mod-swap-main*:  
**assumes** *Lin**vmw*: *LLL-invariant-mod-weak fs mfs dm**u p first b*  
**and** *k*:  $k < m$   
**and** *k0*:  $k \neq 0$   
**and** *mu-F1-i*:  $|\mu \text{ fs } k \text{ (} k-1)| \leq 1 / 2$   
**and** *norm-ineq*:  $\text{sq-norm (gso fs (} k-1))} > \alpha * \text{sq-norm (gso fs } k)$   
**and** *mfs'-def*:  $\text{mfs}' = \text{mfs}[k := \text{mfs}' (k-1), k-1 := \text{mfs}' k]$   
**and** *dmu'-def*:  $\text{dmu}' = (\text{mat } m \text{ m } (\lambda(i,j). ($   
   *if*  $j < i$  *then*  
     *if*  $i = k-1$  *then*  
        $\text{dmu } \$\$ (k, j)$   
     *else if*  $i = k \wedge j \neq k-1$  *then*  
        $\text{dmu } \$\$ (k-1, j)$   
     *else if*  $i > k \wedge j = k$  *then*  
        $((\text{d-of dm}u (\text{Suc } k)) * \text{dmu } \$\$ (i, k-1) - \text{dmu } \$\$ (k, k-1) * \text{dmu } \$\$$   
        $(i, j))$   
        $\text{div (d-of dm}u \text{ } k)$   
     *else if*  $i > k \wedge j = k-1$  *then*  
        $(\text{dmu } \$\$ (k, k-1) * \text{dmu } \$\$ (i, j) + \text{dmu } \$\$ (i, k) * (\text{d-of dm}u (k-1)))$   
        $\text{div (d-of dm}u \text{ } k)$   
     *else*  $\text{dmu } \$\$ (i, j)$   
   *else if*  $i = j$  *then*  
     *if*  $i = k-1$  *then*  
        $((\text{d-of dm}u (\text{Suc } k)) * (\text{d-of dm}u (k-1)) + \text{dmu } \$\$ (k, k-1) * \text{dmu } \$\$$   
        $(k, k-1))$   
        $\text{div (d-of dm}u \text{ } k)$   
     *else*  $(\text{d-of dm}u (\text{Suc } i))$   
     *else*  $\text{dmu } \$\$ (i, j)$   
   ))  
**and** *dmu'-mod-def*:  $\text{dmu}'\text{-mod} = \text{mat } m \text{ m } (\lambda(i, j). ($   
   *if*  $j < i \wedge (j = k \vee j = k-1)$  *then*  
      $\text{dmu}' \$\$ (i, j) \text{ symmod } (p * (\text{d-of dm}u' \text{ } j) * (\text{d-of dm}u' (\text{Suc } j)))$   
     *else*  $\text{dmu}' \$\$ (i, j))$   
**shows**  $(\exists \text{fs}' . \text{LLL-invariant-mod-weak fs}' \text{ mfs}' \text{ dm}u'\text{-mod } p \text{ first } b \wedge$   
 $\text{LLL-measure (} k-1) \text{ fs}' < \text{LLL-measure } k \text{ fs } \wedge$

(LLL-invariant-mod fs mfs dmu p first b k  $\longrightarrow$  LLL-invariant-mod fs' mfs' dmu'-mod p first b (k-1)))  
 <proof>

**lemma** *dmu-quot-is-round-of- $\mu$* :

**assumes** *Lin*v: LLL-invariant-mod fs mfs dmu p first b i'  
**and** *c*:  $c = \text{round-num-denom } (dmu \ \$\$ (i,j)) \ (d\text{-of } dmu \ (Suc \ j))$   
**and** *i*:  $i < m$   
**and** *j*:  $j < i$   
**shows**  $c = \text{round}(\mu \ fs \ i \ j)$   
 <proof>

**lemma** *dmu-quot-is-round-of- $\mu$ -weak*:

**assumes** *Lin*v: LLL-invariant-mod-weak fs mfs dmu p first b  
**and** *c*:  $c = \text{round-num-denom } (dmu \ \$\$ (i,j)) \ (d\text{-of } dmu \ (Suc \ j))$   
**and** *i*:  $i < m$   
**and** *j*:  $j < i$   
**shows**  $c = \text{round}(\mu \ fs \ i \ j)$   
 <proof>

**lemma** *basis-reduction-mod-add-row*: **assumes**

*Lin*v: LLL-invariant-mod-weak fs mfs dmu p first b  
**and** *res*: *basis-reduction-mod-add-row* p mfs dmu i j = (mfs', dmu')  
**and** *i*:  $i < m$   
**and** *j*:  $j < i$   
**and** *igtz*:  $i \neq 0$   
**shows**  $(\exists fs'. \text{LLL-invariant-mod-weak } fs' \ mfs' \ dmu' \ p \ \text{first } b \wedge$   
 $\text{LLL-measure } i \ fs' = \text{LLL-measure } i \ fs \wedge$   
 $(\mu\text{-small-row } i \ fs \ (Suc \ j) \longrightarrow \mu\text{-small-row } i \ fs' \ j) \wedge$   
 $|\mu \ fs' \ i \ j| \leq 1 / 2 \wedge$   
 $(\forall i' \ j'. \ i' < i \longrightarrow j' \leq i' \longrightarrow \mu \ fs' \ i' \ j' = \mu \ fs \ i' \ j') \wedge$   
 $(\text{LLL-invariant-mod } fs \ mfs \ dmu \ p \ \text{first } b \ i \longrightarrow \text{LLL-invariant-mod } fs' \ mfs'$   
 $dmu' \ p \ \text{first } b \ i) \wedge$   
 $(\forall ii \leq m. \ d \ fs' \ ii = d \ fs \ ii))$   
 <proof>

**lemma** *basis-reduction-mod-swap*: **assumes**

*Lin*v: LLL-invariant-mod-weak fs mfs dmu p first b  
**and** *mu*:  $|\mu \ fs \ k \ (k-1)| \leq 1 / 2$   
**and** *res*: *basis-reduction-mod-swap* p mfs dmu k = (mfs', dmu'-mod)  
**and** *cond*:  $\text{sq-norm } (gso \ fs \ (k - 1)) > \alpha * \text{sq-norm } (gso \ fs \ k)$   
**and** *i*:  $k < m \ k \neq 0$   
**shows**  $(\exists fs'. \text{LLL-invariant-mod-weak } fs' \ mfs' \ dmu'\text{-mod } p \ \text{first } b \wedge$   
 $\text{LLL-measure } (k - 1) \ fs' < \text{LLL-measure } k \ fs \wedge$   
 $(\text{LLL-invariant-mod } fs \ mfs \ dmu \ p \ \text{first } b \ k \longrightarrow \text{LLL-invariant-mod } fs' \ mfs'$   
 $dmu'\text{-mod } p \ \text{first } b \ (k-1)))$   
 <proof>

**lemma** *basis-reduction-adjust-mod*: **assumes**

*Lin*: *LLL-invariant-mod-weak fs mfs dm* *p first b*  
**and** *res*: *basis-reduction-adjust-mod p first mfs dm* = (*p'*, *mfs'*, *dmu'*, *g-id**x'*)  
**shows**  $(\exists fs' b'. (LLL\text{-invariant-mod } fs \ mfs \ dm \ p \ \text{first } b \ i \ \longrightarrow \ LLL\text{-invariant-mod } fs' \ mfs' \ dmu' \ p' \ \text{first } b' \ i) \wedge$   
 $LLL\text{-invariant-mod-weak } fs' \ mfs' \ dmu' \ p' \ \text{first } b' \ \wedge$   
 $LLL\text{-measure } i \ fs' = LLL\text{-measure } i \ fs)$   
 ⟨*proof*⟩

**lemma** *alpha-comparison*: **assumes**

*Lin*: *LLL-invariant-mod-weak fs mfs dm* *p first b*  
**and** *alph*: *quotient-of*  $\alpha = (num, denom)$   
**and** *i*:  $i < m$   
**and** *i0*:  $i \neq 0$   
**shows**  $(d\text{-of } dm \ i \ * \ d\text{-of } dm \ i \ * \ denom \leq \ num \ * \ d\text{-of } dm \ (i - 1) \ * \ d\text{-of } dm \ (Suc \ i))$   
 $= (sq\text{-norm } (gso \ fs \ (i - 1)) \leq \ \alpha \ * \ sq\text{-norm } (gso \ fs \ i))$   
 ⟨*proof*⟩

**lemma** *basis-reduction-adjust-swap-add-step*: **assumes**

*Lin*: *LLL-invariant-mod-weak fs mfs dm* *p first b*  
**and** *res*: *basis-reduction-adjust-swap-add-step p first mfs dm* *g-id**x i* = (*p'*, *mfs'*, *dmu'*, *g-id**x'*)  
**and** *alph*: *quotient-of*  $\alpha = (num, denom)$   
**and** *ineq*:  $\neg (d\text{-of } dm \ i \ * \ d\text{-of } dm \ i \ * \ denom \leq \ num \ * \ d\text{-of } dm \ (i - 1) \ * \ d\text{-of } dm \ (Suc \ i))$   
**and** *i*:  $i < m$   
**and** *i0*:  $i \neq 0$   
**shows**  $\exists fs' b'. LLL\text{-invariant-mod-weak } fs' \ mfs' \ dmu' \ p' \ \text{first } b' \ \wedge$   
 $LLL\text{-measure } (i - 1) \ fs' < LLL\text{-measure } i \ fs \ \wedge$   
 $LLL\text{-measure } (m - 1) \ fs' < LLL\text{-measure } (m - 1) \ fs \ \wedge$   
 $(LLL\text{-invariant-mod } fs \ mfs \ dm \ p \ \text{first } b \ i \ \longrightarrow$   
 $LLL\text{-invariant-mod } fs' \ mfs' \ dmu' \ p' \ \text{first } b' \ (i - 1))$   
 ⟨*proof*⟩

**lemma** *basis-reduction-mod-step*: **assumes**

*Lin*: *LLL-invariant-mod fs mfs dm* *p first b i*  
**and** *res*: *basis-reduction-mod-step p first mfs dm* *g-id**x i j* = (*p'*, *mfs'*, *dmu'*, *g-id**x'*, *i'*, *j'*)  
**and** *i*:  $i < m$   
**shows**  $\exists fs' b'. LLL\text{-measure } i' \ fs' < LLL\text{-measure } i \ fs \ \wedge \ LLL\text{-invariant-mod } fs' \ mfs' \ dmu' \ p' \ \text{first } b' \ i'$   
 ⟨*proof*⟩

**lemma** *basis-reduction-mod-main*: **assumes** *LLL-invariant-mod fs mfs dm* *p first b i*

**and** *res*: *basis-reduction-mod-main p first mfs dm* *g-id**x i j* = (*p'*, *mfs'*, *dmu'*)  
**shows**  $\exists fs' b'. LLL\text{-invariant-mod } fs' \ mfs' \ dmu' \ p' \ \text{first } b' \ m$   
 ⟨*proof*⟩

**lemma** *compute-max-gso-quot-alpha*:

**assumes** *inv*: *LLL-invariant-mod-weak fs mfs dm<sub>u</sub> p first b*  
**and** *max*: *compute-max-gso-quot dm<sub>u</sub> = (msq-num, msq-denum, idx)*  
**and** *alph*: *quotient-of α = (num, denum)*  
**and** *cmp*: *(msq-num \* denum > num \* msq-denum) = cmp*  
**and** *m*: *m > 1*  
**shows** *cmp*  $\implies$  *idx*  $\neq$  0  $\wedge$  *idx* < *m*  $\wedge$   $\neg$  (*d-of dm<sub>u</sub> idx* \* *d-of dm<sub>u</sub> idx* \* *denum*  
 $\leq$  *num* \* *d-of dm<sub>u</sub> (idx - 1)* \* *d-of dm<sub>u</sub> (Suc idx)*)  
**and**  $\neg$  *cmp*  $\implies$  *LLL-invariant-mod fs mfs dm<sub>u</sub> p first b m*  
*<proof>*

**lemma** *small-m*:

**assumes** *inv*: *LLL-invariant-mod-weak fs mfs dm<sub>u</sub> p first b*  
**and** *m*: *m  $\leq$  1*  
**shows** *LLL-invariant-mod fs mfs dm<sub>u</sub> p first b m*  
*<proof>*

**lemma** *basis-reduction-iso-main*: **assumes** *LLL-invariant-mod-weak fs mfs dm<sub>u</sub> p first b*

**and** *res*: *basis-reduction-iso-main p first mfs dm<sub>u</sub> g-idx j = (p', mfs', dm<sub>u</sub>)*  
**shows**  $\exists$  *fs' b'*. *LLL-invariant-mod fs' mfs' dm<sub>u</sub>' p' first b' m*  
*<proof>*

**lemma** *basis-reduction-mod-add-rows-loop-inv'*: **assumes**

*fsinv*: *LLL-invariant-mod fs mfs dm<sub>u</sub> p first b m*  
**and** *res*: *basis-reduction-mod-add-rows-loop p mfs dm<sub>u</sub> i i = (mfs', dm<sub>u</sub>)*  
**and** *i*: *i < m*  
**shows**  $\exists$  *fs'*. *LLL-invariant-mod fs' mfs' dm<sub>u</sub>' p first b m*  $\wedge$   
 $(\forall i' j'. i' < i \longrightarrow j' \leq i' \longrightarrow \mu \text{ fs } i' j' = \mu \text{ fs' } i' j') \wedge$   
 $\mu\text{-small fs' } i$   
*<proof>*

**lemma** *basis-reduction-mod-add-rows-outer-loop-inv*:

**assumes** *inv*: *LLL-invariant-mod fs mfs dm<sub>u</sub> p first b m*  
**and** *(mfs', dm<sub>u</sub>) = basis-reduction-mod-add-rows-outer-loop p mfs dm<sub>u</sub> i*  
**and** *i*: *i < m*  
**shows**  $(\exists$  *fs'*. *LLL-invariant-mod fs' mfs' dm<sub>u</sub>' p first b m*  $\wedge$   
 $(\forall j. j \leq i \longrightarrow \mu\text{-small fs' } j))$   
*<proof>*

**lemma** *basis-reduction-mod-fs-bound*:

**assumes** *Lin<sub>v</sub>*: *LLL-invariant-mod fs mfs dm<sub>u</sub> p first b k*  
**and** *mu-small*:  $\mu\text{-small fs } i$   
**and** *i*: *i < m*  
**and** *nFirst*:  $\neg$  *first*  
**shows** *fs ! i = mfs ! i*  
*<proof>*

**lemma** *basis-reduction-mod-fs-bound-first*:  
**assumes** *Lin*: *LLL-invariant-mod fs mfs dm $\mu$  p first b k*  
**and** *m0*:  $m > 0$   
**and** *first*: *first*  
**shows**  $fs ! 0 = mfs ! 0$   
 $\langle proof \rangle$

**lemma** *dmu-initial*:  $dmu\text{-}initial = mat\ m\ m\ (\lambda\ (i,j).\ d\mu\ fs\text{-}init\ i\ j)$   
 $\langle proof \rangle$

**lemma** *LLL-initial-invariant-mod*: **assumes** *res*: *compute-initial-state first = (p, mfs, dm $\mu$ ' , g-idx)*  
**shows**  $\exists fs\ b.$  *LLL-invariant-mod fs mfs dm $\mu$ ' p first b 0*  
 $\langle proof \rangle$

### 3.3 Soundness of Storjohann's algorithm

For all of these abstract algorithms, we actually formulate their soundness proofs by linking to the LLL-invariant (which implies that *fs* is reduced (*LLL-invariant True m fs*) or that the first vector of *fs* is short (*LLL-invariant-weak fs  $\wedge$  gram-schmidt-fs.weakly-reduced n (map of-int-hom.vec-hom fs)  $\alpha$  m*).

Soundness of Storjohann's algorithm

**lemma** *reduce-basis-mod-inv*: **assumes** *res*: *reduce-basis-mod = fs*  
**shows** *LLL-invariant True m fs*  
 $\langle proof \rangle$

Soundness of Storjohann's algorithm for computing a short vector.

**lemma** *short-vector-mod-inv*: **assumes** *res*: *short-vector-mod = v*  
**and** *m*:  $m > 0$   
**shows**  $\exists fs.$  *LLL-invariant-weak fs  $\wedge$  weakly-reduced fs m  $\wedge$  v = hd fs*  
 $\langle proof \rangle$

Soundness of Storjohann's algorithm with improved swap order

**lemma** *reduce-basis-iso-inv*: **assumes** *res*: *reduce-basis-iso = fs*  
**shows** *LLL-invariant True m fs*  
 $\langle proof \rangle$

Soundness of Storjohann's algorithm to compute short vectors with improved swap order

**lemma** *short-vector-iso-inv*: **assumes** *res*: *short-vector-iso = v*  
**and** *m*:  $m > 0$   
**shows**  $\exists fs.$  *LLL-invariant-weak fs  $\wedge$  weakly-reduced fs m  $\wedge$  v = hd fs*  
 $\langle proof \rangle$

**end**

From the soundness results of these abstract versions of the algorithms, one just needs to derive actual implementations that may integrate low-level optimizations.

end

## 4 Storjohann's basis reduction algorithm (concrete implementation)

We refine the abstract algorithm into a more efficient executable one.

```
theory Storjohann-Impl
  imports
    Storjohann
begin
```

### 4.1 Implementation

We basically store four components:

- The  $f$ -basis (as list, all values taken modulo  $p$ )
- The  $d\mu$ -matrix (as nested arrays, all values taken modulo  $d_i d_{i+1} p$ )
- The  $d$ -values (as array)
- The modulo-values  $d_i d_{i+1} p$  (as array)

```
type-synonym state-impl = int vec list  $\times$  int iarray iarray  $\times$  int iarray  $\times$  int iarray
```

```
fun di-of :: state-impl  $\Rightarrow$  int iarray where
  di-of (mfsi, dmui, di, mods) = di
```

```
context LLL
begin
```

```
fun state-impl-inv :: -  $\Rightarrow$  -  $\Rightarrow$  -  $\Rightarrow$  state-impl  $\Rightarrow$  bool where
  state-impl-inv p mfs dmui (mfsi, dmui, di, mods) = (mfsi = mfs  $\wedge$  di = IArray.of-fun (d-of dmui) (Suc m)
     $\wedge$  dmui = IArray.of-fun ( $\lambda$  i. IArray.of-fun ( $\lambda$  j. dmu $$ (i,j)) i) m
     $\wedge$  mods = IArray.of-fun ( $\lambda$  j. p * di !! j * di !! (Suc j)) (m - 1))
```

```
definition state-iso-inv :: (int  $\times$  int) iarray  $\Rightarrow$  int iarray  $\Rightarrow$  bool where
  state-iso-inv prods di = (prods = IArray.of-fun
    ( $\lambda$  i. (di !! (i+1) * di !! (i+1), di !! (i+2) * di !! i)) (m - 1))
```

```
definition perform-add-row :: int  $\Rightarrow$  state-impl  $\Rightarrow$  nat  $\Rightarrow$  nat  $\Rightarrow$  int  $\Rightarrow$  int iarray
 $\Rightarrow$  int  $\Rightarrow$  int  $\Rightarrow$  state-impl where
```

```

perform-add-row p state i j c rowi muij dij1 = (let
  (mfsi, dmui, di, mods) = state;
  fsj = mfsi ! j;
  rowj = dmui !! j
  in
  (case split-at i mfsi of (start, fsi # end) => start @ vec n (λ k. (fsi $ k - c
* fsj $ k) symmod p) # end,
  IArray.of-fun (λ ii. if i = ii then
    IArray.of-fun (λ jj. if jj < j then
      (rowi !! jj - c * rowj !! jj) symmod (mods !! jj)
    else if jj = j then muij - c * dij1
    else rowi !! jj) i
  else dmui !! ii) m,
  di, mods))

```

**definition** *LLL-add-row* :: *int* ⇒ *state-impl* ⇒ *nat* ⇒ *nat* ⇒ *state-impl* **where**

```

LLL-add-row p state i j = (let
  (-, dmui, di, -) = state;
  rowi = dmui !! i;
  dij1 = di !! (Suc j);
  muij = rowi !! j;
  c = round-num-denom muij dij1
  in if c = 0 then state
  else perform-add-row p state i j c rowi muij dij1)

```

**definition** *LLL-swap-row* :: *int* ⇒ *state-impl* ⇒ *nat* ⇒ *state-impl* **where**

```

LLL-swap-row p state k = (case state of (mfsi, dmui, di, mods) => let
  k1 = k - 1;
  kS1 = Suc k;
  muk = dmui !! k;
  muk1 = dmui !! k1;
  mukk1 = muk !! k1;
  dk1 = di !! k1;
  dkS1 = di !! kS1;
  dk = di !! k;
  dk' = (dkS1 * dk1 + mukk1 * mukk1) div dk;
  mod1 = p * dk1 * dk';
  modk = p * dk' * dkS1
  in
  (case split-at k1 mfsi
  of (start, fsk1 # fsk # end) => start @ fsk # fsk1 # end,
  IArray.of-fun (λ i.
    if i < k1 then dmui !! i
    else if i > k then
      let row-i = dmui !! i; muik = row-i !! k; muik1 = row-i !! k1 in
      IArray.of-fun
        (λ j. if j = k1 then ((mukk1 * muik1 + muik * dk1) div dk) symmod
mod1

```

```

else if j = k then ((dkS1 * muik1 - mukk1 * muik) div dk)
symmod modk
else row-i !! j) i
else if i = k then IArray.of-fun (λ j. if j = k1 then mukk1 symmod mod1
else muk1 !! j) i
else IArray.of-fun ((!!) muk) i
) m,
IArray.of-fun (λ i. if i = k then dk' else di !! i) (Suc m),
IArray.of-fun (λ j. if j = k1 then mod1 else if j = k then modk else mods !!
j) (m - 1)))

```

**definition** *perform-swap-add* **where** *perform-swap-add p state k k1 c row-k mukk1 dk =*

```

(let (fs, dmU, dd, mods) = state;
row-k1 = dmU !! k1;
kS1 = Suc k;
mukk1' = mukk1 - c * dk;
dk1 = dd !! k1;
dkS1 = dd !! kS1;
dk' = (dkS1 * dk1 + mukk1' * mukk1') div dk;
mod1 = p * dk1 * dk';
modk = p * dk' * dkS1
in
(case split-at k1 fs of (start, fsk1 # fsk # end) =>
start @ vec n (λk. (fsk $ k - c * fsk1 $ k) symmod p) # fsk1 # end,
IArray.of-fun
(λi. if i < k1
then dmU !! i
else if k < i
then let row-i = dmU !! i;
muik1 = row-i !! k1;
muik = row-i !! k
in IArray.of-fun
(λj. if j = k1 then (mukk1' * muik1 + muik * dk1) div dk
symmod mod1
else if j = k then (dkS1 * muik1 - mukk1' * muik) div
dk symmod modk
else row-i !! j)
i
else if i = k then IArray.of-fun (λj. if j = k1 then mukk1' symmod
mod1 else row-k1 !! j) k
else IArray.of-fun (λj. (row-k !! j - c * row-k1 !! j) symmod mods
!! j) i)
m,
IArray.of-fun (λi. if i = k then dk' else dd !! i) (Suc m),
IArray.of-fun (λj. if j = k1 then mod1 else if j = k then modk else mods !! j)
(m - 1)))

```

**definition** *LLL-swap-add* **where**

```

LLL-swap-add p state i = (let
  i1 = i - 1;
  (-, dmui, di, -) = state;
  rowi = dmui !! i;
  dii = di !! i;
  muij = rowi !! i1;
  c = round-num-denom muij dii
  in if c = 0 then LLL-swap-row p state i
  else perform-swap-add p state i i1 c rowi muij dii)

```

**definition** *LLL-max-gso-norm-di* :: *bool*  $\Rightarrow$  *int iarray*  $\Rightarrow$  *rat*  $\times$  *nat* **where**

```

LLL-max-gso-norm-di first di =
  (if first then (of-int (di !! 1), 0)
   else case max-list-rats-with-index (map (\ i. (di !! (Suc i), di !! i, i)) [0 ..<
m ])
  of (num, denom, i)  $\Rightarrow$  (of-int num / of-int denom, i))

```

**definition** *LLL-max-gso-quot*:: (*int*  $\times$  *int*) *iarray*  $\Rightarrow$  (*int*  $\times$  *int*  $\times$  *nat*) **where**

```

LLL-max-gso-quot di-prods = max-list-rats-with-index
  (map (\i. case di-prods !! i of (l,r)  $\Rightarrow$  (l, r, Suc i)) [0..<(m-1)])

```

**definition** *LLL-max-gso-norm* :: *bool*  $\Rightarrow$  *state-impl*  $\Rightarrow$  *rat*  $\times$  *nat* **where**

```

LLL-max-gso-norm first state = (case state of (-, -, di, mods)  $\Rightarrow$  LLL-max-gso-norm-di
first di)

```

**definition** *perform-adjust-mod* :: *int*  $\Rightarrow$  *state-impl*  $\Rightarrow$  *state-impl* **where**

```

perform-adjust-mod p state = (case state of (mfsi, dmui, di, -)  $\Rightarrow$ 
  let mfsi' = map (map-vec (\x. x symmod p)) mfsi;
      mods = IArray.of-fun (\ j. p * di !! j * di !! (Suc j)) (m - 1);
      dmui' = IArray.of-fun (\ i. let row = dmui !! i in IArray.of-fun (\ j.
row !! j symmod (mods !! j)) i) m
  in
  ((mfsi', dmui', di, mods)))

```

**definition** *mod-of-gso-norm* :: *bool*  $\Rightarrow$  *rat*  $\Rightarrow$  *int* **where**

```

mod-of-gso-norm first mn = log-base ^ (log-ceiling log-base (max 2 (
  root-rat-ceiling 2 (mn * (rat-of-nat (if first then 4 else m + 3))) + 1)))

```

**definition** *LLL-adjust-mod* :: *int*  $\Rightarrow$  *bool*  $\Rightarrow$  *state-impl*  $\Rightarrow$  *int*  $\times$  *state-impl*  $\times$  *nat* **where**

```

LLL-adjust-mod p first state = (
  let (b', g-idx) = LLL-max-gso-norm first state;
      p' = mod-of-gso-norm first b'
  in if p' < p then (p', perform-adjust-mod p' state, g-idx)
  else (p, state, g-idx)
)

```

**definition** *LLL-adjust-swap-add* **where**

*LLL-adjust-swap-add* *p* *first state* *g-idx* *i* = (  
 let *state1* = *LLL-swap-add* *p* *state* *i*  
 in if *i* - 1 = *g-idx* then  
*LLL-adjust-mod* *p* *first state1* else (*p*, *state1*, *g-idx*))

**definition** *LLL-step* :: *int* ⇒ *bool* ⇒ *state-impl* ⇒ *nat* ⇒ *nat* ⇒ *int* ⇒ (*int* × *state-impl* × *nat*) × *nat* × *int* **where**

*LLL-step* *p* *first state* *g-idx* *i* *j* = (if *i* = 0 then ((*p*, *state*, *g-idx*), *Suc* *i*, *j*)  
 else let  
*i1* = *i* - 1;  
*iS* = *Suc* *i*;  
(-, -, *di*, -) = *state*;  
(*num*, *denom*) = *quotient-of* α;  
*d-i* = *di* !! *i*;  
*d-i1* = *di* !! *i1*;  
*d-Si* = *di* !! *iS*  
in if *d-i* \* *d-i* \* *denom* ≤ *num* \* *d-i1* \* *d-Si* then  
((*p*, *state*, *g-idx*), *iS*, *j*)  
else (*LLL-adjust-swap-add* *p* *first state* *g-idx* *i*, *i1*, *j* + 1))

**partial-function** (*tailrec*) *LLL-main* :: *int* ⇒ *bool* ⇒ *state-impl* ⇒ *nat* ⇒ *nat* ⇒ *int* ⇒ *int* × *state-impl*

**where**

*LLL-main* *p* *first state* *g-idx* *i* (*j* :: *int*) = (  
 (if *i* < *m*  
 then case *LLL-step* *p* *first state* *g-idx* *i* *j* of  
 ((*p'*, *state'*, *g-idx'*), *i'*, *j')* ⇒  
*LLL-main* *p'* *first state'* *g-idx'* *i'* *j'*  
 else  
(*p*, *state*)))

**partial-function** (*tailrec*) *LLL-iso-main-inner* **where**

*LLL-iso-main-inner* *p* *first state* *di-prods* *g-idx* (*j* :: *int*) = (  
 case *state* of (-, -, *di*, -) ⇒  
 (  
 (let (*max-gso-num*, *max-gso-denum*, *indx*) = *LLL-max-gso-quot* *di-prods*;  
 (*num*, *denum*) = *quotient-of* α in  
 (if *max-gso-num* \* *denum* > *num* \* *max-gso-denum* then  
 case *LLL-adjust-swap-add* *p* *first state* *g-idx* *indx* of  
 (*p'*, *state'*, *g-idx'*) ⇒ case *state'* of (-, -, *di'*, -) ⇒  
 let *di-prods'* = *IArray.of-fun* (λ *i*. case *di-prods* !! *i* of *lr* ⇒  
 if *i* > *indx* ∨ *i* + 2 < *indx* then *lr*  
 else case *lr* of (*l*, *r*)  
 ⇒ if *i* + 1 = *indx* then let *d-idx* = *di'* !! *indx* in (*d-idx* \* *d-idx*, *r*)  
 else (*l*, *di'* !! (*i* + 2) \* *di'* !! *i*)) (*m* - 1)  
 in *LLL-iso-main-inner* *p'* *first state'* *di-prods'* *g-idx'* (*j* + 1)  
 else  
 else

(p, state))))))

**definition** *LLL-iso-main* **where**

*LLL-iso-main* p first state g-idx j = (if m > 1 then  
 case state of (-, -, di, -) ⇒  
 let di-prods = IArray.of-fun (λ i. (di !! (i+1) \* di !! (i+1), di !! (i+2) \* di !!  
 i)) (m - 1)  
 in *LLL-iso-main-inner* p first state di-prods g-idx j else (p, state))

**definition** *LLL-initial* :: bool ⇒ int × state-impl × nat **where**

*LLL-initial* first = (let init = dμ-impl fs-init;  
 di = IArray.of-fun (λ i. if i = 0 then 1 else let i1 = i - 1 in init !! i1 !! i1)  
 (Suc m);  
 (b, g-idx) = *LLL-max-gso-norm-di* first di;  
 p = mod-of-gso-norm first b;  
 mods = IArray.of-fun (λ j. p \* di !! j \* di !! (Suc j)) (m - 1);  
 dmui = IArray.of-fun (λ i. let row = init !! i in IArray.of-fun (λ j. row !! j  
 symmod (mods !! j)) i) m  
 in (p, (compute-initial-mfs p, dmui, di, mods), g-idx))

**primrec** *LLL-add-rows-loop* **where**

*LLL-add-rows-loop* p state i 0 = state  
 | *LLL-add-rows-loop* p state i (Suc j) = (  
 let state' = *LLL-add-row* p state i j  
 in *LLL-add-rows-loop* p state' i j)

**primrec** *LLL-add-rows-outer-loop* **where**

*LLL-add-rows-outer-loop* p state 0 = state |  
*LLL-add-rows-outer-loop* p state (Suc i) =  
 (let state' = *LLL-add-rows-outer-loop* p state i in  
*LLL-add-rows-loop* p state' (Suc i) (Suc i))

**definition**

*LLL-reduce-basis* = (if m = 0 then [] else  
 let first = False;  
 (p0, state0, g-idx0) = *LLL-initial* first;  
 (p, state) = *LLL-main* p0 first state0 g-idx0 0 0;  
 (mfs, -, -, -) = *LLL-add-rows-outer-loop* p state (m - 1)  
 in mfs)

**definition**

*LLL-reduce-basis-iso* = (if m = 0 then [] else  
 let first = False;  
 (p0, state0, g-idx0) = *LLL-initial* first;  
 (p, state) = *LLL-iso-main* p0 first state0 g-idx0 0;  
 (mfs, -, -, -) = *LLL-add-rows-outer-loop* p state (m - 1)  
 in mfs)

**definition**

```

LLL-short-vector = (
  let first = True;
    (p0, state0, g-idx0) = LLL-initial first;
    (p, (mfs,-,-)) = LLL-main p0 first state0 g-idx0 0 0
  in hd mfs)

```

**definition**

```

LLL-short-vector-iso = (
  let first = True;
    (p0, state0, g-idx0) = LLL-initial first;
    (p, (mfs,-,-)) = LLL-iso-main p0 first state0 g-idx0 0
  in hd mfs)

```

**end**

```

declare LLL.LLL-short-vector-def[code]
declare LLL.LLL-short-vector-iso-def[code]
declare LLL.LLL-reduce-basis-def[code]
declare LLL.LLL-reduce-basis-iso-def[code]
declare LLL.LLL-iso-main-def[code]
declare LLL.LLL-iso-main-inner.simps[code]
declare LLL.LLL-initial-def[code]
declare LLL.LLL-main.simps[code]
declare LLL.LLL-adjust-mod-def[code]
declare LLL.LLL-max-gso-norm-def[code]
declare LLL.perform-adjust-mod-def[code]
declare LLL.LLL-max-gso-norm-di-def[code]
declare LLL.LLL-max-gso-quot-def[code]
declare LLL.LLL-step-def[code]
declare LLL.LLL-add-row-def[code]
declare LLL.perform-add-row-def[code]
declare LLL.LLL-swap-row-def[code]
declare LLL.LLL-swap-add-def[code]
declare LLL.LLL-adjust-swap-add-def[code]
declare LLL.perform-swap-add-def[code]
declare LLL.mod-of-gso-norm-def[code]
declare LLL.compute-initial-mfs-def[code]
declare LLL.log-base-def[code]

```

**4.2 Towards soundness proof of implementation****context** *LLL***begin****lemma** *perform-swap-add*: **assumes**  $k: k \neq 0 \ k < m$  **and**  $fs: \text{length } fs = m$ 

**shows**  $LLL\text{-swap-row } p \ (perform\text{-add-row } p \ (fs, \text{dmu}, di, \text{mods}) \ k \ (k - 1) \ c \ (\text{dmu} !! k) \ (\text{dmu} !! k \ (k - 1)) \ (di !! k)) \ k$   
 $= perform\text{-swap-add } p \ (fs, \text{dmu}, di, \text{mods}) \ k \ (k - 1) \ c \ (\text{dmu} !! k) \ (\text{dmu} !! k \ (k - 1)) \ (di !! k)$

*<proof>*

**lemma** *LLL-swap-add-eq*: **assumes**  $i: i \neq 0 \ i < m$  **and**  $fs: \text{length } fs = m$   
**shows**  $LLL\text{-swap-add } p \ (fs, dmu, di, mods) \ i = (LLL\text{-swap-row } p \ (LLL\text{-add-row } p \ (fs, dmu, di, mods) \ i \ (i - 1)) \ i)$   
*<proof>*  
**end**

**context** *LLL-with-assms*  
**begin**

**lemma** *LLL-mod-inv-to-weak*:  $LLL\text{-invariant-mod } fs \ mfs \ dmu \ p \ \text{first } b \ i \implies LLL\text{-invariant-mod-weak } fs \ mfs \ dmu \ p \ \text{first } b$   
*<proof>*

**declare** *IArray.of-fun-def[simp del]*

**lemma** *LLL-swap-row*: **assumes**  $impl: \text{state-impl-inv } p \ mfs \ dmu \ \text{state}$   
**and**  $Liniv: LLL\text{-invariant-mod-weak } fs \ mfs \ dmu \ p \ \text{first } b$   
**and**  $res: \text{basis-reduction-mod-swap } p \ mfs \ dmu \ k = (mfs', dmu')$   
**and**  $res': LLL\text{-swap-row } p \ \text{state } k = \text{state}'$   
**and**  $k: k < m \ k \neq 0$   
**shows**  $\text{state-impl-inv } p \ mfs' \ dmu' \ \text{state}'$   
*<proof>*

**lemma** *LLL-add-row*: **assumes**  $impl: \text{state-impl-inv } p \ mfs \ dmu \ \text{state}$   
**and**  $Liniv: LLL\text{-invariant-mod-weak } fs \ mfs \ dmu \ p \ \text{first } b$   
**and**  $res: \text{basis-reduction-mod-add-row } p \ mfs \ dmu \ i \ j = (mfs', dmu')$   
**and**  $res': LLL\text{-add-row } p \ \text{state } i \ j = \text{state}'$   
**and**  $i: i < m$   
**and**  $j: j < i$   
**shows**  $\text{state-impl-inv } p \ mfs' \ dmu' \ \text{state}'$   
*<proof>*

**lemma** *LLL-max-gso-norm-di*: **assumes**  $di: di = IArray.of-fun \ (d\text{-of } dmu) \ (Suc \ m)$   
**and**  $m: m \neq 0$   
**shows**  $LLL\text{-max-gso-norm-di } \text{first } di = \text{compute-max-gso-norm } \text{first } dmu$   
*<proof>*

**lemma** *LLL-max-gso-quot*: **assumes**  $di: di = IArray.of-fun \ (d\text{-of } dmu) \ (Suc \ m)$   
**and**  $prods: \text{state-iso-inv } di\text{-prods } di$   
**shows**  $LLL\text{-max-gso-quot } di\text{-prods} = \text{compute-max-gso-quot } dmu$   
*<proof>*

**lemma** *LLL-max-gso-norm*: **assumes** *impl: state-impl-inv p mfs dmu state*  
**and** *m: m ≠ 0*  
**shows** *LLL-max-gso-norm first state = compute-max-gso-norm first dmu*  
*<proof>*

**lemma** *mod-of-gso-norm*: *m ≠ 0 ⇒ mod-of-gso-norm first mn =*  
*compute-mod-of-max-gso-norm first mn*  
*<proof>*

**lemma** *LLL-adjust-mod*: **assumes** *impl: state-impl-inv p mfs dmu state*  
**and** *res: basis-reduction-adjust-mod p first mfs dmu = (p', mfs', dmu', g-idx)*  
**and** *res': LLL-adjust-mod p first state = (p'', state', g-idx')*  
**and** *m: m ≠ 0*  
**shows** *state-impl-inv p' mfs' dmu' state' ∧ p'' = p' ∧ g-idx' = g-idx*  
*<proof>*

**lemma** *LLL-adjust-swap-add*: **assumes** *impl: state-impl-inv p mfs dmu state*  
**and** *Liniv: LLL-invariant-mod-weak fs mfs dmu p first b*  
**and** *res: basis-reduction-adjust-swap-add-step p first mfs dmu g-idx k = (p', mfs', dmu', g-idx')*  
**and** *res': LLL-adjust-swap-add p first state g-idx k = (p'', state', G-idx')*  
**and** *k: k < m and k0: k ≠ 0*  
**shows** *state-impl-inv p' mfs' dmu' state' p'' = p' G-idx' = g-idx'*  
*i ≤ m ⇒ i ≠ k ⇒ di-of state' !! i = di-of state !! i*  
*<proof>*

**lemma** *LLL-step*: **assumes** *impl: state-impl-inv p mfs dmu state*  
**and** *Liniv: LLL-invariant-mod-weak fs mfs dmu p first b*  
**and** *res: basis-reduction-mod-step p first mfs dmu g-idx k j = (p', mfs', dmu', g-idx', k', j')*  
**and** *res': LLL-step p first state g-idx k j = ((p'', state', g-idx''), k'', j'')*  
**and** *k: k < m*  
**shows** *state-impl-inv p' mfs' dmu' state' ∧ k'' = k' ∧ p'' = p' ∧ j'' = j' ∧ g-idx'' = g-idx'*  
*<proof>*

**lemma** *LLL-main*: **assumes** *impl: state-impl-inv p mfs dmu state*  
**and** *Liniv: LLL-invariant-mod fs mfs dmu p first b i*  
**and** *res: basis-reduction-mod-main p first mfs dmu g-idx i k = (p', mfs', dmu')*  
**and** *res': LLL-main p first state g-idx i k = (pi', state')*  
**shows** *state-impl-inv p' mfs' dmu' state' ∧ pi' = p'*  
*<proof>*

**lemma** *LLL-iso-main-inner*: **assumes** *impl: state-impl-inv p mfs dmu state*  
**and** *di-prods: state-iso-inv di-prods (di-of state)*  
**and** *Liniv: LLL-invariant-mod-weak fs mfs dmu p first b*

**and** *res*: *basis-reduction-iso-main* *p first mfs dmU g-idx k* = (*p'*, *mfs'*, *dmU'*)  
**and** *res'*: *LLL-iso-main-inner* *p first state di-prods g-idx k* = (*pi'*, *state'*)  
**and** *m*:  $m > 1$   
**shows** *state-impl-inv* *p' mfs' dmU' state'  $\wedge$  pi' = p'*  
*<proof>*

**lemma** *LLL-iso-main*: **assumes** *impl*: *state-impl-inv* *p mfs dmU state*  
**and** *LinV*: *LLL-invariant-mod-weak* *fs mfs dmU p first b*  
**and** *res*: *basis-reduction-iso-main* *p first mfs dmU g-idx k* = (*p'*, *mfs'*, *dmU'*)  
**and** *res'*: *LLL-iso-main* *p first state g-idx k* = (*pi'*, *state'*)  
**shows** *state-impl-inv* *p' mfs' dmU' state'  $\wedge$  pi' = p'*  
*<proof>*

**lemma** *LLL-initial*: **assumes** *res*: *compute-initial-state* *first* = (*p*, *mfs*, *dmU*, *g-idx*)  
**and** *res'*: *LLL-initial* *first* = (*p'*, *state*, *g-idx'*)  
**and** *m*:  $m \neq 0$   
**shows** *state-impl-inv* *p mfs dmU state  $\wedge$  p' = p  $\wedge$  g-idx' = g-idx*  
*<proof>*

**lemma** *LLL-add-rows-loop*: **assumes** *impl*: *state-impl-inv* *p mfs dmU state*  
**and** *LinV*: *LLL-invariant-mod* *fs mfs dmU p b first i*  
**and** *res*: *basis-reduction-mod-add-rows-loop* *p mfs dmU i j* = (*mfs'*, *dmU'*)  
**and** *res'*: *LLL-add-rows-loop* *p state i j* = *state'*  
**and** *j*:  $j \leq i$   
**and** *i*:  $i < m$   
**shows** *state-impl-inv* *p mfs' dmU' state'*  
*<proof>*

**lemma** *LLL-add-rows-outer-loop*: **assumes** *impl*: *state-impl-inv* *p mfs dmU state*  
**and** *LinV*: *LLL-invariant-mod* *fs mfs dmU p first b m*  
**and** *res*: *basis-reduction-mod-add-rows-outer-loop* *p mfs dmU i* = (*mfs'*, *dmU'*)  
**and** *res'*: *LLL-add-rows-outer-loop* *p state i* = *state'*  
**and** *i*:  $i \leq m - 1$   
**shows** *state-impl-inv* *p mfs' dmU' state'*  
*<proof>*

### 4.3 Soundness of implementation

We just prove that the concrete implementations have the same input-output-behaviour as the abstract versions of Storjohann's algorithms.

**lemma** *LLL-reduce-basis*: *LLL-reduce-basis* = *reduce-basis-mod*  
*<proof>*

**lemma** *LLL-reduce-basis-iso*: *LLL-reduce-basis-iso* = *reduce-basis-iso*  
*<proof>*

**lemma** *LLL-short-vector*: **assumes** *m*:  $m \neq 0$   
**shows** *LLL-short-vector* = *short-vector-mod*

*<proof>*

**lemma** *LLL-short-vector-iso*: **assumes**  $m: m \neq 0$   
**shows** *LLL-short-vector-iso* = *short-vector-iso*  
*<proof>*

**end**

**end**

## 5 Generalization of the statement about the uniqueness of the Hermite normal form

**theory** *Uniqueness-Hermite*  
**imports** *Hermite.Hermite*  
**begin**

**instance** *int* :: *bezout-ring-div*  
*<proof>*

**lemma** *map-matrix-rat-of-int-mult*:  
**shows** *map-matrix rat-of-int* (A\*\*B) = (*map-matrix rat-of-int* A)\*\*(*map-matrix rat-of-int* B)  
*<proof>*

**lemma** *det-map-matrix*:  
**fixes**  $A :: \text{int}^{\wedge n} :: \text{mod-type}^{\wedge n} :: \text{mod-type}$   
**shows** *det* (*map-matrix rat-of-int* A) = *rat-of-int* (*det* A)  
*<proof>*

**lemma** *inv-Z-imp-inv-Q*:  
**fixes**  $A :: \text{int}^{\wedge n} :: \text{mod-type}^{\wedge n} :: \text{mod-type}$   
**assumes** *inv-A*: *invertible* A  
**shows** *invertible* (*map-matrix rat-of-int* A)  
*<proof>*

**lemma** *upper-triangular-Z-eq-Q*:  
*upper-triangular* (*map-matrix rat-of-int* A) = *upper-triangular* A  
*<proof>*

**lemma** *invertible-and-upper-diagonal-not0*:  
**fixes**  $H :: \text{int}^{\wedge n} :: \text{mod-type}^{\wedge n} :: \text{mod-type}$   
**assumes** *inv-H*: *invertible* (*map-matrix rat-of-int* H) **and** *up-H*: *upper-triangular* H  
**shows**  $H \$ i \$ i \neq 0$   
*<proof>*

**lemma** *diagonal-least-nonzero*:  
**fixes**  $H :: \text{int}^{\wedge} n :: \text{mod-type}^{\wedge} n :: \text{mod-type}$   
**assumes**  $H$ : *Hermite associates residues*  $H$   
**and**  $\text{inv-}H$ : *invertible* ( $\text{map-matrix rat-of-int } H$ ) **and**  $\text{up-}H$ : *upper-triangular*  $H$   
**shows** ( $\text{LEAST } n. H \$ i \$ n \neq 0$ ) =  $i$   
 $\langle \text{proof} \rangle$

**lemma** *diagonal-in-associates*:  
**fixes**  $H :: \text{int}^{\wedge} n :: \text{mod-type}^{\wedge} n :: \text{mod-type}$   
**assumes**  $H$ : *Hermite associates residues*  $H$   
**and**  $\text{inv-}H$ : *invertible* ( $\text{map-matrix rat-of-int } H$ ) **and**  $\text{up-}H$ : *upper-triangular*  $H$   
**shows**  $H \$ i \$ i \in \text{associates}$   
 $\langle \text{proof} \rangle$

**lemma** *above-diagonal-in-residues*:  
**fixes**  $H :: \text{int}^{\wedge} n :: \text{mod-type}^{\wedge} n :: \text{mod-type}$   
**assumes**  $H$ : *Hermite associates residues*  $H$   
**and**  $\text{inv-}H$ : *invertible* ( $\text{map-matrix rat-of-int } H$ ) **and**  $\text{up-}H$ : *upper-triangular*  $H$   
**and**  $j-i: j < i$   
**shows**  $H \$ j \$ (\text{LEAST } n. H \$ i \$ n \neq 0) \in \text{residues } (H \$ i \$ (\text{LEAST } n. H \$ i \$ n \neq 0))$   
 $\langle \text{proof} \rangle$

**lemma** *Hermite-unique-generalized*:  
**fixes**  $K :: \text{int}^{\wedge} n :: \text{mod-type}^{\wedge} n :: \text{mod-type}$   
**assumes**  $A$ - $PH$ :  $A = P ** H$   
**and**  $A$ - $QK$ :  $A = Q ** K$   
**and**  $\text{inv-}A$ : *invertible* ( $\text{map-matrix rat-of-int } A$ )  
**and**  $\text{inv-}P$ : *invertible*  $P$   
**and**  $\text{inv-}Q$ : *invertible*  $Q$   
**and**  $H$ : *Hermite associates residues*  $H$   
**and**  $K$ : *Hermite associates residues*  $K$   
**shows**  $H = K$   
 $\langle \text{proof} \rangle$

**end**

## 6 Uniqueness of Hermite normal form in JNF

This theory contains the proof of the uniqueness theorem of the Hermite normal form in JNF, moved from HOL Analysis.

**theory** *Uniqueness-Hermite-JNF*  
**imports**  
*Hermite.Hermite*  
*Uniqueness-Hermite*  
*Smith-Normal-Form.SNF-Missing-Lemmas*

*Smith-Normal-Form.Mod-Type-Connect*  
*Smith-Normal-Form.Finite-Field-Mod-Type-Connection*

**begin**

**hide-const** (**open**) *residues*

We first define some properties that currently exist in HOL Analysis, but not in JNF, namely a predicate for being in echelon form, another one for being in Hermite normal form, definition of a row of zeros up to a concrete position, and so on.

**definition** *is-zero-row-upt-k-JNF* :: *nat => nat => 'a::{zero} mat => bool*  
**where** *is-zero-row-upt-k-JNF* *i k A* = ( $\forall j. j < k \longrightarrow A \$\$ (i, j) = 0$ )

**definition** *is-zero-row-JNF* :: *nat => 'a::{zero} mat => bool*  
**where** *is-zero-row-JNF* *i A* = ( $\forall j < \dim\text{-col } A. A \$\$ (i, j) = 0$ )

**lemma** *echelon-form-def'*:

*echelon-form* *A* = (  
 $(\forall i. \text{is-zero-row } i \ A \longrightarrow \neg (\exists j. j > i \wedge \neg \text{is-zero-row } j \ A))$   
 $\wedge$   
 $(\forall i \ j. i < j \wedge \neg (\text{is-zero-row } i \ A) \wedge \neg (\text{is-zero-row } j \ A)$   
 $\longrightarrow ((\text{LEAST } n. A \$ i \$ n \neq 0) < (\text{LEAST } n. A \$ j \$ n \neq 0)))$   
 $\langle \text{proof} \rangle$

**definition**

*echelon-form-JNF* :: *'a::{bezout-ring} mat => bool*  
**where**  
*echelon-form-JNF* *A* = (  
 $(\forall i < \dim\text{-row } A. \text{is-zero-row-JNF } i \ A \longrightarrow \neg (\exists j. j < \dim\text{-row } A \wedge j > i \wedge \neg \text{is-zero-row-JNF } j \ A))$   
 $\wedge$   
 $(\forall i \ j. i < j \wedge j < \dim\text{-row } A \wedge \neg (\text{is-zero-row-JNF } i \ A) \wedge \neg (\text{is-zero-row-JNF } j \ A)$   
 $\longrightarrow ((\text{LEAST } n. A \$\$ (i, n) \neq 0) < (\text{LEAST } n. A \$\$ (j, n) \neq 0)))$   
 $A$ )

Now, we connect the existing definitions in HOL Analysis to the ones just defined in JNF by means of transfer rules.

**context includes** *lifting-syntax*

**begin**

**lemma** *HMA-is-zero-row-mod-type[transfer-rule]*:

$((\text{Mod-Type-Connect.HMA-I}) \implies (\text{Mod-Type-Connect.HMA-M} :: - \Rightarrow 'a :: \text{comm-ring-1 } \hat{\ } 'n :: \text{mod-type } \hat{\ } 'm :: \text{mod-type} \Rightarrow -)$   
 $\implies (=)) \text{is-zero-row-JNF is-zero-row}$   
 $\langle \text{proof} \rangle$

**lemma** *HMA-echelon-form-mod-type[transfer-rule]*:

$((\text{Mod-Type-Connect.HMA-M} :: - \Rightarrow 'a :: \text{bezout-ring} \wedge 'n :: \text{mod-type} \wedge 'm :: \text{mod-type} \Rightarrow -) \implies (=))$   
*echelon-form-JNF echelon-form*  
 <proof>

**definition Hermite-JNF** ::  $'a :: \{\text{bezout-ring-div}, \text{normalization-semidom}\} \text{ set} \Rightarrow ('a \Rightarrow 'a \text{ set}) \Rightarrow 'a \text{ mat} \Rightarrow \text{bool}$   
**where** *Hermite-JNF associates residues*  $A = ($   
*Complete-set-non-associates associates*  $\wedge ($  *Complete-set-residues residues*  $) \wedge$  *echelon-form-JNF*  $A$   
 $\wedge (\forall i < \text{dim-row } A. \neg \text{is-zero-row-JNF } i \ A \longrightarrow A \ \$\$ (i, \text{LEAST } n. A \ \$\$ (i, n) \neq 0) \in \text{associates})$   
 $\wedge (\forall i < \text{dim-row } A. \neg \text{is-zero-row-JNF } i \ A \longrightarrow (\forall j. j < i \longrightarrow A \ \$\$ (j, (\text{LEAST } n. A \ \$\$ (i, n) \neq 0)))$   
 $\in \text{residues } (A \ \$\$ (i, (\text{LEAST } n. A \ \$\$ (i, n) \neq 0)))$   
 $)))$

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

**assumes**  $AA'$ :  $(\text{Mod-Type-Connect.HMA-M} :: - \Rightarrow 'a :: \text{comm-ring-1} \wedge 'n :: \text{mod-type} \wedge 'm :: \text{mod-type} \Rightarrow -) \ A \ A'$   
**and**  $ii'$ :  $\text{Mod-Type-Connect.HMA-I } i \ i'$  **and**  $\text{zero-i}$ :  $\neg \text{is-zero-row-JNF } i \ A$   
**shows**  $\text{Mod-Type-Connect.HMA-I } (\text{LEAST } n. A \ \$\$ (i, n) \neq 0) \ (\text{LEAST } n. \text{index-hma } A' \ i' \ n \neq 0)$   
 <proof>

**lemma element-least-not-zero-eq-HMA-JNF**:

**fixes**  $A'$ :  $'a :: \text{comm-ring-1} \wedge 'n :: \text{mod-type} \wedge 'm :: \text{mod-type}$   
**assumes**  $AA'$ :  $\text{Mod-Type-Connect.HMA-M } A \ A'$  **and**  $jj'$ :  $\text{Mod-Type-Connect.HMA-I } j \ j'$   
**and**  $ii'$ :  $\text{Mod-Type-Connect.HMA-I } i \ i'$  **and**  $\text{zero-i'}$ :  $\neg \text{is-zero-row } i' \ A'$   
**shows**  $A \ \$\$ (j, \text{LEAST } n. A \ \$\$ (i, n) \neq 0) = A' \ \$\$ j' \ \$\$ (\text{LEAST } n. A' \ \$\$ i' \ \$\$ n \neq 0)$   
 <proof>

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

**shows**  $((\text{Mod-Type-Connect.HMA-M} :: - \Rightarrow 'a :: \{\text{bezout-ring-div}, \text{normalization-semidom}\} \wedge 'n :: \text{mod-type} \wedge 'm :: \text{mod-type} \Rightarrow -) \implies (=))$   
*(Hermite-JNF associates residues) (Hermite associates residues)*  
 <proof>

**corollary HMA-Hermite2**[transfer-rule]:

**shows**  $((=) \implies (=) \implies (\text{Mod-Type-Connect.HMA-M} :: - \Rightarrow 'a :: \{\text{bezout-ring-div}, \text{normalization-semidom}\} \wedge 'n :: \text{mod-type} \wedge 'm :: \text{mod-type} \Rightarrow -) \implies (=))$

(*Hermite-JNF*) (*Hermite*)  
 ⟨*proof*⟩

Once the definitions of both libraries are connected, we start to move the theorem about the uniqueness of the Hermite normal form (stated in HOL Analysis, named *Hermite-unique*) to JNF.

Using the previous transfer rules, we get an statement in JNF. However, the matrices have  $CARD('n::mod-type)$  rows and columns. We want to get rid of that type variable and just state that they are of dimension  $n \times n$  (expressed via the predicate *carrier-mat*

**lemma** *Hermite-unique-JNF'*:

**fixes**  $A::'a::\{bezout-ring-div,normalization-euclidean-semiring,unique-euclidean-ring\}$   
*mat*

**assumes**  $A \in carrier-mat\ CARD('n::mod-type)\ CARD('n::mod-type)$   
 $P \in carrier-mat\ CARD('n::mod-type)\ CARD('n::mod-type)$   
 $H \in carrier-mat\ CARD('n::mod-type)\ CARD('n::mod-type)$   
 $Q \in carrier-mat\ CARD('n::mod-type)\ CARD('n::mod-type)$   
 $K \in carrier-mat\ CARD('n::mod-type)\ CARD('n::mod-type)$

**assumes**  $A = P * H$

**and**  $A = Q * K$  **and** *invertible-mat*  $A$  **and** *invertible-mat*  $P$

**and** *invertible-mat*  $Q$  **and** *Hermite-JNF associates res*  $H$  **and** *Hermite-JNF associates res*  $K$

**shows**  $H = K$

⟨*proof*⟩

Since the *mod-type* restriction relies on many things, the shortcut is to use the *mod-ring* typedef developed in the Berlekamp-Zassenhaus development. This type definition allows us to apply local type definitions easily. Since *mod-ring* is just an instance of *mod-type*, it is straightforward to obtain the following lemma, where  $CARD('n::mod-type)$  has now been substituted by  $CARD('n::nontriv\ mod-ring)$

**corollary** *Hermite-unique-JNF-with-nontriv-mod-ring*:

**fixes**  $A::'a::\{bezout-ring-div,normalization-euclidean-semiring,unique-euclidean-ring\}$   
*mat*

**assumes**  $A \in carrier-mat\ CARD('n)\ CARD('n::nontriv\ mod-ring)$   
 $P \in carrier-mat\ CARD('n)\ CARD('n)$   
 $H \in carrier-mat\ CARD('n)\ CARD('n)$   
 $Q \in carrier-mat\ CARD('n)\ CARD('n)$   
 $K \in carrier-mat\ CARD('n)\ CARD('n)$

**assumes**  $A = P * H$

**and**  $A = Q * K$  **and** *invertible-mat*  $A$  **and** *invertible-mat*  $P$

**and** *invertible-mat*  $Q$  **and** *Hermite-JNF associates res*  $H$  **and** *Hermite-JNF associates res*  $K$

**shows**  $H = K$  ⟨*proof*⟩

Now, we assume in a context that there exists a type text  $'b$  of cardinality  $n$  and we prove inside this context the lemma.

```

context
  fixes  $n::nat$ 
  assumes local-typedef:  $\exists (Rep :: ('b \Rightarrow int)) Abs. type-definition Rep Abs \{0..<n$ 
   $:: int\}$ 
  and  $p: n>1$ 
begin

```

```

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

```

```

lemma Hermite-unique-JNF-aux:
  fixes  $A::'a::\{bezout-ring-div,normalization-euclidean-semiring,unique-euclidean-ring\}$ 
  mat
  assumes  $A \in carrier-mat\ n\ n$ 
   $P \in carrier-mat\ n\ n$ 
   $H \in carrier-mat\ n\ n$ 
   $Q \in carrier-mat\ n\ n$ 
   $K \in carrier-mat\ n\ n$ 
  assumes  $A = P * H$ 
  and  $A = Q * K$  and invertible-mat A and invertible-mat P
  and invertible-mat Q and Hermite-JNF associates res H and Hermite-JNF
associates res K
shows  $H = K$ 
   $\langle proof \rangle$ 
end

```

Now, we cancel the local type definition of the previous context. Since the *mod-type* restriction imposes the type to have cardinality greater than 1, the cases  $n = 0$  and  $n = 1$  must be proved separately (they are trivial)

```

lemma Hermite-unique-JNF:
  fixes  $A::'a::\{bezout-ring-div,normalization-euclidean-semiring,unique-euclidean-ring\}$ 
  mat
  assumes  $A: A \in carrier-mat\ n\ n$  and  $P: P \in carrier-mat\ n\ n$  and  $H: H \in$ 
carrier-mat n n
  and  $Q: Q \in carrier-mat\ n\ n$  and  $K: K \in carrier-mat\ n\ n$ 
  assumes A-PH: A = P * H and A-QK: A = Q * K
  and inv-A: invertible-mat A and inv-P: invertible-mat P and inv-Q: invert-
ible-mat Q
  and HNF-H: Hermite-JNF associates res H and HNF-K: Hermite-JNF asso-
ciates res K
  shows  $H = K$ 
   $\langle proof \rangle$ 
end

```

From here on, we apply the same approach to move the new generalized statement about the uniqueness Hermite normal form, i.e., the version re-

stricted to integer matrices, but imposing invertibility over the rationals.

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

((=) ==> *Mod-Type-Connect.HMA-M* ==> *Mod-Type-Connect.HMA-M*)  
*map-mat map-matrix*  
 ⟨*proof*⟩

**lemma** *Hermite-unique-generalized-JNF'*:

**fixes** *A::int mat*  
**assumes** *A ∈ carrier-mat CARD('n::mod-type) CARD('n::mod-type)*  
*P ∈ carrier-mat CARD('n::mod-type) CARD('n::mod-type)*  
*H ∈ carrier-mat CARD('n::mod-type) CARD('n::mod-type)*  
*Q ∈ carrier-mat CARD('n::mod-type) CARD('n::mod-type)*  
*K ∈ carrier-mat CARD('n::mod-type) CARD('n::mod-type)*  
**assumes** *A = P \* H*  
**and** *A = Q \* K* **and** *invertible-mat (map-mat rat-of-int A)* **and** *invertible-mat P*  
**and** *invertible-mat Q* **and** *Hermite-JNF associates res H* **and** *Hermite-JNF associates res K*  
**shows** *H = K*  
 ⟨*proof*⟩

**corollary** *Hermite-unique-generalized-JNF-with-nontriv-mod-ring*:

**fixes** *A::int mat*  
**assumes** *A ∈ carrier-mat CARD('n) CARD('n::nontriv mod-ring)*  
*P ∈ carrier-mat CARD('n) CARD('n)*  
*H ∈ carrier-mat CARD('n) CARD('n)*  
*Q ∈ carrier-mat CARD('n) CARD('n)*  
*K ∈ carrier-mat CARD('n) CARD('n)*  
**assumes** *A = P \* H*  
**and** *A = Q \* K* **and** *invertible-mat (map-mat rat-of-int A)* **and** *invertible-mat P*  
**and** *invertible-mat Q* **and** *Hermite-JNF associates res H* **and** *Hermite-JNF associates res K*  
**shows** *H = K* ⟨*proof*⟩

**context**

**fixes** *p::nat*  
**assumes** *local-typedef: ∃(Rep :: ('b ⇒ int)) Abs. type-definition Rep Abs {0..<p :: int}*  
**and** *p: p>1*  
**begin**

**private lemma** *type-to-set2*:

**shows** *class.nontriv TYPE('b) (is ?a) and p=CARD('b) (is ?b)*  
 ⟨*proof*⟩

**lemma** *Hermite-unique-generalized-JNF-aux:*

**fixes** *A::int mat*  
**assumes** *A ∈ carrier-mat p p*  
*P ∈ carrier-mat p p*  
*H ∈ carrier-mat p p*  
*Q ∈ carrier-mat p p*  
*K ∈ carrier-mat p p*  
**assumes** *A = P \* H*  
**and** *A = Q \* K* **and** *invertible-mat (map-mat rat-of-int A)* **and** *invertible-mat P*  
**and** *invertible-mat Q* **and** *Hermite-JNF associates res H* **and** *Hermite-JNF associates res K*  
**shows** *H = K*  
 ⟨*proof*⟩  
**end**

**lemma** *HNF-unique-generalized-JNF:*

**fixes** *A::int mat*  
**assumes** *A: A ∈ carrier-mat n n* **and** *P: P ∈ carrier-mat n n* **and** *H: H ∈ carrier-mat n n*  
**and** *Q: Q ∈ carrier-mat n n* **and** *K: K ∈ carrier-mat n n*  
**assumes** *A-PH: A = P \* H* **and** *A-QK: A = Q \* K*  
**and** *inv-A: invertible-mat (map-mat rat-of-int A)* **and** *inv-P: invertible-mat P*  
**and** *inv-Q: invertible-mat Q*  
**and** *HNF-H: Hermite-JNF associates res H* **and** *HNF-K: Hermite-JNF associates res K*  
**shows** *H = K*  
 ⟨*proof*⟩

**end**

## 7 Formalization of an efficient Hermite normal form algorithm

We formalize a version of the Hermite normal form algorithm based on reductions modulo the determinant. This avoids the growth of the intermediate coefficients.

## 7.1 Implementation of the algorithm using generic modulo operation

Exception on generic modulo: currently in Hermite-reduce-above, ordinary div/mod is used, since that is our choice for the complete set of residues.

**theory** *HNF-Mod-Det-Algorithm*

**imports**

*Jordan-Normal-Form.Gauss-Jordan-IArray-Impl*

*Show.Show-Instances*

*Jordan-Normal-Form.Determinant-Impl*

*Jordan-Normal-Form.Show-Matrix*

*LLL-Basis-Reduction.LLL-Certification*

*Smith-Normal-Form.SNF-Algorithm-Euclidean-Domain*

*Smith-Normal-Form.SNF-Missing-Lemmas*

*Uniqueness-Hermite-JNF*

**begin**

### 7.1.1 Echelon form algorithm

**fun** *make-first-column-positive* :: *int mat*  $\Rightarrow$  *int mat* **where**

*make-first-column-positive* *A* = (

*Matrix.mat* (*dim-row A*) (*dim-col A*) — Create a matrix of the same dimensions

( $\lambda(i,j).$  if  $A \ \mathbb{S}(i,0) < 0$  then  $- A \ \mathbb{S}(i,j)$  else  $A \ \mathbb{S}(i,j)$ )

)

)

**locale** *mod-operation* =

**fixes** *generic-mod* :: *int*  $\Rightarrow$  *int*  $\Rightarrow$  *int* (**infixl**  $\langle gmod \rangle$  70)

**and** *generic-div* :: *int*  $\Rightarrow$  *int*  $\Rightarrow$  *int* (**infixl**  $\langle gdiv \rangle$  70)

**begin**

Version for reducing all elements

**fun** *reduce* :: *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *int*  $\Rightarrow$  *int mat*  $\Rightarrow$  *int mat* **where**

*reduce* *a b D A* = (let *Aaj* =  $A \ \mathbb{S}(a,0)$ ; *Abj* =  $A \ \mathbb{S}(b,0)$ )

*in*

if *Aaj* = 0 then *A* else

case *euclid-ext2 Aaj Abj of* (*p,q,u,v,d*)  $\Rightarrow$   $- p * Aaj + q * Abj = d, u = - Abj/d,$

*v* =  $Aaj/d$

*Matrix.mat* (*dim-row A*) (*dim-col A*) — Create a matrix of the same dimensions

( $\lambda(i,k).$  if  $i = a$  then let  $r = (p * A \ \mathbb{S}(a,k) + q * A \ \mathbb{S}(b,k))$  in

if  $k = 0$  then if  $D$  dvd  $r$  then  $D$  else  $r$  else  $r \ gmod \ D$  —

Row *a* is multiplied by *p* and added row *b* multiplied by *q*, modulo *D*

else if  $i = b$  then let  $r = u * A \ \mathbb{S}(a,k) + v * A \ \mathbb{S}(b,k)$  in

if  $k = 0$  then  $r$  else  $r \ gmod \ D$  — Row *b* is multiplied by *v*

and added row *a* multiplied by *u*, modulo *D*

else  $A \ \mathbb{S}(i,k)$  — All the other rows remain unchanged

)

)

Version for reducing, with abs-checking

**fun** *reduce-abs* :: *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *int*  $\Rightarrow$  *int mat*  $\Rightarrow$  *int mat* **where**  
*reduce-abs* *a b D A* = (*let* *Aaj* =  $A\$(a,0)$ ; *Abj* =  $A\$(b,0)$   
*in*  
*if* *Aaj* = 0 *then* *A* *else*  
*case* *euclid-ext2* *Aaj Abj* *of* (*p,q,u,v,d*)  $\Rightarrow$   $-p * Aaj + q * Abj = d, u = -Abj/d,$   
 $v = Aaj/d$   
*Matrix.mat* (*dim-row A*) (*dim-col A*) — Create a matrix of the same dimensions  
 $(\lambda(i,k). \textit{if } i = a \textit{ then let } r = (p * A\$(a,k) + q * A\$(b,k)) \textit{ in}$   
 $\textit{if } \textit{abs } r > D \textit{ then if } k = 0 \wedge D \textit{ dvd } r \textit{ then } D \textit{ else } r \textit{ gmod } D$   
*else* *r*  
 $\textit{else if } i = b \textit{ then let } r = u * A\$(a,k) + v * A\$(b,k) \textit{ in}$   
 $\textit{if } \textit{abs } r > D \textit{ then } r \textit{ gmod } D \textit{ else } r$   
 $\textit{else } A\$(i,k)$  — All the other rows remain unchanged  
 $)$   
 $)$

**definition** *reduce-impl* :: *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *int*  $\Rightarrow$  *int mat*  $\Rightarrow$  *int mat* **where**  
*reduce-impl* *a b D A* = (*let*  
*row-a* = *Matrix.row A a*;  
*Aaj* = *row-a \$v 0*  
*in*  
*if* *Aaj* = 0 *then* *A* *else let*  
*row-b* = *Matrix.row A b*;  
*Abj* = *row-b \$v 0* *in*  
*case* *euclid-ext2* *Aaj Abj* *of* (*p,q,u,v,d*)  $\Rightarrow$   
 $\textit{let } \textit{row-a}' = (\lambda k ak. \textit{let } r = (p * ak + q * \textit{row-b } \$v k) \textit{ in}$   
 $\textit{if } k = 0 \textit{ then if } D \textit{ dvd } r \textit{ then } D \textit{ else } r \textit{ else } r \textit{ gmod } D);$   
 $\textit{row-b}' = (\lambda k bk. \textit{let } r = u * \textit{row-a } \$v k + v * bk \textit{ in}$   
 $\textit{if } k = 0 \textit{ then } r \textit{ else } r \textit{ gmod } D)$   
 $\textit{in change-row a row-a}' (\textit{change-row b row-b}' A)$   
 $)$

**definition** *reduce-abs-impl* :: *nat*  $\Rightarrow$  *nat*  $\Rightarrow$  *int*  $\Rightarrow$  *int mat*  $\Rightarrow$  *int mat* **where**  
*reduce-abs-impl* *a b D A* = (*let*  
*row-a* = *Matrix.row A a*;  
*Aaj* = *row-a \$v 0*  
*in*  
*if* *Aaj* = 0 *then* *A* *else let*  
*row-b* = *Matrix.row A b*;  
*Abj* = *row-b \$v 0* *in*  
*case* *euclid-ext2* *Aaj Abj* *of* (*p,q,u,v,d*)  $\Rightarrow$   
 $\textit{let } \textit{row-a}' = (\lambda k ak. \textit{let } r = (p * ak + q * \textit{row-b } \$v k) \textit{ in}$   
 $\textit{if } \textit{abs } r > D \textit{ then if } k = 0 \wedge D \textit{ dvd } r \textit{ then } D \textit{ else } r \textit{ gmod } D \textit{ else } r);$   
 $\textit{row-b}' = (\lambda k bk. \textit{let } r = u * \textit{row-a } \$v k + v * bk \textit{ in}$   
 $\textit{if } \textit{abs } r > D \textit{ then } r \textit{ gmod } D \textit{ else } r)$   
 $\textit{in change-row a row-a}' (\textit{change-row b row-b}' A)$   
 $)$

**lemma** *reduce-impl*:  $a < nr \implies b < nr \implies 0 < nc \implies a \neq b \implies A \in \text{carrier-mat } nr \ nc$   
 $\implies \text{reduce-impl } a \ b \ D \ A = \text{reduce } a \ b \ D \ A$   
 ⟨proof⟩

**lemma** *reduce-abs-impl*:  $a < nr \implies b < nr \implies 0 < nc \implies a \neq b \implies A \in \text{carrier-mat } nr \ nc$   
 $\implies \text{reduce-abs-impl } a \ b \ D \ A = \text{reduce-abs } a \ b \ D \ A$   
 ⟨proof⟩

**fun** *reduce-below* ::  $\text{nat} \Rightarrow \text{nat list} \Rightarrow \text{int} \Rightarrow \text{int mat} \Rightarrow \text{int mat}$   
**where** *reduce-below*  $a \ [] \ D \ A = A$   
 | *reduce-below*  $a \ (x \# \ xs) \ D \ A = \text{reduce-below } a \ xs \ D \ (\text{reduce } a \ x \ D \ A)$

**fun** *reduce-below-impl* ::  $\text{nat} \Rightarrow \text{nat list} \Rightarrow \text{int} \Rightarrow \text{int mat} \Rightarrow \text{int mat}$   
**where** *reduce-below-impl*  $a \ [] \ D \ A = A$   
 | *reduce-below-impl*  $a \ (x \# \ xs) \ D \ A = \text{reduce-below-impl } a \ xs \ D \ (\text{reduce-impl } a \ x \ D \ A)$

**lemma** *reduce-impl-carrier*[*simp,intro*]:  $A \in \text{carrier-mat } m \ n \implies \text{reduce-impl } a \ b \ D \ A \in \text{carrier-mat } m \ n$   
 ⟨proof⟩

**lemma** *reduce-below-impl*:  $a < nr \implies 0 < nc \implies (\bigwedge b. b \in \text{set } bs \implies b < nr) \implies a \notin \text{set } bs$   
 $\implies A \in \text{carrier-mat } nr \ nc \implies \text{reduce-below-impl } a \ bs \ D \ A = \text{reduce-below } a \ bs \ D \ A$   
 ⟨proof⟩

**fun** *reduce-below-abs* ::  $\text{nat} \Rightarrow \text{nat list} \Rightarrow \text{int} \Rightarrow \text{int mat} \Rightarrow \text{int mat}$   
**where** *reduce-below-abs*  $a \ [] \ D \ A = A$   
 | *reduce-below-abs*  $a \ (x \# \ xs) \ D \ A = \text{reduce-below-abs } a \ xs \ D \ (\text{reduce-abs } a \ x \ D \ A)$

**fun** *reduce-below-abs-impl* ::  $\text{nat} \Rightarrow \text{nat list} \Rightarrow \text{int} \Rightarrow \text{int mat} \Rightarrow \text{int mat}$   
**where** *reduce-below-abs-impl*  $a \ [] \ D \ A = A$   
 | *reduce-below-abs-impl*  $a \ (x \# \ xs) \ D \ A = \text{reduce-below-abs-impl } a \ xs \ D \ (\text{reduce-abs-impl } a \ x \ D \ A)$

**lemma** *reduce-abs-impl-carrier*[*simp,intro*]:  $A \in \text{carrier-mat } m \ n \implies \text{reduce-abs-impl } a \ b \ D \ A \in \text{carrier-mat } m \ n$   
 ⟨proof⟩

**lemma** *reduce-abs-below-impl*:  $a < nr \implies 0 < nc \implies (\bigwedge b. b \in \text{set } bs \implies b <$

$nr) \implies a \notin \text{set } bs$   
 $\implies A \in \text{carrier-mat } nr \ nc \implies \text{reduce-below-abs-impl } a \ bs \ D \ A = \text{reduce-below-abs}$   
 $a \ bs \ D \ A$   
 <proof>

This function outputs a matrix in echelon form via reductions modulo the determinant

**function** *FindPreHNF* :: *bool*  $\Rightarrow$  *int*  $\Rightarrow$  *int mat*  $\Rightarrow$  *int mat*  
**where** *FindPreHNF* *abs-flag* *D* *A* =  
 (let *m* = *dim-row* *A*; *n* = *dim-col* *A* in  
 if  $m < 2 \vee n = 0$  then *A* else — No operations are carried out if  $m = 1$   
 let *non-zero-positions* = *filter* ( $\lambda i. A \ \$\$ (i,0) \neq 0$ ) [ $1..<dim\text{-row } A$ ];  
     *A'* = (if  $A \ \$\$ (0,0) \neq 0$  then *A*  
         else let *i* = *non-zero-positions* ! 0 — Select the first non-zero position  
             below the first element  
             in *swaprows* 0 *i* *A*  
         );  
     *Reduce* = (if *abs-flag* then *reduce-below-abs* else *reduce-below*)  
 in  
 if  $n < 2$  then *Reduce* 0 *non-zero-positions* *D* *A'* — If  $n = 1$ , then we have to  
 reduce the column  
 else  
 let  
     (*A-UL*,*A-UR*,*A-DL*,*A-DR*) = *split-block* (*Reduce* 0 *non-zero-positions* *D*  
     (*make-first-column-positive* *A'*)) 1 1;  
     *sub-PreHNF* = *FindPreHNF* *abs-flag* *D* *A-DR* in  
     *four-block-mat* *A-UL* *A-UR* *A-DL* *sub-PreHNF*)  
 <proof>

### termination

<proof>

**lemma** *FindPreHNF-code*: *FindPreHNF* *abs-flag* *D* *A* =  
 (let *m* = *dim-row* *A*; *n* = *dim-col* *A* in  
 if  $m < 2 \vee n = 0$  then *A* else  
 let *non-zero-positions* = *filter* ( $\lambda i. A \ \$\$ (i,0) \neq 0$ ) [ $1..<dim\text{-row } A$ ];  
     *A'* = (if  $A \ \$\$ (0,0) \neq 0$  then *A*  
         else let *i* = *non-zero-positions* ! 0 in *swaprows* 0 *i* *A*  
         );  
     *Reduce-impl* = (if *abs-flag* then *reduce-below-abs-impl* else *reduce-below-impl*)  
 in  
 if  $n < 2$  then *Reduce-impl* 0 *non-zero-positions* *D* *A'*  
 else  
 let  
     (*A-UL*,*A-UR*,*A-DL*,*A-DR*) = *split-block* (*Reduce-impl* 0 *non-zero-positions*  
     *D* (*make-first-column-positive* *A'*)) 1 1;  
     *sub-PreHNF* = *FindPreHNF* *abs-flag* *D* *A-DR* in  
     *four-block-mat* *A-UL* *A-UR* *A-DL* *sub-PreHNF*) (**is** ?lhs = ?rhs)  
 <proof>

**end**

```

declare mod-operation.FindPreHNF-code[code]
declare mod-operation.reduce-below-impl.simps[code]
declare mod-operation.reduce-impl-def[code]
declare mod-operation.reduce-below-abs-impl.simps[code]
declare mod-operation.reduce-abs-impl-def[code]

```

### 7.1.2 From echelon form to Hermite normal form

From here on, we define functions to transform a matrix in echelon form into its Hermite normal form. Essentially, we are defining the functions that are available in the AFP entry Hermite (which uses HOL Analysis + mod-type) in the JNF matrix representation.

**definition** *find-fst-non0-in-row* ::  $\langle \text{nat} \Rightarrow 'a::\text{comm-ring-1 mat} \Rightarrow \text{nat option} \rangle$   
 — Find the first nonzero element of row  $l$  if  $A$  is upper triangular  
**where**  $\langle \text{find-fst-non0-in-row } l \ A = \text{find } (\lambda j. A \ \$\$ \ (l, j) \neq 0) \ [l \ ..< \ \text{dim-col } A] \rangle$

**primrec** *Hermite-reduce-above*

**where** *Hermite-reduce-above* ( $A::\text{int mat}$ )  $0 \ i \ j = A$   
 $|$  *Hermite-reduce-above*  $A \ (\text{Suc } n) \ i \ j = (\text{let}$   
 $A_{ij} = A \ \$\$ \ (i, j);$   
 $A_{nj} = A \ \$\$ \ (n, j)$   
*in*  
*Hermite-reduce-above* ( $\text{addrow } (- (A_{nj} \ \text{div } A_{ij})) \ n \ i \ A$ )  $n \ i \ j$ )

**definition** *Hermite-of-row-i* ::  $\text{int mat} \Rightarrow \text{nat} \Rightarrow \text{int mat}$

**where** *Hermite-of-row-i*  $A \ i =$   
 $\text{case } \text{find-fst-non0-in-row } i \ A \ \text{of } \text{None} \Rightarrow A \ | \ \text{Some } j \Rightarrow$   
 $\text{let } A_{ij} = A \ \$\$ \ (i, j) \ \text{in}$   
 $\text{if } A_{ij} < 0 \ \text{then } \text{Hermite-reduce-above } (\text{multrow } i \ (-1) \ A) \ i \ i \ j$   
 $\text{else } \text{Hermite-reduce-above } A \ i \ i \ j$ )

**primrec** *Hermite-of-list-of-rows*

**where**  
*Hermite-of-list-of-rows*  $A \ [] = A \ |$   
*Hermite-of-list-of-rows*  $A \ (a\#\text{xs}) = \text{Hermite-of-list-of-rows } (\text{Hermite-of-row-i } A \ a) \ \text{xs}$

We combine the previous functions to assemble the algorithm

**definition** (*in mod-operation*) *Hermite-mod-det abs-flag*  $A =$

( $\text{let } m = \text{dim-row } A; \ n = \text{dim-col } A;$   
 $D = \text{abs}(\text{det-int } A);$   
 $A' = A \ @_r \ D \ \cdot_m \ I_m \ n;$   
 $E = \text{FindPreHNF } \text{abs-flag } D \ A';$   
 $H = \text{Hermite-of-list-of-rows } E \ [0..<m+n]$ )

*in mat-of-rows n (map (Matrix.row H) [0..<m]))*

### 7.1.3 Some examples of execution

**declare** *mod-operation.Hermite-mod-det-def*[code]

**value** *let B = mat-of-rows-list 4 ([[0,3,1,4],[7,1,0,0],[8,0,19,16],[2,0,0,3::int]])*  
*in*  
*show (mod-operation.Hermite-mod-det (mod) True B)*

**value** *let B = mat-of-rows-list 7 ([*  
*[ 1, 17, -41, -1, 1, 0, 0],*  
*[ 0, -1, 2, 0, -6, 2, 1],*  
*[ 9, 2, 1, 1, -2, 2, -5],*  
*[-1, -3, -1, 0, -9, 0, 0],*  
*[ 9, -1, -9, 0, 0, 0, 1],*  
*[ 1, -1, 1, 0, 1, -8, 0],*  
*[ 1, -1, 0, -2, -1, -1, 0::int]]) in*  
*show (mod-operation.Hermite-mod-det (mod) True B)*

**end**

## 7.2 Soundness of the algorithm

**theory** *HNF-Mod-Det-Soundness*

**imports**

*HNF-Mod-Det-Algorithm*

*Signed-Modulo*

**begin**

**hide-const**(**open**) *Determinants.det Determinants2.upper-triangular*

*Finite-Cartesian-Product.row Finite-Cartesian-Product.rows*

*Finite-Cartesian-Product.vec*

### 7.2.1 Results connecting lattices and Hermite normal form

The following results will also be useful for proving the soundness of the certification approach.

**lemma** *of-int-mat-hom-int-id[simp]*:

**fixes** *A::int mat*

**shows** *of-int-hom.mat-hom A = A*  $\langle$ *proof* $\rangle$

**definition** *is-sound-HNF algorithm associates res*

$= (\forall A. \text{let } (P,H) = \text{algorithm } A; m = \text{dim-row } A; n = \text{dim-col } A \text{ in}$   
 $P \in \text{carrier-mat } m \ m \wedge H \in \text{carrier-mat } m \ n \wedge \text{invertible-mat } P \wedge A = P$   
 $* H$   
 $\wedge \text{Hermite-JNF associates res } H)$

**lemma** *HNF-A-eq-HNF-PA:*

**fixes**  $A::'a::\{\text{bezout-ring-div}, \text{normalization-euclidean-semiring}, \text{unique-euclidean-ring}\}$   
*mat*

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

**and**  $\text{inv-}P: \text{invertible-mat } P$  **and**  $P: P \in \text{carrier-mat } n \ n$

**and**  $\text{sound-HNF}: \text{is-sound-HNF } HNF \text{ associates res}$

**and**  $P1\text{-}H1: (P1, H1) = HNF (P * A)$

**and**  $P2\text{-}H2: (P2, H2) = HNF A$

**shows**  $H1 = H2$

*<proof>*

**context** *vec-module*

**begin**

**lemma** *mat-mult-invertible-lattice-eq:*

**assumes**  $fs: \text{set } fs \subseteq \text{carrier-vec } n$

**and**  $gs: \text{set } gs \subseteq \text{carrier-vec } n$

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

**and**  $\text{length-}fs: \text{length } fs = m$  **and**  $\text{length-}gs: \text{length } gs = m$

**and**  $\text{prod}: \text{mat-of-rows } n \ fs = (\text{map-mat of-int } P) * \text{mat-of-rows } n \ gs$

**shows**  $\text{lattice-of } fs = \text{lattice-of } gs$

*<proof>*

**end**

**context**

**fixes**  $n :: \text{nat}$

**begin**

**interpretation** *vec-module*  $TYPE(\text{int})$  *<proof>*

**lemma** *lattice-of-HNF:*

**assumes**  $\text{sound-HNF}: \text{is-sound-HNF } HNF \text{ associates res}$

**and**  $P1\text{-}H1: (P, H) = HNF (\text{mat-of-rows } n \ fs)$

**and**  $fs: \text{set } fs \subseteq \text{carrier-vec } n$  **and**  $\text{len}: \text{length } fs = m$

**shows**  $\text{lattice-of } fs = \text{lattice-of } (\text{rows } H)$

*<proof>*

**end**

**context** *LLL-with-assms*

**begin**

**lemma** *certification-via-eq-HNF*:  
**assumes** *sound-HNF*: *is-sound-HNF HNF associates res*  
**and** *P1-H1*:  $(P1, H1) = \text{HNF } (\text{mat-of-rows } n \text{ fs-init})$   
**and** *P2-H2*:  $(P2, H2) = \text{HNF } (\text{mat-of-rows } n \text{ gs})$   
**and** *H1-H2*:  $H1 = H2$   
**and** *gs*:  $\text{set } gs \subseteq \text{carrier-vec } n$  **and** *len-gs*:  $\text{length } gs = m$   
**shows**  $\text{lattice-of } gs = \text{lattice-of } fs\text{-init LLL-with-assms } n \text{ m } gs \alpha$   
*<proof>*

**end**

**context** *vec-space*  
**begin**

**lemma** *lin-indpt-cols-imp-det-not-0*:  
**fixes** *A*::'a mat  
**assumes** *A*:  $A \in \text{carrier-mat } n \text{ n}$  **and** *li*:  $\text{lin-indpt } (\text{set } (\text{cols } A))$  **and** *d*: *distinct*  
*(cols A)*  
**shows**  $\text{det } A \neq 0$   
*<proof>*

**corollary** *lin-indpt-rows-imp-det-not-0*:  
**fixes** *A*::'a mat  
**assumes** *A*:  $A \in \text{carrier-mat } n \text{ n}$  **and** *li*:  $\text{lin-indpt } (\text{set } (\text{rows } A))$  **and** *d*: *distinct*  
*(rows A)*  
**shows**  $\text{det } A \neq 0$   
*<proof>*

**end**

**context** *LLL*  
**begin**

**lemma** *eq-lattice-imp-mat-mult-invertible-cols*:  
**assumes** *fs*:  $\text{set } fs \subseteq \text{carrier-vec } n$   
**and** *gs*:  $\text{set } gs \subseteq \text{carrier-vec } n$  **and** *ind-fs*: *lin-indep fs*  
**and** *length-fs*:  $\text{length } fs = n$  **and** *length-gs*:  $\text{length } gs = n$   
**and** *l*:  $\text{lattice-of } fs = \text{lattice-of } gs$   
**shows**  $\exists Q \in \text{carrier-mat } n \text{ n. invertible-mat } Q \wedge \text{mat-of-cols } n \text{ fs} = \text{mat-of-cols } n \text{ gs} * Q$   
*<proof>*

**corollary** *eq-lattice-imp-mat-mult-invertible-rows*:  
**assumes** *fs*:  $\text{set } fs \subseteq \text{carrier-vec } n$   
**and** *gs*:  $\text{set } gs \subseteq \text{carrier-vec } n$  **and** *ind-fs*: *lin-indep fs*  
**and** *length-fs*:  $\text{length } fs = n$  **and** *length-gs*:  $\text{length } gs = n$   
**and** *l*:  $\text{lattice-of } fs = \text{lattice-of } gs$

**shows**  $\exists P \in \text{carrier-mat } n \ n. \text{invertible-mat } P \wedge \text{mat-of-rows } n \text{ fs} = P * \text{mat-of-rows } n \text{ gs}$   
 <proof>  
**end**

## 7.2.2 Missing results

This is a new definition for upper triangular matrix, valid for rectangular matrices. This definition will allow us to prove that echelon form implies upper triangular for any matrix.

**definition** *upper-triangular'*  $A = (\forall i < \text{dim-row } A. \forall j < \text{dim-col } A. j < i \longrightarrow A \text{ $$ } (i,j) = 0)$

**lemma** *upper-triangular'D[elim]* :  
 $\text{upper-triangular}' A \Longrightarrow j < \text{dim-col } A \Longrightarrow j < i \Longrightarrow i < \text{dim-row } A \Longrightarrow A \text{ $$ } (i,j) = 0$   
 <proof>

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

**lemma** *prod-list-abs*:  
**fixes**  $xs :: \text{int list}$   
**shows**  $\text{prod-list } (\text{map } \text{abs } xs) = \text{abs } (\text{prod-list } xs)$   
 <proof>

**lemma** *euclid-ext2-works*:  
**assumes**  $\text{euclid-ext2 } a \ b = (p,q,u,v,d)$   
**shows**  $p*a+q*b = d$  **and**  $d = \text{gcd } a \ b$  **and**  $\text{gcd } a \ b * u = -b$  **and**  $\text{gcd } a \ b * v = a$   
**and**  $u = -b \ \text{div} \ \text{gcd } a \ b$  **and**  $v = a \ \text{div} \ \text{gcd } a \ b$   
 <proof>

**lemma** *res-function-euclidean2*:  
*res-function*  $(\lambda b \ n :: 'a :: \{\text{unique-euclidean-ring}\}. n \ \text{mod } b)$   
 <proof>

**lemma** *mult-row-1-id*:  
**fixes**  $A :: 'a :: \text{semiring-1}^n \ ^m$   
**shows**  $\text{mult-row } A \ b \ 1 = A$  <proof>

Results about appending rows

**lemma** *row-append-rows1*:  
**assumes**  $A: A \in \text{carrier-mat } m \ n$   
**and**  $B: B \in \text{carrier-mat } p \ n$   
**assumes**  $i: i < \text{dim-row } A$

**shows**  $Matrix.row (A @_r B) i = Matrix.row A i$   
 $\langle proof \rangle$

**lemma** *row-append-rows2*:  
**assumes**  $A: A \in carrier\text{-}mat\ m\ n$   
**and**  $B: B \in carrier\text{-}mat\ p\ n$   
**assumes**  $i: i \in \{m..<m+p\}$   
**shows**  $Matrix.row (A @_r B) i = Matrix.row B (i - m)$   
 $\langle proof \rangle$

**lemma** *rows-append-rows*:  
**assumes**  $A: A \in carrier\text{-}mat\ m\ n$   
**and**  $B: B \in carrier\text{-}mat\ p\ n$   
**shows**  $Matrix.rows (A @_r B) = Matrix.rows A @ Matrix.rows B$   
 $\langle proof \rangle$

**lemma** *append-rows-nth2*:  
**assumes**  $A': A' \in carrier\text{-}mat\ m\ n$   
**and**  $B: B \in carrier\text{-}mat\ p\ n$   
**and**  $A\text{-}def: A = (A' @_r B)$   
**and**  $a: a < m$  **and**  $ap: a < p$  **and**  $j: j < n$   
**shows**  $A \$\$ (a + m, j) = B \$\$ (a, j)$   
 $\langle proof \rangle$

**lemma** *append-rows-nth3*:  
**assumes**  $A': A' \in carrier\text{-}mat\ m\ n$   
**and**  $B: B \in carrier\text{-}mat\ p\ n$   
**and**  $A\text{-}def: A = (A' @_r B)$   
**and**  $a: a \geq m$  **and**  $ap: a < m + p$  **and**  $j: j < n$   
**shows**  $A \$\$ (a, j) = B \$\$ (a - m, j)$   
 $\langle proof \rangle$

Results about submatrices

**lemma** *pick-first-id*: **assumes**  $i: i < n$  **shows**  $pick \{0..<n\} i = i$   
 $\langle proof \rangle$

**lemma** *submatrix-index-id*:  
**assumes**  $H: H \in carrier\text{-}mat\ m\ n$  **and**  $i: i < k1$  **and**  $j: j < k2$   
**and**  $k1: k1 \leq m$  **and**  $k2: k2 \leq n$   
**shows**  $(submatrix\ H\ \{0..<k1\}\ \{0..<k2\}) \$\$ (i, j) = H \$\$ (i, j)$   
 $\langle proof \rangle$

**lemma** *submatrix-carrier-first*:  
**assumes**  $H: H \in carrier\text{-}mat\ m\ n$   
**and**  $k1: k1 \leq m$  **and**  $k2: k2 \leq n$

**shows** *submatrix*  $H \{0..<k1\} \{0..<k2\} \in \text{carrier-mat } k1 \ k2$   
 <proof>

**lemma** *Units-eq-invertible-mat:*

**assumes**  $A \in \text{carrier-mat } n \ n$   
**shows**  $A \in \text{Group.Units } (\text{ring-mat } \text{TYPE}('a::\text{comm-ring-1}) \ n \ b) = \text{invertible-mat } A$  (is ?lhs = ?rhs)  
 <proof>

**lemma** *map-first-rows-index:*

**assumes**  $A \in \text{carrier-mat } M \ n$  and  $m \leq M$  and  $i < m$  and  $ja < n$   
**shows**  $\text{map } (\text{Matrix.row } A) \ [0..<m] \ ! \ i \ \$v \ ja = A \ \$\$ \ (i, \ ja)$   
 <proof>

**lemma** *matrix-append-rows-eq-if-preserves:*

**assumes**  $A: A \in \text{carrier-mat } (m+p) \ n$  and  $B: B \in \text{carrier-mat } p \ n$   
**and**  $\text{eq}: \forall i \in \{m..<m+p\}. \forall j < n. A \ \$\$ \ (i, \ j) = B \ \$\$ \ (i-m, \ j)$   
**shows**  $A = \text{mat-of-rows } n \ [\text{Matrix.row } A \ i. \ i \leftarrow [0..<m]] \ @_r \ B$  (is - = ?A' @<sub>r</sub> -)  
 <proof>

**lemma** *invertible-mat-first-column-not0:*

**fixes**  $A::'a :: \text{comm-ring-1 mat}$   
**assumes**  $A: A \in \text{carrier-mat } n \ n$  and  $\text{inv-A}: \text{invertible-mat } A$  and  $n0: 0 < n$   
**shows**  $\text{col } A \ 0 \neq (0_v \ n)$   
 <proof>

**lemma** *invertible-mat-mult-int:*

**assumes**  $A = P * B$   
**and**  $P \in \text{carrier-mat } n \ n$   
**and**  $B \in \text{carrier-mat } n \ n$   
**and** *invertible-mat*  $P$   
**and** *invertible-mat*  $(\text{map-mat rat-of-int } B)$   
**shows** *invertible-mat*  $(\text{map-mat rat-of-int } A)$   
 <proof>

**lemma** *echelon-form-JNF-intro:*

**assumes**  $(\forall i < \text{dim-row } A. \text{is-zero-row-JNF } i \ A \longrightarrow \neg (\exists j. j < \text{dim-row } A \wedge j > i \wedge \neg \text{is-zero-row-JNF } j \ A))$   
**and**  $(\forall i \ j. i < j \wedge j < \text{dim-row } A \wedge \neg (\text{is-zero-row-JNF } i \ A) \wedge \neg (\text{is-zero-row-JNF } j \ A) \longrightarrow ((\text{LEAST } n. A \ \$\$ \ (i, \ n) \neq 0) < (\text{LEAST } n. A \ \$\$ \ (j, \ n) \neq 0)))$   
**shows** *echelon-form-JNF*  $A$  <proof>

**lemma** *echelon-form-submatrix:*

**assumes** *ef-H: echelon-form-JNF H* **and** *H: H ∈ carrier-mat m n*  
**and** *k: k ≤ min m n*  
**shows** *echelon-form-JNF (submatrix H {0..<k} {0..<k})*  
 ⟨*proof*⟩

**lemma** *HNF-submatrix:*

**assumes** *HNF-H: Hermite-JNF associates res H* **and** *H: H ∈ carrier-mat m n*  
**and** *k: k ≤ min m n*  
**shows** *Hermite-JNF associates res (submatrix H {0..<k} {0..<k})*  
 ⟨*proof*⟩

**lemma** *HNF-of-HNF-id:*

**fixes** *H :: int mat*  
**assumes** *HNF-H: Hermite-JNF associates res H*  
**and** *H: H ∈ carrier-mat n n*  
**and** *H-P1-H1: H = P1 \* H1*  
**and** *inv-P1: invertible-mat P1*  
**and** *H1: H1 ∈ carrier-mat n n*  
**and** *P1: P1 ∈ carrier-mat n n*  
**and** *HNF-H1: Hermite-JNF associates res H1*  
**and** *inv-H: invertible-mat (map-mat rat-of-int H)*  
**shows** *H1 = H*  
 ⟨*proof*⟩

**context**

**fixes** *n :: nat*  
**begin**

**interpretation** *vec-module TYPE(int)* ⟨*proof*⟩

**lemma** *lattice-is-monotone:*

**fixes** *S T*  
**assumes** *S: set S ⊆ carrier-vec n*  
**assumes** *T: set T ⊆ carrier-vec n*  
**assumes** *subs: set S ⊆ set T*  
**shows** *lattice-of S ⊆ lattice-of T*  
 ⟨*proof*⟩

**lemma** *lattice-of-append:*

**assumes** *fs: set fs ⊆ carrier-vec n*  
**assumes** *gs: set gs ⊆ carrier-vec n*  
**shows** *lattice-of (fs @ gs) = lattice-of (gs @ fs)*  
 ⟨*proof*⟩

**lemma** *lattice-of-append-cons:*

**assumes**  $fs: \text{set } fs \subseteq \text{carrier-vec } n$  **and**  $v: v \in \text{carrier-vec } n$   
**shows**  $\text{lattice-of } (v \# fs) = \text{lattice-of } (fs @ [v])$   
 ⟨proof⟩

**lemma** *already-in-lattice-subset*:  
**assumes**  $fs: \text{set } fs \subseteq \text{carrier-vec } n$  **and**  $inlattice: v \in \text{lattice-of } fs$   
**and**  $v: v \in \text{carrier-vec } n$   
**shows**  $\text{lattice-of } (v \# fs) \subseteq \text{lattice-of } fs$   
 ⟨proof⟩

**lemma** *already-in-lattice*:  
**assumes**  $fs: \text{set } fs \subseteq \text{carrier-vec } n$  **and**  $inlattice: v \in \text{lattice-of } fs$   
**and**  $v: v \in \text{carrier-vec } n$   
**shows**  $\text{lattice-of } fs = \text{lattice-of } (v \# fs)$   
 ⟨proof⟩

**lemma** *already-in-lattice-append*:  
**assumes**  $fs: \text{set } fs \subseteq \text{carrier-vec } n$  **and**  $inlattice: \text{lattice-of } gs \subseteq \text{lattice-of } fs$   
**and**  $gs: \text{set } gs \subseteq \text{carrier-vec } n$   
**shows**  $\text{lattice-of } fs = \text{lattice-of } (fs @ gs)$   
 ⟨proof⟩

**lemma** *zero-in-lattice*:  
**assumes**  $fs\text{-carrier}: \text{set } fs \subseteq \text{carrier-vec } n$   
**shows**  $0_v \ n \in \text{lattice-of } fs$   
 ⟨proof⟩

**lemma** *lattice-zero-rows-subset*:  
**assumes**  $H: H \in \text{carrier-mat } a \ n$   
**shows**  $\text{lattice-of } (\text{Matrix.rows } (0_m \ m \ n)) \subseteq \text{lattice-of } (\text{Matrix.rows } H)$   
 ⟨proof⟩

**lemma** *lattice-of-append-zero-rows*:  
**assumes**  $H': H' \in \text{carrier-mat } m \ n$   
**and**  $H: H = H' @_r (0_m \ m \ n)$   
**shows**  $\text{lattice-of } (\text{Matrix.rows } H) = \text{lattice-of } (\text{Matrix.rows } H')$   
 ⟨proof⟩  
**end**

Lemmas about echelon form

**lemma** *echelon-form-JNF-1xn*:  
**assumes**  $A \in \text{carrier-mat } m \ n$  **and**  $m < 2$   
**shows**  $\text{echelon-form-JNF } A$   
 ⟨proof⟩

**lemma** *echelon-form-JNF-mx1*:  
**assumes**  $A \in \text{carrier-mat } m \ n$  **and**  $n < 2$   
**and**  $\forall i \in \{1..<m\}. A \$\$ (i,0) = 0$   
**shows** *echelon-form-JNF A*  
 $\langle \text{proof} \rangle$

**lemma** *echelon-form-mx0*:  
**assumes**  $A \in \text{carrier-mat } m \ 0$   
**shows** *echelon-form-JNF A*  $\langle \text{proof} \rangle$

**lemma** *echelon-form-JNF-first-column-0*:  
**assumes**  $eA: \text{echelon-form-JNF } A$  **and**  $A: A \in \text{carrier-mat } m \ n$   
**and**  $i0: 0 < i$  **and**  $im: i < m$  **and**  $n0: 0 < n$   
**shows**  $A \$\$ (i,0) = 0$   
 $\langle \text{proof} \rangle$

**lemma** *is-zero-row-JNF-multrow[simp]*:  
**fixes**  $A: 'a::\text{comm-ring-1 mat}$   
**assumes**  $i < \text{dim-row } A$   
**shows**  $\text{is-zero-row-JNF } i \ (\text{multrow } j \ (- \ 1) \ A) = \text{is-zero-row-JNF } i \ A$   
 $\langle \text{proof} \rangle$

**lemma** *echelon-form-JNF-multrow*:  
**assumes**  $A : \text{carrier-mat } m \ n$  **and**  $i < m$  **and**  $eA: \text{echelon-form-JNF } A$   
**shows**  $\text{echelon-form-JNF } (\text{multrow } i \ (- \ 1) \ A)$   
 $\langle \text{proof} \rangle$

**thm** *echelon-form-imp-upper-triangular*

**lemma** *echelon-form-JNF-least-position-ge-diagonal*:  
**assumes**  $eA: \text{echelon-form-JNF } A$   
**and**  $A: A: \text{carrier-mat } m \ n$   
**and**  $\text{nz-}iA: \neg \text{is-zero-row-JNF } i \ A$   
**and**  $im: i < m$   
**shows**  $i \leq (\text{LEAST } n. A \$\$ (i,n) \neq 0)$   
 $\langle \text{proof} \rangle$

**lemma** *echelon-form-JNF-imp-upper-triangular*:  
**assumes**  $eA: \text{echelon-form-JNF } A$   
**shows** *upper-triangular A*  
 $\langle \text{proof} \rangle$

**lemma** *echelon-form-JNF-imp-upper-triangular*:

**assumes** *eA*: *echelon-form-JNF* *A*

**shows** *upper-triangular'* *A*

*<proof>*

**lemma** *upper-triangular-append-zero*:

**assumes** *uH*: *upper-triangular'* *H*

**and** *H*: *H*  $\in$  *carrier-mat*  $(m+m)$  *n* **and** *mn*:  $n \leq m$

**shows** *H* = *mat-of-rows* *n* (*map* (*Matrix.row* *H*)  $[0..<m]$ )  $\textcircled{r}$   $0_m$  *m* *n* (**is** - =  
*?H'*  $\textcircled{r}$   $0_m$  *m* *n*)

*<proof>*

### 7.2.3 The algorithm is sound

**lemma** *find-fst-non0-in-row-None'*:

**assumes** *l* < *m*

**shows** *find-fst-non0-in-row* *l* *A* = *None*  $\longleftrightarrow$   $(\forall j \in \{l..<dim-col\ A\}. A \ \$\$ (l,j) = 0)$  (**is** *?lhs* = *?rhs*)

*<proof>*

**lemma** *find-fst-non0-in-row-None*:

**assumes** *A*: *A*  $\in$  *carrier-mat* *m* *n*

**and** *ut-A*: *upper-triangular'* *A*

**and** *lm*: *l* < *m*

**shows** *find-fst-non0-in-row* *l* *A* = *None*  $\longleftrightarrow$  *is-zero-row-JNF* *l* *A* (**is** *?lhs* = *?rhs*)

*<proof>*

**lemma**

**assumes** *res*: *find-fst-non0-in-row* *l* *A* = *Some* *j*

**shows** *find-fst-non0-in-row*: *A*  $\ \$\$ (l,j) \neq 0$   $l \leq j < dim-col\ A$

**and** *find-fst-non0-in-row-zero-before*:  $\forall j' \in \{l..<j\}. A \ \$\$ (l, j') = 0$

*<proof>*

**corollary** *find-fst-non0-in-row-zero-before'*:

**assumes** *res*: *find-fst-non0-in-row* *l* *A* = *Some* *j*

**and** *j'*  $\in \{l..<j\}$

**shows** *A*  $\ \$\$ (l,j') = 0$  *<proof>*

**lemma** *find-fst-non0-in-row-LEAST*:

**assumes** *A*: *A*  $\in$  *carrier-mat* *m* *n*

**and** *ut-A*: *upper-triangular'* *A*

**and** *res*: *find-fst-non0-in-row* *l* *A* = *Some* *j*

**and** *lm*: *l* < *m*

**shows** *j* = (*LEAST* *n*. *A*  $\ \$\$ (l,n) \neq 0$ )

*<proof>*

**lemma** *make-first-column-positive-preserves-dimensions*:  
**shows** [simp]:  $\dim\text{-row } (\text{make-first-column-positive } A) = \dim\text{-row } A$   
**and** [simp]:  $\dim\text{-col } (\text{make-first-column-positive } A) = \dim\text{-col } A$   
 ⟨proof⟩

**lemma** *make-first-column-positive-works*:  
**assumes**  $A \in \text{carrier-mat } m \ n$  **and**  $i < m$  **and**  $0 < n$   
**shows**  $\text{make-first-column-positive } A \ \$\$ (i,0) \geq 0$   
**and**  $j < n \implies A \ \$\$ (i,0) < 0 \implies (\text{make-first-column-positive } A) \ \$\$ (i,j) = - A \ \$\$ (i,j)$   
**and**  $j < n \implies A \ \$\$ (i,0) \geq 0 \implies (\text{make-first-column-positive } A) \ \$\$ (i,j) = A \ \$\$ (i,j)$   
 ⟨proof⟩

**lemma** *make-first-column-positive-invertible*:  
**shows**  $\exists P. \text{invertible-mat } P \wedge P \in \text{carrier-mat } (\dim\text{-row } A) (\dim\text{-row } A)$   
 $\wedge \text{make-first-column-positive } A = P * A$   
 ⟨proof⟩

**locale** *proper-mod-operation = mod-operation +*  
**assumes** *dvd-gdiv-mult-right*[simp]:  $b > 0 \implies b \text{ dvd } a \implies (a \text{ gdiv } b) * b = a$   
**and** *gmod-gdiv*:  $y > 0 \implies x \text{ gmod } y = x - x \text{ gdiv } y * y$   
**and** *dvd-imp-gmod-0*:  $0 < a \implies a \text{ dvd } b \implies b \text{ gmod } a = 0$   
**and** *gmod-0-imp-dvd*:  $a \text{ gmod } b = 0 \implies b \text{ dvd } a$   
**and** *gmod-0*[simp]:  $n \text{ gmod } 0 = n \ n > 0 \implies 0 \text{ gmod } n = 0$

**begin**

**lemma** *reduce-alt-def-not0*:

**assumes**  $A \ \$\$ (a,0) \neq 0$  **and** *pquvd*:  $(p,q,u,v,d) = \text{euclid-ext2 } (A \ \$\$ (a,0)) (A \ \$\$ (b,0))$

**shows**  $\text{reduce } a \ b \ D \ A =$

$\text{Matrix.mat } (\dim\text{-row } A) (\dim\text{-col } A)$   
 $(\lambda(i,k). \text{if } i = a \text{ then let } r = (p * A \ \$\$ (a,k) + q * A \ \$\$ (b,k)) \text{ in}$   
 $\quad \text{if } k = 0 \text{ then if } D \text{ dvd } r \text{ then } D \text{ else } r \text{ else } r \text{ gmod } D$   
 $\text{else if } i = b \text{ then let } r = u * A \ \$\$ (a,k) + v * A \ \$\$ (b,k) \text{ in}$   
 $\quad \text{if } k = 0 \text{ then } r \text{ else } r \text{ gmod } D$   
 $\text{else } A \ \$\$ (i,k)) \text{ (is - = ?rhs)}$

**and**

$\text{reduce-abs } a \ b \ D \ A =$

$\text{Matrix.mat } (\dim\text{-row } A) (\dim\text{-col } A)$   
 $(\lambda(i,k). \text{if } i = a \text{ then let } r = (p * A \ \$\$ (a,k) + q * A \ \$\$ (b,k)) \text{ in}$   
 $\quad \text{if abs } r > D \text{ then if } k = 0 \wedge D \text{ dvd } r \text{ then } D \text{ else } r \text{ gmod}$   
 $D \text{ else } r$   
 $\text{else if } i = b \text{ then let } r = u * A \ \$\$ (a,k) + v * A \ \$\$ (b,k) \text{ in}$   
 $\quad \text{if abs } r > D \text{ then } r \text{ gmod } D \text{ else } r$   
 $\text{else } A \ \$\$ (i,k)) \text{ (is - = ?rhs-abs)}$

⟨proof⟩

**lemma** *reduce-preserves-dimensions*:

**shows** [simp]:  $\dim\text{-row } (\text{reduce } a \ b \ D \ A) = \dim\text{-row } A$   
**and** [simp]:  $\dim\text{-col } (\text{reduce } a \ b \ D \ A) = \dim\text{-col } A$   
**and** [simp]:  $\dim\text{-row } (\text{reduce-abs } a \ b \ D \ A) = \dim\text{-row } A$   
**and** [simp]:  $\dim\text{-col } (\text{reduce-abs } a \ b \ D \ A) = \dim\text{-col } A$   
{proof}

**lemma** *reduce-carrier*:

**assumes**  $A \in \text{carrier-mat } m \ n$   
**shows**  $(\text{reduce } a \ b \ D \ A) \in \text{carrier-mat } m \ n$   
**and**  $(\text{reduce-abs } a \ b \ D \ A) \in \text{carrier-mat } m \ n$   
{proof}

**lemma** *reduce-gcd*:

**assumes**  $A: A \in \text{carrier-mat } m \ n$  **and**  $a: a < m$  **and**  $j: 0 < n$   
**and**  $A_{aj}: A \ \$\$ \ (a, 0) \neq 0$   
**shows**  $(\text{reduce } a \ b \ D \ A) \ \$\$ \ (a, 0) = (\text{let } r = \text{gcd } (A \ \$\$ \ (a, 0)) \ (A \ \$\$ \ (b, 0)) \ \text{in if } D \ \text{dvd } r \ \text{then } D \ \text{else } r) \ (\text{is } ?lhs = ?rhs)$   
**and**  $(\text{reduce-abs } a \ b \ D \ A) \ \$\$ \ (a, 0) = (\text{let } r = \text{gcd } (A \ \$\$ \ (a, 0)) \ (A \ \$\$ \ (b, 0)) \ \text{in if } D < r \ \text{then if } D \ \text{dvd } r \ \text{then } D \ \text{else } r \ \text{gmod } D \ \text{else } r) \ (\text{is } ?lhs\text{-abs} = ?rhs\text{-abs})$   
{proof}

**lemma** *reduce-preserves*:

**assumes**  $A: A \in \text{carrier-mat } m \ n$  **and**  $j: j < n$   
**and**  $A_{aj}: A \ \$\$ \ (a, 0) \neq 0$  **and**  $ib: i \neq b$  **and**  $ia: i \neq a$  **and**  $im: i < m$   
**shows**  $(\text{reduce } a \ b \ D \ A) \ \$\$ \ (i, j) = A \ \$\$ \ (i, j) \ (\text{is } ?thesis1)$   
**and**  $(\text{reduce-abs } a \ b \ D \ A) \ \$\$ \ (i, j) = A \ \$\$ \ (i, j) \ (\text{is } ?thesis2)$   
{proof}

**lemma** *reduce-0*:

**assumes**  $A: A \in \text{carrier-mat } m \ n$  **and**  $a: a < m$  **and**  $j: 0 < n$  **and**  $b: b < m$  **and**  
 $ab: a \neq b$   
**and**  $A_{aj}: A \ \$\$ \ (a, 0) \neq 0$   
**and**  $D: D \geq 0$   
**shows**  $(\text{reduce } a \ b \ D \ A) \ \$\$ \ (b, 0) = 0 \ (\text{is } ?thesis1)$   
**and**  $(\text{reduce-abs } a \ b \ D \ A) \ \$\$ \ (b, 0) = 0 \ (\text{is } ?thesis2)$   
{proof}  
**end**

Let us show the key lemma: operations modulo determinant don't modify the (integer) row span.

**context** *LLL-with-assms*  
**begin**

**lemma** *lattice-of-kId-subset-fs-init*:  
**assumes** *k-det*:  $k = \text{Determinant.det } (\text{mat-of-rows } n \text{ fs-init})$   
**and** *mn*:  $m=n$   
**shows**  $\text{lattice-of } (\text{Matrix.rows } (k \cdot_m (1_m \ m))) \subseteq \text{lattice-of fs-init}$   
 $\langle \text{proof} \rangle$

**end**

**context** *LLL-with-assms*  
**begin**

**lemma** *lattice-of-append-det-preserves*:  
**assumes** *k-det*:  $k = \text{abs } (\text{Determinant.det } (\text{mat-of-rows } n \text{ fs-init}))$   
**and** *mn*:  $m = n$   
**and** *A*:  $A = (\text{mat-of-rows } n \text{ fs-init}) @_r (k \cdot_m (1_m \ m))$   
**shows**  $\text{lattice-of } (\text{Matrix.rows } A) = \text{lattice-of fs-init}$   
 $\langle \text{proof} \rangle$

This is another key lemma. Here,  $A$  is the initial matrix (*mat-of-rows*  $n$  *fs-init*) augmented with  $m$  rows  $(k, 0, \dots, 0), (0, k, 0, \dots, 0), \dots, (0, \dots, 0, k)$  where  $k$  is the determinant of (*mat-of-rows*  $n$  *fs-init*). With the algorithm of the article, we obtain  $H = H' @_r (0_m \ m \ n)$  by means of an invertible matrix  $P$  (which is computable). Then,  $H$  is the HNF of  $A$ . The lemma shows that  $H'$  is the HNF of (*mat-of-rows*  $n$  *fs-init*) and that there exists an invertible matrix to carry out the transformation.

**lemma** *Hermite-append-det-id*:  
**assumes** *k-det*:  $k = \text{abs } (\text{Determinant.det } (\text{mat-of-rows } n \text{ fs-init}))$   
**and** *mn*:  $m = n$   
**and** *A*:  $A = (\text{mat-of-rows } n \text{ fs-init}) @_r (k \cdot_m (1_m \ m))$   
**and** *H'*:  $H' \in \text{carrier-mat } m \ n$   
**and** *H-append*:  $H = H' @_r (0_m \ m \ n)$   
**and** *P*:  $P \in \text{carrier-mat } (m+m) \ (m+m)$   
**and** *inv-P*: *invertible-mat*  $P$   
**and** *A-PH*:  $A = P * H$   
**and** *HNF-H*: *Hermite-JNF associates res*  $H$   
**shows** *Hermite-JNF associates res*  $H'$   
**and**  $(\exists P'. \text{invertible-mat } P' \wedge P' \in \text{carrier-mat } m \ m \wedge (\text{mat-of-rows } n \text{ fs-init}) = P' * H')$   
 $\langle \text{proof} \rangle$

**end**

**context** *proper-mod-operation*  
**begin**

**definition** *reduce-element-mod-D* ( $A::\text{int mat}$ )  $a\ j\ D\ m =$   
 (if  $j = 0$  then if  $D\ \text{dvd}\ A\ \$(a,j)$  then  $\text{addrow } (-(A\ \$(a,j)\ \text{gdiv}\ D)) + 1) a\ (j + m)$   $A$  else  $A$   
 else  $\text{addrow } (-(A\ \$(a,j)\ \text{gdiv}\ D)) a\ (j + m) A$

**definition** *reduce-element-mod-D-abs* ( $A::\text{int mat}$ )  $a\ j\ D\ m =$   
 (if  $j = 0 \wedge D\ \text{dvd}\ A\ \$(a,j)$  then  $\text{addrow } (-(A\ \$(a,j)\ \text{gdiv}\ D)) + 1) a\ (j + m)$   
 $A$   
 else  $\text{addrow } (-(A\ \$(a,j)\ \text{gdiv}\ D)) a\ (j + m) A$

**lemma** *reduce-element-mod-D-preserves-dimensions:*

**shows**  $[simp]: \text{dim-row } (\text{reduce-element-mod-D } A\ a\ j\ D\ m) = \text{dim-row } A$   
**and**  $[simp]: \text{dim-col } (\text{reduce-element-mod-D } A\ a\ j\ D\ m) = \text{dim-col } A$   
**and**  $[simp]: \text{dim-row } (\text{reduce-element-mod-D-abs } A\ a\ j\ D\ m) = \text{dim-row } A$   
**and**  $[simp]: \text{dim-col } (\text{reduce-element-mod-D-abs } A\ a\ j\ D\ m) = \text{dim-col } A$   
 $\langle \text{proof} \rangle$

**lemma** *reduce-element-mod-D-carrier:*

**shows**  $\text{reduce-element-mod-D } A\ a\ j\ D\ m \in \text{carrier-mat } (\text{dim-row } A) (\text{dim-col } A)$

**and**  $\text{reduce-element-mod-D-abs } A\ a\ j\ D\ m \in \text{carrier-mat } (\text{dim-row } A) (\text{dim-col } A)$   $\langle \text{proof} \rangle$

**lemma** *reduce-element-mod-D-invertible-mat:*

**assumes**  $A\text{-def}: A = A' @_r (D \cdot_m (1_m\ n))$   
**and**  $A': A' \in \text{carrier-mat } m\ n$  **and**  $a: a < m$  **and**  $j: j < n$  **and**  $mn: m \geq n$   
**shows**  $\exists P. P \in \text{carrier-mat } (m+n) (m+n) \wedge \text{invertible-mat } P \wedge$   
 $\text{reduce-element-mod-D } A\ a\ j\ D\ m = P * A$  (**is** *?thesis1*)  
**and**  $\exists P. P \in \text{carrier-mat } (m+n) (m+n) \wedge \text{invertible-mat } P \wedge$   
 $\text{reduce-element-mod-D-abs } A\ a\ j\ D\ m = P * A$  (**is** *?thesis2*)  
 $\langle \text{proof} \rangle$

**lemma** *reduce-element-mod-D-append:*

**assumes**  $A\text{-def}: A = A' @_r (D \cdot_m (1_m\ n))$   
**and**  $A': A' \in \text{carrier-mat } m\ n$  **and**  $a: a < m$  **and**  $j: j < n$  **and**  $mn: m \geq n$   
**shows**  $\text{reduce-element-mod-D } A\ a\ j\ D\ m$   
 $= \text{mat-of-rows } n [\text{Matrix.row } (\text{reduce-element-mod-D } A\ a\ j\ D\ m)\ i.\ i \leftarrow [0..<m]]$   
 $@_r (D \cdot_m (1_m\ n))$  (**is** *?lhs = ?A' @\_r ?D*)  
**and**  $\text{reduce-element-mod-D-abs } A\ a\ j\ D\ m$   
 $= \text{mat-of-rows } n [\text{Matrix.row } (\text{reduce-element-mod-D-abs } A\ a\ j\ D\ m)\ i.\ i \leftarrow$   
 $[0..<m]] @_r (D \cdot_m (1_m\ n))$  (**is** *?lhs-abs = ?A'-abs @\_r ?D*)  
 $\langle \text{proof} \rangle$

**lemma** *reduce-append-rows-eq:*

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

**and** *A-def*:  $A = A' @_r (D \cdot_m (1_m n))$  **and**  $a: a < m$  **and**  $xm: x < m$  **and**  $0 < n$   
**and** *Aaj*:  $A \$\$ (a, 0) \neq 0$   
**shows** *reduce a x D A*  
 $= \text{mat-of-rows } n \text{ [Matrix.row ((reduce a x D A)) } i. i \leftarrow [0..<m]] @_r D \cdot_m 1_m n$   
**(is ?thesis1)**  
**and** *reduce-abs a x D A*  
 $= \text{mat-of-rows } n \text{ [Matrix.row ((reduce-abs a x D A)) } i. i \leftarrow [0..<m]] @_r D \cdot_m$   
 $1_m n$  **(is ?thesis2)**  
 ⟨*proof*⟩

**fun** *reduce-row-mod-D*  
**where** *reduce-row-mod-D A a [] D m = A |*  
*reduce-row-mod-D A a (x # xs) D m = reduce-row-mod-D (reduce-element-mod-D*  
*A a x D m) a xs D m*

**fun** *reduce-row-mod-D-abs*  
**where** *reduce-row-mod-D-abs A a [] D m = A |*  
*reduce-row-mod-D-abs A a (x # xs) D m = reduce-row-mod-D-abs (reduce-element-mod-D-abs*  
*A a x D m) a xs D m*

**lemma** *reduce-row-mod-D-preserves-dimensions*:  
**shows** [*simp*]:  $\text{dim-row (reduce-row-mod-D A a xs D m)} = \text{dim-row A}$   
**and** [*simp*]:  $\text{dim-col (reduce-row-mod-D A a xs D m)} = \text{dim-col A}$   
 ⟨*proof*⟩

**lemma** *reduce-row-mod-D-preserves-dimensions-abs*:  
**shows** [*simp*]:  $\text{dim-row (reduce-row-mod-D-abs A a xs D m)} = \text{dim-row A}$   
**and** [*simp*]:  $\text{dim-col (reduce-row-mod-D-abs A a xs D m)} = \text{dim-col A}$   
 ⟨*proof*⟩

**lemma** *reduce-row-mod-D-invertible-mat*:  
**assumes** *A-def*:  $A = A' @_r (D \cdot_m (1_m n))$   
**and** *A'*:  $A' \in \text{carrier-mat } m \ n$  **and**  $a: a < m$  **and**  $j: \forall j \in \text{set } xs. j < n$  **and**  $mn:$   
 $m \geq n$   
**shows**  $\exists P. P \in \text{carrier-mat } (m+n) \ (m+n) \wedge \text{invertible-mat } P \wedge$   
 $\text{reduce-row-mod-D A a xs D m} = P * A$   
 ⟨*proof*⟩

**lemma** *reduce-row-mod-D-abs-invertible-mat*:  
**assumes** *A-def*:  $A = A' @_r (D \cdot_m (1_m n))$   
**and** *A'*:  $A' \in \text{carrier-mat } m \ n$  **and**  $a: a < m$  **and**  $j: \forall j \in \text{set } xs. j < n$  **and**  $mn:$   
 $m \geq n$   
**shows**  $\exists P. P \in \text{carrier-mat } (m+n) \ (m+n) \wedge \text{invertible-mat } P \wedge$   
 $\text{reduce-row-mod-D-abs A a xs D m} = P * A$   
 ⟨*proof*⟩  
**end**

**context** *proper-mod-operation*

**begin**

**lemma** *dvd-gdiv-mult-left[simp]*: **assumes**  $b > 0$   $b \text{ dvd } a$  **shows**  $b * (a \text{ gdiv } b) = a$   
*<proof>*

**lemma** *reduce-element-mod-D*:

**assumes** *A-def*:  $A = A' @_r (D \cdot_m (1_m \ n))$

**and** *A'*:  $A' \in \text{carrier-mat } m \ n$  **and**  $a \leq m$  **and**  $j < n$  **and**  $mn: m \geq n$

**and** *D*:  $D > 0$

**shows** *reduce-element-mod-D*  $A \ a \ j \ D \ m = \text{Matrix.mat } (\text{dim-row } A) (\text{dim-col } A)$

$(\lambda(i,k). \text{ if } i = a \wedge k = j \text{ then if } j = 0 \text{ then if } D \text{ dvd } A\$\$(i,k)$

$\text{ then } D \text{ else } A\$\$(i,k) \text{ else } A\$\$(i,k) \text{ gmod } D \text{ else } A\$\$(i,k)) (\text{is } - = ?A)$

**and** *reduce-element-mod-D-abs*  $A \ a \ j \ D \ m = \text{Matrix.mat } (\text{dim-row } A) (\text{dim-col } A)$

$(\lambda(i,k). \text{ if } i = a \wedge k = j \text{ then if } j = 0 \wedge D \text{ dvd } A\$\$(i,k) \text{ then } D \text{ else } A\$\$(i,k)$

$\text{ gmod } D \text{ else } A\$\$(i,k)) (\text{is } - = ?A\text{-abs})$

*<proof>*

**lemma** *reduce-row-mod-D*:

**assumes** *A-def*:  $A = A' @_r (D \cdot_m (1_m \ n))$

**and** *A'*:  $A' \in \text{carrier-mat } m \ n$  **and**  $a < m$  **and**  $j: \forall j \in \text{set } xs. j < n$

**and** *d*: *distinct xs* **and**  $m \geq n$

**and**  $D > 0$

**shows** *reduce-row-mod-D*  $A \ a \ xs \ D \ m = \text{Matrix.mat } (\text{dim-row } A) (\text{dim-col } A)$

$(\lambda(i,k). \text{ if } i = a \wedge k \in \text{set } xs \text{ then if } k = 0 \text{ then if } D \text{ dvd } A\$\$(i,k)$

$\text{ then } D \text{ else } A\$\$(i,k) \text{ else } A\$\$(i,k) \text{ gmod } D \text{ else } A\$\$(i,k))$

*<proof>*

**lemma** *reduce-row-mod-D-abs*:

**assumes** *A-def*:  $A = A' @_r (D \cdot_m (1_m \ n))$

**and** *A'*:  $A' \in \text{carrier-mat } m \ n$  **and**  $a < m$  **and**  $j: \forall j \in \text{set } xs. j < n$

**and** *d*: *distinct xs* **and**  $m \geq n$

**and**  $D > 0$

**shows** *reduce-row-mod-D-abs*  $A \ a \ xs \ D \ m = \text{Matrix.mat } (\text{dim-row } A) (\text{dim-col } A)$

$(\lambda(i,k). \text{ if } i = a \wedge k \in \text{set } xs \text{ then if } k = 0 \wedge D \text{ dvd } A\$\$(i,k)$

$\text{ then } D \text{ else } A\$\$(i,k) \text{ gmod } D \text{ else } A\$\$(i,k))$

*<proof>*

**end**

Now, we prove some transfer rules to connect Bézout matrices in HOL Analysis and JNF

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

```

shows ((Mod-Type-Connect.HMA-M :: -  $\Rightarrow$  'a :: {bezout-ring}  $\wedge$  'n :: mod-type
 $\wedge$  'm :: mod-type  $\Rightarrow$  -)
 $\implies$  (Mod-Type-Connect.HMA-I :: -  $\Rightarrow$  'm  $\Rightarrow$  -)  $\implies$  (Mod-Type-Connect.HMA-I
:: -  $\Rightarrow$  'm  $\Rightarrow$  -)
 $\implies$  (Mod-Type-Connect.HMA-I :: -  $\Rightarrow$  'n  $\Rightarrow$  -)  $\implies$  (=)  $\implies$  (Mod-Type-Connect.HMA-M))

(bezout-matrix-JNF) (bezout-matrix)
<proof>

```

```

context
begin

```

```

private lemma invertible-bezout-matrix-JNF-mod-type:
  fixes A::'a::{bezout-ring-div} mat
  assumes A  $\in$  carrier-mat CARD('m::mod-type) CARD('n::mod-type)
  assumes ib: is-bezout-ext bezout
  and a-less-b: a < b and b: b < CARD('m) and j: j < CARD('n)
  and aj: A $$ (a, j)  $\neq$  0
shows invertible-mat (bezout-matrix-JNF A a b j bezout)
<proof> lemma invertible-bezout-matrix-JNF-nontriv-mod-ring:
  fixes A::'a::{bezout-ring-div} mat
  assumes A  $\in$  carrier-mat CARD('m::nontriv mod-ring) CARD('n::nontriv mod-ring)
  assumes ib: is-bezout-ext bezout
  and a-less-b: a < b and b: b < CARD('m) and j: j < CARD('n)
  and aj: A $$ (a, j)  $\neq$  0
shows invertible-mat (bezout-matrix-JNF A a b j bezout)
<proof>

```

```

lemmas invertible-bezout-matrix-JNF-internalized =
  invertible-bezout-matrix-JNF-nontriv-mod-ring[unfolded CARD-mod-ring,
  internalize-sort 'm::nontriv, internalize-sort 'c::nontriv]

```

```

context
  fixes m::nat and n::nat
  assumes local-typedef1:  $\exists$  (Rep :: ('b  $\Rightarrow$  int)) Abs. type-definition Rep Abs {0..m
:: int}
  assumes local-typedef2:  $\exists$  (Rep :: ('c  $\Rightarrow$  int)) Abs. type-definition Rep Abs {0..n
:: int}
  and m: m > 1
  and n: n > 1
begin

```

```

lemma type-to-set1:
  shows class.nontriv TYPE('b) (is ?a) and m=CARD('b) (is ?b)
<proof>

```

**lemma** *type-to-set2*:  
 shows *class.nontriv TYPE('c) (is ?a) and n=CARD('c) (is ?b)*  
 ⟨*proof*⟩

**lemma** *invertible-bezout-matrix-JNF-nontriv-mod-ring-aux*:  
 fixes *A::'a::{bezout-ring-div} mat*  
 assumes *A ∈ carrier-mat m n*  
 assumes *ib: is-bezout-ext bezout*  
 and *a-less-b: a < b and b: b < m and j: j < n*  
 and *aj: A \$\$ (a, j) ≠ 0*  
 shows *invertible-mat (bezout-matrix-JNF A a b j bezout)*  
 ⟨*proof*⟩  
 end

**context**  
**begin**

**private lemma** *invertible-bezout-matrix-JNF-cancelled-first*:  
 $\exists \text{Rep Abs. type-definition Rep Abs } \{0..<\text{int } n\} \implies \{0..<\text{int } m\} \neq \{\} \implies$   
 $1 < m \implies 1 < n \implies$   
 $(A::'a::\text{bezout-ring-div mat}) \in \text{carrier-mat } m \ n \implies \text{is-bezout-ext bezout}$   
 $\implies a < b \implies b < m \implies j < n \implies A \ \$\$ (a, j) \neq 0 \implies \text{invertible-mat}$   
 $(\text{bezout-matrix-JNF } A \ a \ b \ j \ \text{bezout})$   
 ⟨*proof*⟩ **lemma** *invertible-bezout-matrix-JNF-cancelled-both*:  
 $\{0..<\text{int } n\} \neq \{\} \implies \{0..<\text{int } m\} \neq \{\} \implies 1 < m \implies 1 < n \implies$   
 $1 < m \implies 1 < n \implies$   
 $(A::'a::\text{bezout-ring-div mat}) \in \text{carrier-mat } m \ n \implies \text{is-bezout-ext bezout}$   
 $\implies a < b \implies b < m \implies j < n \implies A \ \$\$ (a, j) \neq 0 \implies \text{invertible-mat}$   
 $(\text{bezout-matrix-JNF } A \ a \ b \ j \ \text{bezout})$   
 ⟨*proof*⟩

**lemma** *invertible-bezout-matrix-JNF'*:  
 fixes *A::'a::{bezout-ring-div} mat*  
 assumes *A ∈ carrier-mat m n*  
 assumes *ib: is-bezout-ext bezout*  
 and *a-less-b: a < b and b: b < m and j: j < n*  
 and *n > 1*  
 and *aj: A \$\$ (a, j) ≠ 0*  
 shows *invertible-mat (bezout-matrix-JNF A a b j bezout)*  
 ⟨*proof*⟩

**lemma** *invertible-bezout-matrix-JNF-n1*:

**fixes**  $A::'a::\{\text{bezout-ring-div}\}$  *mat*  
**assumes**  $A: A \in \text{carrier-mat } m \ n$   
**assumes**  $ib: \text{is-bezout-ext } \text{bezout}$   
**and**  $a\text{-less-}b: a < b$  **and**  $b: b < m$  **and**  $j: j < n$   
**and**  $n1: n=1$   
**and**  $aj: A \ \$\$ \ (a, j) \neq 0$   
**shows** *invertible-mat* (*bezout-matrix-JNF*  $A \ a \ b \ j \ \text{bezout}$ )  
*<proof>*

**corollary** *invertible-bezout-matrix-JNF*:  
**fixes**  $A::'a::\{\text{bezout-ring-div}\}$  *mat*  
**assumes**  $A \in \text{carrier-mat } m \ n$   
**assumes**  $ib: \text{is-bezout-ext } \text{bezout}$   
**and**  $a\text{-less-}b: a < b$  **and**  $b: b < m$  **and**  $j: j < n$   
**and**  $aj: A \ \$\$ \ (a, j) \neq 0$   
**shows** *invertible-mat* (*bezout-matrix-JNF*  $A \ a \ b \ j \ \text{bezout}$ )  
*<proof>*

**end**  
**end**

We continue with the soundness of the algorithm

**lemma** *bezout-matrix-JNF-mult-eq*:  
**assumes**  $A': A' \in \text{carrier-mat } m \ n$  **and**  $a: a \leq m$  **and**  $b: b \leq m$  **and**  $ab: a \neq b$   
**and**  $A\text{-def}: A = A' \ @_r \ B$  **and**  $B: B \in \text{carrier-mat } n \ n$   
**assumes**  $pquvd: (p, q, u, v, d) = \text{euclid-ext2 } (A \ \$\$ \ (a, j)) \ (A \ \$\$ \ (b, j))$   
**shows** *Matrix.mat* (*dim-row*  $A$ ) (*dim-col*  $A$ )  
 $(\lambda(i, k). \text{if } i = a \text{ then } (p * A \ \$\$ \ (a, k) + q * A \ \$\$ \ (b, k))$   
 $\quad \text{else if } i = b \text{ then } u * A \ \$\$ \ (a, k) + v * A \ \$\$ \ (b, k)$   
 $\quad \text{else } A \ \$\$ \ (i, k)$   
 $) = (\text{bezout-matrix-JNF } A \ a \ b \ j \ \text{euclid-ext2}) * A \ (\text{is } ?A = ?BM * A)$   
*<proof>*

**context** *proper-mod-operation*  
**begin**

**lemma** *reduce-invertible-mat*:  
**assumes**  $A': A' \in \text{carrier-mat } m \ n$  **and**  $a: a < m$  **and**  $j: 0 < n$  **and**  $b: b < m$  **and**  
 $ab: a \neq b$   
**and**  $A\text{-def}: A = A' \ @_r \ (D \cdot_m \ (1_m \ n))$   
**and**  $Aaj: A \ \$\$ \ (a, 0) \neq 0$   
**and**  $a\text{-less-}b: a < b$   
**and**  $mn: m \geq n$   
**and**  $D\text{-ge}0: D > 0$   
**shows**  $\exists P. \text{invertible-mat } P \wedge P \in \text{carrier-mat } (m+n) \ (m+n) \wedge (\text{reduce } a \ b \ D$

$A) = P * A$  (is ?thesis1)  
 <proof>

**lemma** *reduce-abs-invertible-mat*:

**assumes**  $A'$ :  $A' \in \text{carrier-mat } m \ n$  **and**  $a$ :  $a < m$  **and**  $j$ :  $0 < n$  **and**  $b$ :  $b < m$  **and**  
 $ab$ :  $a \neq b$   
**and**  $A\text{-def}$ :  $A = A' @_r (D \cdot_m (1_m \ n))$   
**and**  $Aaj$ :  $A \ \$\$ (a, 0) \neq 0$   
**and**  $a\text{-less-}b$ :  $a < b$   
**and**  $mn$ :  $m \geq n$   
**and**  $D\text{-ge}0$ :  $D > 0$   
**shows**  $\exists P$ . *invertible-mat*  $P \wedge P \in \text{carrier-mat } (m+n) \ (m+n) \wedge (\text{reduce-abs } a \ b \ D \ A) = P * A$  (is ?thesis1)  
 <proof>

**lemma** *reduce-element-mod-D-case-m'*:

**assumes**  $A\text{-def}$ :  $A = A' @_r \ B$  **and**  $B$ :  $B \in \text{carrier-mat } n \ n$   
**and**  $A'$ :  $A' \in \text{carrier-mat } m \ n$  **and**  $a$ :  $a \leq m$  **and**  $j$ :  $j < n$   
**and**  $mn$ :  $m \geq n$  **and**  $B1$ :  $B \ \$\$ (j, j) = D$  **and**  $B2$ :  $(\forall j' \in \{0..<n\} - \{j\}. B \ \$\$ (j, j') = 0)$   
**and**  $D0$ :  $D > 0$   
**shows** *reduce-element-mod-D*  $A \ a \ j \ D \ m = \text{Matrix.mat } (dim\text{-row } A) \ (dim\text{-col } A)$   
 $(\lambda(i, k). \text{if } i = a \wedge k = j \text{ then if } j = 0 \text{ then if } D \ \text{dvd } A \ \$\$ (i, k) \text{ then } D$   
 $\text{else } A \ \$\$ (i, k) \text{ else } A \ \$\$ (i, k) \ \text{gmod } D \text{ else } A \ \$\$ (i, k))$  (is - = ?A)  
 <proof>

**lemma** *reduce-element-mod-D-abs-case-m'*:

**assumes**  $A\text{-def}$ :  $A = A' @_r \ B$  **and**  $B$ :  $B \in \text{carrier-mat } n \ n$   
**and**  $A'$ :  $A' \in \text{carrier-mat } m \ n$  **and**  $a$ :  $a \leq m$  **and**  $j$ :  $j < n$   
**and**  $mn$ :  $m \geq n$  **and**  $B1$ :  $B \ \$\$ (j, j) = D$  **and**  $B2$ :  $(\forall j' \in \{0..<n\} - \{j\}. B \ \$\$ (j, j') = 0)$   
**and**  $D0$ :  $D > 0$   
**shows** *reduce-element-mod-D-abs*  $A \ a \ j \ D \ m = \text{Matrix.mat } (dim\text{-row } A) \ (dim\text{-col } A)$   
 $(\lambda(i, k). \text{if } i = a \wedge k = j \text{ then if } j = 0 \wedge D \ \text{dvd } A \ \$\$ (i, k) \text{ then } D \text{ else } A \ \$\$ (i, k) \ \text{gmod } D \text{ else } A \ \$\$ (i, k))$  (is - = ?A)  
 <proof>

**lemma** *reduce-row-mod-D-case-m'*:

**assumes**  $A\text{-def}$ :  $A = A' @_r \ B$  **and**  $B \in \text{carrier-mat } n \ n$   
**and**  $A'$ :  $A' \in \text{carrier-mat } m \ n$  **and**  $a < m$

**and**  $j: \forall j \in \text{set } xs. j < n \wedge (B \text{ \textasciitilde\textasciitilde } (j, j) = D) \wedge (\forall j' \in \{0..<n\} - \{j\}. B \text{ \textasciitilde\textasciitilde } (j, j') = 0)$   
**and**  $d: \text{distinct } xs \text{ and } m \geq n$   
**and**  $D: D > 0$   
**shows**  $\text{reduce-row-mod-}D \ A \ a \ xs \ D \ m = \text{Matrix.mat } (dim\text{-row } A) \ (dim\text{-col } A)$   
 $(\lambda(i,k). \text{if } i = a \wedge k \in \text{set } xs \text{ then if } k = 0 \text{ then if } D \text{ dvd } A \text{ \textasciitilde\textasciitilde } (i,k) \text{ then } D$   
 $\text{else } A \text{ \textasciitilde\textasciitilde } (i,k) \text{ else } A \text{ \textasciitilde\textasciitilde } (i,k) \text{ gmod } D \text{ else } A \text{ \textasciitilde\textasciitilde } (i,k))$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{reduce-row-mod-}D\text{-abs-case-}m'$ :  
**assumes**  $A\text{-def}: A = A' \text{ \textasciitilde\textasciitilde }_r \ B$  **and**  $B \in \text{carrier-mat } n \ n$   
**and**  $A': A' \in \text{carrier-mat } m \ n$  **and**  $a < m$   
**and**  $j: \forall j \in \text{set } xs. j < n \wedge (B \text{ \textasciitilde\textasciitilde } (j, j) = D) \wedge (\forall j' \in \{0..<n\} - \{j\}. B \text{ \textasciitilde\textasciitilde } (j, j') = 0)$   
**and**  $d: \text{distinct } xs \text{ and } m \geq n$   
**and**  $D: D > 0$   
**shows**  $\text{reduce-row-mod-}D\text{-abs } A \ a \ xs \ D \ m = \text{Matrix.mat } (dim\text{-row } A) \ (dim\text{-col } A)$   
 $(\lambda(i,k). \text{if } i = a \wedge k \in \text{set } xs \text{ then if } k = 0 \wedge D \text{ dvd } A \text{ \textasciitilde\textasciitilde } (i,k) \text{ then } D$   
 $\text{else } A \text{ \textasciitilde\textasciitilde } (i,k) \text{ gmod } D \text{ else } A \text{ \textasciitilde\textasciitilde } (i,k))$   
 $\langle \text{proof} \rangle$

**lemma**  
**assumes**  $A\text{-def}: A = A' \text{ \textasciitilde\textasciitilde }_r \ B$  **and**  $B: B \in \text{carrier-mat } n \ n$   
**and**  $A': A' \in \text{carrier-mat } m \ n$  **and**  $a < m$  **and**  $j: j < n$  **and**  $mn: m \geq n$   
**shows**  $\text{reduce-element-mod-}D\text{-invertible-mat-case-}m$ :  
 $\exists P. P \in \text{carrier-mat } (m+n) \ (m+n) \wedge \text{invertible-mat } P \wedge \text{reduce-element-mod-}D$   
 $A \ a \ j \ D \ m = P * A$  (**is**  $?thesis1$ )  
**and**  $\text{reduce-element-mod-}D\text{-abs-invertible-mat-case-}m$ :  
 $\exists P. P \in \text{carrier-mat } (m+n) \ (m+n) \wedge \text{invertible-mat } P \wedge$   
 $\text{reduce-element-mod-}D\text{-abs } A \ a \ j \ D \ m = P * A$  (**is**  $?thesis2$ )  
 $\langle \text{proof} \rangle$

**lemma**  $\text{reduce-row-mod-}D\text{-invertible-mat-case-}m$ :  
**assumes**  $A\text{-def}: A = A' \text{ \textasciitilde\textasciitilde }_r \ B$  **and**  $B \in \text{carrier-mat } n \ n$   
**and**  $A': A' \in \text{carrier-mat } m \ n$  **and**  $a < m$   
**and**  $j: \forall j \in \text{set } xs. j < n \wedge (B \text{ \textasciitilde\textasciitilde } (j, j) = D) \wedge (\forall j' \in \{0..<n\} - \{j\}. B \text{ \textasciitilde\textasciitilde } (j, j') = 0)$   
**and**  $mn: m \geq n$   
**shows**  $\exists P. P \in \text{carrier-mat } (m+n) \ (m+n) \wedge \text{invertible-mat } P \wedge$   
 $\text{reduce-row-mod-}D \ A \ a \ xs \ D \ m = P * A$   
 $\langle \text{proof} \rangle$

**lemma** *reduce-row-mod-D-abs-invertible-mat-case-m:*

**assumes** *A-def:*  $A = A' @_r B$  **and**  $B \in \text{carrier-mat } n \ n$   
**and**  $A'$ :  $A' \in \text{carrier-mat } m \ n$  **and**  $a: a < m$   
**and**  $j$ :  $\forall j \in \text{set } xs. j < n \wedge (B \$\$ (j, j) = D) \wedge (\forall j' \in \{0..<n\} - \{j\}. B \$\$ (j, j')$   
 $= 0)$   
**and**  $mn$ :  $m \geq n$   
**shows**  $\exists P. P \in \text{carrier-mat } (m+n) \ (m+n) \wedge \text{invertible-mat } P \wedge$   
 $\text{reduce-row-mod-D-abs } A \ a \ xs \ D \ m = P * A$   
*<proof>*

**lemma** *reduce-row-mod-D-case-m'':*

**assumes** *A-def:*  $A = A' @_r B$  **and**  $B \in \text{carrier-mat } n \ n$   
**and**  $A'$ :  $A' \in \text{carrier-mat } m \ n$  **and**  $a \leq m$   
**and**  $j$ :  $\forall j \in \text{set } xs. j < n \wedge (B \$\$ (j, j) = D) \wedge (\forall j' \in \{0..<n\} - \{j\}. B \$\$ (j, j')$   
 $= 0)$   
**and**  $d$ : *distinct*  $xs$  **and**  $m \geq n$  **and**  $0 \notin \text{set } xs$   
**and**  $D > 0$   
**shows**  $\text{reduce-row-mod-D } A \ a \ xs \ D \ m = \text{Matrix.mat } (dim\text{-row } A) \ (dim\text{-col } A)$   
 $(\lambda(i,k). \text{if } i = a \wedge k \in \text{set } xs \text{ then if } k = 0 \text{ then if } D \text{ dvd } A \$\$ (i,k) \text{ then } D$   
 $\text{else } A \$\$ (i,k) \text{ else } A \$\$ (i,k) \text{ gmod } D \text{ else } A \$\$ (i,k))$   
*<proof>*

**lemma** *reduce-row-mod-D-abs-case-m'':*

**assumes** *A-def:*  $A = A' @_r B$  **and**  $B \in \text{carrier-mat } n \ n$   
**and**  $A'$ :  $A' \in \text{carrier-mat } m \ n$  **and**  $a \leq m$   
**and**  $j$ :  $\forall j \in \text{set } xs. j < n \wedge (B \$\$ (j, j) = D) \wedge (\forall j' \in \{0..<n\} - \{j\}. B \$\$ (j, j')$   
 $= 0)$   
**and**  $d$ : *distinct*  $xs$  **and**  $m \geq n$  **and**  $0 \notin \text{set } xs$   
**and**  $D > 0$   
**shows**  $\text{reduce-row-mod-D-abs } A \ a \ xs \ D \ m = \text{Matrix.mat } (dim\text{-row } A) \ (dim\text{-col } A)$   
 $(\lambda(i,k). \text{if } i = a \wedge k \in \text{set } xs \text{ then if } k = 0 \wedge D \text{ dvd } A \$\$ (i,k) \text{ then } D$   
 $\text{else } A \$\$ (i,k) \text{ gmod } D \text{ else } A \$\$ (i,k))$   
*<proof>*

**lemma**

**assumes**  $A$ -def:  $A = A' @_r B$  **and**  $B: B \in \text{carrier-mat } n \ n$   
**and**  $A': A' \in \text{carrier-mat } m \ n$  **and**  $a: a \leq m$  **and**  $j: j < n$  **and**  $mn: m \geq n$  **and**  $j0: j \neq 0$   
**shows** *reduce-element-mod-D-invertible-mat-case-m'*:  
 $\exists P. P \in \text{carrier-mat } (m+n) \ (m+n) \wedge \text{invertible-mat } P \wedge \text{reduce-element-mod-D}$   
 $A \ a \ j \ D \ m = P * A$  (**is** ?thesis1)  
**and** *reduce-element-mod-D-abs-invertible-mat-case-m'*:  
 $\exists P. P \in \text{carrier-mat } (m+n) \ (m+n) \wedge \text{invertible-mat } P \wedge \text{reduce-element-mod-D-abs}$   
 $A \ a \ j \ D \ m = P * A$  (**is** ?thesis2)  
 <proof>

**lemma** *reduce-row-mod-D-invertible-mat-case-m'*:  
**assumes**  $A$ -def:  $A = A' @_r B$  **and**  $B \in \text{carrier-mat } n \ n$   
**and**  $A': A' \in \text{carrier-mat } m \ n$  **and**  $a: a \leq m$   
**and**  $j: \forall j \in \text{set } xs. j < n \wedge (B \ \$\$ (j, j) = D) \wedge (\forall j' \in \{0..<n\} - \{j\}. B \ \$\$ (j, j') = 0)$   
**and**  $d: \text{distinct } xs$  **and**  $mn: m \geq n$  **and**  $0 \notin \text{set } xs$   
**shows**  $\exists P. P \in \text{carrier-mat } (m+n) \ (m+n) \wedge \text{invertible-mat } P \wedge$   
 $\text{reduce-row-mod-D } A \ a \ xs \ D \ m = P * A$   
 <proof>

**lemma** *reduce-row-mod-D-abs-invertible-mat-case-m'*:  
**assumes**  $A$ -def:  $A = A' @_r B$  **and**  $B \in \text{carrier-mat } n \ n$   
**and**  $A': A' \in \text{carrier-mat } m \ n$  **and**  $a: a \leq m$   
**and**  $j: \forall j \in \text{set } xs. j < n \wedge (B \ \$\$ (j, j) = D) \wedge (\forall j' \in \{0..<n\} - \{j\}. B \ \$\$ (j, j') = 0)$   
**and**  $d: \text{distinct } xs$  **and**  $mn: m \geq n$  **and**  $0 \notin \text{set } xs$   
**shows**  $\exists P. P \in \text{carrier-mat } (m+n) \ (m+n) \wedge \text{invertible-mat } P \wedge$   
 $\text{reduce-row-mod-D-abs } A \ a \ xs \ D \ m = P * A$   
 <proof>

**lemma** *reduce-invertible-mat-case-m*:  
**assumes**  $A': A' \in \text{carrier-mat } m \ n$  **and**  $B: B \in \text{carrier-mat } n \ n$   
**and**  $a: a < m$  **and**  $ab: a \neq m$   
**and**  $A$ -def:  $A = A' @_r B$   
**and**  $j: \forall j \in \text{set } xs. j < n \wedge (B \ \$\$ (j, j) = D) \wedge (\forall j' \in \{0..<n\} - \{j\}. B \ \$\$ (j, j') = 0)$   
**and**  $Aaj: A \ \$\$ (a, 0) \neq 0$   
**and**  $mn: m \geq n$   
**and**  $n0: 0 < n$   
**and**  $pquvd: (p, q, u, v, d) = \text{euclid-ext2 } (A \ \$\$ (a, 0)) \ (A \ \$\$ (m, 0))$   
**and**  $A2$ -def:  $A2 = \text{Matrix.mat } (\text{dim-row } A) \ (\text{dim-col } A)$   
 $(\lambda(i, k). \text{if } i = a \text{ then } (p * A \ \$\$ (a, k) + q * A \ \$\$ (m, k))$   
 $\text{else if } i = m \text{ then } u * A \ \$\$ (a, k) + v * A \ \$\$ (m, k)$   
 $\text{else } A \ \$\$ (i, k))$

)  
**and** *xs-def*:  $xs = [1..<n]$   
**and** *ys-def*:  $ys = [1..<n]$   
**and** *j-ys*:  $\forall j \in \text{set } ys. j < n \wedge (B \text{ \$\$ } (j, j) = D) \wedge (\forall j' \in \{0..<n\} - \{j\}. B \text{ \$\$ } (j, j') = 0)$   
**and** *D0*:  $D > 0$   
**and** *Am0-D*:  $A \text{ \$\$ } (m, 0) \in \{0, D\}$   
**and** *Am0-D2*:  $A \text{ \$\$ } (m, 0) = 0 \longrightarrow A \text{ \$\$ } (a, 0) = D$   
**shows**  $\exists P. \text{invertible-mat } P \wedge P \in \text{carrier-mat } (m+n) (m+n) \wedge (\text{reduce } a \text{ } m \text{ } D \text{ } A) = P * A$   
 <proof>

**lemma** *reduce-abs-invertible-mat-case-m*:

**assumes** *A'*:  $A' \in \text{carrier-mat } m \text{ } n$  **and** *B*:  $B \in \text{carrier-mat } n \text{ } n$   
**and** *a*:  $a < m$  **and** *ab*:  $a \neq m$   
**and** *A-def*:  $A = A' @_r B$   
**and** *j*:  $\forall j \in \text{set } xs. j < n \wedge (B \text{ \$\$ } (j, j) = D) \wedge (\forall j' \in \{0..<n\} - \{j\}. B \text{ \$\$ } (j, j') = 0)$   
**and** *Aaj*:  $A \text{ \$\$ } (a, 0) \neq 0$   
**and** *mn*:  $m \geq n$   
**and** *n0*:  $0 < n$   
**and** *pqvd*:  $(p, q, u, v, d) = \text{euclid-ext2 } (A \text{ \$\$ } (a, 0)) (A \text{ \$\$ } (m, 0))$   
**and** *A2-def*:  $A2 = \text{Matrix.mat } (\text{dim-row } A) (\text{dim-col } A)$   
 $(\lambda(i, k). \text{if } i = a \text{ then } (p * A \text{ \$\$ } (a, k) + q * A \text{ \$\$ } (m, k))$   
 $\text{else if } i = m \text{ then } u * A \text{ \$\$ } (a, k) + v * A \text{ \$\$ } (m, k)$   
 $\text{else } A \text{ \$\$ } (i, k))$   
 )  
**and** *xs-def*:  $xs = \text{filter } (\lambda i. \text{abs } (A2 \text{ \$\$ } (a, i)) > D) [0..<n]$   
**and** *ys-def*:  $ys = \text{filter } (\lambda i. \text{abs } (A2 \text{ \$\$ } (m, i)) > D) [0..<n]$   
**and** *j-ys*:  $\forall j \in \text{set } ys. j < n \wedge (B \text{ \$\$ } (j, j) = D) \wedge (\forall j' \in \{0..<n\} - \{j\}. B \text{ \$\$ } (j, j') = 0)$   
**and** *D0*:  $D > 0$   
**shows**  $\exists P. \text{invertible-mat } P \wedge P \in \text{carrier-mat } (m+n) (m+n) \wedge (\text{reduce-abs } a \text{ } m \text{ } D \text{ } A) = P * A$   
 <proof>

**lemma** *reduce-not0*:

**assumes** *A*:  $A \in \text{carrier-mat } m \text{ } n$  **and** *a*:  $a < m$  **and** *a-less-b*:  $a < b$  **and** *j*:  $0 < n$   
**and** *b*:  $b < m$   
**and** *Aaj*:  $A \text{ \$\$ } (a, 0) \neq 0$  **and** *D0*:  $D \neq 0$   
**shows**  $\text{reduce } a \text{ } b \text{ } D \text{ } A \text{ \$\$ } (a, 0) \neq 0$  (**is**  $? \text{reduce } \text{\$\$ } (a, 0) \neq -$ )  
**and**  $\text{reduce-abs } a \text{ } b \text{ } D \text{ } A \text{ \$\$ } (a, 0) \neq 0$  (**is**  $? \text{reduce-abs } \text{\$\$ } (a, 0) \neq -$ )  
 <proof>

**lemma** *reduce-below-not0*:  
**assumes**  $A: A \in \text{carrier-mat } m \ n$  **and**  $a: a < m$  **and**  $j: 0 < n$   
**and**  $Aaj: A \ \$\$ (a, 0) \neq 0$   
**and** *distinct xs* **and**  $\forall x \in \text{set } xs. x < m \wedge a < x$   
**and**  $D \neq 0$   
**shows** *reduce-below a xs D A*  $\ \$\$ (a, 0) \neq 0$  (**is**  $?R \ \$\$ (a, 0) \neq -$ )  
*<proof>*

**lemma** *reduce-below-abs-not0*:  
**assumes**  $A: A \in \text{carrier-mat } m \ n$  **and**  $a: a < m$  **and**  $j: 0 < n$   
**and**  $Aaj: A \ \$\$ (a, 0) \neq 0$   
**and** *distinct xs* **and**  $\forall x \in \text{set } xs. x < m \wedge a < x$   
**and**  $D \neq 0$   
**shows** *reduce-below-abs a xs D A*  $\ \$\$ (a, 0) \neq 0$  (**is**  $?R \ \$\$ (a, 0) \neq -$ )  
*<proof>*

**lemma** *reduce-below-not0-case-m*:  
**assumes**  $A': A' \in \text{carrier-mat } m \ n$  **and**  $a: a < m$  **and**  $j: 0 < n$   
**and**  $A\text{-def}: A = A' \ @_r (D \cdot_m (1_m \ n))$   
**and**  $Aaj: A \ \$\$ (a, 0) \neq 0$   
**and**  $mn: m \geq n$   
**and**  $\forall x \in \text{set } xs. x < m \wedge a < x$   
**and**  $D \neq 0$   
**shows** *reduce-below a (xs@[m]) D A*  $\ \$\$ (a, 0) \neq 0$  (**is**  $?R \ \$\$ (a, 0) \neq -$ )  
*<proof>*

**lemma** *reduce-below-abs-not0-case-m*:  
**assumes**  $A': A' \in \text{carrier-mat } m \ n$  **and**  $a: a < m$  **and**  $j: 0 < n$   
**and**  $A\text{-def}: A = A' \ @_r (D \cdot_m (1_m \ n))$   
**and**  $Aaj: A \ \$\$ (a, 0) \neq 0$   
**and**  $mn: m \geq n$   
**and**  $\forall x \in \text{set } xs. x < m \wedge a < x$   
**and**  $D \neq 0$   
**shows** *reduce-below-abs a (xs@[m]) D A*  $\ \$\$ (a, 0) \neq 0$  (**is**  $?R \ \$\$ (a, 0) \neq -$ )  
*<proof>*

**lemma** *reduce-below-invertible-mat*:  
**assumes**  $A': A' \in \text{carrier-mat } m \ n$  **and**  $a: a < m$  **and**  $j: 0 < n$   
**and**  $A\text{-def}: A = A' \ @_r (D \cdot_m (1_m \ n))$   
**and**  $Aaj: A \ \$\$ (a, 0) \neq 0$   
**and** *distinct xs* **and**  $\forall x \in \text{set } xs. x < m \wedge a < x$

**and**  $m \geq n$   
**and**  $D > 0$   
**shows**  $(\exists P. \text{invertible-mat } P \wedge P \in \text{carrier-mat } (m+n) (m+n) \wedge \text{reduce-below}$   
 $a \text{ xs } D \ A = P * A)$   
 $\langle \text{proof} \rangle$

**lemma** *reduce-below-abs-invertible-mat*:

**assumes**  $A': A' \in \text{carrier-mat } m \ n$  **and**  $a: a < m$  **and**  $j: 0 < n$   
**and**  $A\text{-def}: A = A' @_r (D \cdot_m (1_m \ n))$   
**and**  $Aaj: A \ \$\$ (a, 0) \neq 0$   
**and** *distinct xs* **and**  $\forall x \in \text{set } xs. x < m \wedge a < x$   
**and**  $m \geq n$   
**and**  $D > 0$   
**shows**  $(\exists P. \text{invertible-mat } P \wedge P \in \text{carrier-mat } (m+n) (m+n) \wedge \text{reduce-below-abs}$   
 $a \text{ xs } D \ A = P * A)$   
 $\langle \text{proof} \rangle$

**lemma** *reduce-below-preserves*:

**assumes**  $A': A' \in \text{carrier-mat } m \ n$  **and**  $a: a < m$  **and**  $j: j < n$   
**and**  $A\text{-def}: A = A' @_r (D \cdot_m (1_m \ n))$   
**and**  $Aaj: A \ \$\$ (a, 0) \neq 0$   
**and**  $mn: m \geq n$   
**assumes**  $i \notin \text{set } xs$  **and** *distinct xs* **and**  $\forall x \in \text{set } xs. x < m \wedge a < x$   
**and**  $i \neq a$  **and**  $i < m+n$   
**and**  $D > 0$   
**shows** *reduce-below*  $a \text{ xs } D \ A \ \$\$ (i, j) = A \ \$\$ (i, j)$   
 $\langle \text{proof} \rangle$

**lemma** *reduce-below-abs-preserves*:

**assumes**  $A': A' \in \text{carrier-mat } m \ n$  **and**  $a: a < m$  **and**  $j: j < n$   
**and**  $A\text{-def}: A = A' @_r (D \cdot_m (1_m \ n))$   
**and**  $Aaj: A \ \$\$ (a, 0) \neq 0$   
**and**  $mn: m \geq n$   
**assumes**  $i \notin \text{set } xs$  **and** *distinct xs* **and**  $\forall x \in \text{set } xs. x < m \wedge a < x$   
**and**  $i \neq a$  **and**  $i < m+n$   
**and**  $D > 0$   
**shows** *reduce-below-abs*  $a \text{ xs } D \ A \ \$\$ (i, j) = A \ \$\$ (i, j)$   
 $\langle \text{proof} \rangle$

**lemma** *reduce-below-0*:

**assumes**  $A': A' \in \text{carrier-mat } m \ n$  **and**  $a: a < m$  **and**  $j: 0 < n$

**and**  $A$ -def:  $A = A' @_r (D \cdot_m (1_m n))$   
**and**  $Aaj$ :  $A \ \$\$ (a,0) \neq 0$   
**and**  $mn$ :  $m \geq n$   
**assumes**  $i \in \text{set } xs$  **and** *distinct*  $xs$  **and**  $\forall x \in \text{set } xs. x < m \wedge a < x$   
**and**  $D > 0$   
**shows** *reduce-below a xs*  $D A \ \$\$ (i,0) = 0$   
*<proof>*

**lemma** *reduce-below-abs-0*:

**assumes**  $A'$ :  $A' \in \text{carrier-mat } m n$  **and**  $a: a < m$  **and**  $j: 0 < n$   
**and**  $A$ -def:  $A = A' @_r (D \cdot_m (1_m n))$   
**and**  $Aaj$ :  $A \ \$\$ (a,0) \neq 0$   
**and**  $mn$ :  $m \geq n$   
**assumes**  $i \in \text{set } xs$  **and** *distinct*  $xs$  **and**  $\forall x \in \text{set } xs. x < m \wedge a < x$   
**and**  $D > 0$   
**shows** *reduce-below-abs a xs*  $D A \ \$\$ (i,0) = 0$   
*<proof>*

**lemma** *reduce-below-preserves-case-m*:

**assumes**  $A'$ :  $A' \in \text{carrier-mat } m n$  **and**  $a: a < m$  **and**  $j: j < n$   
**and**  $A$ -def:  $A = A' @_r (D \cdot_m (1_m n))$   
**and**  $Aaj$ :  $A \ \$\$ (a,0) \neq 0$   
**and**  $mn$ :  $m \geq n$   
**assumes**  $i \notin \text{set } xs$  **and** *distinct*  $xs$  **and**  $\forall x \in \text{set } xs. x < m \wedge a < x$   
**and**  $i \neq a$  **and**  $i < m+n$  **and**  $i \neq m$   
**and**  $D > 0$   
**shows** *reduce-below a (xs @ [m])*  $D A \ \$\$ (i,j) = A \ \$\$ (i,j)$   
*<proof>*

**lemma** *reduce-below-abs-preserves-case-m*:

**assumes**  $A'$ :  $A' \in \text{carrier-mat } m n$  **and**  $a: a < m$  **and**  $j: j < n$   
**and**  $A$ -def:  $A = A' @_r (D \cdot_m (1_m n))$   
**and**  $Aaj$ :  $A \ \$\$ (a,0) \neq 0$   
**and**  $mn$ :  $m \geq n$   
**assumes**  $i \notin \text{set } xs$  **and** *distinct*  $xs$  **and**  $\forall x \in \text{set } xs. x < m \wedge a < x$   
**and**  $i \neq a$  **and**  $i < m+n$  **and**  $i \neq m$   
**and**  $D > 0$   
**shows** *reduce-below-abs a (xs @ [m])*  $D A \ \$\$ (i,j) = A \ \$\$ (i,j)$   
*<proof>*

**lemma** *reduce-below-0-case-m1*:

**assumes**  $A'$ :  $A' \in \text{carrier-mat } m n$  **and**  $a: a < m$  **and**  $j: 0 < n$   
**and**  $A$ -def:  $A = A' @_r (D \cdot_m (1_m n))$

**and**  $Aaj: A \text{ \textcircled{\$} } (a,0) \neq 0$   
**and**  $mn: m \geq n$   
**assumes** *distinct xs* **and**  $\forall x \in \text{set } xs. x < m \wedge a < x$   
**and**  $m \neq a$   
**and**  $D > 0$   
**shows** *reduce-below a*  $(xs \text{ \textcircled{[} } m] ) D A \text{ \textcircled{\$} } (m,0) = 0$   
*\langle proof \rangle*

**lemma** *reduce-below-abs-0-case-m1:*

**assumes**  $A': A' \in \text{carrier-mat } m \ n$  **and**  $a: a < m$  **and**  $j: 0 < n$   
**and**  $A\text{-def}: A = A' \text{ \textcircled{@}_r } (D \cdot_m (1_m \ n))$   
**and**  $Aaj: A \text{ \textcircled{\$} } (a,0) \neq 0$   
**and**  $mn: m \geq n$   
**assumes** *distinct xs* **and**  $\forall x \in \text{set } xs. x < m \wedge a < x$   
**and**  $m \neq a$   
**and**  $D > 0$   
**shows** *reduce-below-abs a*  $(xs \text{ \textcircled{[} } m] ) D A \text{ \textcircled{\$} } (m,0) = 0$   
*\langle proof \rangle*

**lemma** *reduce-below-preserves-case-m2:*

**assumes**  $A': A' \in \text{carrier-mat } m \ n$  **and**  $a: a < m$  **and**  $j: 0 < n$   
**and**  $A\text{-def}: A = A' \text{ \textcircled{@}_r } (D \cdot_m (1_m \ n))$   
**and**  $Aaj: A \text{ \textcircled{\$} } (a,0) \neq 0$   
**and**  $mn: m \geq n$   
**assumes**  $i \in \text{set } xs$  **and** *distinct xs* **and**  $\forall x \in \text{set } xs. x < m \wedge a < x$   
**and**  $i \neq a$  **and**  $i < m+n$   
**and**  $D > 0$   
**shows** *reduce-below a*  $(xs \text{ \textcircled{[} } m] ) D A \text{ \textcircled{\$} } (i,0) = \text{reduce-below a } xs \ D \ A \ \text{\textcircled{\$} } (i,0)$   
*\langle proof \rangle*

**lemma** *reduce-below-abs-preserves-case-m2:*

**assumes**  $A': A' \in \text{carrier-mat } m \ n$  **and**  $a: a < m$  **and**  $j: 0 < n$   
**and**  $A\text{-def}: A = A' \text{ \textcircled{@}_r } (D \cdot_m (1_m \ n))$   
**and**  $Aaj: A \text{ \textcircled{\$} } (a,0) \neq 0$   
**and**  $mn: m \geq n$   
**assumes**  $i \in \text{set } xs$  **and** *distinct xs* **and**  $\forall x \in \text{set } xs. x < m \wedge a < x$   
**and**  $i \neq a$  **and**  $i < m+n$   
**and**  $D > 0$   
**shows** *reduce-below-abs a*  $(xs \text{ \textcircled{[} } m] ) D A \text{ \textcircled{\$} } (i,0) = \text{reduce-below-abs a } xs \ D \ A \ \text{\textcircled{\$} } (i,0)$   
*\langle proof \rangle*

**lemma** *reduce-below-0-case-m:*

**assumes**  $A': A' \in \text{carrier-mat } m \ n$  **and**  $a: a < m$  **and**  $j: 0 < n$   
**and**  $A\text{-def}: A = A' \text{ \textcircled{@}_r } (D \cdot_m (1_m \ n))$

**and**  $Aaj: A \ \$\$ (a,0) \neq 0$   
**and**  $mn: m \geq n$   
**assumes**  $i \in \text{set } (xs \ @ \ [m])$  **and** *distinct xs* **and**  $\forall x \in \text{set } xs. x < m \wedge a < x$   
**and**  $D > 0$   
**shows** *reduce-below a*  $(xs \ @ \ [m]) \ D \ A \ \$\$ (i,0) = 0$   
 $\langle \text{proof} \rangle$

**lemma** *reduce-below-abs-0-case-m:*

**assumes**  $A': A' \in \text{carrier-mat } m \ n$  **and**  $a: a < m$  **and**  $j: 0 < n$   
**and**  $A\text{-def}: A = A' \ @_r (D \cdot_m (1_m \ n))$   
**and**  $Aaj: A \ \$\$ (a,0) \neq 0$   
**and**  $mn: m \geq n$   
**assumes**  $i \in \text{set } (xs \ @ \ [m])$  **and** *distinct xs* **and**  $\forall x \in \text{set } xs. x < m \wedge a < x$   
**and**  $D > 0$   
**shows** *reduce-below-abs a*  $(xs \ @ \ [m]) \ D \ A \ \$\$ (i,0) = 0$   
 $\langle \text{proof} \rangle$

**lemma** *reduce-below-0-case-m-complete:*

**assumes**  $A': A' \in \text{carrier-mat } m \ n$  **and**  $a: 0 < m$  **and**  $j: 0 < n$   
**and**  $A\text{-def}: A = A' \ @_r (D \cdot_m (1_m \ n))$   
**and**  $Aaj: A \ \$\$ (0,0) \neq 0$   
**and**  $mn: m \geq n$   
**assumes**  $i\text{-mn}: i < m+n$  **and**  $d\text{-xs}: \text{distinct } xs$  **and**  $xs: \forall x \in \text{set } xs. x < m \wedge 0 < x$   
**and**  $ia: i \neq 0$   
**and**  $xs\text{-def}: xs = \text{filter } (\lambda i. A \ \$\$ (i,0) \neq 0) [1..<dim\text{-row } A]$   
**and**  $D: D > 0$   
**shows** *reduce-below 0*  $(xs \ @ \ [m]) \ D \ A \ \$\$ (i,0) = 0$   
 $\langle \text{proof} \rangle$

**lemma** *reduce-below-abs-0-case-m-complete:*

**assumes**  $A': A' \in \text{carrier-mat } m \ n$  **and**  $a: 0 < m$  **and**  $j: 0 < n$   
**and**  $A\text{-def}: A = A' \ @_r (D \cdot_m (1_m \ n))$   
**and**  $Aaj: A \ \$\$ (0,0) \neq 0$   
**and**  $mn: m \geq n$   
**assumes**  $i\text{-mn}: i < m+n$  **and**  $d\text{-xs}: \text{distinct } xs$  **and**  $xs: \forall x \in \text{set } xs. x < m \wedge 0 < x$   
**and**  $ia: i \neq 0$   
**and**  $xs\text{-def}: xs = \text{filter } (\lambda i. A \ \$\$ (i,0) \neq 0) [1..<dim\text{-row } A]$   
**and**  $D: D > 0$   
**shows** *reduce-below-abs 0*  $(xs \ @ \ [m]) \ D \ A \ \$\$ (i,0) = 0$   
 $\langle \text{proof} \rangle$

**lemma** *reduce-below-invertible-mat-case-m:*  
**assumes**  $A'$ :  $A' \in \text{carrier-mat } m \ n$  **and**  $a$ :  $a < m$  **and**  $n0$ :  $0 < n$   
**and**  $A$ -def:  $A = A' @_r (D \cdot_m (1_m \ n))$   
**and**  $Aaj$ :  $A \ \$\$ (a, 0) \neq 0$   
**and**  $mn$ :  $m \geq n$  **and** *distinct*  $xs$  **and**  $\forall x \in \text{set } xs. x < m \wedge a < x$   
**and**  $D0$ :  $D > 0$   
**shows**  $(\exists P. \text{invertible-mat } P \wedge P \in \text{carrier-mat } (m+n) \ (m+n) \wedge \text{reduce-below}$   
 $a \ (xs@[m]) \ D \ A = P * A)$   
 $\langle \text{proof} \rangle$

**lemma** *reduce-below-abs-invertible-mat-case-m:*  
**assumes**  $A'$ :  $A' \in \text{carrier-mat } m \ n$  **and**  $a$ :  $a < m$  **and**  $n0$ :  $0 < n$   
**and**  $A$ -def:  $A = A' @_r (D \cdot_m (1_m \ n))$   
**and**  $Aaj$ :  $A \ \$\$ (a, 0) \neq 0$   
**and**  $mn$ :  $m \geq n$  **and** *distinct*  $xs$  **and**  $\forall x \in \text{set } xs. x < m \wedge a < x$   
**and**  $D0$ :  $D > 0$   
**shows**  $(\exists P. \text{invertible-mat } P \wedge P \in \text{carrier-mat } (m+n) \ (m+n) \wedge \text{reduce-below-abs}$   
 $a \ (xs@[m]) \ D \ A = P * A)$   
 $\langle \text{proof} \rangle$

**end**

**hide-const** (open)  $C$

This lemma will be very important, since it will allow us to prove that the output matrix is in echelon form.

**lemma** *echelon-form-four-block-mat:*  
**assumes**  $A$ :  $A \in \text{carrier-mat } 1 \ 1$   
**and**  $B$ :  $B \in \text{carrier-mat } 1 \ (n-1)$   
**and**  $D$ :  $D \in \text{carrier-mat } (m-1) \ (n-1)$   
**and**  $H$ -def:  $H = \text{four-block-mat } A \ B \ (0_m \ (m-1) \ 1) \ D$   
**and**  $A00$ :  $A \ \$\$ (0, 0) \neq 0$   
**and**  $e$ - $D$ : *echelon-form-JNF*  $D$   
**and**  $m$ :  $m > 0$  **and**  $n$ :  $n > 0$   
**shows** *echelon-form-JNF*  $H$   
 $\langle \text{proof} \rangle$

**context** *mod-operation*  
**begin**

**lemma** *reduce-below:*  
**assumes**  $A \in \text{carrier-mat } m \ n$   
**shows** *reduce-below*  $a \ xs \ D \ A \in \text{carrier-mat } m \ n$   
 $\langle \text{proof} \rangle$

**lemma** *reduce-below-preserves-dimensions*:  
**shows**  $[simp]: \dim\text{-row } (\text{reduce-below } a \text{ } xs \ D \ A) = \dim\text{-row } A$   
**and**  $[simp]: \dim\text{-col } (\text{reduce-below } a \text{ } xs \ D \ A) = \dim\text{-col } A$   
 $\langle \text{proof} \rangle$

**lemma** *reduce-below-abs*:  
**assumes**  $A \in \text{carrier-mat } m \ n$   
**shows**  $\text{reduce-below-abs } a \text{ } xs \ D \ A \in \text{carrier-mat } m \ n$   
 $\langle \text{proof} \rangle$

**lemma** *reduce-below-abs-preserves-dimensions*:  
**shows**  $[simp]: \dim\text{-row } (\text{reduce-below-abs } a \text{ } xs \ D \ A) = \dim\text{-row } A$   
**and**  $[simp]: \dim\text{-col } (\text{reduce-below-abs } a \text{ } xs \ D \ A) = \dim\text{-col } A$   
 $\langle \text{proof} \rangle$

**lemma** *FindPreHNF-1xn*:  
**assumes**  $A: A \in \text{carrier-mat } m \ n$  **and**  $m < 2 \vee n = 0$   
**shows**  $\text{FindPreHNF abs-flag } D \ A \in \text{carrier-mat } m \ n$   $\langle \text{proof} \rangle$

**lemma** *FindPreHNF-mx1*:  
**assumes**  $A: A \in \text{carrier-mat } m \ n$  **and**  $m \geq 2$  **and**  $n \neq 0 \ n < 2$   
**shows**  $\text{FindPreHNF abs-flag } D \ A \in \text{carrier-mat } m \ n$   
 $\langle \text{proof} \rangle$

**lemma** *FindPreHNF-mxn2*:  
**assumes**  $A: A \in \text{carrier-mat } m \ n$  **and**  $m: m \geq 2$  **and**  $n: n \geq 2$   
**shows**  $\text{FindPreHNF abs-flag } D \ A \in \text{carrier-mat } m \ n$   
 $\langle \text{proof} \rangle$

**lemma** *FindPreHNF*:  
**assumes**  $A: A \in \text{carrier-mat } m \ n$   
**shows**  $\text{FindPreHNF abs-flag } D \ A \in \text{carrier-mat } m \ n$   
 $\langle \text{proof} \rangle$   
**end**

**lemma** *make-first-column-positive-append-id*:  
**assumes**  $A': A' \in \text{carrier-mat } m \ n$   
**and**  $A\text{-def}: A = A' @_r (D \cdot_m (1_m \ n))$   
**and**  $D0: D > 0$   
**and**  $n0: 0 < n$   
**shows**  $\text{make-first-column-positive } A$   
 $= \text{mat-of-rows } n \ (\text{map } (\text{Matrix.row } (\text{make-first-column-positive } A)) \ [0..<m]) @_r$   
 $(D \cdot_m (1_m \ n))$   
 $\langle \text{proof} \rangle$

**lemma** *A'-swaprows-invertible-mat:*

**fixes**  $A::\text{int mat}$   
**assumes**  $A: A \in \text{carrier-mat } m \ n$   
**assumes**  $A'\text{-def: } A' = (\text{if } A \ \$\$ (0, 0) \neq 0 \ \text{then } A \ \text{else let } i = \text{non-zero-positions}$   
 $! \ 0 \ \text{in swaprows } 0 \ i \ A)$   
**and**  $\text{nz-def: non-zero-positions} = \text{filter } (\lambda i. A \ \$\$ (i, 0) \neq 0) [1..<\text{dim-row } A]$   
**and**  $\text{nz-empty: } A \ \$\$ (0, 0) = 0 \implies \text{non-zero-positions} \neq []$   
**and**  $m0: 0 < m$   
**shows**  $\exists P. P \in \text{carrier-mat } m \ m \wedge \text{invertible-mat } P \wedge A' = P * A$   
 $\langle \text{proof} \rangle$

**lemma** *swaprows-append-id:*

**assumes**  $A': A' \in \text{carrier-mat } m \ n$   
**and**  $A\text{-def: } A = A' @_r (D \cdot_m (1_m \ n))$   
**and**  $i: i < m$   
**shows**  $\text{swaprows } 0 \ i \ A$   
 $= \text{mat-of-rows } n \ (\text{map } (\text{Matrix.row } (\text{swaprows } 0 \ i \ A)) [0..<m]) @_r (D \cdot_m (1_m$   
 $n))$   
 $\langle \text{proof} \rangle$

**lemma** *non-zero-positions-xs-m:*

**fixes**  $A::'a::\text{comm-ring-1 mat}$   
**assumes**  $A\text{-def: } A = A' @_r D \cdot_m 1_m \ n$   
**and**  $A': A' \in \text{carrier-mat } m \ n$   
**and**  $\text{nz-def: non-zero-positions} = \text{filter } (\lambda i. A \ \$\$ (i, 0) \neq 0) [1..<\text{dim-row } A]$   
**and**  $m0: 0 < m$  **and**  $n0: 0 < n$   
**and**  $D0: D \neq 0$   
**shows**  $\exists xs. \text{non-zero-positions} = xs @ [m] \wedge \text{distinct } xs \wedge (\forall x \in \text{set } xs. x < m \wedge 0 < x)$   
 $\langle \text{proof} \rangle$

**lemma** *non-zero-positions-xs-m':*

**fixes**  $A::'a::\text{comm-ring-1 mat}$   
**assumes**  $A\text{-def: } A = A' @_r D \cdot_m 1_m \ n$   
**and**  $A': A' \in \text{carrier-mat } m \ n$   
**and**  $\text{nz-def: non-zero-positions} = \text{filter } (\lambda i. A \ \$\$ (i, 0) \neq 0) [1..<\text{dim-row } A]$   
**and**  $m0: 0 < m$  **and**  $n0: 0 < n$   
**and**  $D0: D \neq 0$   
**shows**  $\text{non-zero-positions} = (\text{filter } (\lambda i. A \ \$\$ (i, 0) \neq 0) [1..<m]) @ [m]$   
 $\wedge \text{distinct } (\text{filter } (\lambda i. A \ \$\$ (i, 0) \neq 0) [1..<m])$   
 $\wedge (\forall x \in \text{set } (\text{filter } (\lambda i. A \ \$\$ (i, 0) \neq 0) [1..<m]). x < m \wedge 0 < x)$   
 $\langle \text{proof} \rangle$

**lemma** *A-A'D-eq-first-n-rows*:  
**assumes** *A-def*:  $A = A' @_r D \cdot_m 1_m n$   
**and** *A'*:  $A' \in \text{carrier-mat } m \ n$   
**and** *mn*:  $m \geq n$   
**shows**  $(\text{mat-of-rows } n \ (\text{map } (\text{Matrix.row } A') \ [0..<n]))$   
 $= (\text{mat-of-rows } n \ (\text{map } (\text{Matrix.row } A) \ [0..<n]))$  (**is** *?lhs = ?rhs*)  
*<proof>*

**lemma** *non-zero-positions-xs-m-invertible*:  
**assumes** *A-def*:  $A = A' @_r D \cdot_m 1_m n$   
**and** *A'*:  $A' \in \text{carrier-mat } m \ n$   
**and** *nz-def*:  $\text{non-zero-positions} = \text{filter } (\lambda i. A \ \$\$ (i,0) \neq 0) \ [1..<\text{dim-row } A]$   
**and** *m0*:  $0 < m$  **and** *n0*:  $0 < n$   
**and** *D0*:  $D \neq 0$   
**and** *inv-A''*:  $\text{invertible-mat } (\text{map-mat rat-of-int } (\text{mat-of-rows } n \ (\text{map } (\text{Matrix.row } A') \ [0..<n])))$   
**and** *A'00*:  $A' \ \$\$ (0,0) = 0$   
**and** *mn*:  $m \geq n$   
**shows**  $\text{length non-zero-positions} > 1$   
*<proof>*

**corollary** *non-zero-positions-length-xs*:  
**assumes** *A-def*:  $A = A' @_r D \cdot_m 1_m n$   
**and** *A'*:  $A' \in \text{carrier-mat } m \ n$   
**and** *nz-def*:  $\text{non-zero-positions} = \text{filter } (\lambda i. A \ \$\$ (i,0) \neq 0) \ [1..<\text{dim-row } A]$   
**and** *m0*:  $0 < m$  **and** *n0*:  $0 < n$   
**and** *D0*:  $D \neq 0$   
**and** *inv-A''*:  $\text{invertible-mat } (\text{map-mat rat-of-int } (\text{mat-of-rows } n \ (\text{map } (\text{Matrix.row } A') \ [0..<n])))$   
**and** *A'00*:  $A' \ \$\$ (0,0) = 0$   
**and** *mn*:  $m \geq n$   
**and** *nz-xs-m*:  $\text{non-zero-positions} = \text{xs } @ \ [m]$   
**shows**  $\text{length xs} > 0$   
*<proof>*

**lemma** *make-first-column-positive-nz-conv*:  
**assumes**  $i < \text{dim-row } A$  **and**  $j < \text{dim-col } A$   
**shows**  $(\text{make-first-column-positive } A \ \$\$ (i, j) \neq 0) = (A \ \$\$ (i, j) \neq 0)$   
*<proof>*

**lemma** *make-first-column-positive-00*:  
**assumes** *A-def*:  $A = A'' @_r D \cdot_m 1_m n$

**and**  $A''$ :  $A''$  : carrier-mat  $m$   $n$   
**assumes**  $nz$ -def: non-zero-positions = filter ( $\lambda i. A \text{ \#\# } (i,0) \neq 0$ ) [ $1..<dim$ -row  $A$ ]  
**and**  $A'$ -def:  $A' = (if\ A \text{ \#\# } (0, 0) \neq 0\ then\ A\ else\ let\ i = non-zero-positions !$   
 $0\ in\ swaprows\ 0\ i\ A)$   
**and**  $m0$ :  $0 < m$  **and**  $n0$ :  $0 < n$  **and**  $D0$ :  $D \neq 0$  **and**  $mn$ :  $m \geq n$   
**shows** make-first-column-positive  $A' \text{ \#\# } (0, 0) \neq 0$   
 $\langle proof \rangle$

**context** proper-mod-operation

**begin**

**lemma** reduce-below-0-case-m-make-first-column-positive:

**assumes**  $A'$ :  $A' \in carrier\ mat\ m\ n$  **and**  $m0$ :  $0 < m$  **and**  $n0$ :  $0 < n$   
**and**  $A$ -def:  $A = A' @_r (D \cdot_m (1_m\ n))$   
**and**  $mn$ :  $m \geq n$   
**assumes**  $i$ - $mn$ :  $i < m+n$  **and**  $d$ - $xs$ : distinct  $xs$  **and**  $xs$ :  $\forall x \in set\ xs. x < m \wedge 0 < x$   
**and**  $ia$ :  $i \neq 0$   
**and**  $A''$ -def:  $A'' = (if\ A \text{ \#\# } (0, 0) \neq 0\ then\ A\ else\ let\ i = non-zero-positions !$   
 $0\ in\ swaprows\ 0\ i\ A)$   
**and**  $D0$ :  $D > 0$   
**and**  $nz$ -def: non-zero-positions = filter ( $\lambda i. A \text{ \#\# } (i,0) \neq 0$ ) [ $1..<dim$ -row  $A$ ]  
**shows** reduce-below 0 non-zero-positions  $D$  (make-first-column-positive  $A''$ )  $\text{ \#\# } (i,0) = 0$   
 $\langle proof \rangle$

**lemma** reduce-below-abs-0-case-m-make-first-column-positive:

**assumes**  $A'$ :  $A' \in carrier\ mat\ m\ n$  **and**  $m0$ :  $0 < m$  **and**  $n0$ :  $0 < n$   
**and**  $A$ -def:  $A = A' @_r (D \cdot_m (1_m\ n))$   
**and**  $mn$ :  $m \geq n$   
**assumes**  $i$ - $mn$ :  $i < m+n$  **and**  $d$ - $xs$ : distinct  $xs$  **and**  $xs$ :  $\forall x \in set\ xs. x < m \wedge 0 < x$   
**and**  $ia$ :  $i \neq 0$   
**and**  $A''$ -def:  $A'' = (if\ A \text{ \#\# } (0, 0) \neq 0\ then\ A\ else\ let\ i = non-zero-positions !$   
 $0\ in\ swaprows\ 0\ i\ A)$   
**and**  $D0$ :  $D > 0$   
**and**  $nz$ -def: non-zero-positions = filter ( $\lambda i. A \text{ \#\# } (i,0) \neq 0$ ) [ $1..<dim$ -row  $A$ ]  
**shows** reduce-below-abs 0 non-zero-positions  $D$  (make-first-column-positive  $A''$ )  
 $\text{ \#\# } (i,0) = 0$   
 $\langle proof \rangle$

**lemma** FindPreHNF-invertible-mat-2xn:

**assumes**  $A$ :  $A \in carrier\ mat\ m\ n$  **and**  $m < 2$   
**shows**  $\exists P. P \in carrier\ mat\ m\ m \wedge invertible\ mat\ P \wedge FindPreHNF\ abs\ flag\ D$   
 $A = P * A$   
 $\langle proof \rangle$

**lemma** *FindPreHNF-invertible-mat-mx2*:

**assumes** *A-def*:  $A = A'' @_r D \cdot_m 1_m n$

**and** *A''*:  $A'' \in \text{carrier-mat } m \ n$  **and** *n2*:  $n < 2$  **and** *n0*:  $0 < n$  **and** *D-g0*:  $D > 0$

**and** *mn*:  $m \geq n$

**shows**  $\exists P. P \in \text{carrier-mat } (m+n) \ (m+n) \wedge \text{invertible-mat } P \wedge \text{FindPreHNF}$   
*abs-flag*  $D \ A = P * A$

*<proof>*

**corollary** *FindPreHNF-echelon-form-mx0*:

**assumes**  $A \in \text{carrier-mat } m \ 0$

**shows** *echelon-form-JNF* (*FindPreHNF abs-flag*  $D \ A$ )

*<proof>*

**lemma** *FindPreHNF-echelon-form-mx1*:

**assumes** *A-def*:  $A = A'' @_r D \cdot_m 1_m n$

**and** *A''*:  $A'' \in \text{carrier-mat } m \ n$  **and** *n2*:  $n < 2$  **and** *D-g0*:  $D > 0$  **and** *mn*:  $m \geq n$

**shows** *echelon-form-JNF* (*FindPreHNF abs-flag*  $D \ A$ )

*<proof>*

**lemma** *FindPreHNF-works-n-ge2*:

**assumes** *A-def*:  $A = A'' @_r D \cdot_m 1_m n$

**and** *A''*:  $A'' \in \text{carrier-mat } m \ n$  **and** *n2*:  $n \geq 2$  **and** *m-le-n*:  $m \geq n$  **and** *D>0*

**shows**  $\exists P. P \in \text{carrier-mat } (m+n) \ (m+n) \wedge \text{invertible-mat } P \wedge \text{FindPreHNF}$   
*abs-flag*  $D \ A = P * A \wedge \text{echelon-form-JNF}$  (*FindPreHNF abs-flag*  $D \ A$ )

*<proof>*

**lemma**

**assumes** *A-def*:  $A = A'' @_r D \cdot_m 1_m n$

**and** *A''*:  $A'' \in \text{carrier-mat } m \ n$  **and** *n2*:  $n \geq 2$  **and** *m-le-n*:  $m \geq n$  **and** *D>0*

**shows** *FindPreHNF-invertible-mat-n-ge2*:  $\exists P. P \in \text{carrier-mat } (m+n) \ (m+n) \wedge$   
*invertible-mat*  $P \wedge \text{FindPreHNF abs-flag}$   $D \ A = P * A$

**and** *FindPreHNF-echelon-form-n-ge2*: *echelon-form-JNF* (*FindPreHNF abs-flag*  $D \ A$ )

*<proof>*

**lemma** *FindPreHNF-invertible-mat*:

**assumes** *A-def*:  $A = A'' @_r D \cdot_m 1_m n$

**and** *A''*:  $A'' \in \text{carrier-mat } m \ n$  **and** *n0*:  $0 < n$  **and** *mn*:  $m \geq n$  **and** *D*:  $D > 0$

**shows**  $\exists P. P \in \text{carrier-mat } (m+n) \ (m+n) \wedge \text{invertible-mat } P \wedge \text{FindPreHNF}$   
*abs-flag*  $D \ A = P * A$

*<proof>*

**lemma** *FindPreHNF-echelon-form*:

```

assumes A-def:  $A = A'' @_r D \cdot_m 1_m n$ 
and A'':  $A'' \in \text{carrier-mat } m \ n$  and mn:  $m \geq n$  and D:  $D > 0$ 
shows echelon-form-JNF (FindPreHNF abs-flag D A)
<proof>
end

```

We connect the algorithm developed in the Hermite AFP entry with ours. This would permit to reuse many existing results and prove easily the soundness.

```

thm Hermite.Hermite-reduce-above.simps
thm Hermite.Hermite-of-row-i-def
thm Hermite.Hermite-of-upt-row-i-def
thm Hermite.Hermite-of-def

```

```

thm Hermite-reduce-above.simps
thm Hermite-of-row-i-def
thm Hermite-of-list-of-rows.simps
thm mod-operation.Hermite-mod-det-def

```

```

thm Hermite.Hermite-reduce-above.simps Hermite-reduce-above.simps

```

```

context includes lifting-syntax
begin

```

```

definition res-int =  $(\lambda b \ n::\text{int}. \ n \ \text{mod} \ b)$ 

```

```

lemma res-function-res-int:
  res-function res-int
  <proof>

```

```

lemma HMA-Hermite-reduce-above[transfer-rule]:
  assumes  $n < \text{CARD}('m)$ 
  shows  $((\text{Mod-Type-Connect.HMA-M} :: - \Rightarrow \text{int} \wedge 'n :: \text{mod-type} \wedge 'm :: \text{mod-type}$ 
 $\Rightarrow -)$ 
     $\implies (\text{Mod-Type-Connect.HMA-I}) \implies (\text{Mod-Type-Connect.HMA-I}) \implies$ 
 $(\text{Mod-Type-Connect.HMA-M}))$ 
     $(\lambda A \ i \ j. \ \text{Hermite-reduce-above } A \ n \ i \ j)$ 
     $(\lambda A \ i \ j. \ \text{Hermite.Hermite-reduce-above } A \ n \ i \ j \ \text{res-int})$ 
  <proof>

```

```

corollary HMA-Hermite-reduce-above':
  assumes  $n < \text{CARD}('m)$ 
  and  $\text{Mod-Type-Connect.HMA-M } A \ (A':: \text{int} \wedge 'n :: \text{mod-type} \wedge 'm :: \text{mod-type})$ 
  and  $\text{Mod-Type-Connect.HMA-I } i \ i'$  and  $\text{Mod-Type-Connect.HMA-I } j \ j'$ 
  shows  $\text{Mod-Type-Connect.HMA-M } (\text{Hermite-reduce-above } A \ n \ i \ j) \ (\text{Hermite.Hermite-reduce-above}$ 
 $A' \ n \ i' \ j' \ \text{res-int})$ 

```

*<proof>*

**lemma** *HMA-Hermite-of-row-i[transfer-rule]*:

**assumes** *upt-A: upper-triangular' A*

**and** *AA': Mod-Type-Connect.HMA-M A (A':: int ^'n :: mod-type ^'m :: mod-type)*

**and** *ii': Mod-Type-Connect.HMA-I i i'*

**shows** *Mod-Type-Connect.HMA-M (Hermite-of-row-i A i)*

*(Hermite.Hermite-of-row-i ass-function-euclidean res-int A' i')*

*<proof>*

**lemma** *Hermite-of-list-of-rows-append*:

*Hermite-of-list-of-rows A (xs @ [x]) = Hermite-of-row-i (Hermite-of-list-of-rows A xs) x*

*<proof>*

**lemma** *Hermite-reduce-above[simp]*: *Hermite-reduce-above A n i j ∈ carrier-mat (dim-row A) (dim-col A)*

*<proof>*

**lemma** *Hermite-of-row-i*: *Hermite-of-row-i A i ∈ carrier-mat (dim-row A) (dim-col A)*

*<proof>*

**end**

We now move more lemmas from HOL Analysis (with mod-type restrictions) to the JNF matrix representation.

**context**

**begin**

**private lemma** *echelon-form-Hermite-of-row-mod-type*:

**fixes** *A::int mat*

**assumes** *A ∈ carrier-mat CARD('m::mod-type) CARD('n::mod-type)*

**assumes** *eA: echelon-form-JNF A*

**and** *i: i < CARD('m)*

**shows** *echelon-form-JNF (Hermite-of-row-i A i)*

*<proof>* **lemma** *echelon-form-Hermite-of-row-nontriv-mod-ring*:

**fixes** *A::int mat*

**assumes** *A ∈ carrier-mat CARD('m::nontriv mod-ring) CARD('n::nontriv mod-ring)*

**assumes** *eA: echelon-form-JNF A*

**and** *i < CARD('m)*

**shows** *echelon-form-JNF (Hermite-of-row-i A i)*

*<proof>*

**lemmas** *echelon-form-Hermite-of-row-nontriv-mod-ring-internalized* =  
*echelon-form-Hermite-of-row-nontriv-mod-ring[unfolded CARD-mod-ring,*  
*internalize-sort 'm::nontriv, internalize-sort 'b::nontriv]*

**context**

**fixes** *m::nat and n::nat*  
**assumes** *local-typedef1:  $\exists (Rep :: ('b \Rightarrow int))$  Abs. type-definition Rep Abs  $\{0..<m$   
 $:: int\}$*   
**assumes** *local-typedef2:  $\exists (Rep :: ('c \Rightarrow int))$  Abs. type-definition Rep Abs  $\{0..<n$   
 $:: int\}$*   
**and** *m:  $m > 1$*   
**and** *n:  $n > 1$*   
**begin**

**lemma** *echelon-form-Hermite-of-row-nontriv-mod-ring-aux:*

**fixes** *A::int mat*  
**assumes** *A  $\in$  carrier-mat m n*  
**assumes** *eA: echelon-form-JNF A*  
**and** *i < m*  
**shows** *echelon-form-JNF (Hermite-of-row-i A i)*  
*<proof>*

**end**

**context**

**begin**

**private lemma** *echelon-form-Hermite-of-row-i-cancelled-first:*

$\exists Rep$  Abs. type-definition Rep Abs  $\{0..<int\ n\} \Longrightarrow 1 < m \Longrightarrow 1 < n$   
 $\Longrightarrow A \in$  carrier-mat m n  $\Longrightarrow$  echelon-form-JNF A  $\Longrightarrow i < m$   
 $\Longrightarrow$  echelon-form-JNF (HNF-Mod-Det-Algorithm.Hermite-of-row-i A i)  
*<proof>* **lemma** *echelon-form-Hermite-of-row-i-cancelled-both:*  
 $1 < m \Longrightarrow 1 < n \Longrightarrow A \in$  carrier-mat m n  $\Longrightarrow$  echelon-form-JNF A  $\Longrightarrow i < m$   
 $\Longrightarrow$  echelon-form-JNF (HNF-Mod-Det-Algorithm.Hermite-of-row-i A i)  
*<proof>*

**lemma** *echelon-form-JNF-Hermite-of-row-i':*

**fixes** *A::int mat*  
**assumes** *A  $\in$  carrier-mat m n*  
**assumes** *eA: echelon-form-JNF A*  
**and** *i < m*  
**and** *1 < m and 1 < n*  
**shows** *echelon-form-JNF (Hermite-of-row-i A i)*  
*<proof>*

**corollary** *echelon-form-JNF-Hermite-of-row-i*:

**fixes**  $A::\text{int mat}$

**assumes**  $eA: \text{echelon-form-JNF } A$

**and**  $i: i < \text{dim-row } A$

**shows**  $\text{echelon-form-JNF } (\text{Hermite-of-row-}i \ A \ i)$

*<proof>*

**lemma** *Hermite-of-list-of-rows*:

$(\text{Hermite-of-list-of-rows } A \ xs) \in \text{carrier-mat } (\text{dim-row } A) \ (\text{dim-col } A)$

*<proof>*

**lemma** *echelon-form-JNF-Hermite-of-list-of-rows*:

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

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

**and**  $\text{echelon-form-JNF } A$

**shows**  $\text{echelon-form-JNF } (\text{Hermite-of-list-of-rows } A \ xs)$

*<proof>*

**lemma** *HMA-Hermite-of-upt-row-i[transfer-rule]*:

**assumes**  $xs = [0..<i]$

**and**  $\forall x \in \text{set } xs. x < \text{CARD } ('m)$

**assumes**  $\text{Mod-Type-Connect.HMA-M } A \ (A':: \text{int } ^n :: \text{mod-type } ^m :: \text{mod-type})$

**and**  $\text{echelon-form-JNF } A$

**shows**  $\text{Mod-Type-Connect.HMA-M } (\text{Hermite-of-list-of-rows } A \ xs)$

$(\text{Hermite.Hermite-of-upt-row-}i \ A' \ i \ \text{ass-function-euclidean-res-int})$

*<proof>*

**lemma** *Hermite-Hermite-of-upt-row-i*:

**assumes**  $a: \text{ass-function } \text{ass}$

**and**  $r: \text{res-function } \text{res}$

**and**  $eA: \text{echelon-form } A$

**shows**  $\text{Hermite } (\text{range } \text{ass}) \ (\lambda c. \text{range } (\text{res } c)) \ (\text{Hermite-of-upt-row-}i \ A \ (\text{nrows } A) \ \text{ass } \text{res})$

*<proof>*

**lemma** *Hermite-of-row-i-0*:

$\text{Hermite-of-row-}i \ A \ 0 = A \ \vee \ \text{Hermite-of-row-}i \ A \ 0 = \text{multrow } 0 \ (- \ 1) \ A$

*<proof>*

**lemma** *Hermite-JNF-intro:*

**assumes**

*Complete-set-non-associates associates (Complete-set-residues res) echelon-form-JNF*  
 $A$

$(\forall i < \dim\text{-row } A. \neg \text{is-zero-row-JNF } i \ A \longrightarrow A \ \$\$ (i, \text{LEAST } n. A \ \$\$ (i, n) \neq 0)$   
 $\in \text{associates})$

$(\forall i < \dim\text{-row } A. \neg \text{is-zero-row-JNF } i \ A \longrightarrow (\forall j. j < i \longrightarrow A \ \$\$ (j, (\text{LEAST } n. A \ \$\$ (i, n) \neq 0)))$

$\in \text{res } (A \ \$\$ (i, (\text{LEAST } n. A \ \$\$ (i, n) \neq 0))))$

**shows** *Hermite-JNF associates res A*

*<proof>*

**lemma** *least-multrow:*

**assumes**  $A \in \text{carrier-mat } m \ n$  **and**  $i < m$  **and**  $eA: \text{echelon-form-JNF } A$

**assumes**  $ia: ia < \dim\text{-row } A$  **and**  $\text{nz-ia-mr}A: \neg \text{is-zero-row-JNF } ia$  (*multrow*  $i$   
 $(- 1) \ A$ )

**shows**  $(\text{LEAST } n. \text{multrow } i \ (- 1) \ A \ \$\$ (ia, n) \neq 0) = (\text{LEAST } n. A \ \$\$ (ia,$   
 $n) \neq 0)$

*<proof>*

**lemma** *Hermite-Hermite-of-row-i:*

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

**shows** *Hermite-JNF (range ass-function-euclidean) ( $\lambda c. \text{range } (\text{res-int } c)$ ) (Hermite-of-row- $i$   
 $A \ 0$ )*

*<proof>*

**lemma** *Hermite-of-row-i-0-eq-0:*

**assumes**  $A: A \in \text{carrier-mat } m \ n$  **and**  $i: i > 0$  **and**  $eA: \text{echelon-form-JNF } A$  **and**  
 $im: i < m$

**and**  $n0: 0 < n$

**shows** *Hermite-of-row-i A 0*  $\ \$\$ (i, 0) = 0$

*<proof>*

**lemma** *Hermite-Hermite-of-row-i-mx1:*

**assumes**  $A: A \in \text{carrier-mat } m \ 1$  **and**  $eA: \text{echelon-form-JNF } A$

**shows** *Hermite-JNF (range ass-function-euclidean) ( $\lambda c. \text{range } (\text{res-int } c)$ ) (Hermite-of-row- $i$   
 $A \ 0$ )*

*<proof>*

**lemma** *Hermite-of-list-of-rows-1xn:*

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

**and**  $eA: \text{echelon-form-JNF } A$

**and**  $x: \forall x \in \text{set } xs. x < 1$  **and**  $xs: xs \neq []$

**shows** *Hermite-JNF (range ass-function-euclidean)*

$(\lambda c. \text{range } (\text{res-int } c))$  (*Hermite-of-list-of-rows A xs*)

*<proof>*

**lemma** *Hermite-of-row-i-id-mx1*:

**assumes**  $H'$ : *Hermite-JNF* (*range ass-function-euclidean*) ( $\lambda c.$  *range* (*res-int*  $c$ ))  
 $A$

**and**  $x$ :  $x < \text{dim-row } A$  **and**  $A$ :  $A \in \text{carrier-mat } m \ 1$

**shows** *Hermite-of-row-i*  $A \ x = A$

*<proof>*

**lemma** *Hermite-of-row-i-id-mx1'*:

**assumes**  $eA$ : *echelon-form-JNF*  $A$

**and**  $x$ :  $x < \text{dim-row } A$  **and**  $A$ :  $A \in \text{carrier-mat } m \ 1$

**shows** *Hermite-of-row-i*  $A \ x = A \vee$  *Hermite-of-row-i*  $A \ x = \text{multrow } 0 \ (- \ 1) \ A$

*<proof>*

**lemma** *Hermite-of-list-of-rows-mx1*:

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

**and**  $eA$ : *echelon-form-JNF*  $A$

**and**  $x$ :  $\forall x \in \text{set } xs. \ x < m$  **and**  $xs$ :  $xs = [0..<i]$  **and**  $i$ :  $i > 0$

**shows** *Hermite-JNF* (*range ass-function-euclidean*)

( $\lambda c.$  *range* (*res-int*  $c$ )) (*Hermite-of-list-of-rows*  $A \ xs$ )

*<proof>*

**lemma** *invertible-Hermite-of-list-of-rows-1xn*:

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

**shows**  $\exists P. \ P \in \text{carrier-mat } 1 \ 1 \wedge \text{invertible-mat } P \wedge \text{Hermite-of-list-of-rows } A$   
 $[0..<1] = P * A$

*<proof>*

**lemma** *invertible-Hermite-of-list-of-rows-mx1'*:

**assumes**  $A$ :  $A \in \text{carrier-mat } m \ 1$  **and**  $eA$ : *echelon-form-JNF*  $A$

**and**  $xs-i$ :  $xs = [0..<i]$  **and**  $xs-m$ :  $\forall x \in \text{set } xs. \ x < m$  **and**  $i$ :  $i > 0$

**shows**  $\exists P. \ P \in \text{carrier-mat } m \ m \wedge \text{invertible-mat } P \wedge \text{Hermite-of-list-of-rows}$   
 $A \ xs = P * A$

*<proof>*

**corollary** *invertible-Hermite-of-list-of-rows-mx1*:

**assumes**  $A \in \text{carrier-mat } m \ 1$  **and**  $eA$ : *echelon-form-JNF*  $A$

**shows**  $\exists P. \ P \in \text{carrier-mat } m \ m \wedge \text{invertible-mat } P \wedge \text{Hermite-of-list-of-rows}$   
 $A \ [0..<m] = P * A$

*<proof>*

**lemma** *Hermite-of-list-of-rows-mx0*:  
**assumes**  $A: A \in \text{carrier-mat } m \ 0$   
**and**  $xs: xs = [0..<i]$  **and**  $x: \forall x \in \text{set } xs. x < m$   
**shows** *Hermite-of-list-of-rows*  $A \ xs = A$   
 $\langle \text{proof} \rangle$

Again, we move more lemmas from HOL Analysis (with mod-type restrictions) to the JNF matrix representation.

**context**  
**begin**

**private lemma** *Hermite-Hermite-of-list-of-rows-mod-type*:  
**fixes**  $A::\text{int mat}$   
**assumes**  $A \in \text{carrier-mat } \text{CARD}('m::\text{mod-type}) \ \text{CARD}('n::\text{mod-type})$   
**assumes**  $eA: \text{echelon-form-JNF } A$   
**shows** *Hermite-JNF* (*range ass-function-euclidean*)  
 $(\lambda c. \text{range } (\text{res-int } c)) (\text{Hermite-of-list-of-rows } A \ [0..<\text{CARD}('m)])$   
 $\langle \text{proof} \rangle$  **lemma** *invertible-Hermite-of-list-of-rows-mod-type*:  
**fixes**  $A::\text{int mat}$   
**assumes**  $A \in \text{carrier-mat } \text{CARD}('m::\text{mod-type}) \ \text{CARD}('n::\text{mod-type})$   
**assumes**  $eA: \text{echelon-form-JNF } A$   
**shows**  $\exists P. P \in \text{carrier-mat } \text{CARD}('m) \ \text{CARD}('m) \wedge$   
 $\text{invertible-mat } P \wedge \text{Hermite-of-list-of-rows } A \ [0..<\text{CARD}('m)] = P * A$   
 $\langle \text{proof} \rangle$  **lemma** *Hermite-Hermite-of-list-of-rows-nontriv-mod-ring*:  
**fixes**  $A::\text{int mat}$   
**assumes**  $A \in \text{carrier-mat } \text{CARD}('m::\text{nontriv mod-ring}) \ \text{CARD}('n::\text{nontriv mod-ring})$   
**assumes**  $eA: \text{echelon-form-JNF } A$   
**shows** *Hermite-JNF* (*range ass-function-euclidean*)  
 $(\lambda c. \text{range } (\text{res-int } c)) (\text{Hermite-of-list-of-rows } A \ [0..<\text{CARD}('m)])$   
 $\langle \text{proof} \rangle$  **lemma** *invertible-Hermite-of-list-of-rows-nontriv-mod-ring*:  
**fixes**  $A::\text{int mat}$   
**assumes**  $A \in \text{carrier-mat } \text{CARD}('m::\text{nontriv mod-ring}) \ \text{CARD}('n::\text{nontriv mod-ring})$   
**assumes**  $eA: \text{echelon-form-JNF } A$   
**shows**  $\exists P. P \in \text{carrier-mat } \text{CARD}('m) \ \text{CARD}('m) \wedge$   
 $\text{invertible-mat } P \wedge \text{Hermite-of-list-of-rows } A \ [0..<\text{CARD}('m)] = P * A$   
 $\langle \text{proof} \rangle$

**lemmas** *Hermite-Hermite-of-list-of-rows-nontriv-mod-ring-internalized* =  
*Hermite-Hermite-of-list-of-rows-nontriv-mod-ring*[*unfolded* *CARD-mod-ring*,  
*internalize-sort*  $'m::\text{nontriv}$ , *internalize-sort*  $'b::\text{nontriv}$ ]

**lemmas** *invertible-Hermite-of-list-of-rows-nontriv-mod-ring-internalized* =  
*invertible-Hermite-of-list-of-rows-nontriv-mod-ring*[*unfolded* *CARD-mod-ring*,  
*internalize-sort*  $'m::\text{nontriv}$ , *internalize-sort*  $'b::\text{nontriv}$ ]

**context**  
**fixes**  $m::nat$  **and**  $n::nat$   
**assumes**  $local\text{-}typedef1: \exists (Rep :: ('b \Rightarrow int)) Abs. type\text{-}definition\ Rep\ Abs\ \{0..<m$   
 $:: int\}$   
**assumes**  $local\text{-}typedef2: \exists (Rep :: ('c \Rightarrow int)) Abs. type\text{-}definition\ Rep\ Abs\ \{0..<n$   
 $:: int\}$   
**and**  $m: m>1$   
**and**  $n: n>1$   
**begin**

**lemma** *Hermite-Hermite-of-list-of-rows-nontriv-mod-ring-aux:*  
**fixes**  $A::int\ mat$   
**assumes**  $A \in carrier\text{-}mat\ m\ n$   
**assumes**  $eA: echelon\text{-}form\text{-}JNF\ A$   
**shows** *Hermite-JNF (range ass-function-euclidean)*  
 $(\lambda c. range\ (res\text{-}int\ c))\ (Hermite\text{-}of\text{-}list\text{-}of\text{-}rows\ A\ [0..<m])$   
 $\langle proof \rangle$

**lemma** *invertible-Hermite-of-list-of-rows-nontriv-mod-ring-aux:*  
**fixes**  $A::int\ mat$   
**assumes**  $A \in carrier\text{-}mat\ m\ n$   
**assumes**  $eA: echelon\text{-}form\text{-}JNF\ A$   
**shows**  $\exists P. P \in carrier\text{-}mat\ m\ m \wedge invertible\text{-}mat\ P \wedge Hermite\text{-}of\text{-}list\text{-}of\text{-}rows$   
 $A\ [0..<m] = P * A$   
 $\langle proof \rangle$   
**end**

**context**  
**begin**

**private lemma** *invertible-Hermite-of-list-of-rows-cancelled-first:*  
 $\exists Rep\ Abs. type\text{-}definition\ Rep\ Abs\ \{0..<int\ n\}$   
 $\implies 1 < m \implies 1 < n \implies A \in carrier\text{-}mat\ m\ n \implies echelon\text{-}form\text{-}JNF\ A$   
 $\implies \exists P. P \in carrier\text{-}mat\ m\ m \wedge invertible\text{-}mat\ P \wedge Hermite\text{-}of\text{-}list\text{-}of\text{-}rows\ A$   
 $[0..<m] = P * A$   
 $\langle proof \rangle$  **lemma** *invertible-Hermite-of-list-of-rows-cancelled-both:*  
 $1 < m \implies 1 < n \implies A \in carrier\text{-}mat\ m\ n \implies echelon\text{-}form\text{-}JNF\ A$   
 $\implies \exists P. P \in carrier\text{-}mat\ m\ m \wedge invertible\text{-}mat\ P \wedge Hermite\text{-}of\text{-}list\text{-}of\text{-}rows\ A$   
 $[0..<m] = P * A$   
 $\langle proof \rangle$  **lemma** *Hermite-Hermite-of-list-of-rows-cancelled-first:*  
 $\exists Rep\ Abs. type\text{-}definition\ Rep\ Abs\ \{0..<int\ n\} \implies$   
 $1 < m \implies$   
 $1 < n \implies$   
 $A \in carrier\text{-}mat\ m\ n \implies$

```

    echelon-form-JNF A
  ⇒ Hermite-JNF (range ass-function-euclidean) (λc. range (res-int c)) (Hermite-of-list-of-rows
A [0.. $m$ ])
  ⟨proof⟩ lemma Hermite-Hermite-of-list-of-rows-cancelled-both:
1 < m ⇒
  1 < n ⇒
  A ∈ carrier-mat m n ⇒
  echelon-form-JNF A
  ⇒ Hermite-JNF (range ass-function-euclidean) (λc. range (res-int c)) (Hermite-of-list-of-rows
A [0.. $m$ ])
  ⟨proof⟩

```

```

lemma Hermite-Hermite-of-list-of-rows':
  fixes A::int mat
  assumes A ∈ carrier-mat m n
    and echelon-form-JNF A
    and 1 < m and 1 < n
  shows Hermite-JNF (range ass-function-euclidean)
    (λc. range (res-int c)) (Hermite-of-list-of-rows A [0.. $m$ ])
  ⟨proof⟩

```

```

corollary Hermite-Hermite-of-list-of-rows:
  fixes A::int mat
  assumes A: A ∈ carrier-mat m n
    and eA: echelon-form-JNF A
  shows Hermite-JNF (range ass-function-euclidean)
    (λc. range (res-int c)) (Hermite-of-list-of-rows A [0.. $m$ ])
  ⟨proof⟩

```

```

lemma invertible-Hermite-of-list-of-rows:
  assumes A: A ∈ carrier-mat m n
    and eA: echelon-form-JNF A
  shows ∃ P. P ∈ carrier-mat m m ∧ invertible-mat P ∧ Hermite-of-list-of-rows A
    [0.. $m$ ] = P * A
  ⟨proof⟩
end
end
end
end

```

Now we have all the required stuff to prove the soundness of the algorithm.

```

context proper-mod-operation
begin

```

**lemma** *Hermite-mod-det-mx0*:  
**assumes**  $A \in \text{carrier-mat } m \ 0$   
**shows** *Hermite-mod-det abs-flag*  $A = A$   
 $\langle \text{proof} \rangle$

**lemma** *Hermite-JNF-mx0*:  
**assumes**  $A: A \in \text{carrier-mat } m \ 0$   
**shows** *Hermite-JNF (range ass-function-euclidean)*  $(\lambda c. \text{range } (\text{res-int } c)) \ A$   
 $\langle \text{proof} \rangle$

**lemma** *Hermite-mod-det-soundness-mx0*:  
**assumes**  $A: A \in \text{carrier-mat } m \ n$   
**and**  $n0: n=0$   
**shows** *Hermite-JNF (range ass-function-euclidean)*  $(\lambda c. \text{range } (\text{res-int } c)) \ (\text{Hermite-mod-det abs-flag } A)$   
**and**  $(\exists P. \text{invertible-mat } P \wedge P \in \text{carrier-mat } m \ m \wedge (\text{Hermite-mod-det abs-flag } A) = P * A)$   
 $\langle \text{proof} \rangle$

**lemma** *Hermite-mod-det-soundness-mxn*:  
**assumes**  $mn: m = n$   
**and**  $A: A \in \text{carrier-mat } m \ n$   
**and**  $n0: 0 < n$   
**and** *inv-RAT-A: invertible-mat (map-mat rat-of-int A)*  
**shows** *Hermite-JNF (range ass-function-euclidean)*  $(\lambda c. \text{range } (\text{res-int } c)) \ (\text{Hermite-mod-det abs-flag } A)$   
**and**  $(\exists P. \text{invertible-mat } P \wedge P \in \text{carrier-mat } m \ m \wedge (\text{Hermite-mod-det abs-flag } A) = P * A)$   
 $\langle \text{proof} \rangle$

**lemma** *Hermite-mod-det-soundness*:  
**assumes**  $mn: m = n$   
**and** *A-def: A ∈ carrier-mat m n*  
**and** *i: invertible-mat (map-mat rat-of-int A)*  
**shows** *Hermite-JNF (range ass-function-euclidean)*  $(\lambda c. \text{range } (\text{res-int } c)) \ (\text{Hermite-mod-det abs-flag } A)$   
**and**  $(\exists P. \text{invertible-mat } P \wedge P \in \text{carrier-mat } m \ m \wedge (\text{Hermite-mod-det abs-flag } A) = P * A)$   
 $\langle \text{proof} \rangle$

We can even move the whole echelon form algorithm *echelon-form-of* from HOL Analysis to JNF and then we can combine it with *Hermite-of-list-of-rows* to have another HNF algorithm which is not efficient, but valid for arbitrary matrices.

**lemma** *reduce-D0*:  
 $\text{reduce } a \ b \ 0 \ A = (\text{let } Aaj = A\$\$(a,0); \text{Abj} = A \ \$\$ (b,0)$

*in*  
*if*  $A_{aj} = 0$  *then*  $A$  *else*  
*case* *euclid-ext2*  $A_{aj}$   $A_{bj}$  *of*  $(p, q, u, v, d) \Rightarrow$   
 $Matrix.mat$   $(dim-row\ A)$   $(dim-col\ A)$   
 $(\lambda(i, k). \text{if } i = a \text{ then } (p * A_{ak} + q * A_{bk}))$   
 $\text{else if } i = b \text{ then } u * A_{ak} + v * A_{bk}$   
 $\text{else } A_{ik}$   
 $)$   
 $)$  **(is ?lhs = ?rhs)**  
 $\langle proof \rangle$

**lemma** *bezout-matrix-JNF-mult-eq'*:  
**assumes**  $A'$ :  $A' \in carrier-mat\ m\ n$  **and**  $a$ :  $a < m$  **and**  $b$ :  $b < m$  **and**  $ab$ :  $a \neq b$   
**and**  $A-def$ :  $A = A' @_r B$  **and**  $B$ :  $B \in carrier-mat\ t\ n$   
**assumes**  $pquvd$ :  $(p, q, u, v, d) = euclid-ext2\ (A_{aj})\ (A_{bj})$   
**shows**  $Matrix.mat$   $(dim-row\ A)$   $(dim-col\ A)$   
 $(\lambda(i, k). \text{if } i = a \text{ then } (p * A_{ak} + q * A_{bk}))$   
 $\text{else if } i = b \text{ then } u * A_{ak} + v * A_{bk}$   
 $\text{else } A_{ik}$   
 $) = (bezout-matrix-JNF\ A\ a\ b\ j\ euclid-ext2) * A$  **(is ?A = ?BM \* A)**  
 $\langle proof \rangle$

**lemma** *bezout-matrix-JNF-mult-eq2*:  
**assumes**  $A$ :  $A \in carrier-mat\ m\ n$  **and**  $a$ :  $a < m$  **and**  $b$ :  $b < m$  **and**  $ab$ :  $a \neq b$   
**assumes**  $pquvd$ :  $(p, q, u, v, d) = euclid-ext2\ (A_{aj})\ (A_{bj})$   
**shows**  $Matrix.mat$   $(dim-row\ A)$   $(dim-col\ A)$   
 $(\lambda(i, k). \text{if } i = a \text{ then } (p * A_{ak} + q * A_{bk}))$   
 $\text{else if } i = b \text{ then } u * A_{ak} + v * A_{bk}$   
 $\text{else } A_{ik}$   
 $) = (bezout-matrix-JNF\ A\ a\ b\ j\ euclid-ext2) * A$  **(is ?A = ?BM \* A)**  
 $\langle proof \rangle$

**lemma** *reduce-invertible-mat-D0-BM*:  
**assumes**  $A$ :  $A \in carrier-mat\ m\ n$   
**and**  $a$ :  $a < m$   
**and**  $b$ :  $b < m$   
**and**  $ab$ :  $a \neq b$   
**and**  $Aa0$ :  $A_{a0} \neq 0$   
**shows**  $reduce\ a\ b\ 0\ A = (bezout-matrix-JNF\ A\ a\ b\ 0\ euclid-ext2) * A$   
 $\langle proof \rangle$

**lemma** *reduce-invertible-mat-D0*:  
**assumes**  $A$ :  $A \in carrier-mat\ m\ n$

**and**  $a: a < m$   
**and**  $b: b < m$   
**and**  $n0: 0 < n$   
**and**  $ab: a \neq b$   
**and**  $a\text{-less-}b: a < b$   
**shows**  $\exists P. \text{invertible-mat } P \wedge P \in \text{carrier-mat } m \ m \wedge \text{reduce } a \ b \ 0 \ A = P * A$   
 $\langle \text{proof} \rangle$

**lemma** *reduce-below-invertible-mat-D0*:  
**assumes**  $A': A \in \text{carrier-mat } m \ n$  **and**  $a: a < m$  **and**  $j: 0 < n$   
**and** *distinct xs* **and**  $\forall x \in \text{set } xs. x < m \wedge a < x$   
**and**  $D=0$   
**shows**  $(\exists P. \text{invertible-mat } P \wedge P \in \text{carrier-mat } m \ m \wedge \text{reduce-below } a \ xs \ D \ A = P * A)$   
 $\langle \text{proof} \rangle$

**lemma** *reduce-not0'*:  
**assumes**  $A: A \in \text{carrier-mat } m \ n$  **and**  $a: a < m$  **and**  $a\text{-less-}b: a < b$  **and**  $j: 0 < n$   
**and**  $b: b < m$   
**and**  $Aaj: A \ \$\$ \ (a,0) \neq 0$   
**shows**  $\text{reduce } a \ b \ 0 \ A \ \$\$ \ (a, 0) \neq 0$  (**is**  $?\text{reduce-ab } \ \$\$ \ (a,0) \neq -$ )  
 $\langle \text{proof} \rangle$

**lemma** *reduce-below-preserves-D0*:  
**assumes**  $A': A \in \text{carrier-mat } m \ n$  **and**  $a: a < m$  **and**  $j: j < n$   
**and**  $Aaj: A \ \$\$ \ (a,0) \neq 0$   
**assumes**  $i \notin \text{set } xs$  **and** *distinct xs* **and**  $\forall x \in \text{set } xs. x < m \wedge a < x$   
**and**  $i \neq a$  **and**  $i < m$   
**and**  $D=0$   
**shows**  $\text{reduce-below } a \ xs \ D \ A \ \$\$ \ (i,j) = A \ \$\$ \ (i,j)$   
 $\langle \text{proof} \rangle$

**lemma** *reduce-below-0-D0*:  
**assumes**  $A: A \in \text{carrier-mat } m \ n$  **and**  $a: a < m$  **and**  $j: 0 < n$   
**and**  $Aaj: A \ \$\$ \ (a,0) \neq 0$   
**assumes**  $i \in \text{set } xs$  **and** *distinct xs* **and**  $\forall x \in \text{set } xs. x < m \wedge a < x$   
**and**  $D=0$   
**shows**  $\text{reduce-below } a \ xs \ D \ A \ \$\$ \ (i,0) = 0$   
 $\langle \text{proof} \rangle$

**end**

Definition of the echelon form algorithm in JNF

**primrec** *bezout-iterate-JNF*

**where** *bezout-iterate-JNF*  $A\ 0\ i\ j\ \text{bezout} = A$   
 | *bezout-iterate-JNF*  $A\ (\text{Suc } n)\ i\ j\ \text{bezout} =$   
   (*if*  $(\text{Suc } n) \leq i$  *then*  $A$  *else*  
     *bezout-iterate-JNF* (*bezout-matrix-JNF*  $A\ i\ ((\text{Suc } n))\ j\ \text{bezout} * A$ )  $n\ i$   
   *j bezout*)

**definition**

*echelon-form-of-column-k-JNF* *bezout*  $A'\ k =$   
 (*let*  $(A, i) = A'$   
   *in if*  $(i = \text{dim-row } A) \vee (\forall m \in \{i..<\text{dim-row } A\}. A\ \$\$ (m, k) = 0)$  *then*  $(A,$   
 $i)$  *else*  
   *if*  $(\forall m \in \{i+1..<\text{dim-row } A\}. A\ \$\$ (m, k) = 0)$  *then*  $(A, i + 1)$  *else*  
     *let*  $n = (\text{LEAST } n. A\ \$\$ (n, k) \neq 0 \wedge i \leq n);$   
     *interchange-A* = *swaprows*  $i\ n\ A$   
   *in*  
   (*bezout-iterate-JNF* (*interchange-A*)  $(\text{dim-row } A - 1)\ i\ k\ \text{bezout}, i + 1)$ )

**definition** *echelon-form-of-upt-k-JNF*  $A\ k\ \text{bezout} = (\text{fst } (\text{foldl } (\text{echelon-form-of-column-k-JNF } \text{bezout}) (A, 0) [0..<\text{Suc } k]))$

**definition** *echelon-form-of-JNF*  $A\ \text{bezout} = \text{echelon-form-of-upt-k-JNF } A\ (\text{dim-col } A - 1)\ \text{bezout}$

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

**lemma** *HMA-bezout-iterate*[*transfer-rule*]:

**assumes**  $n < \text{CARD } ('m)$   
   **shows**  $((\text{Mod-Type-Connect.HMA-M} :: - \Rightarrow \text{int } ^\wedge n :: \text{mod-type } ^\wedge m :: \text{mod-type}$   
 $\Rightarrow -)$   
    $====> (\text{Mod-Type-Connect.HMA-I}) ====> (\text{Mod-Type-Connect.HMA-I}) ====>$   
 $(=) ====> (\text{Mod-Type-Connect.HMA-M}))$   
 ( $\lambda A\ i\ j\ \text{bezout}. \text{bezout-iterate-JNF } A\ n\ i\ j\ \text{bezout}$ )  
 ( $\lambda A\ i\ j\ \text{bezout}. \text{bezout-iterate } A\ n\ i\ j\ \text{bezout}$ )

*<proof>*

**corollary** *HMA-bezout-iterate'*[*transfer-rule*]:

**fixes**  $A' :: \text{int } ^\wedge n :: \text{mod-type } ^\wedge m :: \text{mod-type}$   
   **assumes**  $n < \text{CARD } ('m)$   
   **and**  $\text{Mod-Type-Connect.HMA-M } A\ A'$   
   **and**  $\text{Mod-Type-Connect.HMA-I } i\ i'$  **and**  $\text{Mod-Type-Connect.HMA-I } j\ j'$   
   **shows**  $\text{Mod-Type-Connect.HMA-M } (\text{bezout-iterate-JNF } A\ n\ i\ j\ \text{bezout})\ (\text{bezout-iterate}$   
 $A'\ n\ i'\ j'\ \text{bezout})$   
 (*proof*)

**lemma** *snd-echelon-form-of-column-k-JNF-le-dim-row*:  
**assumes**  $i < \text{dim-row } A$   
**shows**  $\text{snd } (\text{echelon-form-of-column-k-JNF bezout } (A, i) k) \leq \text{dim-row } A$   
 $\langle \text{proof} \rangle$

**lemma** *HMA-echelon-form-of-column-k[transfer-rule]*:  
**assumes**  $k < \text{CARD}('n)$   
**shows**  $((=) \implies \text{rel-prod } (\text{Mod-Type-Connect.HMA-M} :: - \Rightarrow \text{int } ^\wedge 'n :: \text{mod-type } ^\wedge 'm :: \text{mod-type} \Rightarrow -) (\lambda a b. a=b \wedge a \leq \text{CARD}('m)) \implies (\text{rel-prod } (\text{Mod-Type-Connect.HMA-M}) (\lambda a b. a=b \wedge a \leq \text{CARD}('m)))) (\lambda \text{bezout } A. \text{echelon-form-of-column-k-JNF bezout } A k) (\lambda \text{bezout } A. \text{echelon-form-of-column-k bezout } A k)$   
 $\langle \text{proof} \rangle$

**corollary** *HMA-echelon-form-of-column-k'[transfer-rule]*:  
**assumes**  $k < \text{CARD}('n)$  **and**  $i \leq \text{CARD}('m)$   
**and**  $(\text{Mod-Type-Connect.HMA-M} :: - \Rightarrow \text{int } ^\wedge 'n :: \text{mod-type } ^\wedge 'm :: \text{mod-type} \Rightarrow -) A A'$   
**shows**  $(\text{rel-prod } (\text{Mod-Type-Connect.HMA-M}) (\lambda a b. a=b \wedge a \leq \text{CARD}('m))) (\text{echelon-form-of-column-k-JNF bezout } (A, i) k) (\text{echelon-form-of-column-k bezout } (A', i) k)$   
 $\langle \text{proof} \rangle$

**lemma** *HMA-foldl-echelon-form-of-column-k*:  
**assumes**  $k \leq \text{CARD}('n)$   
**shows**  $((\text{Mod-Type-Connect.HMA-M} :: - \Rightarrow \text{int } ^\wedge 'n :: \text{mod-type } ^\wedge 'm :: \text{mod-type} \Rightarrow -) \implies (=) \implies (\text{rel-prod } (\text{Mod-Type-Connect.HMA-M}) (\lambda a b. a=b \wedge a \leq \text{CARD}('m)))) (\lambda A \text{ bezout}. (\text{foldl } (\text{echelon-form-of-column-k-JNF bezout}) (A, 0) [0..<k])) (\lambda A \text{ bezout}. (\text{foldl } (\text{echelon-form-of-column-k bezout}) (A, 0) [0..<k]))$   
 $\langle \text{proof} \rangle$

**lemma** *HMA-echelon-form-of-upt-k[transfer-rule]*:  
**assumes**  $k < \text{CARD}('n)$   
**shows**  $((\text{Mod-Type-Connect.HMA-M} :: - \Rightarrow \text{int } ^\wedge 'n :: \text{mod-type } ^\wedge 'm :: \text{mod-type} \Rightarrow -) \implies (=) \implies (\text{Mod-Type-Connect.HMA-M})) (\lambda A \text{ bezout}. \text{echelon-form-of-upt-k-JNF } A k \text{ bezout}) (\lambda A \text{ bezout}. \text{echelon-form-of-upt-k } A k \text{ bezout})$   
 $\langle \text{proof} \rangle$

**lemma** *HMA-echelon-form-of*[*transfer-rule*]:  
**shows**  $((\text{Mod-Type-Connect.HMA-M} :: - \Rightarrow \text{int} \hat{\ } 'n :: \text{mod-type} \hat{\ } 'm :: \text{mod-type}$   
 $\Rightarrow -) \text{====> } (=)$   
 $\text{====> } (\text{Mod-Type-Connect.HMA-M}))$   
 $(\lambda A \text{ bezout. echelon-form-of-JNF } A \text{ bezout})$   
 $(\lambda A \text{ bezout. echelon-form-of } A \text{ bezout})$

*<proof>*  
**end**

**context**  
**begin**

**private lemma** *echelon-form-of-euclidean-invertible-mod-type*:

**fixes**  $A :: \text{int mat}$   
**assumes**  $A \in \text{carrier-mat } \text{CARD}('m :: \text{mod-type}) \text{ CARD}('n :: \text{mod-type})$   
**shows**  $\exists P. \text{invertible-mat } P \wedge P \in \text{carrier-mat } (\text{CARD}('m :: \text{mod-type})) (\text{CARD}('n :: \text{mod-type}))$

$\wedge P * A = \text{echelon-form-of-JNF } A \text{ euclid-ext2}$   
 $\wedge \text{echelon-form-JNF } (\text{echelon-form-of-JNF } A \text{ euclid-ext2})$

*<proof>* **lemma** *echelon-form-of-euclidean-invertible-nontriv-mod-ring*:

**fixes**  $A :: \text{int mat}$   
**assumes**  $A \in \text{carrier-mat } \text{CARD}('m :: \text{nontriv mod-ring}) \text{ CARD}('n :: \text{nontriv mod-ring})$   
**shows**  $\exists P. \text{invertible-mat } P \wedge P \in \text{carrier-mat } (\text{CARD}('m)) (\text{CARD}('n))$

$\wedge P * A = \text{echelon-form-of-JNF } A \text{ euclid-ext2}$   
 $\wedge \text{echelon-form-JNF } (\text{echelon-form-of-JNF } A \text{ euclid-ext2})$

*<proof>*

**lemmas** *echelon-form-of-euclidean-invertible-nontriv-mod-ring-internalized* =  
*echelon-form-of-euclidean-invertible-nontriv-mod-ring*[*unfolded CARD-mod-ring*,  
*internalize-sort 'm::nontriv, internalize-sort 'b::nontriv*]

**context**

**fixes**  $m :: \text{nat}$  **and**  $n :: \text{nat}$   
**assumes** *local-typedef1*:  $\exists (\text{Rep} :: ('b \Rightarrow \text{int})) \text{ Abs. type-definition } \text{Rep } \text{Abs } \{0..<m$   
 $:: \text{int}\}$   
**assumes** *local-typedef2*:  $\exists (\text{Rep} :: ('c \Rightarrow \text{int})) \text{ Abs. type-definition } \text{Rep } \text{Abs } \{0..<n$   
 $:: \text{int}\}$   
**and**  $m > 1$   
**and**  $n > 1$

**begin**

**lemma** *echelon-form-of-euclidean-invertible-nontriv-mod-ring-aux*:

**fixes**  $A :: \text{int mat}$   
**assumes**  $A \in \text{carrier-mat } m \ n$   
**shows**  $\exists P. \text{invertible-mat } P \wedge P \in \text{carrier-mat } m \ m$   
 $\wedge P * A = \text{echelon-form-of-JNF } A \text{ euclid-ext2}$

$\wedge$  echelon-form-JNF (echelon-form-of-JNF A euclid-ext2)  
 ⟨proof⟩

**end**

**context**  
**begin**

**private lemma** echelon-form-of-euclidean-invertible-cancelled-first:  
 $\exists Rep\ Abs.\ type\ definition\ Rep\ Abs\ \{0..<int\ n\} \implies 1 < m \implies 1 < n \implies$   
 $A \in carrier\ mat\ m\ n \implies \exists P.\ invertible\ mat\ P \wedge P \in carrier\ mat\ m\ m$   
 $\wedge P * (A::int\ mat) = echelon\ form\ of\ JNF\ A\ euclid\ ext2 \wedge echelon\ form\ of\ JNF$   
 (echelon-form-of-JNF A euclid-ext2)  
 ⟨proof⟩ **lemma** echelon-form-of-euclidean-invertible-cancelled-both:  
 $1 < m \implies 1 < n \implies A \in carrier\ mat\ m\ n \implies \exists P.\ invertible\ mat\ P \wedge P \in$   
 carrier-mat m m  
 $\wedge P * (A::int\ mat) = echelon\ form\ of\ JNF\ A\ euclid\ ext2 \wedge echelon\ form\ of\ JNF$   
 (echelon-form-of-JNF A euclid-ext2)  
 ⟨proof⟩

**lemma** echelon-form-of-euclidean-invertible':  
**fixes** A::int mat  
**assumes** A ∈ carrier-mat m n  
**and** 1 < m **and** 1 < n  
**shows**  $\exists P.\ invertible\ mat\ P \wedge$   
 $P \in carrier\ mat\ m\ m \wedge P * A = echelon\ form\ of\ JNF\ A\ euclid\ ext2$   
 $\wedge echelon\ form\ of\ JNF\ (echelon\ form\ of\ JNF\ A\ euclid\ ext2)$   
 ⟨proof⟩  
**end**  
**end**

**context** mod-operation  
**begin**

**definition** FindPreHNF-rectangular A  
 = (let m = dim-row A; n = dim-col A in  
 if m < 2  $\vee$  n = 0 then A else — No operations are carried out if m = 1  
 if n = 1 then  
 let non-zero-positions = filter ( $\lambda i.\ A\ \$\$ (i,0) \neq 0$ ) [1..<dim-row A] in  
 if non-zero-positions = [] then A  
 else let A' = (if A\$\$ (0,0)  $\neq$  0 then A else let i = non-zero-positions ! 0 in  
 swaprows 0 i A)  
 in reduce-below-impl 0 non-zero-positions 0 A')

*else (echelon-form-of-JNF A euclid-ext2)*

This is the (non-efficient) HNF algorithm obtained from the echelon form and Hermite normal form AFP entries

**definition** *HNF-algorithm-from-HA* A  
 = *Hermite-of-list-of-rows (FindPreHNF-rectangular A) [0.. $\langle$ dim-row A]*

Now we can combine *FindPreHNF-rectangular*, *FindPreHNF* and *Hermite-of-list-of-rows* to get an algorithm to compute the HNF of any matrix (if it is square and invertible, then the HNF is computed reducing entries modulo D)

**definition** *HNF-algorithm abs-flag* A =  
 (let m = dim-row A; n = dim-col A in  
 if m  $\neq$  n then *Hermite-of-list-of-rows (FindPreHNF-rectangular A) [0.. $\langle$ m]*  
 else  
 let D = abs (det-int A) in  
 if D = 0 then *Hermite-of-list-of-rows (FindPreHNF-rectangular A) [0.. $\langle$ m]*  
 else  
 let A' = A @<sub>r</sub> D ·<sub>m</sub> 1<sub>m</sub> n;  
 E = *FindPreHNF abs-flag D A'*;  
 H = *Hermite-of-list-of-rows E [0.. $\langle$ m+n]*  
 in mat-of-rows n (map (Matrix.row H) [0.. $\langle$ m]))

**end**

**declare** *mod-operation.FindPreHNF-rectangular-def*[code]  
**declare** *mod-operation.HNF-algorithm-from-HA-def*[code]  
**declare** *mod-operation.HNF-algorithm-def*[code]

**context** *proper-mod-operation*  
**begin**

**lemma** *FindPreHNF-rectangular-soundness:*

**fixes** A::int mat

**assumes** A: A  $\in$  carrier-mat m n

**shows**  $\exists P. invertible\text{-}mat\ P \wedge P \in carrier\text{-}mat\ m\ m \wedge P * A = FindPreHNF\text{-}rectangular\ A$

$\wedge echelon\text{-}form\text{-}JNF\ (FindPreHNF\text{-}rectangular\ A)$

*<proof>*

**lemma** *HNF-algorithm-from-HA-soundness:*

**assumes** A: A  $\in$  carrier-mat m n

**shows** *Hermite-JNF (range ass-function-euclidean) ( $\lambda c. range (res-int c)$ ) (HNF-algorithm-from-HA A)*

$\wedge (\exists P. P \in carrier\text{-}mat\ m\ m \wedge invertible\text{-}mat\ P \wedge (HNF\text{-}algorithm\text{-}from\text{-}HA\ A) = P * A)$

*<proof>*

Soundness theorem for any matrix

**lemma** *HNF-algorithm-soundness*:  
**assumes**  $A: A \in \text{carrier-mat } m \ n$   
**shows** *Hermite-JNF (range ass-function-euclidean)*  $(\lambda c. \text{range } (\text{res-int } c))$  *(HNF-algorithm abs-flag A)*  
 $\wedge (\exists P. P \in \text{carrier-mat } m \ m \wedge \text{invertible-mat } P \wedge (\text{HNF-algorithm abs-flag } A) = P * A)$   
 $\langle \text{proof} \rangle$   
**end**

New predicate of soundness of a HNF algorithm, without providing explicitly the transformation matrix.

**definition** *is-sound-HNF' algorithm associates res*  
 $= (\forall A. \text{let } H = \text{algorithm } A; m = \text{dim-row } A; n = \text{dim-col } A \text{ in Hermite-JNF associates res } H$   
 $\wedge H \in \text{carrier-mat } m \ n \wedge (\exists P. P \in \text{carrier-mat } m \ m \wedge \text{invertible-mat } P \wedge A = P * H))$

**lemma** *is-sound-HNF-conv*:  
**assumes**  $s: \text{is-sound-HNF}' \text{ algorithm associates res}$   
**shows** *is-sound-HNF*  $(\lambda A. \text{let } H = \text{algorithm } A \text{ in } (\text{SOME } P. P \in \text{carrier-mat } (\text{dim-row } A) \ (\text{dim-row } A) \wedge \text{invertible-mat } P \wedge A = P * H, H)) \text{ associates res}$   
 $\langle \text{proof} \rangle$

**context** *proper-mod-operation*

**begin**

**corollary** *is-sound-HNF'-HNF-algorithm*:

*is-sound-HNF'* *(HNF-algorithm abs-flag)* *(range ass-function-euclidean)*  $(\lambda c. \text{range } (\text{res-int } c))$   
 $\langle \text{proof} \rangle$

**corollary** *is-sound-HNF'-HNF-algorithm-from-HA*:

*is-sound-HNF'* *(HNF-algorithm-from-HA)* *(range ass-function-euclidean)*  $(\lambda c. \text{range } (\text{res-int } c))$   
 $\langle \text{proof} \rangle$

**end**

Some work to make the algorithm executable

**definition** *find-non0'* ::  $\langle \text{nat} \Rightarrow \text{nat} \Rightarrow 'a::\text{comm-ring-1 mat} \Rightarrow \text{nat option} \rangle$  **where**  
 $\langle \text{find-non0}' \ i \ k \ A = \text{find } (\lambda j. A \ \$\$ \ (j, k) \neq 0) \ [i \ ..< \ \text{dim-row } A] \rangle$

**lemma**

**assumes**  $\text{res}: \text{find-non0}' \ i \ k \ A = \text{Some } j$   
**shows**  $\text{find-non0}' : A \ \$\$ \ (j, k) \neq 0 \ i \leq j \ j < \text{dim-row } A$   
**and**  $\text{find-non0}'\text{-w-zero-before}: \forall j' \in \{i..<j\}. A \ \$\$ \ (j', k) = 0$   
 $\langle \text{proof} \rangle$

**lemma** *find-non0'-LEAST*:

**assumes** *res*: *find-non0' i k A = Some j*  
**shows**  $j = (\text{LEAST } n. A \ \$\$ (n,k) \neq 0 \wedge i \leq n)$   
*<proof>*

**lemma** *echelon-form-of-column-k-JNF-code*[*code*]:  
*echelon-form-of-column-k-JNF bezout (A,i) k =*  
*(if (i = dim-row A)  $\vee$  ( $\forall m \in \{i..<dim\text{-row } A\}. A \ \$\$ (m, k) = 0$ ) then (A, i)*  
*else*  
*if ( $\forall m \in \{i+1..<dim\text{-row } A\}. A \ \$\$ (m,k) = 0$ ) then (A, i + 1) else*  
*let n = the (find-non0' i k A);*  
*interchange-A = swaprows i n A*  
*in*  
*(bezout-iterate-JNF (interchange-A) (dim-row A - 1) i k bezout, i + 1))*  
*<proof>*

### 7.3 Instantiation of the HNF-algorithm with modulo-operation

We currently use a Boolean flag to indicate whether standard-mod or symmetric modulo should be used.

**lemma** *sym-mod: proper-mod-operation sym-mod sym-div*  
*<proof>*

**lemma** *standard-mod: proper-mod-operation (mod) (div)*  
*<proof>*

**definition** *HNF-algorithm* :: *bool*  $\Rightarrow$  *int mat*  $\Rightarrow$  *int mat* **where**  
*HNF-algorithm use-sym-mod = (if use-sym-mod*  
*then mod-operation.HNF-algorithm sym-mod False else mod-operation.HNF-algorithm*  
*(mod) True)*

**definition** *HNF-algorithm-from-HA* :: *bool*  $\Rightarrow$  *int mat*  $\Rightarrow$  *int mat* **where**  
*HNF-algorithm-from-HA use-sym-mod = (if use-sym-mod*  
*then mod-operation.HNF-algorithm-from-HA sym-mod else mod-operation.HNF-algorithm-from-HA*  
*(mod))*

**corollary** *is-sound-HNF'-HNF-algorithm*:  
*is-sound-HNF' (HNF-algorithm use-sym-mod) (range ass-function-euclidean)*  
*( $\lambda c. \text{range } (\text{res-int } c)$ )*  
*<proof>*

**corollary** *is-sound-HNF'-HNF-algorithm-from-HA*:  
*is-sound-HNF' (HNF-algorithm-from-HA use-sym-mod) (range ass-function-euclidean)*  
*( $\lambda c. \text{range } (\text{res-int } c)$ )*  
*<proof>*

```

value [code]let A = mat-of-rows-list 4 (
  [[0,3,1,4],
   [7,1,0,0],
   [8,0,19,16],
   [2,0,0,3::int],
   [9,-3,2,5],
   [6,3,2,4]]) in
  show (HNF-algorithm True A)

```

```

value [code]let A = mat-of-rows-list 6 (
  [[0,3,1,4,8,7],
   [7,1,0,0,4,1],
   [8,0,19,16,33,5],
   [2,0,0,3::int,-5,8]]) in
  show (HNF-algorithm False A)

```

```

value [code]let A = mat-of-rows-list 6 (
  [[0,3,1,4,8,7],
   [7,1,0,0,4,1],
   [8,0,19,16,33,5],
   [0,3,1,4,8,7],
   [2,0,0,3::int,-5,8],
   [2,4,6,8,10,12]]) in
  show (Determinant.det A, HNF-algorithm True A)

```

```

value [code]let A = mat-of-rows-list 6 (
  [[0,3,1,4,8,7],
   [7,1,0,0,4,1],
   [8,0,19,16,33,5],
   [5,6,1,2,8,7],
   [2,0,0,3::int,-5,8],
   [2,4,6,8,10,12]]) in
  show (Determinant.det A, HNF-algorithm True A)

```

**end**

## 8 LLL certification via Hermite normal forms

In this file, we define the new certified approach and prove its soundness.

```

theory LLL-Certification-via-HNF
imports
  LLL-Basis-Reduction.LLL-Certification

```

*Jordan-Normal-Form.DL-Rank*  
*HNF-Mod-Det-Soundness*

**begin**

**context** *LLL-with-assms*  
**begin**

**lemma** *m-le-n:  $m \leq n$*   
 $\langle$ *proof* $\rangle$

**end**

This lemma is a generalization of the theorem named *HNF-A-eq-HNF-PA*, using the new uniqueness statement of the HNF. We provide two versions, one assuming the existence and the other one obtained from a sound algorithm.

**lemma** *HNF-A-eq-HNF-PA'-exist:*

**fixes** *A::int mat*

**assumes** *A:  $A \in \text{carrier-mat } n \ n$  and  $\text{inv-A: invertible-mat (map-mat rat-of-int } A)$*

**and** *inv-P: invertible-mat P and  $P: P \in \text{carrier-mat } n \ n$*

**and** *HNF-H1: Hermite-JNF associates res H1*

**and** *H1:  $H1 \in \text{carrier-mat } n \ n$*

**and** *HNF-H2: Hermite-JNF associates res H2*

**and** *H2:  $H2 \in \text{carrier-mat } n \ n$*

**and** *sound-HNF1:  $\exists P1. P1 \in \text{carrier-mat } n \ n \wedge \text{invertible-mat } P1 \wedge (P * A) = P1 * H1$*

**and** *sound-HNF2:  $\exists P2. P2 \in \text{carrier-mat } n \ n \wedge \text{invertible-mat } P2 \wedge A = P2 * H2$*

**shows**  *$H1 = H2$*

$\langle$ *proof* $\rangle$

**corollary** *HNF-A-eq-HNF-PA':*

**fixes** *A::int mat*

**assumes** *A:  $A \in \text{carrier-mat } n \ n$  and  $\text{inv-A: invertible-mat (map-mat rat-of-int } A)$*

**and** *inv-P: invertible-mat P and  $P: P \in \text{carrier-mat } n \ n$*

**and** *sound-HNF: is-sound-HNF HNF associates res*

**and** *P1-H1:  $(P1, H1) = \text{HNF } (P * A)$*

**and** *P2-H2:  $(P2, H2) = \text{HNF } A$*

**shows**  *$H1 = H2$*

$\langle$ *proof* $\rangle$

**context** *LLL-with-assms*  
**begin**

**lemma** *certification-via-eq-HNF2-exist*:  
**assumes** *HNF-H1*: Hermite-JNF associates res *H1*  
**and** *H1*:  $H1 \in \text{carrier-mat } n \ n$   
**and** *HNF-H2*: Hermite-JNF associates res *H2*  
**and** *H2*:  $H2 \in \text{carrier-mat } n \ n$   
**and** *sound-HNF1*:  $\exists P1. P1 \in \text{carrier-mat } n \ n \wedge \text{invertible-mat } P1 \wedge (\text{mat-of-rows } n \ \text{fs-init}) = P1 * H1$   
**and** *sound-HNF2*:  $\exists P2. P2 \in \text{carrier-mat } n \ n \wedge \text{invertible-mat } P2 \wedge (\text{mat-of-rows } n \ \text{gs}) = P2 * H2$   
**and** *gs*:  $\text{set } gs \subseteq \text{carrier-vec } n$   
**and** *l*:  $\text{lattice-of } \text{fs-init} = \text{lattice-of } gs$   
**and** *mn*:  $m = n$  **and** *len-gs*:  $\text{length } gs = n$   
**shows**  $H1 = H2$   
 $\langle \text{proof} \rangle$

**lemma** *certification-via-eq-HNF2*:  
**assumes** *sound-HNF*: *is-sound-HNF* *HNF* associates res  
**and** *P1-H1*:  $(P1, H1) = \text{HNF } (\text{mat-of-rows } n \ \text{fs-init})$   
**and** *P2-H2*:  $(P2, H2) = \text{HNF } (\text{mat-of-rows } n \ \text{gs})$   
**and** *gs*:  $\text{set } gs \subseteq \text{carrier-vec } n$   
**and** *l*:  $\text{lattice-of } \text{fs-init} = \text{lattice-of } gs$   
**and** *mn*:  $m = n$  **and** *len-gs*:  $\text{length } gs = n$   
**shows**  $H1 = H2$   
 $\langle \text{proof} \rangle$

**corollary** *lattice-of-eq-via-HNF*:  
**assumes** *sound-HNF*: *is-sound-HNF* *HNF* associates res  
**and** *P1-H1*:  $(P1, H1) = \text{HNF } (\text{mat-of-rows } n \ \text{fs-init})$   
**and** *P2-H2*:  $(P2, H2) = \text{HNF } (\text{mat-of-rows } n \ \text{gs})$   
**and** *gs*:  $\text{set } gs \subseteq \text{carrier-vec } n$   
**and** *mn*:  $m = n$  **and** *len-gs*:  $\text{length } gs = n$   
**shows**  $(H1 = H2) \longleftrightarrow (\text{lattice-of } \text{fs-init} = \text{lattice-of } gs)$   
 $\langle \text{proof} \rangle$   
**end**

**context**  
**begin**

**interpretation** *vec-module*  $\text{TYPE}(\text{int}) \ n \ \langle \text{proof} \rangle$

**lemma** *lattice-of-eq-via-HNF-paper*:  
**fixes**  $F \ G :: \text{int mat}$  **and**  $\text{HNF} :: \text{int mat} \Rightarrow \text{int mat}$   
**assumes** *sound-HNF'*: *is-sound-HNF'* *HNF*  $\mathcal{A} \ \mathcal{R}$   
**and** *inv-F-Q*: *invertible-mat*  $(\text{map-mat } \text{rat-of-int } F)$   
**and** *FG*:  $\{F, G\} \subseteq \text{carrier-mat } n \ n$

```

shows (HNF F = HNF G)  $\longleftrightarrow$  (lattice-of (rows F) = lattice-of (rows G))
<proof>
end

```

We define a new const similar to *external-lll-solver*, but now it only returns the reduced matrix.

```

consts external-lll-solver' :: integer  $\times$  integer  $\Rightarrow$  integer list list  $\Rightarrow$  integer list list

```

**hide-type (open)** *Finite-Cartesian-Product.vec*

The following definition is an adaptation of *reduce-basis-external*

```

definition reduce-basis-external' :: (int mat  $\Rightarrow$  int mat)  $\Rightarrow$  rat  $\Rightarrow$  int vec list  $\Rightarrow$ 
int vec list where

```

```

  reduce-basis-external' HNF  $\alpha$  fs = (case fs of Nil  $\Rightarrow$  [] | Cons f -  $\Rightarrow$  (let
    rb = reduce-basis  $\alpha$ ;
    fsi = map (map integer-of-int o list-of-vec) fs;
    n = dim-vec f;
    m = length fs;
    gsi = external-lll-solver' (map-prod integer-of-int integer-of-int (quotient-of  $\alpha$ ))
    fsi;
    gs = (map (vec-of-list o map int-of-integer) gsi) in
    if  $\neg$  (length gs = m  $\wedge$  ( $\forall$  gi  $\in$  set gs. dim-vec gi = n)) then
      Code.abort (STR "error in external LLL invocation: dimensions of reduced
basis do not fit"  $\square$  input to external solver: "
+ String.implode (show fs) + STR "' $\square$ '") ( $\lambda$  -. rb fs)
    else
      let Fs = mat-of-rows n fs;
          Gs = mat-of-rows n gs;
          H1 = HNF Fs;
          H2 = HNF Gs in
        if (H1 = H2) then rb gs
        else Code.abort (STR "the reduced matrix does not span the same lat-
tice"  $\square$  f,g,P1,P2,H1,H2 are as follows"  $\square$ 
+ String.implode (show Fs) + STR "' $\square$ '"
+ String.implode (show Gs) + STR "' $\square$ '"
+ String.implode (show H1) + STR "' $\square$ '"
+ String.implode (show H2) + STR "' $\square$ '"
) ( $\lambda$  -. rb fs)
    )

```

```

locale certification = LLL-with-assms +
fixes HNF::int mat  $\Rightarrow$  int mat and associates res
assumes sound-HNF': is-sound-HNF' HNF associates res
begin

```

```

lemma reduce-basis-external': assumes res: reduce-basis-external' HNF  $\alpha$  fs-init
= fs

```

```

shows reduced fs m LLL-invariant True m fs

```

*<proof>*  
**end**

**context** *LLL-with-assms*  
**begin**

We interpret the certification context using our formalized *HNF-algorithm*

**interpretation** *efficient-cert: certification n m fs-init  $\alpha$  HNF-algorithm use-sym-mod range ass-function-euclidean  $\lambda c.$  range (res-int c)*  
*<proof>*

**thm** *efficient-cert.reduce-basis-external'*

Same, but applying the naive HNF algorithm, moved to JNF library from the echelon form and Hermite normal form AFP entries

**interpretation** *cert: certification n m fs-init  $\alpha$  HNF-algorithm-from-HA use-sym-mod range ass-function-euclidean  $\lambda c.$  range (res-int c)*  
*<proof>*

**thm** *cert.reduce-basis-external'*

**lemma** *RBE-HNF-algorithm-efficient:*

**assumes** *reduce-basis-external' (HNF-algorithm use-sym-mod)  $\alpha$  fs-init = fs*  
**shows** *gram-schmidt-fs.reduced n (map of-int-hom.vec-hom fs)  $\alpha$  m*  
**and** *LLL-invariant True m fs <proof>*

**lemma** *RBE-HNF-algorithm-naive:*

**assumes** *reduce-basis-external' (HNF-algorithm-from-HA use-sym-mod)  $\alpha$  fs-init = fs*  
**shows** *gram-schmidt-fs.reduced n (map of-int-hom.vec-hom fs)  $\alpha$  m*  
**and** *LLL-invariant True m fs <proof>*

**end**

**lemma** *external-lll-solver'-code[code]:*

*external-lll-solver' = Code.abort (STR "require proper implementation of external-lll-solver'") ( $\lambda -.$  external-lll-solver')*  
*<proof>*

**end**