

# Matrix

Steven Obua

December 17, 2025

```
theory Matrix
imports Main HOL-Library.Lattice-Algebras
begin

type-synonym 'a infmatrix = nat  $\Rightarrow$  nat  $\Rightarrow$  'a

definition nonzero-positions :: (nat  $\Rightarrow$  nat  $\Rightarrow$  'a::zero)  $\Rightarrow$  (nat  $\times$  nat) set where
  nonzero-positions A = {pos. A (fst pos) (snd pos)  $\sim$  0}

definition matrix = {(f::(nat  $\Rightarrow$  nat  $\Rightarrow$  'a::zero)). finite (nonzero-positions f)}

typedef (overloaded) 'a matrix = matrix :: (nat  $\Rightarrow$  nat  $\Rightarrow$  'a::zero) set
  <proof>

declare Rep-matrix-inverse[simp]

lemma matrix-eqI:
  fixes A B :: 'a::zero matrix
  assumes  $\bigwedge m n. \text{Rep-matrix } A \ m \ n = \text{Rep-matrix } B \ m \ n$ 
  shows A=B
  <proof>

lemma finite-nonzero-positions : finite (nonzero-positions (Rep-matrix A))
  <proof>

definition nrows :: ('a::zero) matrix  $\Rightarrow$  nat where
  nrows A == if nonzero-positions(Rep-matrix A) = {} then 0 else Suc(Max ((image
fst) (nonzero-positions (Rep-matrix A))))

definition ncols :: ('a::zero) matrix  $\Rightarrow$  nat where
  ncols A == if nonzero-positions(Rep-matrix A) = {} then 0 else Suc(Max ((image
snd) (nonzero-positions (Rep-matrix A))))

lemma nrows:
  assumes hyp: nrows A  $\leq$  m
  shows (Rep-matrix A m n) = 0
```

$\langle proof \rangle$

**definition** *transpose-infmatrix* :: 'a infmatrix  $\Rightarrow$  'a infmatrix **where**  
*transpose-infmatrix* A j i == A i j

**definition** *transpose-matrix* :: ('a::zero) matrix  $\Rightarrow$  'a matrix **where**  
*transpose-matrix* == Abs-matrix o *transpose-infmatrix* o Rep-matrix

**declare** *transpose-infmatrix-def*[simp]

**lemma** *transpose-infmatrix-twice*[simp]: *transpose-infmatrix* (*transpose-infmatrix* A) = A  
 $\langle proof \rangle$

**lemma** *transpose-infmatrix*: *transpose-infmatrix* ( $\lambda j\ i. P\ j\ i$ ) = ( $\lambda j\ i. P\ i\ j$ )  
 $\langle proof \rangle$

**lemma** *transpose-infmatrix-closed*[simp]: Rep-matrix (Abs-matrix (*transpose-infmatrix* (Rep-matrix x))) = *transpose-infmatrix* (Rep-matrix x)  
 $\langle proof \rangle$

**lemma** *infmatrixforward*: (x::'a infmatrix) = y  $\implies \forall\ a\ b. x\ a\ b = y\ a\ b$   
 $\langle proof \rangle$

**lemma** *transpose-infmatrix-inject*: (*transpose-infmatrix* A = *transpose-infmatrix* B) = (A = B)  
 $\langle proof \rangle$

**lemma** *transpose-matrix-inject*: (*transpose-matrix* A = *transpose-matrix* B) = (A = B)  
 $\langle proof \rangle$

**lemma** *transpose-matrix*[simp]: Rep-matrix(*transpose-matrix* A) j i = Rep-matrix A i j  
 $\langle proof \rangle$

**lemma** *transpose-transpose-id*[simp]: *transpose-matrix* (*transpose-matrix* A) = A  
 $\langle proof \rangle$

**lemma** *nrows-transpose*[simp]: nrows (*transpose-matrix* A) = ncols A  
 $\langle proof \rangle$

**lemma** *ncols-transpose*[simp]: ncols (*transpose-matrix* A) = nrows A  
 $\langle proof \rangle$

**lemma** *ncols*: ncols A  $\leq n \implies$  Rep-matrix A m n = 0  
 $\langle proof \rangle$

**lemma** *ncols-le*: (ncols A  $\leq n$ )  $\longleftrightarrow (\forall j\ i. n \leq i \longrightarrow (Rep-matrix\ A\ j\ i) = 0)$  (is

- = ?st)  
 <proof>

**lemma** *less-ncols*:  $(n < \text{ncols } A) = (\exists j \ i. \ n \leq i \wedge (\text{Rep-matrix } A \ j \ i) \neq 0)$   
 <proof>

**lemma** *le-ncols*:  $(n \leq \text{ncols } A) = (\forall m. (\forall j \ i. \ m \leq i \longrightarrow (\text{Rep-matrix } A \ j \ i) = 0) \longrightarrow n \leq m)$   
 <proof>

**lemma** *nrows-le*:  $(\text{nrows } A \leq n) = (\forall j \ i. \ n \leq j \longrightarrow (\text{Rep-matrix } A \ j \ i) = 0)$  (is ?s)  
 <proof>

**lemma** *less-nrows*:  $(m < \text{nrows } A) = (\exists j \ i. \ m \leq j \wedge (\text{Rep-matrix } A \ j \ i) \neq 0)$   
 <proof>

**lemma** *le-nrows*:  $(n \leq \text{nrows } A) = (\forall m. (\forall j \ i. \ m \leq j \longrightarrow (\text{Rep-matrix } A \ j \ i) = 0) \longrightarrow n \leq m)$   
 <proof>

**lemma** *nrows-notzero*:  $\text{Rep-matrix } A \ m \ n \neq 0 \implies m < \text{nrows } A$   
 <proof>

**lemma** *ncols-notzero*:  $\text{Rep-matrix } A \ m \ n \neq 0 \implies n < \text{ncols } A$   
 <proof>

**lemma** *finite-natarray1*:  $\text{finite } \{x. \ x < (n::\text{nat})\}$   
 <proof>

**lemma** *finite-natarray2*:  $\text{finite } \{(x, y). \ x < (m::\text{nat}) \wedge y < (n::\text{nat})\}$   
 <proof>

**lemma** *RepAbs-matrix*:  
 assumes  $\exists m. \forall j \ i. \ m \leq j \longrightarrow x \ j \ i = 0$   
 and  $\exists n. \forall j \ i. \ (n \leq i \longrightarrow x \ j \ i = 0)$   
 shows  $(\text{Rep-matrix } (\text{Abs-matrix } x)) = x$   
 <proof>

**definition** *apply-infmatrix* ::  $('a \Rightarrow 'b) \Rightarrow 'a \ \text{infmatrix} \Rightarrow 'b \ \text{infmatrix}$  **where**  
 $\text{apply-infmatrix } f == \lambda A. (\lambda j \ i. \ f \ (A \ j \ i))$

**definition** *apply-matrix* ::  $('a \Rightarrow 'b) \Rightarrow ('a::\text{zero}) \ \text{matrix} \Rightarrow ('b::\text{zero}) \ \text{matrix}$  **where**  
 $\text{apply-matrix } f == \lambda A. \text{Abs-matrix } (\text{apply-infmatrix } f \ (\text{Rep-matrix } A))$

**definition** *combine-infmatrix* ::  $('a \Rightarrow 'b \Rightarrow 'c) \Rightarrow 'a \ \text{infmatrix} \Rightarrow 'b \ \text{infmatrix} \Rightarrow 'c \ \text{infmatrix}$  **where**  
 $\text{combine-infmatrix } f == \lambda A \ B. (\lambda j \ i. \ f \ (A \ j \ i) \ (B \ j \ i))$

**definition** *combine-matrix* :: ( $'a \Rightarrow 'b \Rightarrow 'c \Rightarrow ('a::\text{zero}) \text{ matrix} \Rightarrow ('b::\text{zero}) \text{ matrix} \Rightarrow ('c::\text{zero}) \text{ matrix}$  **where**  
*combine-matrix*  $f == \lambda A B. \text{Abs-matrix } (\text{combine-infmatrix } f (\text{Rep-matrix } A) (\text{Rep-matrix } B))$

**lemma** *expand-apply-infmatrix[simp]*:  $\text{apply-infmatrix } f A j i = f (A j i)$   
 $\langle \text{proof} \rangle$

**lemma** *expand-combine-infmatrix[simp]*:  $\text{combine-infmatrix } f A B j i = f (A j i) (B j i)$   
 $\langle \text{proof} \rangle$

**definition** *commutative* :: ( $'a \Rightarrow 'a \Rightarrow 'b \Rightarrow \text{bool}$  **where**  
*commutative*  $f == \forall x y. f x y = f y x$

**definition** *associative* :: ( $'a \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{bool}$  **where**  
*associative*  $f == \forall x y z. f (f x y) z = f x (f y z)$

To reason about associativity and commutativity of operations on matrices, let's take a step back and look at the general situation: Assume that we have sets  $A$  and  $B$  with  $B \subset A$  and an abstraction  $u : A \rightarrow B$ . This abstraction has to fulfill  $u(b) = b$  for all  $b \in B$ , but is arbitrary otherwise. Each function  $f : A \times A \rightarrow A$  now induces a function  $f' : B \times B \rightarrow B$  by  $f' = u \circ f$ . It is obvious that commutativity of  $f$  implies commutativity of  $f'$ :  $f'xy = u(fxy) = u(fyx) = f'yx$ .

**lemma** *combine-infmatrix-commute*:  
 $\text{commutative } f \implies \text{commutative } (\text{combine-infmatrix } f)$   
 $\langle \text{proof} \rangle$

**lemma** *combine-matrix-commute*:  
 $\text{commutative } f \implies \text{commutative } (\text{combine-matrix } f)$   
 $\langle \text{proof} \rangle$

On the contrary, given an associative function  $f$  we cannot expect  $f'$  to be associative. A counterexample is given by  $A = \mathbb{Z}$ ,  $B = \{-1, 0, 1\}$ , as  $f$  we take addition on  $\mathbb{Z}$ , which is clearly associative. The abstraction is given by  $u(a) = 0$  for  $a \notin B$ . Then we have

$$f'(f'11) - 1 = u(f(u(f11)) - 1) = u(f(u2) - 1) = u(f0 - 1) = -1,$$

but on the other hand we have

$$f'1(f'1 - 1) = u(f1(u(f1 - 1))) = u(f10) = 1.$$

A way out of this problem is to assume that  $f(A \times A) \subset A$  holds, and this is what we are going to do:

**lemma** *nonzero-positions-combine-infmatrix[simp]*:  $f 0 0 = 0 \implies \text{nonzero-positions } (\text{combine-infmatrix } f A B) \subseteq (\text{nonzero-positions } A) \cup (\text{nonzero-positions } B)$

$\langle \text{proof} \rangle$

**lemma** *finite-nonzero-positions-Rep[simp]*: *finite (nonzero-positions (Rep-matrix A))*

$\langle \text{proof} \rangle$

**lemma** *combine-infmatrix-closed [simp]*:

$f \ 0 \ 0 = 0 \implies \text{Rep-matrix } (\text{Abs-matrix } (\text{combine-infmatrix } f \ (\text{Rep-matrix } A) \ (\text{Rep-matrix } B))) = \text{combine-infmatrix } f \ (\text{Rep-matrix } A) \ (\text{Rep-matrix } B)$

$\langle \text{proof} \rangle$

We need the next two lemmas only later, but it is analog to the above one, so we prove them now:

**lemma** *nonzero-positions-apply-infmatrix[simp]*:  $f \ 0 = 0 \implies \text{nonzero-positions } (\text{apply-infmatrix } f \ A) \subseteq \text{nonzero-positions } A$

$\langle \text{proof} \rangle$

**lemma** *apply-infmatrix-closed [simp]*:

$f \ 0 = 0 \implies \text{Rep-matrix } (\text{Abs-matrix } (\text{apply-infmatrix } f \ (\text{Rep-matrix } A))) = \text{apply-infmatrix } f \ (\text{Rep-matrix } A)$

$\langle \text{proof} \rangle$

**lemma** *combine-infmatrix-assoc[simp]*:  $f \ 0 \ 0 = 0 \implies \text{associative } f \implies \text{associative } (\text{combine-infmatrix } f)$

$\langle \text{proof} \rangle$

**lemma** *combine-matrix-assoc*:  $f \ 0 \ 0 = 0 \implies \text{associative } f \implies \text{associative } (\text{combine-matrix } f)$

$\langle \text{proof} \rangle$

**lemma** *Rep-apply-matrix[simp]*:  $f \ 0 = 0 \implies \text{Rep-matrix } (\text{apply-matrix } f \ A) \ j \ i = f \ (\text{Rep-matrix } A \ j \ i)$

$\langle \text{proof} \rangle$

**lemma** *Rep-combine-matrix[simp]*:  $f \ 0 \ 0 = 0 \implies \text{Rep-matrix } (\text{combine-matrix } f \ A \ B) \ j \ i = f \ (\text{Rep-matrix } A \ j \ i) \ (\text{Rep-matrix } B \ j \ i)$

$\langle \text{proof} \rangle$

**lemma** *combine-nrows-max*:  $f \ 0 \ 0 = 0 \implies \text{nrows } (\text{combine-matrix } f \ A \ B) \leq \max (\text{nrows } A) (\text{nrows } B)$

$\langle \text{proof} \rangle$

**lemma** *combine-ncols-max*:  $f \ 0 \ 0 = 0 \implies \text{ncols } (\text{combine-matrix } f \ A \ B) \leq \max (\text{ncols } A) (\text{ncols } B)$

$\langle \text{proof} \rangle$

**lemma** *combine-nrows*:  $f \ 0 \ 0 = 0 \implies \text{nrows } A \leq q \implies \text{nrows } B \leq q \implies \text{nrows } (\text{combine-matrix } f \ A \ B) \leq q$

$\langle \text{proof} \rangle$

**lemma** *combine-ncols*:  $f\ 0\ 0 = 0 \implies \text{ncols } A \leq q \implies \text{ncols } B \leq q \implies \text{ncols}(\text{combine-matrix } f\ A\ B) \leq q$   
 ⟨proof⟩

**definition** *zero-r-neutral* ::  $('a \Rightarrow 'b :: \text{zero} \Rightarrow 'a) \Rightarrow \text{bool}$  **where**  
*zero-r-neutral*  $f == \forall a. f\ a\ 0 = a$

**definition** *zero-l-neutral* ::  $('a :: \text{zero} \Rightarrow 'b \Rightarrow 'b) \Rightarrow \text{bool}$  **where**  
*zero-l-neutral*  $f == \forall a. f\ 0\ a = a$

**definition** *zero-closed* ::  $(( 'a :: \text{zero} \Rightarrow ('b :: \text{zero} \Rightarrow ('c :: \text{zero}))) \Rightarrow \text{bool})$  **where**  
*zero-closed*  $f == (\forall x. f\ x\ 0 = 0) \wedge (\forall y. f\ 0\ y = 0)$

**primrec** *foldseq* ::  $('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow (\text{nat} \Rightarrow 'a) \Rightarrow \text{nat} \Rightarrow 'a$   
**where**

*foldseq*  $f\ s\ 0 = s\ 0$   
 | *foldseq*  $f\ s\ (\text{Suc } n) = f\ (s\ 0)\ (\text{foldseq } f\ (\lambda k. s(\text{Suc } k))\ n)$

**primrec** *foldseq-transposed* ::  $('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow (\text{nat} \Rightarrow 'a) \Rightarrow \text{nat} \Rightarrow 'a$   
**where**

*foldseq-transposed*  $f\ s\ 0 = s\ 0$   
 | *foldseq-transposed*  $f\ s\ (\text{Suc } n) = f\ (\text{foldseq-transposed } f\ s\ n)\ (s\ (\text{Suc } n))$

**lemma** *foldseq-assoc*:  
**assumes** *a:associative* *f*  
**shows** *associative*  $f \implies \text{foldseq } f = \text{foldseq-transposed } f$   
 ⟨proof⟩

**lemma** *foldseq-distr*:  
**assumes** *assoc: associative* *f* **and** *comm: commutative* *f*  
**shows** *foldseq*  $f\ (\lambda k. f\ (u\ k)\ (v\ k))\ n = f\ (\text{foldseq } f\ u\ n)\ (\text{foldseq } f\ v\ n)$   
 ⟨proof⟩

**theorem**  $\llbracket \text{associative } f; \text{ associative } g; \forall a\ b\ c\ d. g\ (f\ a\ b)\ (f\ c\ d) = f\ (g\ a\ c)\ (g\ b\ d); \exists x\ y. (f\ x) \neq (f\ y); \exists x\ y. (g\ x) \neq (g\ y); f\ x\ x = x; g\ x\ x = x \rrbracket \implies f=g \mid (\forall y. f\ y\ x = y) \mid (\forall y. g\ y\ x = y)$   
 ⟨proof⟩

**lemma** *foldseq-zero*:  
**assumes** *fz: f 0 0 = 0* **and** *sz:  $\forall i. i \leq n \longrightarrow s\ i = 0$*   
**shows** *foldseq*  $f\ s\ n = 0$   
 ⟨proof⟩

**lemma** *foldseq-significant-positions*:  
**assumes** *p:  $\forall i. i \leq N \longrightarrow S\ i = T\ i$*   
**shows** *foldseq*  $f\ S\ N = \text{foldseq } f\ T\ N$   
 ⟨proof⟩

**lemma** *foldseq-tail*:

**assumes**  $M \leq N$

**shows**  $\text{foldseq } f \ S \ N = \text{foldseq } f \ (\lambda k. \ (\text{if } k < M \text{ then } (S \ k) \text{ else } (\text{foldseq } f \ (\lambda k. \ S(k+M)) \ (N-M)))) \ M$   
 $\langle \text{proof} \rangle$

**lemma** *foldseq-zerotail*:

**assumes**  $fz: f \ 0 \ 0 = 0$  **and**  $sz: \forall i. \ n \leq i \longrightarrow s \ i = 0$  **and**  $nm: n \leq m$

**shows**  $\text{foldseq } f \ s \ n = \text{foldseq } f \ s \ m$

$\langle \text{proof} \rangle$

**lemma** *foldseq-zerotail2*:

**assumes**  $\forall x. \ f \ x \ 0 = x$

**and**  $\forall i. \ n < i \longrightarrow s \ i = 0$

**and**  $nm: n \leq m$

**shows**  $\text{foldseq } f \ s \ n = \text{foldseq } f \ s \ m$

$\langle \text{proof} \rangle$

**lemma** *foldseq-zerostart*:

**assumes**  $f00x: \forall x. \ f \ 0 \ (f \ 0 \ x) = f \ 0 \ x$  **and**  $0: \forall i. \ i \leq n \longrightarrow s \ i = 0$

**shows**  $\text{foldseq } f \ s \ (\text{Suc } n) = f \ 0 \ (s \ (\text{Suc } n))$

$\langle \text{proof} \rangle$

**lemma** *foldseq-zerostart2*:

**assumes**  $x: \forall x. \ f \ 0 \ x = x$  **and**  $0: \forall i. \ i < n \longrightarrow s \ i = 0$

**shows**  $\text{foldseq } f \ s \ n = s \ n$

$\langle \text{proof} \rangle$

**lemma** *foldseq-almostzero*:

**assumes**  $f0x: \forall x. \ f \ 0 \ x = x$  **and**  $fx0: \forall x. \ f \ x \ 0 = x$  **and**  $s0: \forall i. \ i \neq j \longrightarrow s \ i = 0$

**shows**  $\text{foldseq } f \ s \ n = (\text{if } (j \leq n) \text{ then } (s \ j) \text{ else } 0)$

$\langle \text{proof} \rangle$

**lemma** *foldseq-distr-unary*:

**assumes**  $\bigwedge a \ b. \ g \ (f \ a \ b) = f \ (g \ a) \ (g \ b)$

**shows**  $g(\text{foldseq } f \ s \ n) = \text{foldseq } f \ (\lambda x. \ g(s \ x)) \ n$

$\langle \text{proof} \rangle$

**definition** *mult-matrix-n* ::  $\text{nat} \Rightarrow (('a::\text{zero}) \Rightarrow ('b::\text{zero}) \Rightarrow ('c::\text{zero})) \Rightarrow ('c \Rightarrow 'c \Rightarrow 'c) \Rightarrow 'a \text{ matrix} \Rightarrow 'b \text{ matrix} \Rightarrow 'c \text{ matrix}$  **where**

$\text{mult-matrix-n } n \text{ fmul fadd } A \ B == \text{Abs-matrix}(\lambda j \ i. \ \text{foldseq fadd } (\lambda k. \ \text{fmul } (\text{Rep-matrix } A \ j \ k) \ (\text{Rep-matrix } B \ k \ i)) \ n)$

**definition** *mult-matrix* ::  $((('a::\text{zero}) \Rightarrow ('b::\text{zero}) \Rightarrow ('c::\text{zero})) \Rightarrow ('c \Rightarrow 'c \Rightarrow 'c) \Rightarrow 'a \text{ matrix} \Rightarrow 'b \text{ matrix} \Rightarrow 'c \text{ matrix})$  **where**

$\text{mult-matrix fmul fadd } A \ B == \text{mult-matrix-n } (\max (\text{ncols } A) (\text{nrows } B)) \text{ fmul fadd } A \ B$

**lemma** *mult-matrix-n*:

**assumes**  $\text{ncols } A \leq n \text{ nrows } B \leq n$   $\text{fadd } 0 \ 0 = 0$   $\text{fmul } 0 \ 0 = 0$

**shows**  $\text{mult-matrix } \text{fmul } \text{fadd } A \ B = \text{mult-matrix-n } n \ \text{fmul } \text{fadd } A \ B$

*<proof>*

**lemma** *mult-matrix-nm*:

**assumes**  $\text{ncols } A \leq n \text{ nrows } B \leq n \text{ ncols } A \leq m \text{ nrows } B \leq m$   $\text{fadd } 0 \ 0 = 0$   
 $\text{fmul } 0 \ 0 = 0$

**shows**  $\text{mult-matrix-n } n \ \text{fmul } \text{fadd } A \ B = \text{mult-matrix-n } m \ \text{fmul } \text{fadd } A \ B$

*<proof>*

**definition** *r-distributive* ::  $('a \Rightarrow 'b \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'b \Rightarrow 'b) \Rightarrow \text{bool}$  **where**

*r-distributive*  $\text{fmul } \text{fadd} == \forall a \ u \ v. \text{fmul } a \ (\text{fadd } u \ v) = \text{fadd } (\text{fmul } a \ u) \ (\text{fmul } a \ v)$

**definition** *l-distributive* ::  $('a \Rightarrow 'b \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow \text{bool}$  **where**

*l-distributive*  $\text{fmul } \text{fadd} == \forall a \ u \ v. \text{fmul } (\text{fadd } u \ v) \ a = \text{fadd } (\text{fmul } u \ a) \ (\text{fmul } v \ a)$

**definition** *distributive* ::  $('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow \text{bool}$  **where**

*distributive*  $\text{fmul } \text{fadd} == \text{l-distributive } \text{fmul } \text{fadd} \wedge \text{r-distributive } \text{fmul } \text{fadd}$

**lemma** *max1*: !!  $a \ x \ y. (a::\text{nat}) \leq x \implies a \leq \text{max } x \ y$  *<proof>*

**lemma** *max2*: !!  $b \ x \ y. (b::\text{nat}) \leq y \implies b \leq \text{max } x \ y$  *<proof>*

**lemma** *r-distributive-matrix*:

**assumes**

*r-distributive*  $\text{fmul } \text{fadd}$

*associative*  $\text{fadd}$

*commutative*  $\text{fadd}$

$\text{fadd } 0 \ 0 = 0$

$\forall a. \text{fmul } a \ 0 = 0$

$\forall a. \text{fmul } 0 \ a = 0$

**shows** *r-distributive*  $(\text{mult-matrix } \text{fmul } \text{fadd}) \ (\text{combine-matrix } \text{fadd})$

*<proof>*

**lemma** *l-distributive-matrix*:

**assumes**

*l-distributive*  $\text{fmul } \text{fadd}$

*associative*  $\text{fadd}$

*commutative*  $\text{fadd}$

$\text{fadd } 0 \ 0 = 0$

$\forall a. \text{fmul } a \ 0 = 0$

$\forall a. \text{fmul } 0 \ a = 0$

**shows** *l-distributive*  $(\text{mult-matrix } \text{fmul } \text{fadd}) \ (\text{combine-matrix } \text{fadd})$

*<proof>*

**instantiation** *matrix* ::  $(\text{zero}) \ \text{zero}$



**begin**

**definition** *zero-matrix-def*:  $0 = \text{Abs-matrix } (\lambda j \ i. \ 0)$

**instance**  $\langle \text{proof} \rangle$

**end**

**lemma** *Rep-zero-matrix-def[simp]*:  $\text{Rep-matrix } 0 \ j \ i = 0$   
 $\langle \text{proof} \rangle$

**lemma** *zero-matrix-def-nrows[simp]*:  $\text{nrows } 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *zero-matrix-def-ncols[simp]*:  $\text{ncols } 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *combine-matrix-zero-l-neutral*:  $\text{zero-l-neutral } f \implies \text{zero-l-neutral } (\text{combine-matrix } f)$   
 $\langle \text{proof} \rangle$

**lemma** *combine-matrix-zero-r-neutral*:  $\text{zero-r-neutral } f \implies \text{zero-r-neutral } (\text{combine-matrix } f)$   
 $\langle \text{proof} \rangle$

**lemma** *mult-matrix-zero-closed*:  $\llbracket \text{fadd } 0 \ 0 = 0; \text{zero-closed } \text{fmul} \rrbracket \implies \text{zero-closed}$   
 $(\text{mult-matrix } \text{fmul } \text{fadd})$   
 $\langle \text{proof} \rangle$

**lemma** *mult-matrix-n-zero-right[simp]*:  $\llbracket \text{fadd } 0 \ 0 = 0; \forall a. \text{fmul } a \ 0 = 0 \rrbracket \implies$   
 $\text{mult-matrix-n } n \ \text{fmul } \text{fadd } A \ 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *mult-matrix-n-zero-left[simp]*:  $\llbracket \text{fadd } 0 \ 0 = 0; \forall a. \text{fmul } 0 \ a = 0 \rrbracket \implies$   
 $\text{mult-matrix-n } n \ \text{fmul } \text{fadd } 0 \ A = 0$   
 $\langle \text{proof} \rangle$

**lemma** *mult-matrix-zero-left[simp]*:  $\llbracket \text{fadd } 0 \ 0 = 0; \forall a. \text{fmul } 0 \ a = 0 \rrbracket \implies \text{mult-matrix}$   
 $\text{fmul } \text{fadd } 0 \ A = 0$   
 $\langle \text{proof} \rangle$

**lemma** *mult-matrix-zero-right[simp]*:  $\llbracket \text{fadd } 0 \ 0 = 0; \forall a. \text{fmul } a \ 0 = 0 \rrbracket \implies$   
 $\text{mult-matrix } \text{fmul } \text{fadd } A \ 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *apply-matrix-zero[simp]*:  $f \ 0 = 0 \implies \text{apply-matrix } f \ 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *combine-matrix-zero*:  $f \ 0 \ 0 = 0 \implies \text{combine-matrix } f \ 0 \ 0 = 0$

$\langle \text{proof} \rangle$

**lemma** *transpose-matrix-zero*[simp]: *transpose-matrix* 0 = 0  
 $\langle \text{proof} \rangle$

**lemma** *apply-zero-matrix-def*[simp]: *apply-matrix* ( $\lambda x. 0$ ) A = 0  
 $\langle \text{proof} \rangle$

**definition** *singleton-matrix* ::  $\text{nat} \Rightarrow \text{nat} \Rightarrow ('a::\text{zero}) \Rightarrow 'a \text{ matrix}$  **where**  
*singleton-matrix* j i a == *Abs-matrix*( $\lambda m n. \text{if } j = m \wedge i = n \text{ then } a \text{ else } 0$ )

**definition** *move-matrix* ::  $('a::\text{zero}) \text{ matrix} \Rightarrow \text{int} \Rightarrow \text{int} \Rightarrow 'a \text{ matrix}$  **where**  
*move-matrix* A y x == *Abs-matrix*( $\lambda j i. \text{if } (((\text{int } j) - y) < 0) \mid (((\text{int } i) - x) < 0) \text{ then } 0 \text{ else } \text{Rep-matrix } A (\text{nat } ((\text{int } j) - y)) (\text{nat } ((\text{int } i) - x)))$ )

**definition** *take-rows* ::  $('a::\text{zero}) \text{ matrix} \Rightarrow \text{nat} \Rightarrow 'a \text{ matrix}$  **where**  
*take-rows* A r == *Abs-matrix*( $\lambda j i. \text{if } (j < r) \text{ then } (\text{Rep-matrix } A j i) \text{ else } 0$ )

**definition** *take-columns* ::  $('a::\text{zero}) \text{ matrix} \Rightarrow \text{nat} \Rightarrow 'a \text{ matrix}$  **where**  
*take-columns* A c == *Abs-matrix*( $\lambda j i. \text{if } (i < c) \text{ then } (\text{Rep-matrix } A j i) \text{ else } 0$ )

**definition** *column-of-matrix* ::  $('a::\text{zero}) \text{ matrix} \Rightarrow \text{nat} \Rightarrow 'a \text{ matrix}$  **where**  
*column-of-matrix* A n == *take-columns* (*move-matrix* A 0 (- int n)) 1

**definition** *row-of-matrix* ::  $('a::\text{zero}) \text{ matrix} \Rightarrow \text{nat} \Rightarrow 'a \text{ matrix}$  **where**  
*row-of-matrix* A m == *take-rows* (*move-matrix* A (- int m) 0) 1

**lemma** *Rep-singleton-matrix*[simp]: *Rep-matrix* (*singleton-matrix* j i e) m n = (if  
j = m  $\wedge$  i = n then e else 0)  
 $\langle \text{proof} \rangle$

**lemma** *apply-singleton-matrix*[simp]: f 0 = 0  $\implies$  *apply-matrix* f (*singleton-matrix*  
j i x) = (*singleton-matrix* j i (f x))  
 $\langle \text{proof} \rangle$

**lemma** *singleton-matrix-zero*[simp]: *singleton-matrix* j i 0 = 0  
 $\langle \text{proof} \rangle$

**lemma** *nrows-singleton*[simp]: *nrows*(*singleton-matrix* j i e) = (if e = 0 then 0 else  
Suc j)  
 $\langle \text{proof} \rangle$

**lemma** *ncols-singleton*[simp]: *ncols*(*singleton-matrix* j i e) = (if e = 0 then 0 else  
Suc i)  
 $\langle \text{proof} \rangle$

**lemma** *combine-singleton*: f 0 0 = 0  $\implies$  *combine-matrix* f (*singleton-matrix* j i  
a) (*singleton-matrix* j i b) = *singleton-matrix* j i (f a b)  
 $\langle \text{proof} \rangle$

**lemma** *transpose-singleton[simp]*:  $\text{transpose-matrix } (\text{singleton-matrix } j \ i \ a) = \text{singleton-matrix } i \ j \ a$   
 ⟨proof⟩

**lemma** *Rep-move-matrix[simp]*:  
 $\text{Rep-matrix } (\text{move-matrix } A \ y \ x) \ j \ i =$   
 $(\text{if } (((\text{int } j) - y) < 0) \mid (((\text{int } i) - x) < 0) \text{ then } 0 \text{ else } \text{Rep-matrix } A \ (\text{nat}((\text{int } j) - y)) \ (\text{nat}((\text{int } i) - x)))$   
 ⟨proof⟩

**lemma** *move-matrix-0-0[simp]*:  $\text{move-matrix } A \ 0 \ 0 = A$   
 ⟨proof⟩

**lemma** *move-matrix-ortho*:  $\text{move-matrix } A \ j \ i = \text{move-matrix } (\text{move-matrix } A \ j \ 0) \ 0 \ i$   
 ⟨proof⟩

**lemma** *transpose-move-matrix[simp]*:  
 $\text{transpose-matrix } (\text{move-matrix } A \ x \ y) = \text{move-matrix } (\text{transpose-matrix } A) \ y \ x$   
 ⟨proof⟩

**lemma** *move-matrix-singleton[simp]*:  $\text{move-matrix } (\text{singleton-matrix } u \ v \ x) \ j \ i =$   
 $(\text{if } (j + \text{int } u < 0) \mid (i + \text{int } v < 0) \text{ then } 0 \text{ else } (\text{singleton-matrix } (\text{nat } (j + \text{int } u)) \ (\text{nat } (i + \text{int } v)) \ x))$   
 ⟨proof⟩

**lemma** *Rep-take-columns[simp]*:  
 $\text{Rep-matrix } (\text{take-columns } A \ c) \ j \ i = (\text{if } i < c \text{ then } (\text{Rep-matrix } A \ j \ i) \text{ else } 0)$   
 ⟨proof⟩

**lemma** *Rep-take-rows[simp]*:  
 $\text{Rep-matrix } (\text{take-rows } A \ r) \ j \ i = (\text{if } j < r \text{ then } (\text{Rep-matrix } A \ j \ i) \text{ else } 0)$   
 ⟨proof⟩

**lemma** *Rep-column-of-matrix[simp]*:  
 $\text{Rep-matrix } (\text{column-of-matrix } A \ c) \ j \ i = (\text{if } i = 0 \text{ then } (\text{Rep-matrix } A \ j \ c) \text{ else } 0)$   
 ⟨proof⟩

**lemma** *Rep-row-of-matrix[simp]*:  
 $\text{Rep-matrix } (\text{row-of-matrix } A \ r) \ j \ i = (\text{if } j = 0 \text{ then } (\text{Rep-matrix } A \ r \ i) \text{ else } 0)$   
 ⟨proof⟩

**lemma** *column-of-matrix*:  $\text{ncols } A \leq n \implies \text{column-of-matrix } A \ n = 0$   
 ⟨proof⟩

**lemma** *row-of-matrix*:  $\text{nrows } A \leq n \implies \text{row-of-matrix } A \ n = 0$   
 ⟨proof⟩

**lemma** *mult-matrix-singleton-right*[simp]:  
**assumes**  $\forall x. \text{fmul } x \ 0 = 0 \ \forall x. \text{fmul } 0 \ x = 0 \ \forall x. \text{fadd } 0 \ x = x \ \forall x. \text{fadd } x \ 0 = x$   
**shows**  $(\text{mult-matrix } \text{fmul } \text{fadd } A \ (\text{singleton-matrix } j \ i \ e)) = \text{apply-matrix } (\lambda x. \text{fmul } x \ e) \ (\text{move-matrix } (\text{column-of-matrix } A \ j) \ 0 \ (\text{int } i))$   
 $\langle \text{proof} \rangle$

**lemma** *mult-matrix-ext*:  
**assumes**  
*eprem*:  
 $\exists e. (\forall a \ b. a \neq b \longrightarrow \text{fmul } a \ e \neq \text{fmul } b \ e)$   
**and** *fpreds*:  
 $\forall a. \text{fmul } 0 \ a = 0$   
 $\forall a. \text{fmul } a \ 0 = 0$   
 $\forall a. \text{fadd } a \ 0 = a$   
 $\forall a. \text{fadd } 0 \ a = a$   
**and** *contrapreds*:  $\text{mult-matrix } \text{fmul } \text{fadd } A = \text{mult-matrix } \text{fmul } \text{fadd } B$   
**shows**  $A = B$   
 $\langle \text{proof} \rangle$

**definition** *foldmatrix* ::  $('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow ('a \text{ infmatrix}) \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a$  **where**  
 $\text{foldmatrix } f \ g \ A \ m \ n == \text{foldseq-transposed } g \ (\lambda j. \text{foldseq } f \ (A \ j) \ n) \ m$

**definition** *foldmatrix-transposed* ::  $('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow ('a \text{ infmatrix}) \Rightarrow \text{nat} \Rightarrow \text{nat} \Rightarrow 'a$  **where**  
 $\text{foldmatrix-transposed } f \ g \ A \ m \ n == \text{foldseq } g \ (\lambda j. \text{foldseq-transposed } f \ (A \ j) \ n) \ m$

**lemma** *foldmatrix-transpose*:  
**assumes**  $\forall a \ b \ c \ d. g(f \ a \ b) \ (f \ c \ d) = f \ (g \ a \ c) \ (g \ b \ d)$   
**shows**  $\text{foldmatrix } f \ g \ A \ m \ n = \text{foldmatrix-transposed } g \ f \ (\text{transpose-infmatrix } A) \ n \ m$   
 $\langle \text{proof} \rangle$

**lemma** *foldseq-foldseq*:  
**assumes** *associative*  $f$  *associative*  $g \ \forall a \ b \ c \ d. g(f \ a \ b) \ (f \ c \ d) = f \ (g \ a \ c) \ (g \ b \ d)$   
**shows**  
 $\text{foldseq } g \ (\lambda j. \text{foldseq } f \ (A \ j) \ n) \ m = \text{foldseq } f \ (\lambda j. \text{foldseq } g \ ((\text{transpose-infmatrix } A) \ j) \ m) \ n$   
 $\langle \text{proof} \rangle$

**lemma** *mult-n-nrows*:  
**assumes**  $\forall a. \text{fmul } 0 \ a = 0 \ \forall a. \text{fmul } a \ 0 = 0 \ \text{fadd } 0 \ 0 = 0$   
**shows**  $\text{nrows } (\text{mult-matrix-n } n \ \text{fmul } \text{fadd } A \ B) \leq \text{nrows } A$   
 $\langle \text{proof} \rangle$

**lemma** *mult-n-ncols*:  
**assumes**  $\forall a. \text{fmul } 0 \ a = 0 \ \forall a. \text{fmul } a \ 0 = 0 \ \text{fadd } 0 \ 0 = 0$

**shows**  $ncols \text{ (mult-matrix-n } n \text{ fmul fadd } A \text{ } B) \leq ncols \text{ } B$   
 ⟨proof⟩

**lemma** *mult-nrows*:

**assumes**

$\forall a. \text{fmul } 0 \text{ } a = 0$

$\forall a. \text{fmul } a \text{ } 0 = 0$

$\text{fadd } 0 \text{ } 0 = 0$

**shows**  $nrows \text{ (mult-matrix fmul fadd } A \text{ } B) \leq nrows \text{ } A$

⟨proof⟩

**lemma** *mult-ncols*:

**assumes**

$\forall a. \text{fmul } 0 \text{ } a = 0$

$\forall a. \text{fmul } a \text{ } 0 = 0$

$\text{fadd } 0 \text{ } 0 = 0$

**shows**  $ncols \text{ (mult-matrix fmul fadd } A \text{ } B) \leq ncols \text{ } B$

⟨proof⟩

**lemma** *nrows-move-matrix-le*:  $nrows \text{ (move-matrix } A \text{ } j \text{ } i) \leq nat((int \text{ (nrows } A)) + j)$

⟨proof⟩

**lemma** *ncols-move-matrix-le*:  $ncols \text{ (move-matrix } A \text{ } j \text{ } i) \leq nat((int \text{ (ncols } A)) + i)$

⟨proof⟩

**lemma** *mult-matrix-assoc*:

**assumes**

$\forall a. \text{fmul1 } 0 \text{ } a = 0$

$\forall a. \text{fmul1 } a \text{ } 0 = 0$

$\forall a. \text{fmul2 } 0 \text{ } a = 0$

$\forall a. \text{fmul2 } a \text{ } 0 = 0$

$\text{fadd1 } 0 \text{ } 0 = 0$

$\text{fadd2 } 0 \text{ } 0 = 0$

$\forall a \text{ } b \text{ } c \text{ } d. \text{fadd2 (fadd1 } a \text{ } b) \text{ (fadd1 } c \text{ } d) = \text{fadd1 (fadd2 } a \text{ } c) \text{ (fadd2 } b \text{ } d)$

*associative fadd1*

*associative fadd2*

$\forall a \text{ } b \text{ } c. \text{fmul2 (fmul1 } a \text{ } b) \text{ } c = \text{fmul1 } a \text{ (fmul2 } b \text{ } c)$

$\forall a \text{ } b \text{ } c. \text{fmul2 (fadd1 } a \text{ } b) \text{ } c = \text{fadd1 (fmul2 } a \text{ } c) \text{ (fmul2 } b \text{ } c)$

$\forall a \text{ } b \text{ } c. \text{fmul1 } c \text{ (fadd2 } a \text{ } b) = \text{fadd2 (fmul1 } c \text{ } a) \text{ (fmul1 } c \text{ } b)$

**shows**  $\text{mult-matrix fmul2 fadd2 (mult-matrix fmul1 fadd1 } A \text{ } B) \text{ } C = \text{mult-matrix fmul1 fadd1 } A \text{ (mult-matrix fmul2 fadd2 } B \text{ } C)$

⟨proof⟩

**lemma** *mult-matrix-assoc-simple*:

**assumes**

$\forall a. \text{fmul } 0 \text{ } a = 0$

$\forall a. \text{fmul } a \text{ } 0 = 0$

*associative fadd*  
*commutative fadd*  
*associative fmul*  
*distributive fmul fadd*  
**shows**  $\text{mult-matrix fmul fadd (mult-matrix fmul fadd A B) C} = \text{mult-matrix fmul fadd A (mult-matrix fmul fadd B C)}$   
 <proof>

**lemma** *transpose-apply-matrix*:  $f\ 0 = 0 \implies \text{transpose-matrix (apply-matrix f A)}$   
 $= \text{apply-matrix f (transpose-matrix A)}$   
 <proof>

**lemma** *transpose-combine-matrix*:  $f\ 0\ 0 = 0 \implies \text{transpose-matrix (combine-matrix f A B)}$   
 $= \text{combine-matrix f (transpose-matrix A) (transpose-matrix B)}$   
 <proof>

**lemma** *Rep-mult-matrix*:  
**assumes**  $\forall a. \text{fmul } 0\ a = 0 \ \forall a. \text{fmul } a\ 0 = 0 \ \text{fadd } 0\ 0 = 0$   
**shows**  
 $\text{Rep-matrix(mult-matrix fmul fadd A B) } j\ i =$   
 $\text{foldseq fadd } (\lambda k. \text{fmul (Rep-matrix A } j\ k) (\text{Rep-matrix B } k\ i)) (\text{max (ncols A)}$   
 $(\text{nrows B}))$   
 <proof>

**lemma** *transpose-mult-matrix*:  
**assumes**  
 $\forall a. \text{fmul } 0\ a = 0$   
 $\forall a. \text{fmul } a\ 0 = 0$   
 $\text{fadd } 0\ 0 = 0$   
 $\forall x\ y. \text{fmul } y\ x = \text{fmul } x\ y$   
**shows**  
 $\text{transpose-matrix (mult-matrix fmul fadd A B)} = \text{mult-matrix fmul fadd (transpose-matrix B) (transpose-matrix A)}$   
 <proof>

**lemma** *column-transpose-matrix*:  $\text{column-of-matrix (transpose-matrix A) } n = \text{transpose-matrix (row-of-matrix A } n)$   
 <proof>

**lemma** *take-columns-transpose-matrix*:  $\text{take-columns (transpose-matrix A) } n = \text{transpose-matrix (take-rows A } n)$   
 <proof>

**instantiation** *matrix* ::  $(\{\text{zero}, \text{ord}\})\ \text{ord}$   
**begin**

**definition**  
*le-matrix-def*:  $A \leq B \longleftrightarrow (\forall j\ i. \text{Rep-matrix A } j\ i \leq \text{Rep-matrix B } j\ i)$

**definition**

*less-def*:  $A < (B :: 'a \text{ matrix}) \longleftrightarrow A \leq B \wedge \neg B \leq A$

**instance**  $\langle \text{proof} \rangle$

**end**

**instance** *matrix* ::  $(\{\text{zero}, \text{order}\}) \text{ order}$   
 $\langle \text{proof} \rangle$

**lemma** *le-apply-matrix*:

**assumes**

$f \ 0 = 0$

$\forall x \ y. x \leq y \longrightarrow f \ x \leq f \ y$

$(a :: 'a :: \{\text{ord}, \text{zero}\}) \text{ matrix} \leq b$

**shows**  $\text{apply-matrix } f \ a \leq \text{apply-matrix } f \ b$

$\langle \text{proof} \rangle$

**lemma** *le-combine-matrix*:

**assumes**

$f \ 0 \ 0 = 0$

$\forall a \ b \ c \ d. a \leq b \wedge c \leq d \longrightarrow f \ a \ c \leq f \ b \ d$

$A \leq B$

$C \leq D$

**shows**  $\text{combine-matrix } f \ A \ C \leq \text{combine-matrix } f \ B \ D$

$\langle \text{proof} \rangle$

**lemma** *le-left-combine-matrix*:

**assumes**

$f \ 0 \ 0 = 0$

$\forall a \ b \ c. a \leq b \longrightarrow f \ c \ a \leq f \ c \ b$

$A \leq B$

**shows**  $\text{combine-matrix } f \ C \ A \leq \text{combine-matrix } f \ C \ B$

$\langle \text{proof} \rangle$

**lemma** *le-right-combine-matrix*:

**assumes**

$f \ 0 \ 0 = 0$

$\forall a \ b \ c. a \leq b \longrightarrow f \ a \ c \leq f \ b \ c$

$A \leq B$

**shows**  $\text{combine-matrix } f \ A \ C \leq \text{combine-matrix } f \ B \ C$

$\langle \text{proof} \rangle$

**lemma** *le-transpose-matrix*:  $(A \leq B) = (\text{transpose-matrix } A \leq \text{transpose-matrix } B)$

$\langle \text{proof} \rangle$

**lemma** *le-foldseq*:

**assumes**

$\forall a\ b\ c\ d. a \leq b \wedge c \leq d \longrightarrow f\ a\ c \leq f\ b\ d$   
 $\forall i. i \leq n \longrightarrow s\ i \leq t\ i$   
**shows**  $\text{foldseq}\ f\ s\ n \leq \text{foldseq}\ f\ t\ n$   
 $\langle \text{proof} \rangle$

**lemma** *le-left-mult*:

**assumes**  
 $\forall a\ b\ c\ d. a \leq b \wedge c \leq d \longrightarrow \text{fadd}\ a\ c \leq \text{fadd}\ b\ d$   
 $\forall c\ a\ b. 0 \leq c \wedge a \leq b \longrightarrow \text{fmul}\ c\ a \leq \text{fmul}\ c\ b$   
 $\forall a. \text{fmul}\ 0\ a = 0$   
 $\forall a. \text{fmul}\ a\ 0 = 0$   
 $\text{fadd}\ 0\ 0 = 0$   
 $0 \leq C$   
 $A \leq B$   
**shows**  $\text{mult-matrix}\ \text{fmul}\ \text{fadd}\ C\ A \leq \text{mult-matrix}\ \text{fmul}\ \text{fadd}\ C\ B$   
 $\langle \text{proof} \rangle$

**lemma** *le-right-mult*:

**assumes**  
 $\forall a\ b\ c\ d. a \leq b \wedge c \leq d \longrightarrow \text{fadd}\ a\ c \leq \text{fadd}\ b\ d$   
 $\forall c\ a\ b. 0 \leq c \wedge a \leq b \longrightarrow \text{fmul}\ a\ c \leq \text{fmul}\ b\ c$   
 $\forall a. \text{fmul}\ 0\ a = 0$   
 $\forall a. \text{fmul}\ a\ 0 = 0$   
 $\text{fadd}\ 0\ 0 = 0$   
 $0 \leq C$   
 $A \leq B$   
**shows**  $\text{mult-matrix}\ \text{fmul}\ \text{fadd}\ A\ C \leq \text{mult-matrix}\ \text{fmul}\ \text{fadd}\ B\ C$   
 $\langle \text{proof} \rangle$

**lemma** *spec2*:  $\forall j\ i. P\ j\ i \Longrightarrow P\ j\ i$   $\langle \text{proof} \rangle$

**lemma** *singleton-matrix-le[simp]*:  $(\text{singleton-matrix}\ j\ i\ a \leq \text{singleton-matrix}\ j\ i\ b)$   
 $= (a \leq (b:::\text{order}))$   
 $\langle \text{proof} \rangle$

**lemma** *singleton-le-zero[simp]*:  $(\text{singleton-matrix}\ j\ i\ x \leq 0) = (x \leq (0::'a::\{\text{order}, \text{zero}\}))$   
 $\langle \text{proof} \rangle$

**lemma** *singleton-ge-zero[simp]*:  $(0 \leq \text{singleton-matrix}\ j\ i\ x) = ((0::'a::\{\text{order}, \text{zero}\}) \leq x)$   
 $\langle \text{proof} \rangle$

**lemma** *move-matrix-le-zero[simp]*:  
**fixes**  $A::'a::\{\text{order}, \text{zero}\}\ \text{matrix}$   
**assumes**  $0 \leq j\ 0 \leq i$   
**shows**  $(\text{move-matrix}\ A\ j\ i \leq 0) = (A \leq 0)$   
 $\langle \text{proof} \rangle$

**lemma** *move-matrix-zero-le[simp]*:



**fixes**  $A :: 'a :: \{\text{order}, \text{zero}\} \text{ matrix}$   
**assumes**  $0 \leq j \ 0 \leq i$   
**shows**  $(0 \leq \text{move-matrix } A \ j \ i) = (0 \leq A)$   
 $\langle \text{proof} \rangle$

**lemma** *move-matrix-le-move-matrix-iff[simp]*:  
**fixes**  $A :: 'a :: \{\text{order}, \text{zero}\} \text{ matrix}$   
**assumes**  $0 \leq j \ 0 \leq i$   
**shows**  $(\text{move-matrix } A \ j \ i \leq \text{move-matrix } B \ j \ i) = (A \leq B)$   
 $\langle \text{proof} \rangle$

**instantiation**  $\text{matrix} :: (\{\text{lattice}, \text{zero}\}) \text{ lattice}$   
**begin**

**definition**  $\text{inf} = \text{combine-matrix inf}$

**definition**  $\text{sup} = \text{combine-matrix sup}$

**instance**  
 $\langle \text{proof} \rangle$

**end**

**instantiation**  $\text{matrix} :: (\{\text{plus}, \text{zero}\}) \text{ plus}$   
**begin**

**definition**  
 $\text{plus-matrix-def: } A + B = \text{combine-matrix } (+) \ A \ B$

**instance**  $\langle \text{proof} \rangle$

**end**

**instantiation**  $\text{matrix} :: (\{\text{uminus}, \text{zero}\}) \text{ uminus}$   
**begin**

**definition**  
 $\text{minus-matrix-def: } - \ A = \text{apply-matrix uminus } A$

**instance**  $\langle \text{proof} \rangle$

**end**

**instantiation**  $\text{matrix} :: (\{\text{minus}, \text{zero}\}) \text{ minus}$   
**begin**

**definition**  
 $\text{diff-matrix-def: } A - B = \text{combine-matrix } (-) \ A \ B$

```

instance  $\langle proof \rangle$ 

end

instantiation matrix :: (plus, times, zero) times
begin

definition
  times-matrix-def:  $A * B = \text{mult-matrix } ((*) \text{ } (+) \text{ } A \text{ } B$ 

instance  $\langle proof \rangle$ 

end

instantiation matrix :: (lattice, uminus, zero) abs
begin

definition
  abs-matrix-def:  $|A :: 'a \text{ matrix}| = \text{sup } A \text{ } (- \text{ } A)$ 

instance  $\langle proof \rangle$ 

end

instance matrix :: (monoid-add) monoid-add
 $\langle proof \rangle$ 

instance matrix :: (comm-monoid-add) comm-monoid-add
 $\langle proof \rangle$ 

instance matrix :: (group-add) group-add
 $\langle proof \rangle$ 

instance matrix :: (ab-group-add) ab-group-add
 $\langle proof \rangle$ 

instance matrix :: (ordered-ab-group-add) ordered-ab-group-add
 $\langle proof \rangle$ 

instance matrix :: (lattice-ab-group-add) semilattice-inf-ab-group-add  $\langle proof \rangle$ 
instance matrix :: (lattice-ab-group-add) semilattice-sup-ab-group-add  $\langle proof \rangle$ 

instance matrix :: (semiring-0) semiring-0
 $\langle proof \rangle$ 

instance matrix :: (ring) ring  $\langle proof \rangle$ 

instance matrix :: (ordered-ring) ordered-ring
 $\langle proof \rangle$ 

```

**instance** *matrix* :: (*lattice-ring*) *lattice-ring*

*<proof>*

**instance** *matrix* :: (*lattice-ab-group-add-abs*) *lattice-ab-group-add-abs*

*<proof>*

**lemma** *Rep-matrix-add[simp]*:

*Rep-matrix* ((*a*::('a::monoid-add)*matrix*)+*b*) *j i* = (*Rep-matrix a j i*) + (*Rep-matrix b j i*)

*<proof>*

**lemma** *Rep-matrix-mult*: *Rep-matrix* ((*a*::('a::semiring-0) *matrix*) \* *b*) *j i* =

*foldseq* (+) ( $\lambda k.$  (*Rep-matrix a j k*) \* (*Rep-matrix b k i*)) (*max* (*ncols a*) (*nrows b*))

*<proof>*

**lemma** *apply-matrix-add*:  $\forall x y. f (x+y) = (f x) + (f y) \implies f 0 = (0::'a)$

$\implies \text{apply-matrix } f ((a::('a::monoid-add) \text{ matrix}) + b) = (\text{apply-matrix } f a) + (\text{apply-matrix } f b)$

*<proof>*

**lemma** *singleton-matrix-add*: *singleton-matrix j i* ((*a*::monoid-add)+*b*) = (*singleton-matrix j i a*) + (*singleton-matrix j i b*)

*<proof>*

**lemma** *nrows-mult*: *nrows* ((*A*::('a::semiring-0) *matrix*) \* *B*)  $\leq$  *nrows A*

*<proof>*

**lemma** *ncols-mult*: *ncols* ((*A*::('a::semiring-0) *matrix*) \* *B*)  $\leq$  *ncols B*

*<proof>*

**definition**

*one-matrix* :: *nat*  $\Rightarrow$  ('a::{zero,one}) *matrix* **where**

*one-matrix n* = *Abs-matrix* ( $\lambda j i. \text{if } j = i \wedge j < n \text{ then } 1 \text{ else } 0$ )

**lemma** *Rep-one-matrix[simp]*: *Rep-matrix* (*one-matrix n*) *j i* = (*if* (*j* = *i*  $\wedge$  *j* < *n*) *then 1 else 0*)

*<proof>*

**lemma** *nrows-one-matrix[simp]*: *nrows* ((*one-matrix n*) :: ('a::zero-neq-one)*matrix*) = *n* (**is** ?*r* = -)

*<proof>*

**lemma** *ncols-one-matrix[simp]*: *ncols* ((*one-matrix n*) :: ('a::zero-neq-one)*matrix*) = *n* (**is** ?*r* = -)

*<proof>*

**lemma** *one-matrix-mult-right[simp]*:

**fixes**  $A :: ('a::\text{semiring-1}) \text{ matrix}$   
**shows**  $\text{ncols } A \leq n \implies A * (\text{one-matrix } n) = A$   
 $\langle \text{proof} \rangle$

**lemma** *one-matrix-mult-left*[simp]:  
**fixes**  $A :: ('a::\text{semiring-1}) \text{ matrix}$   
**shows**  $\text{nrows } A \leq n \implies (\text{one-matrix } n) * A = A$   
 $\langle \text{proof} \rangle$

**lemma** *transpose-matrix-mult*:  
**fixes**  $A :: ('a::\text{comm-ring}) \text{ matrix}$   
**shows**  $\text{transpose-matrix } (A*B) = (\text{transpose-matrix } B) * (\text{transpose-matrix } A)$   
 $\langle \text{proof} \rangle$

**lemma** *transpose-matrix-add*:  
**fixes**  $A :: ('a::\text{monoid-add}) \text{ matrix}$   
**shows**  $\text{transpose-matrix } (A+B) = \text{transpose-matrix } A + \text{transpose-matrix } B$   
 $\langle \text{proof} \rangle$

**lemma** *transpose-matrix-diff*:  
**fixes**  $A :: ('a::\text{group-add}) \text{ matrix}$   
**shows**  $\text{transpose-matrix } (A-B) = \text{transpose-matrix } A - \text{transpose-matrix } B$   
 $\langle \text{proof} \rangle$

**lemma** *transpose-matrix-minus*:  
**fixes**  $A :: ('a::\text{group-add}) \text{ matrix}$   
**shows**  $\text{transpose-matrix } (-A) = - \text{transpose-matrix } (A::'a \text{ matrix})$   
 $\langle \text{proof} \rangle$

**definition** *right-inverse-matrix* ::  $('a::\{\text{ring-1}\}) \text{ matrix} \Rightarrow 'a \text{ matrix} \Rightarrow \text{bool}$  **where**  
 $\text{right-inverse-matrix } A \ X == (A * X = \text{one-matrix } (\max (\text{nrows } A) (\text{ncols } X)))$   
 $\wedge \text{nrows } X \leq \text{ncols } A$

**definition** *left-inverse-matrix* ::  $('a::\{\text{ring-1}\}) \text{ matrix} \Rightarrow 'a \text{ matrix} \Rightarrow \text{bool}$  **where**  
 $\text{left-inverse-matrix } A \ X == (X * A = \text{one-matrix } (\max (\text{nrows } X) (\text{ncols } A))) \wedge$   
 $\text{ncols } X \leq \text{nrows } A$

**definition** *inverse-matrix* ::  $('a::\{\text{ring-1}\}) \text{ matrix} \Rightarrow 'a \text{ matrix} \Rightarrow \text{bool}$  **where**  
 $\text{inverse-matrix } A \ X == (\text{right-inverse-matrix } A \ X) \wedge (\text{left-inverse-matrix } A \ X)$

**lemma** *right-inverse-matrix-dim*:  $\text{right-inverse-matrix } A \ X \implies \text{nrows } A = \text{ncols } X$   
 $\langle \text{proof} \rangle$

**lemma** *left-inverse-matrix-dim*:  $\text{left-inverse-matrix } A \ Y \implies \text{ncols } A = \text{nrows } Y$   
 $\langle \text{proof} \rangle$

**lemma** *left-right-inverse-matrix-unique*:  
**assumes**  $\text{left-inverse-matrix } A \ Y \ \text{right-inverse-matrix } A \ X$

**shows**  $X = Y$   
 $\langle \text{proof} \rangle$

**lemma** *inverse-matrix-inject*:  $\llbracket \text{inverse-matrix } A \ X; \text{inverse-matrix } A \ Y \rrbracket \implies X = Y$   
 $\langle \text{proof} \rangle$

**lemma** *one-matrix-inverse*:  $\text{inverse-matrix } (\text{one-matrix } n) \ (\text{one-matrix } n)$   
 $\langle \text{proof} \rangle$

**lemma** *zero-imp-mult-zero*:  $(a :: 'a :: \text{semiring-0}) = 0 \mid b = 0 \implies a * b = 0$   
 $\langle \text{proof} \rangle$

**lemma** *Rep-matrix-zero-imp-mult-zero*:  
 $\forall j \ i \ k. (\text{Rep-matrix } A \ j \ k = 0) \mid (\text{Rep-matrix } B \ k \ i) = 0 \implies A * B = (0 :: ('a :: \text{lattice-ring}) \text{ matrix})$   
 $\langle \text{proof} \rangle$

**lemma** *add-nrows*:  $\text{nrows } (A :: ('a :: \text{monoid-add}) \text{ matrix}) \leq u \implies \text{nrows } B \leq u \implies \text{nrows } (A + B) \leq u$   
 $\langle \text{proof} \rangle$

**lemma** *move-matrix-row-mult*:  
**fixes**  $A :: ('a :: \text{semiring-0}) \text{ matrix}$   
**shows**  $\text{move-matrix } (A * B) \ j \ 0 = (\text{move-matrix } A \ j \ 0) * B$   
 $\langle \text{proof} \rangle$

**lemma** *move-matrix-col-mult*:  
**fixes**  $A :: ('a :: \text{semiring-0}) \text{ matrix}$   
**shows**  $\text{move-matrix } (A * B) \ 0 \ i = A * (\text{move-matrix } B \ 0 \ i)$   
 $\langle \text{proof} \rangle$

**lemma** *move-matrix-add*:  $((\text{move-matrix } (A + B) \ j \ i) :: ('a :: \text{monoid-add}) \text{ matrix}) = (\text{move-matrix } A \ j \ i) + (\text{move-matrix } B \ j \ i)$   
 $\langle \text{proof} \rangle$

**lemma** *move-matrix-mult*:  $\text{move-matrix } ((A :: ('a :: \text{semiring-0}) \text{ matrix}) * B) \ j \ i = (\text{move-matrix } A \ j \ 0) * (\text{move-matrix } B \ 0 \ i)$   
 $\langle \text{proof} \rangle$

**definition** *scalar-mult* ::  $('a :: \text{ring}) \Rightarrow 'a \text{ matrix} \Rightarrow 'a \text{ matrix}$  **where**  
 $\text{scalar-mult } a \ m == \text{apply-matrix } ((* ) \ a) \ m$

**lemma** *scalar-mult-zero[simp]*:  $\text{scalar-mult } y \ 0 = 0$   
 $\langle \text{proof} \rangle$

**lemma** *scalar-mult-add*:  $\text{scalar-mult } y \ (a+b) = (\text{scalar-mult } y \ a) + (\text{scalar-mult } y \ b)$   
 $\langle \text{proof} \rangle$

**lemma** *Rep-scalar-mult[simp]*: *Rep-matrix* (*scalar-mult* *y a*) *j i* = *y* \* (*Rep-matrix* *a j i*)  
 ⟨*proof*⟩

**lemma** *scalar-mult-singleton[simp]*: *scalar-mult* *y* (*singleton-matrix* *j i x*) = *singleton-matrix* *j i* (*y* \* *x*)  
 ⟨*proof*⟩

**lemma** *Rep-minus[simp]*: *Rep-matrix* (-(*A:::group-add*)) *x y* = - (*Rep-matrix* *A x y*)  
 ⟨*proof*⟩

**lemma** *Rep-abs[simp]*: *Rep-matrix* |*A:::lattice-ab-group-add*| *x y* = |*Rep-matrix* *A x y*|  
 ⟨*proof*⟩

**end**

**theory** *SparseMatrix*

**imports** *Matrix*

**begin**

**type-synonym** '*a* *spvec* = (*nat* \* '*a*) *list*

**type-synonym** '*a* *spmat* = '*a* *spvec* *spvec*

**definition** *sparse-row-vector* :: ('*a*::*ab-group-add*) *spvec* ⇒ '*a* *matrix*  
**where** *sparse-row-vector* *arr* = *foldl* (% *m x. m* + (*singleton-matrix* 0 (*fst x*) (*snd x*))) 0 *arr*

**definition** *sparse-row-matrix* :: ('*a*::*ab-group-add*) *spmat* ⇒ '*a* *matrix*  
**where** *sparse-row-matrix* *arr* = *foldl* (% *m r. m* + (*move-matrix* (*sparse-row-vector* (*snd r*)) (*int* (*fst r*)) 0)) 0 *arr*

**code-datatype** *sparse-row-vector* *sparse-row-matrix*

**lemma** *sparse-row-vector-empty [simp]*: *sparse-row-vector* [] = 0  
 ⟨*proof*⟩

**lemma** *sparse-row-matrix-empty [simp]*: *sparse-row-matrix* [] = 0  
 ⟨*proof*⟩

**lemma** [*code*]:  
 ⟨0 = *sparse-row-vector* []⟩  
 ⟨*proof*⟩

**lemma** *foldl-distrstart*: ∀ *a x y. (f (g x y) a = g x (f y a)) ⇒ (foldl f (g x y) l = g x (foldl f y l))*

$\langle \text{proof} \rangle$

**lemma** *sparse-row-vector-cons*[simp]:

*sparse-row-vector* (*a* # *arr*) = (*singleton-matrix* 0 (*fst* *a*) (*snd* *a*)) + (*sparse-row-vector* *arr*)

$\langle \text{proof} \rangle$

**lemma** *sparse-row-vector-append*[simp]:

*sparse-row-vector* (*a* @ *b*) = (*sparse-row-vector* *a*) + (*sparse-row-vector* *b*)

$\langle \text{proof} \rangle$

**lemma** *nrows-spvec*[simp]: *nrows* (*sparse-row-vector* *x*) ≤ (*Suc* 0)

$\langle \text{proof} \rangle$

**lemma** *sparse-row-matrix-cons*: *sparse-row-matrix* (*a* # *arr*) = ((*move-matrix* (*sparse-row-vector* (*snd* *a*)) (*int* (*fst* *a*)) 0)) + *sparse-row-matrix* *arr*

$\langle \text{proof} \rangle$

**lemma** *sparse-row-matrix-append*: *sparse-row-matrix* (*arr* @ *brr*) = (*sparse-row-matrix* *arr*) + (*sparse-row-matrix* *brr*)

$\langle \text{proof} \rangle$

**fun** *sorted-spvec* :: 'a *spvec* ⇒ *bool*

**where**

*sorted-spvec* [] = *True*

| *sorted-spvec-step1*: *sorted-spvec* [*a*] = *True*

| *sorted-spvec-step*: *sorted-spvec* ((*m,x*)#(*n,y*)#*bs*) = ((*m* < *n*) ∧ (*sorted-spvec* ((*n,y*)#*bs*)))

**primrec** *sorted-spmat* :: 'a *spmat* ⇒ *bool*

**where**

*sorted-spmat* [] = *True*

| *sorted-spmat* (*a* # *as*) = ((*sorted-spvec* (*snd* *a*)) ∧ (*sorted-spmat* *as*))

**declare** *sorted-spvec.simps* [simp del]

**lemma** *sorted-spvec-empty*[simp]: *sorted-spvec* [] = *True*

$\langle \text{proof} \rangle$

**lemma** *sorted-spvec-cons1*: *sorted-spvec* (*a* # *as*) ⇒ *sorted-spvec* *as*

$\langle \text{proof} \rangle$

**lemma** *sorted-spvec-cons2*: *sorted-spvec* (*a* # *b* # *t*) ⇒ *sorted-spvec* (*a* # *t*)

$\langle \text{proof} \rangle$

**lemma** *sorted-spvec-cons3*: *sorted-spvec*(*a* # *b* # *t*) ⇒ *fst* *a* < *fst* *b*

$\langle \text{proof} \rangle$

**lemma** *sorted-sparse-row-vector-zero*:

**assumes**  $m \leq n$   
**shows**  $\text{sorted-spvec } ((n,a)\#arr) \implies \text{Rep-matrix } (\text{sparse-row-vector } arr) \ j \ m = 0$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{sorted-sparse-row-matrix-zero}[\text{rule-format}]$ :  
**assumes**  $m \leq n$   
**shows**  $\text{sorted-spvec } ((n,a)\#arr) \implies \text{Rep-matrix } (\text{sparse-row-matrix } arr) \ m \ j = 0$   
 $\langle \text{proof} \rangle$

**primrec**  $\text{minus-spvec} :: ('a::\text{ab-group-add}) \text{ spvec} \Rightarrow 'a \text{ spvec}$   
**where**  
 $\text{minus-spvec } [] = []$   
 $|\ \text{minus-spvec } (a\#as) = (\text{fst } a, -(\text{snd } a))\#(\text{minus-spvec } as)$

**primrec**  $\text{abs-spvec} :: ('a::\text{lattice-ab-group-add-abs}) \text{ spvec} \Rightarrow 'a \text{ spvec}$   
**where**  
 $\text{abs-spvec } [] = []$   
 $|\ \text{abs-spvec } (a\#as) = (\text{fst } a, |\text{snd } a|)\#(\text{abs-spvec } as)$

**lemma**  $\text{sparse-row-vector-minus}$ :  
 $\text{sparse-row-vector } (\text{minus-spvec } v) = - (\text{sparse-row-vector } v)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{sparse-row-vector-abs}$ :  
 $\text{sorted-spvec } (v :: 'a::\text{lattice-ring spvec}) \implies \text{sparse-row-vector } (\text{abs-spvec } v) = |\text{sparse-row-vector } v|$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{sorted-spvec-minus-spvec}$ :  
 $\text{sorted-spvec } v \implies \text{sorted-spvec } (\text{minus-spvec } v)$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{sorted-spvec-abs-spvec}$ :  
 $\text{sorted-spvec } v \implies \text{sorted-spvec } (\text{abs-spvec } v)$   
 $\langle \text{proof} \rangle$

**definition**  $\text{smult-spvec } y = \text{map } (\% a. (\text{fst } a, y * \text{snd } a))$

**lemma**  $\text{smult-spvec-empty}[\text{simp}]$ :  $\text{smult-spvec } y \ [] = []$   
 $\langle \text{proof} \rangle$

**lemma**  $\text{smult-spvec-cons}$ :  $\text{smult-spvec } y \ (a\#arr) = (\text{fst } a, y * (\text{snd } a)) \# (\text{smult-spvec } y \ arr)$   
 $\langle \text{proof} \rangle$

**fun**  $\text{addmult-spvec} :: ('a::\text{ring}) \Rightarrow 'a \text{ spvec} \Rightarrow 'a \text{ spvec} \Rightarrow 'a \text{ spvec}$   
**where**



$\text{addmult-spvec } y \text{ arr } [] = \text{arr}$   
 $| \text{addmult-spvec } y \text{ [] brr} = \text{smult-spvec } y \text{ brr}$   
 $| \text{addmult-spvec } y ((i,a)\# \text{arr}) ((j,b)\# \text{brr}) = ($   
 $\quad \text{if } i < j \text{ then } ((i,a)\#(\text{addmult-spvec } y \text{ arr } ((j,b)\# \text{brr})))$   
 $\quad \text{else (if } (j < i) \text{ then } ((j, y * b)\#(\text{addmult-spvec } y ((i,a)\# \text{arr}) \text{ brr}))$   
 $\quad \text{else } ((i, a + y*b)\#(\text{addmult-spvec } y \text{ arr brr})))$

**lemma** *addmult-spvec-empty1*[simp]:  $\text{addmult-spvec } y \text{ [] } a = \text{smult-spvec } y \text{ } a$   
 $\langle \text{proof} \rangle$

**lemma** *addmult-spvec-empty2*[simp]:  $\text{addmult-spvec } y \text{ } a \text{ []} = a$   
 $\langle \text{proof} \rangle$

**lemma** *sparse-row-vector-map*:  $(\forall x y. f (x+y) = (f x) + (f y)) \implies (f :: 'a \Rightarrow ('a :: \text{lattice-ring}))$   
 $0 = 0 \implies$   
 $\text{sparse-row-vector } (\text{map } (\% x. (fst x, f (snd x))) a) = \text{apply-matrix } f (\text{sparse-row-vector } a)$   
 $\langle \text{proof} \rangle$

**lemma** *sparse-row-vector-smult*:  $\text{sparse-row-vector } (\text{smult-spvec } y \text{ } a) = \text{scalar-mult } y (\text{sparse-row-vector } a)$   
 $\langle \text{proof} \rangle$

**lemma** *sparse-row-vector-addmult-spvec*:  $\text{sparse-row-vector } (\text{addmult-spvec } (y :: 'a :: \text{lattice-ring}) a \text{ } b) =$   
 $(\text{sparse-row-vector } a) + (\text{scalar-mult } y (\text{sparse-row-vector } b))$   
 $\langle \text{proof} \rangle$

**lemma** *sorted-smult-spvec*:  $\text{sorted-spvec } a \implies \text{sorted-spvec } (\text{smult-spvec } y \text{ } a)$   
 $\langle \text{proof} \rangle$

**lemma** *sorted-spvec-addmult-spvec-helper*:  $\llbracket \text{sorted-spvec } (\text{addmult-spvec } y ((a, b) \# \text{arr}) \text{ brr}); aa < a; \text{sorted-spvec } ((a, b) \# \text{arr});$   
 $\text{sorted-spvec } ((aa, ba) \# \text{brr}) \rrbracket \implies \text{sorted-spvec } ((aa, y * ba) \# \text{addmult-spvec } y ((a, b) \# \text{arr}) \text{ brr})$   
 $\langle \text{proof} \rangle$

**lemma** *sorted-spvec-addmult-spvec-helper2*:  
 $\llbracket \text{sorted-spvec } (\text{addmult-spvec } y \text{ arr } ((aa, ba) \# \text{brr})); a < aa; \text{sorted-spvec } ((a, b) \# \text{arr}); \text{sorted-spvec } ((aa, ba) \# \text{brr}) \rrbracket$   
 $\implies \text{sorted-spvec } ((a, b) \# \text{addmult-spvec } y \text{ arr } ((aa, ba) \# \text{brr}))$   
 $\langle \text{proof} \rangle$

**lemma** *sorted-spvec-addmult-spvec-helper3*[rule-format]:  
 $\text{sorted-spvec } (\text{addmult-spvec } y \text{ arr brr}) \implies$   
 $\text{sorted-spvec } ((aa, b) \# \text{arr}) \implies$   
 $\text{sorted-spvec } ((aa, ba) \# \text{brr}) \implies$   
 $\text{sorted-spvec } ((aa, b + y * ba) \# (\text{addmult-spvec } y \text{ arr brr}))$

*<proof>*

**lemma** *sorted-addmult-spvec*: *sorted-spvec a  $\implies$  sorted-spvec b  $\implies$  sorted-spvec (addmult-spvec y a b)*  
*<proof>*

**fun** *mult-spvec-spmat* :: ('a::lattice-ring) spvec  $\Rightarrow$  'a spvec  $\Rightarrow$  'a spmat  $\Rightarrow$  'a spvec  
**where**  
*mult-spvec-spmat c [] brr = c*  
*| mult-spvec-spmat c arr [] = c*  
*| mult-spvec-spmat c ((i,a)#arr) ((j,b)#brr) = (*  
*if (i < j) then mult-spvec-spmat c arr ((j,b)#brr)*  
*else if (j < i) then mult-spvec-spmat c ((i,a)#arr) brr*  
*else mult-spvec-spmat (addmult-spvec a c b) arr brr)*

**lemma** *sparse-row-mult-spvec-spmat*:  
**assumes** *sorted-spvec (a::('a::lattice-ring) spvec) sorted-spvec B*  
**shows** *sparse-row-vector (mult-spvec-spmat c a B) = (sparse-row-vector c) + (sparse-row-vector a) \* (sparse-row-matrix B)*  
*<proof>*

**lemma** *sorted-mult-spvec-spmat*:  
*sorted-spvec (c::('a::lattice-ring) spvec)  $\implies$  sorted-spmat B  $\implies$  sorted-spvec (mult-spvec-spmat c a B)*  
*<proof>*

**primrec** *mult-spmat* :: ('a::lattice-ring) spmat  $\Rightarrow$  'a spmat  $\Rightarrow$  'a spmat  
**where**  
*mult-spmat [] A = []*  
*| mult-spmat (a#as) A = (fst a, mult-spvec-spmat [] (snd a) A)#(mult-spmat as A)*

**lemma** *sparse-row-mult-spmat*:  
*sorted-spmat A  $\implies$  sorted-spvec B  $\implies$*   
*sparse-row-matrix (mult-spmat A B) = (sparse-row-matrix A) \* (sparse-row-matrix B)*  
*<proof>*

**lemma** *sorted-spvec-mult-spmat*:  
**fixes** *A :: ('a::lattice-ring) spmat*  
**shows** *sorted-spvec A  $\implies$  sorted-spvec (mult-spmat A B)*  
*<proof>*

**lemma** *sorted-spmat-mult-spmat*:  
*sorted-spmat (B::('a::lattice-ring) spmat)  $\implies$  sorted-spmat (mult-spmat A B)*  
*<proof>*

**fun** *add-spvec* :: ('a::lattice-ab-group-add) spvec  $\Rightarrow$  'a spvec  $\Rightarrow$  'a spvec

where

```

    add-spvec arr [] = arr
| add-spvec [] brr = brr
| add-spvec ((i,a)#arr) ((j,b)#brr) = (
    if i < j then (i,a)#(add-spvec arr ((j,b)#brr))
    else if (j < i) then (j,b) # add-spvec ((i,a)#arr) brr
    else (i, a+b) # add-spvec arr brr)

```

**lemma** *add-spvec-empty1*[simp]:  $\text{add-spvec } [] \ a = a$   
 ⟨proof⟩

**lemma** *sparse-row-vector-add*:  $\text{sparse-row-vector } (\text{add-spvec } a \ b) = (\text{sparse-row-vector } a) + (\text{sparse-row-vector } b)$   
 ⟨proof⟩

**fun** *add-spmat* :: ('a::lattice-ab-group-add) *spmat*  $\Rightarrow$  'a *spmat*  $\Rightarrow$  'a *spmat*  
 where

```

    add-spmat [] bs = bs
| add-spmat as [] = as
| add-spmat ((i,a)#as) ((j,b)#bs) = (
    if i < j then
      (i,a) # add-spmat as ((j,b)#bs)
    else if j < i then
      (j,b) # add-spmat ((i,a)#as) bs
    else
      (i, add-spvec a b) # add-spmat as bs)

```

**lemma** *add-spmat-Nil2*[simp]:  $\text{add-spmat } as \ [] = as$   
 ⟨proof⟩

**lemma** *sparse-row-add-spmat*:  $\text{sparse-row-matrix } (\text{add-spmat } A \ B) = (\text{sparse-row-matrix } A) + (\text{sparse-row-matrix } B)$   
 ⟨proof⟩

**lemma** [code]:  
 ⟨ $\text{sparse-row-matrix } A + \text{sparse-row-matrix } B = \text{sparse-row-matrix } (\text{add-spmat } A \ B)$ ⟩  
 ⟨ $\text{sparse-row-vector } a + \text{sparse-row-vector } b = \text{sparse-row-vector } (\text{add-spvec } a \ b)$ ⟩  
 ⟨proof⟩

**lemma** *sorted-add-spvec-helper1*[rule-format]:  $\text{add-spvec } ((a,b)\#arr) \ brr = (ab, bb) \# list \longrightarrow (ab = a \mid (brr \neq [] \ \& \ ab = fst \ (hd \ brr)))$   
 ⟨proof⟩

**lemma** *sorted-add-spmat-helper1*[rule-format]:  
 $\text{add-spmat } ((a,b)\#arr) \ brr = (ab, bb) \# list \Longrightarrow (ab = a \mid (brr \neq [] \ \& \ ab = fst \ (hd \ brr)))$

*<proof>*

**lemma** *sorted-add-svvec-helper*:  $\text{add-svvec } arr \ brr = (ab, bb) \# list \implies ((arr \neq [] \ \& \ ab = \text{fst } (hd \ arr)) \mid (brr \neq [] \ \& \ ab = \text{fst } (hd \ brr)))$   
*<proof>*

**lemma** *sorted-add-spmat-helper*:  $\text{add-spmat } arr \ brr = (ab, bb) \# list \implies ((arr \neq [] \ \& \ ab = \text{fst } (hd \ arr)) \mid (brr \neq [] \ \& \ ab = \text{fst } (hd \ brr)))$   
*<proof>*

**lemma** *add-svvec-commute*:  $\text{add-svvec } a \ b = \text{add-svvec } b \ a$   
*<proof>*

**lemma** *add-spmat-commute*:  $\text{add-spmat } a \ b = \text{add-spmat } b \ a$   
*<proof>*

**lemma** *sorted-add-svvec-helper2*:  $\text{add-svvec } ((a,b)\#arr) \ brr = (ab, bb) \# list \implies aa < a \implies \text{sorted-svvec } ((aa, ba) \# brr) \implies aa < ab$   
*<proof>*

**lemma** *sorted-add-spmat-helper2*:  $\text{add-spmat } ((a,b)\#arr) \ brr = (ab, bb) \# list \implies aa < a \implies \text{sorted-svvec } ((aa, ba) \# brr) \implies aa < ab$   
*<proof>*

**lemma** *sorted-svvec-add-svvec*:  $\text{sorted-svvec } a \implies \text{sorted-svvec } b \implies \text{sorted-svvec } (\text{add-svvec } a \ b)$   
*<proof>*

**lemma** *sorted-svvec-add-spmat*:  
 $\text{sorted-svvec } A \implies \text{sorted-svvec } B \implies \text{sorted-svvec } (\text{add-spmat } A \ B)$   
*<proof>*

**lemma** *sorted-spmat-add-spmat[rule-format]*:  $\text{sorted-spmat } A \implies \text{sorted-spmat } B \implies \text{sorted-spmat } (\text{add-spmat } A \ B)$   
*<proof>*

**fun** *le-svvec* :: (*'a*::*lattice-ab-group-add*) *svvec*  $\Rightarrow$  *'a svvec*  $\Rightarrow$  *bool*  
**where**

$\text{le-svvec } [] \ [] = \text{True}$   
 $\mid \text{le-svvec } ((-,a)\#as) \ [] = (a \leq 0 \ \& \ \text{le-svvec } as \ [])$   
 $\mid \text{le-svvec } [] \ ((-,b)\#bs) = (0 \leq b \ \& \ \text{le-svvec } [] \ bs)$   
 $\mid \text{le-svvec } ((i,a)\#as) \ ((j,b)\#bs) = ($   
 $\quad \text{if } (i < j) \text{ then } a \leq 0 \ \& \ \text{le-svvec } as \ ((j,b)\#bs)$   
 $\quad \text{else if } (j < i) \text{ then } 0 \leq b \ \& \ \text{le-svvec } ((i,a)\#as) \ bs$   
 $\quad \text{else } a \leq b \ \& \ \text{le-svvec } as \ bs)$

**fun** *le-spmat* :: (*'a*::*lattice-ab-group-add*) *spmat*  $\Rightarrow$  *'a spat*  $\Rightarrow$  *bool*  
**where**

```

le-spmat [] [] = True
| le-spmat ((i,a)#as) [] = (le-spvec a [] & le-spmat as [])
| le-spmat [] ((j,b)#bs) = (le-spvec [] b & le-spmat [] bs)
| le-spmat ((i,a)#as) ((j,b)#bs) = (
  if i < j then (le-spvec a [] & le-spmat as ((j,b)#bs))
  else if j < i then (le-spvec [] b & le-spmat ((i,a)#as) bs)
  else (le-spvec a b & le-spmat as bs))

```

**definition** *disj-matrices* :: ('a::zero) matrix  $\Rightarrow$  'a matrix  $\Rightarrow$  bool **where**  
*disj-matrices* A B  $\longleftrightarrow$   
 $(\forall j\ i. (\text{Rep-matrix } A\ j\ i \neq 0) \longrightarrow (\text{Rep-matrix } B\ j\ i = 0)) \ \& \ (\forall j\ i. (\text{Rep-matrix } B\ j\ i \neq 0) \longrightarrow (\text{Rep-matrix } A\ j\ i = 0))$

**lemma** *disj-matrices-contr1*: *disj-matrices* A B  $\implies$  Rep-matrix A j i  $\neq$  0  $\implies$  Rep-matrix B j i = 0  
 <proof>

**lemma** *disj-matrices-contr2*: *disj-matrices* A B  $\implies$  Rep-matrix B j i  $\neq$  0  $\implies$  Rep-matrix A j i = 0  
 <proof>

**lemma** *disj-matrices-add*:  
**fixes** A :: ('a::lattice-ab-group-add) matrix  
**shows** *disj-matrices* A B  $\implies$  *disj-matrices* C D  $\implies$  *disj-matrices* A D  
 $\implies$  *disj-matrices* B C  $\implies$  (A + B  $\leq$  C + D) = (A  $\leq$  C  $\wedge$  B  $\leq$  D)  
 <proof>

**lemma** *disj-matrices-zero1[simp]*: *disj-matrices* 0 B  
 <proof>

**lemma** *disj-matrices-zero2[simp]*: *disj-matrices* A 0  
 <proof>

**lemma** *disj-matrices-commute*: *disj-matrices* A B = *disj-matrices* B A  
 <proof>

**lemma** *disj-matrices-add-le-zero*: *disj-matrices* A B  $\implies$   
 (A + B  $\leq$  0) = (A  $\leq$  0 & (B::('a::lattice-ab-group-add) matrix)  $\leq$  0)  
 <proof>

**lemma** *disj-matrices-add-zero-le*: *disj-matrices* A B  $\implies$   
 (0  $\leq$  A + B) = (0  $\leq$  A & 0  $\leq$  (B::('a::lattice-ab-group-add) matrix))  
 <proof>

**lemma** *disj-matrices-add-x-le*: *disj-matrices* A B  $\implies$  *disj-matrices* B C  $\implies$   
 (A  $\leq$  B + C) = (A  $\leq$  C & 0  $\leq$  (B::('a::lattice-ab-group-add) matrix))  
 <proof>

**lemma** *disj-matrices-add-le-x*:  $\text{disj-matrices } A \ B \implies \text{disj-matrices } B \ C \implies$   
 $(B + A \leq C) = (A \leq C \ \& \ (B::('a::\text{lattice-ab-group-add}) \text{ matrix}) \leq 0)$   
 $\langle \text{proof} \rangle$

**lemma** *disj-sparse-row-singleton*:  $i \leq j \implies \text{sorted-spvec}((j,y)\#v) \implies \text{disj-matrices}$   
 $(\text{sparse-row-vector } v) \ (\text{singleton-matrix } 0 \ i \ x)$   
 $\langle \text{proof} \rangle$

**lemma** *disj-matrices-x-add*:  $\text{disj-matrices } A \ B \implies \text{disj-matrices } A \ C \implies \text{disj-matrices}$   
 $(A::('a::\text{lattice-ab-group-add}) \text{ matrix}) \ (B+C)$   
 $\langle \text{proof} \rangle$

**lemma** *disj-matrices-add-x*:  $\text{disj-matrices } A \ B \implies \text{disj-matrices } A \ C \implies \text{disj-matrices}$   
 $(B+C) \ (A::('a::\text{lattice-ab-group-add}) \text{ matrix})$   
 $\langle \text{proof} \rangle$

**lemma** *disj-singleton-matrices[simp]*:  $\text{disj-matrices} \ (\text{singleton-matrix } j \ i \ x) \ (\text{singleton-matrix}$   
 $u \ v \ y) = (j \neq u \mid i \neq v \mid x = 0 \mid y = 0)$   
 $\langle \text{proof} \rangle$

**lemma** *disj-move-sparse-vec-mat*:  
**assumes**  $j \leq a$  **and**  $\text{sorted-spvec} \ ((a, c) \# as)$   
**shows**  $\text{disj-matrices} \ (\text{sparse-row-matrix } as) \ (\text{move-matrix} \ (\text{sparse-row-vector } b)$   
 $(\text{int } j) \ i)$   
 $\langle \text{proof} \rangle$

**lemma** *disj-move-sparse-row-vector-twice*:  
 $j \neq u \implies \text{disj-matrices} \ (\text{move-matrix} \ (\text{sparse-row-vector } a) \ j \ i) \ (\text{move-matrix}$   
 $(\text{sparse-row-vector } b) \ u \ v)$   
 $\langle \text{proof} \rangle$

**lemma** *le-spvec-iff-sparse-row-le*:  
 $\text{sorted-spvec } a \implies \text{sorted-spvec } b \implies (\text{le-spvec } a \ b) \longleftrightarrow (\text{sparse-row-vector } a \leq$   
 $\text{sparse-row-vector } b)$   
 $\langle \text{proof} \rangle$

**lemma** *le-spvec-empty2-sparse-row*:  
 $\text{sorted-spvec } b \implies \text{le-spvec } b \ [] = (\text{sparse-row-vector } b \leq 0)$   
 $\langle \text{proof} \rangle$

**lemma** *le-spvec-empty1-sparse-row*:  
 $(\text{sorted-spvec } b) \implies (\text{le-spvec } [] \ b = (0 \leq \text{sparse-row-vector } b))$   
 $\langle \text{proof} \rangle$

**lemma** *le-spmat-iff-sparse-row-le*:  
 $\llbracket \text{sorted-spvec } A; \text{sorted-spmat } A; \text{sorted-spvec } B; \text{sorted-spmat } B \rrbracket \implies$   
 $\text{le-spmat } A \ B = (\text{sparse-row-matrix } A \leq \text{sparse-row-matrix } B)$   
 $\langle \text{proof} \rangle$

**primrec** *abs-spmat* :: ('a::lattice-ring) *spmat*  $\Rightarrow$  'a *spmat*

**where**

*abs-spmat* [] = []  
| *abs-spmat* (a#as) = (fst a, *abs-spvec* (snd a))#(*abs-spmat* as)

**primrec** *minus-spmat* :: ('a::lattice-ring) *spmat*  $\Rightarrow$  'a *spmat*

**where**

*minus-spmat* [] = []  
| *minus-spmat* (a#as) = (fst a, *minus-spvec* (snd a))#(*minus-spmat* as)

**lemma** *sparse-row-matrix-minus*:

*sparse-row-matrix* (*minus-spmat* A) = - (*sparse-row-matrix* A)  
<proof>

**lemma** *Rep-sparse-row-vector-zero*:

**assumes**  $x \neq 0$

**shows** *Rep-matrix* (*sparse-row-vector* v) x y = 0

<proof>

**lemma** *sparse-row-matrix-abs*:

*sorted-spvec* A  $\Longrightarrow$  *sorted-spmat* A  $\Longrightarrow$  *sparse-row-matrix* (*abs-spmat* A) = |*sparse-row-matrix* A|  
<proof>

**lemma** *sorted-spvec-minus-spmat*: *sorted-spvec* A  $\Longrightarrow$  *sorted-spvec* (*minus-spmat* A)

<proof>

**lemma** *sorted-spvec-abs-spmat*: *sorted-spvec* A  $\Longrightarrow$  *sorted-spvec* (*abs-spmat* A)

<proof>

**lemma** *sorted-spmat-minus-spmat*: *sorted-spmat* A  $\Longrightarrow$  *sorted-spmat* (*minus-spmat* A)

<proof>

**lemma** *sorted-spmat-abs-spmat*: *sorted-spmat* A  $\Longrightarrow$  *sorted-spmat* (*abs-spmat* A)

<proof>

**definition** *diff-spmat* :: ('a::lattice-ring) *spmat*  $\Rightarrow$  'a *spmat*  $\Rightarrow$  'a *spmat*

**where** *diff-spmat* A B = *add-spmat* A (*minus-spmat* B)

**lemma** *sorted-spmat-diff-spmat*: *sorted-spmat* A  $\Longrightarrow$  *sorted-spmat* B  $\Longrightarrow$  *sorted-spmat* (*diff-spmat* A B)

<proof>

**lemma** *sorted-spvec-diff-spmat*: *sorted-spvec* A  $\Longrightarrow$  *sorted-spvec* B  $\Longrightarrow$  *sorted-spvec* (*diff-spmat* A B)

*<proof>*

**lemma** *sparse-row-diff-spmat*: *sparse-row-matrix* (*diff-spmat* *A B*) = (*sparse-row-matrix* *A*) - (*sparse-row-matrix* *B*)  
*<proof>*

**definition** *sorted-sparse-matrix* :: 'a *spmat*  $\Rightarrow$  *bool*  
**where** *sorted-sparse-matrix* *A*  $\longleftrightarrow$  *sorted-spvec* *A* & *sorted-spmat* *A*

**lemma** *sorted-sparse-matrix-imp-spvec*: *sorted-sparse-matrix* *A*  $\Longrightarrow$  *sorted-spvec* *A*  
*<proof>*

**lemma** *sorted-sparse-matrix-imp-spmat*: *sorted-sparse-matrix* *A*  $\Longrightarrow$  *sorted-spmat* *A*  
*<proof>*

**lemmas** *sorted-sp-simps* =  
*sorted-spvec.simps*  
*sorted-spmat.simps*  
*sorted-sparse-matrix-def*

**lemma** *bool1*: ( $\neg$  *True*) = *False* *<proof>*  
**lemma** *bool2*: ( $\neg$  *False*) = *True* *<proof>*  
**lemma** *bool3*: ((*P*::*bool*)  $\wedge$  *True*) = *P* *<proof>*  
**lemma** *bool4*: (*True*  $\wedge$  (*P*::*bool*)) = *P* *<proof>*  
**lemma** *bool5*: ((*P*::*bool*)  $\wedge$  *False*) = *False* *<proof>*  
**lemma** *bool6*: (*False*  $\wedge$  (*P*::*bool*)) = *False* *<proof>*  
**lemma** *bool7*: ((*P*::*bool*)  $\vee$  *True*) = *True* *<proof>*  
**lemma** *bool8*: (*True*  $\vee$  (*P*::*bool*)) = *True* *<proof>*  
**lemma** *bool9*: ((*P*::*bool*)  $\vee$  *False*) = *P* *<proof>*  
**lemma** *bool10*: (*False*  $\vee$  (*P*::*bool*)) = *P* *<proof>*  
**lemmas** *boolarith* = *bool1 bool2 bool3 bool4 bool5 bool6 bool7 bool8 bool9 bool10*

**lemma** *if-case-eq*: (if *b* then *x* else *y*) = (case *b* of *True*  $\Rightarrow$  *x* | *False*  $\Rightarrow$  *y*) *<proof>*

**primrec** *pprt-spvec* :: ('a::*{lattice-ab-group-add}*) *spvec*  $\Rightarrow$  'a *spvec*  
**where**  
*pprt-spvec* [] = []  
| *pprt-spvec* (*a*#*as*) = (*fst* *a*, *pprt* (*snd* *a*)) # (*pprt-spvec* *as*)

**primrec** *nprr-spvec* :: ('a::*{lattice-ab-group-add}*) *spvec*  $\Rightarrow$  'a *spvec*  
**where**  
*nprr-spvec* [] = []  
| *nprr-spvec* (*a*#*as*) = (*fst* *a*, *nprr* (*snd* *a*)) # (*nprr-spvec* *as*)

**primrec** *pprt-spmat* :: ('a::*{lattice-ab-group-add}*) *spmat*  $\Rightarrow$  'a *spmat*  
**where**  
*pprt-spmat* [] = []  
| *pprt-spmat* (*a*#*as*) = (*fst* *a*, *pprt-spvec* (*snd* *a*)) # (*pprt-spmat* *as*)



**primrec** *nprrt-spmat* :: ('a::lattice-ab-group-add) *spmat*  $\Rightarrow$  'a *spmat*

**where**

*nprrt-spmat* [] = []  
| *nprrt-spmat* (a#as) = (fst a, *nprrt-spvec* (snd a))#(*nprrt-spmat* as)

**lemma** *pprrt-add*: *disj-matrices* A (B::(-::lattice-ring) *matrix*)  $\Longrightarrow$  *pprrt* (A+B) =  
*pprrt* A + *pprrt* B  
⟨proof⟩

**lemma** *nprrt-add*: *disj-matrices* A (B::(-::lattice-ring) *matrix*)  $\Longrightarrow$  *nprrt* (A+B) =  
*nprrt* A + *nprrt* B  
⟨proof⟩

**lemma** *pprrt-singleton[simp]*:  
**fixes** x:: -::lattice-ring  
**shows** *pprrt* (singleton-matrix j i x) = singleton-matrix j i (*pprrt* x)  
⟨proof⟩

**lemma** *nprrt-singleton[simp]*:  
**fixes** x:: -::lattice-ring  
**shows** *nprrt* (singleton-matrix j i x) = singleton-matrix j i (*nprrt* x)  
⟨proof⟩

**lemma** *sparse-row-vector-pprrt*:  
**fixes** v:: -::lattice-ring *spvec*  
**shows** *sorted-spvec* v  $\Longrightarrow$  *sparse-row-vector* (*pprrt-spvec* v) = *pprrt* (*sparse-row-vector* v)  
⟨proof⟩

**lemma** *sparse-row-vector-nprrt*:  
**fixes** v:: -::lattice-ring *spvec*  
**shows** *sorted-spvec* v  $\Longrightarrow$  *sparse-row-vector* (*nprrt-spvec* v) = *nprrt* (*sparse-row-vector* v)  
⟨proof⟩

**lemma** *pprrt-move-matrix*: *pprrt* (move-matrix (A::('a::lattice-ring) *matrix*) j i) =  
move-matrix (*pprrt* A) j i  
⟨proof⟩

**lemma** *nprrt-move-matrix*: *nprrt* (move-matrix (A::('a::lattice-ring) *matrix*) j i) =  
move-matrix (*nprrt* A) j i  
⟨proof⟩

**lemma** *sparse-row-matrix-pprrt*:  
**fixes** m:: 'a::lattice-ring *spmat*  
**shows** *sorted-spvec* m  $\Longrightarrow$  *sorted-spmat* m  $\Longrightarrow$  *sparse-row-matrix* (*pprrt-spmat*

$m) = \text{pprt} (\text{sparse-row-matrix } m)$   
 $\langle \text{proof} \rangle$

**lemma** *sparse-row-matrix-nprt*:  
**fixes**  $m :: 'a :: \text{lattice-ring} \text{ spmat}$   
**shows**  $\text{sorted-spvec } m \implies \text{sorted-spmat } m \implies \text{sorted-spmat } m \implies \text{sparse-row-matrix}$   
 $(\text{nprrt-spmat } m) = \text{nprrt} (\text{sparse-row-matrix } m)$   
 $\langle \text{proof} \rangle$

**lemma** *sorted-pprt-spvec*:  $\text{sorted-spvec } v \implies \text{sorted-spvec} (\text{pprt-spvec } v)$   
 $\langle \text{proof} \rangle$

**lemma** *sorted-nprt-spvec*:  $\text{sorted-spvec } v \implies \text{sorted-spvec} (\text{nprrt-spvec } v)$   
 $\langle \text{proof} \rangle$

**lemma** *sorted-spvec-pprt-spmat*:  $\text{sorted-spvec } m \implies \text{sorted-spvec} (\text{pprt-spmat } m)$   
 $\langle \text{proof} \rangle$

**lemma** *sorted-spvec-nprt-spmat*:  $\text{sorted-spvec } m \implies \text{sorted-spvec} (\text{nprrt-spmat } m)$   
 $\langle \text{proof} \rangle$

**lemma** *sorted-spmat-pprt-spmat*:  $\text{sorted-spmat } m \implies \text{sorted-spmat} (\text{pprt-spmat } m)$   
 $\langle \text{proof} \rangle$

**lemma** *sorted-spmat-nprt-spmat*:  $\text{sorted-spmat } m \implies \text{sorted-spmat} (\text{nprrt-spmat } m)$   
 $\langle \text{proof} \rangle$

**definition** *mult-est-spmat* ::  $('a :: \text{lattice-ring}) \text{ spmat} \Rightarrow 'a \text{ spmat} \Rightarrow 'a \text{ spmat} \Rightarrow 'a \text{ spmat} \Rightarrow 'a \text{ spmat}$  **where**  
 $\text{mult-est-spmat } r1 \ r2 \ s1 \ s2 =$   
 $\text{add-spmat} (\text{mult-spmat} (\text{pprt-spmat } s2) (\text{pprt-spmat } r2)) (\text{add-spmat} (\text{mult-spmat}$   
 $(\text{pprt-spmat } s1) (\text{nprrt-spmat } r2))$   
 $(\text{add-spmat} (\text{mult-spmat} (\text{nprrt-spmat } s2) (\text{pprt-spmat } r1)) (\text{mult-spmat} (\text{nprrt-spmat}$   
 $s1) (\text{nprrt-spmat } r1))))$

**lemmas** *sparse-row-matrix-op-simps* =  
 $\text{sorted-sparse-matrix-imp-spmat}$   $\text{sorted-sparse-matrix-imp-spvec}$   
 $\text{sparse-row-add-spmat}$   $\text{sorted-spvec-add-spmat}$   $\text{sorted-spmat-add-spmat}$   
 $\text{sparse-row-diff-spmat}$   $\text{sorted-spvec-diff-spmat}$   $\text{sorted-spmat-diff-spmat}$   
 $\text{sparse-row-matrix-minus}$   $\text{sorted-spvec-minus-spmat}$   $\text{sorted-spmat-minus-spmat}$   
 $\text{sparse-row-mult-spmat}$   $\text{sorted-spvec-mult-spmat}$   $\text{sorted-spmat-mult-spmat}$   
 $\text{sparse-row-matrix-abs}$   $\text{sorted-spvec-abs-spmat}$   $\text{sorted-spmat-abs-spmat}$   
 $\text{le-spmat-iff-sparse-row-le}$   
 $\text{sparse-row-matrix-pprt}$   $\text{sorted-spvec-pprt-spmat}$   $\text{sorted-spmat-pprt-spmat}$   
 $\text{sparse-row-matrix-nprt}$   $\text{sorted-spvec-nprt-spmat}$   $\text{sorted-spmat-nprt-spmat}$

**lemmas** *sparse-row-matrix-arith-simps* =

```

mult-spmat.simps mult-spvec-spmat.simps
addmult-spvec.simps
smult-spvec-empty smult-spvec-cons
add-spmat.simps add-spvec.simps
minus-spmat.simps minus-spvec.simps
abs-spmat.simps abs-spvec.simps
diff-spmat-def
le-spmat.simps le-spvec.simps
pprt-spmat.simps pprt-spvec.simps
nprr-spmat.simps nprr-spvec.simps
mult-est-spmat-def

```

**end**

```

theory LP
imports Main HOL-Library.Lattice-Algebras
begin

```

```

lemma le-add-right-mono:
  assumes
     $a \leq b + (c::'a::ordered-ab-group-add)$ 
     $c \leq d$ 
  shows  $a \leq b + d$ 
  <proof>

```

```

lemma linprog-dual-estimate:
  assumes
     $A * x \leq (b::'a::lattice-ring)$ 
     $0 \leq y$ 
     $|A - A'| \leq \delta \cdot A$ 
     $b \leq b'$ 
     $|c - c'| \leq \delta \cdot c$ 
     $|x| \leq r$ 
  shows
     $c * x \leq y * b' + (y * \delta \cdot A + |y * A' - c'| + \delta \cdot c) * r$ 
  <proof>

```

```

lemma le-ge-imp-abs-diff-1:
  assumes
     $A1 \leq (A::'a::lattice-ring)$ 
     $A \leq A2$ 
  shows  $|A - A1| \leq A2 - A1$ 
  <proof>

```

```

lemma mult-le-prts:

```

```

assumes
  a1 <= (a::'a::lattice-ring)
  a <= a2
  b1 <= b
  b <= b2
shows
  a * b <= pprrt a2 * pprrt b2 + pprrt a1 * nprrt b2 + nprrt a2 * pprrt b1 + nprrt a1
  * nprrt b1
  <proof>

lemma mult-le-dual-prts:
assumes
  A * x ≤ (b::'a::lattice-ring)
  0 ≤ y
  A1 ≤ A
  A ≤ A2
  c1 ≤ c
  c ≤ c2
  r1 ≤ x
  x ≤ r2
shows
  c * x ≤ y * b + (let s1 = c1 - y * A2; s2 = c2 - y * A1 in pprrt s2 * pprrt r2
  + pprrt s1 * nprrt r2 + nprrt s2 * pprrt r1 + nprrt s1 * nprrt r1)
  (is - <= - + ?C)
  <proof>

end

```

## 1 Floating Point Representation of the Reals

```

theory ComputeFloat
imports Complex-Main HOL-Library.Lattice-Algebras
begin

  <ML>

```

```

definition int-of-real :: real ⇒ int
  where int-of-real x = (SOME y. real-of-int y = x)

```

```

definition real-is-int :: real ⇒ bool
  where real-is-int x = (∃ (u::int). x = real-of-int u)

```

```

lemma real-is-int-def2: real-is-int x = (x = real-of-int (int-of-real x))
  <proof>

```

```

lemma real-is-int-real[simp]: real-is-int (real-of-int (x::int))
  <proof>

```

**lemma** *int-of-real-real[simp]*:  $\text{int-of-real } (\text{real-of-int } x) = x$   
 $\langle \text{proof} \rangle$

**lemma** *real-int-of-real[simp]*:  $\text{real-is-int } x \implies \text{real-of-int } (\text{int-of-real } x) = x$   
 $\langle \text{proof} \rangle$

**lemma** *real-is-int-add-int-of-real*:  $\text{real-is-int } a \implies \text{real-is-int } b \implies (\text{int-of-real } (a+b)) = (\text{int-of-real } a) + (\text{int-of-real } b)$   
 $\langle \text{proof} \rangle$

**lemma** *real-is-int-add[simp]*:  $\text{real-is-int } a \implies \text{real-is-int } b \implies \text{real-is-int } (a+b)$   
 $\langle \text{proof} \rangle$

**lemma** *int-of-real-sub*:  $\text{real-is-int } a \implies \text{real-is-int } b \implies (\text{int-of-real } (a-b)) = (\text{int-of-real } a) - (\text{int-of-real } b)$   
 $\langle \text{proof} \rangle$

**lemma** *real-is-int-sub[simp]*:  $\text{real-is-int } a \implies \text{real-is-int } b \implies \text{real-is-int } (a-b)$   
 $\langle \text{proof} \rangle$

**lemma** *real-is-int-rep*:  $\text{real-is-int } x \implies \exists!(a::\text{int}). \text{real-of-int } a = x$   
 $\langle \text{proof} \rangle$

**lemma** *int-of-real-mult*:  
**assumes**  $\text{real-is-int } a \text{ } \text{real-is-int } b$   
**shows**  $(\text{int-of-real } (a*b)) = (\text{int-of-real } a) * (\text{int-of-real } b)$   
 $\langle \text{proof} \rangle$

**lemma** *real-is-int-mult[simp]*:  $\text{real-is-int } a \implies \text{real-is-int } b \implies \text{real-is-int } (a*b)$   
 $\langle \text{proof} \rangle$

**lemma** *real-is-int-0[simp]*:  $\text{real-is-int } (0::\text{real})$   
 $\langle \text{proof} \rangle$

**lemma** *real-is-int-1[simp]*:  $\text{real-is-int } (1::\text{real})$   
 $\langle \text{proof} \rangle$

**lemma** *real-is-int-n1*:  $\text{real-is-int } (-1::\text{real})$   
 $\langle \text{proof} \rangle$

**lemma** *real-is-int-numeral[simp]*:  $\text{real-is-int } (\text{numeral } x)$   
 $\langle \text{proof} \rangle$

**lemma** *real-is-int-neg-numeral[simp]*:  $\text{real-is-int } (- \text{numeral } x)$   
 $\langle \text{proof} \rangle$

**lemma** *int-of-real-0[simp]*:  $\text{int-of-real } (0::\text{real}) = (0::\text{int})$   
 $\langle \text{proof} \rangle$

**lemma** *int-of-real-1*[simp]: *int-of-real* (1::real) = (1::int)

*<proof>*

**lemma** *int-of-real-numeral*[simp]: *int-of-real* (numeral b) = numeral b

*<proof>*

**lemma** *int-of-real-neg-numeral*[simp]: *int-of-real* (- numeral b) = - numeral b

*<proof>*

**lemma** *int-div-zdiv*: *int* (a div b) = (*int* a) div (*int* b)

*<proof>*

**lemma** *int-mod-zmod*: *int* (a mod b) = (*int* a) mod (*int* b)

*<proof>*

**lemma** *abs-div-2-less*:  $a \neq 0 \implies a \neq -1 \implies |(a::int) \text{ div } 2| < |a|$

*<proof>*

**lemma** *norm-0-1*: (1::numeral) = Numeral1

*<proof>*

**lemma** *add-left-zero*:  $0 + a = (a::'a::comm-monoid-add)$

*<proof>*

**lemma** *add-right-zero*:  $a + 0 = (a::'a::comm-monoid-add)$

*<proof>*

**lemma** *mult-left-one*:  $1 * a = (a::'a::semiring-1)$

*<proof>*

**lemma** *mult-right-one*:  $a * 1 = (a::'a::semiring-1)$

*<proof>*

**lemma** *int-pow-0*:  $(a::int)^0 = 1$

*<proof>*

**lemma** *int-pow-1*:  $(a::int)^{(\text{Numeral1})} = a$

*<proof>*

**lemma** *one-eq-Numeral1-nring*: (1::'a::numeral) = Numeral1

*<proof>*

**lemma** *one-eq-Numeral1-nat*: (1::nat) = Numeral1

*<proof>*

**lemma** *zpower-Pls*:  $(z::int)^0 = \text{Numeral1}$

*<proof>*

**lemma** *fst-cong*:  $a=a' \implies \text{fst } (a,b) = \text{fst } (a',b)$   
 ⟨proof⟩

**lemma** *snd-cong*:  $b=b' \implies \text{snd } (a,b) = \text{snd } (a,b')$   
 ⟨proof⟩

**lemma** *lift-bool*:  $x \implies x=\text{True}$   
 ⟨proof⟩

**lemma** *nlift-bool*:  $\sim x \implies x=\text{False}$   
 ⟨proof⟩

**lemma** *not-false-eq-true*:  $(\sim \text{False}) = \text{True}$  ⟨proof⟩

**lemma** *not-true-eq-false*:  $(\sim \text{True}) = \text{False}$  ⟨proof⟩

**lemmas** *powerarith* = *nat-numeral power-numeral-even*  
*power-numeral-odd zpower-Pls*

**definition** *float* ::  $(\text{int} \times \text{int}) \Rightarrow \text{real}$  **where**  
*float* =  $(\lambda(a, b). \text{real-of-int } a * 2^{\text{powr real-of-int } b})$

**lemma** *float-add-l0*:  $\text{float } (0, e) + x = x$   
 ⟨proof⟩

**lemma** *float-add-r0*:  $x + \text{float } (0, e) = x$   
 ⟨proof⟩

**lemma** *float-add*:  
 $\text{float } (a1, e1) + \text{float } (a2, e2) =$   
 $(\text{if } e1 \leq e2 \text{ then } \text{float } (a1 + a2 * 2^{\sim(\text{nat}(e2-e1))}, e1) \text{ else } \text{float } (a1 * 2^{\sim(\text{nat}(e1-e2))} + a2,$   
 $e2))$   
 ⟨proof⟩

**lemma** *float-mult-l0*:  $\text{float } (0, e) * x = \text{float } (0, 0)$   
 ⟨proof⟩

**lemma** *float-mult-r0*:  $x * \text{float } (0, e) = \text{float } (0, 0)$   
 ⟨proof⟩

**lemma** *float-mult*:  
 $\text{float } (a1, e1) * \text{float } (a2, e2) = (\text{float } (a1 * a2, e1 + e2))$   
 ⟨proof⟩

**lemma** *float-minus*:  
 $-(\text{float } (a,b)) = \text{float } (-a, b)$   
 ⟨proof⟩

**lemma** *zero-le-float*:

$(0 \leq \text{float } (a,b)) = (0 \leq a)$   
 $\langle \text{proof} \rangle$

**lemma** *float-le-zero*:  
 $(\text{float } (a,b) \leq 0) = (a \leq 0)$   
 $\langle \text{proof} \rangle$

**lemma** *float-abs*:  
 $|\text{float } (a,b)| = (\text{if } 0 \leq a \text{ then } (\text{float } (a,b)) \text{ else } (\text{float } (-a,b)))$   
 $\langle \text{proof} \rangle$

**lemma** *float-zero*:  
 $\text{float } (0, b) = 0$   
 $\langle \text{proof} \rangle$

**lemma** *float-pprt*:  
 $\text{pprt } (\text{float } (a, b)) = (\text{if } 0 \leq a \text{ then } (\text{float } (a,b)) \text{ else } (\text{float } (0, b)))$   
 $\langle \text{proof} \rangle$

**lemma** *float-nprt*:  
 $\text{nprt } (\text{float } (a, b)) = (\text{if } 0 \leq a \text{ then } (\text{float } (0,b)) \text{ else } (\text{float } (a, b)))$   
 $\langle \text{proof} \rangle$

**definition** *lbound* :: *real*  $\Rightarrow$  *real*  
**where** *lbound* *x* = *min* 0 *x*

**definition** *ubound* :: *real*  $\Rightarrow$  *real*  
**where** *ubound* *x* = *max* 0 *x*

**lemma** *lbound*: *lbound* *x*  $\leq$  *x*  
 $\langle \text{proof} \rangle$

**lemma** *ubound*: *x*  $\leq$  *ubound* *x*  
 $\langle \text{proof} \rangle$

**lemma** *pprt-lbound*: *pprt* (*lbound* *x*) = *float* (0, 0)  
 $\langle \text{proof} \rangle$

**lemma** *nprt-ubound*: *nprt* (*ubound* *x*) = *float* (0, 0)  
 $\langle \text{proof} \rangle$

**lemmas** *floatarith*[*simplified norm-0-1*] = *float-add float-add-l0 float-add-r0 float-mult*  
*float-mult-l0 float-mult-r0*  
*float-minus float-abs zero-le-float float-pprt float-nprt pprt-lbound nprrt-ubound*

**lemmas** *arith* = *arith-simps rel-simps diff-nat-numeral nat-0*  
*nat-neg-numeral powerarith floatarith not-false-eq-true not-true-eq-false*



$\langle ML \rangle$

**end**

**theory** *Compute-Oracle* **imports** *HOL.HOL*  
**begin**

$\langle ML \rangle$

**end**

**theory** *ComputeHOL*  
**imports** *Complex-Main* *Compute-Oracle/Compute-Oracle*  
**begin**

**lemma** *Trueprop-eq-eq*:  $\text{Trueprop } X == (X == \text{True})$   $\langle \text{proof} \rangle$

**lemma** *meta-eq-trivial*:  $x == y \implies x == y$   $\langle \text{proof} \rangle$

**lemma** *meta-eq-imp-eq*:  $x == y \implies x = y$   $\langle \text{proof} \rangle$

**lemma** *eq-trivial*:  $x = y \implies x = y$   $\langle \text{proof} \rangle$

**lemma** *bool-to-true*:  $x :: \text{bool} \implies x == \text{True}$   $\langle \text{proof} \rangle$

**lemma** *transmeta-1*:  $x = y \implies y == z \implies x = z$   $\langle \text{proof} \rangle$

**lemma** *transmeta-2*:  $x == y \implies y = z \implies x = z$   $\langle \text{proof} \rangle$

**lemma** *transmeta-3*:  $x == y \implies y == z \implies x = z$   $\langle \text{proof} \rangle$

**lemma** *If-True*:  $\text{If } \text{True} = (\lambda x y. x)$   $\langle \text{proof} \rangle$

**lemma** *If-False*:  $\text{If } \text{False} = (\lambda x y. y)$   $\langle \text{proof} \rangle$

**lemmas** *compute-if* = *If-True If-False*

**lemma** *bool1*:  $(\neg \text{True}) = \text{False}$   $\langle \text{proof} \rangle$

**lemma** *bool2*:  $(\neg \text{False}) = \text{True}$   $\langle \text{proof} \rangle$

**lemma** *bool3*:  $(P \wedge \text{True}) = P$   $\langle \text{proof} \rangle$

**lemma** *bool4*:  $(\text{True} \wedge P) = P$   $\langle \text{proof} \rangle$

**lemma** *bool5*:  $(P \wedge \text{False}) = \text{False}$   $\langle \text{proof} \rangle$

**lemma** *bool6*:  $(\text{False} \wedge P) = \text{False}$   $\langle \text{proof} \rangle$

**lemma** *bool7*:  $(P \vee \text{True}) = \text{True}$   $\langle \text{proof} \rangle$

**lemma** *bool8*:  $(\text{True} \vee P) = \text{True}$   $\langle \text{proof} \rangle$

**lemma** *bool9*:  $(P \vee \text{False}) = P$   $\langle \text{proof} \rangle$

**lemma** *bool10*:  $(\text{False} \vee P) = P$   $\langle \text{proof} \rangle$

**lemma** *bool11*:  $(\text{True} \longrightarrow P) = P$   $\langle \text{proof} \rangle$

**lemma** *bool12*:  $(P \longrightarrow \text{True}) = \text{True}$   $\langle \text{proof} \rangle$

**lemma** *bool13*:  $(\text{True} \longrightarrow P) = P$   $\langle \text{proof} \rangle$

**lemma** *bool14*:  $(P \longrightarrow \text{False}) = (\neg P)$   $\langle \text{proof} \rangle$

**lemma** *bool15*:  $(\text{False} \longrightarrow P) = \text{True}$   $\langle \text{proof} \rangle$

**lemma** *bool16*:  $(False = False) = True$   $\langle proof \rangle$

**lemma** *bool17*:  $(True = True) = True$   $\langle proof \rangle$

**lemma** *bool18*:  $(False = True) = False$   $\langle proof \rangle$

**lemma** *bool19*:  $(True = False) = False$   $\langle proof \rangle$

**lemmas** *compute-bool* = *bool1 bool2 bool3 bool4 bool5 bool6 bool7 bool8 bool9 bool10  
bool11 bool12 bool13 bool14 bool15 bool16 bool17 bool18 bool19*

**lemma** *compute-fst*:  $fst\ (x,y) = x$   $\langle proof \rangle$

**lemma** *compute-snd*:  $snd\ (x,y) = y$   $\langle proof \rangle$

**lemma** *compute-pair-eq*:  $((a, b) = (c, d)) = (a = c \wedge b = d)$   $\langle proof \rangle$

**lemma** *case-prod-simp*:  $case-prod\ f\ (x,y) = f\ x\ y$   $\langle proof \rangle$

**lemmas** *compute-pair* = *compute-fst compute-snd compute-pair-eq case-prod-simp*

**lemma** *compute-the*:  $the\ (Some\ x) = x$   $\langle proof \rangle$

**lemma** *compute-None-Some-eq*:  $(None = Some\ x) = False$   $\langle proof \rangle$

**lemma** *compute-Some-None-eq*:  $(Some\ x = None) = False$   $\langle proof \rangle$

**lemma** *compute-None-None-eq*:  $(None = None) = True$   $\langle proof \rangle$

**lemma** *compute-Some-Some-eq*:  $(Some\ x = Some\ y) = (x = y)$   $\langle proof \rangle$

**definition** *case-option-compute* ::  $'b\ option \Rightarrow 'a \Rightarrow ('b \Rightarrow 'a) \Rightarrow 'a$

**where** *case-option-compute* *opt a f* = *case-option a f opt*

**lemma** *case-option-compute*:  $case-option = (\lambda\ a\ f\ opt.\ case-option-compute\ opt\ a\ f)$   
 $\langle proof \rangle$

**lemma** *case-option-compute-None*:  $case-option-compute\ None = (\lambda\ a\ f.\ a)$   
 $\langle proof \rangle$

**lemma** *case-option-compute-Some*:  $case-option-compute\ (Some\ x) = (\lambda\ a\ f.\ f\ x)$   
 $\langle proof \rangle$

**lemmas** *compute-case-option* = *case-option-compute case-option-compute-None case-option-compute-Some*

**lemmas** *compute-option* = *compute-the compute-None-Some-eq compute-Some-None-eq  
compute-None-None-eq compute-Some-Some-eq compute-case-option*

**lemma** *length-cons*:  $length\ (x\#\ xs) = 1 + (length\ xs)$   
 $\langle proof \rangle$

**lemma** *length-nil*:  $\text{length } [] = 0$

*<proof>*

**lemmas** *compute-list-length* = *length-nil length-cons*

**definition** *case-list-compute* ::  $'b \text{ list} \Rightarrow 'a \Rightarrow ('b \Rightarrow 'b \text{ list} \Rightarrow 'a) \Rightarrow 'a$

**where** *case-list-compute*  $l\ a\ f = \text{case-list } a\ f\ l$

**lemma** *case-list-compute*: *case-list* =  $(\lambda\ (a::'a)\ f\ (l::'b \text{ list}).\ \text{case-list-compute } l\ a\ f)$

*<proof>*

**lemma** *case-list-compute-empty*: *case-list-compute*  $([]::'b \text{ list}) = (\lambda\ (a::'a)\ f.\ a)$

*<proof>*

**lemma** *case-list-compute-cons*: *case-list-compute*  $(u\#\!v) = (\lambda\ (a::'a)\ f.\ (f\ (u::'b)\ v))$

*<proof>*

**lemmas** *compute-case-list* = *case-list-compute case-list-compute-empty case-list-compute-cons*

**lemma** *compute-list-nth*:  $((x\#\!xs) !\ n) = (\text{if } n = 0 \text{ then } x \text{ else } (xs !\ (n - 1)))$

*<proof>*

**lemmas** *compute-list* = *compute-case-list compute-list-length compute-list-nth*

**lemmas** *compute-let* = *Let-def*

**lemmas** *compute-hol* = *compute-if compute-bool compute-pair compute-option compute-list compute-let*

*<ML>*

**end**

**theory** *ComputeNumeral*

```

imports ComputeHOL ComputeFloat
begin

lemmas biteq = eq-num-simps

lemmas bitless = less-num-simps

lemmas bitle = le-num-simps

lemmas bitadd = add-num-simps

lemmas bitmul = mult-num-simps

lemmas bitarith = arith-simps

lemmas natnorm = one-eq-Numeral1-nat

fun natfac :: nat ⇒ nat
  where natfac n = (if n = 0 then 1 else n * (natfac (n - 1)))

lemmas compute-natarith =
  arith-simps rel-simps
  diff-nat-numeral nat-numeral nat-0 nat-neg-numeral
  numeral-One [symmetric]
  numeral-1-eq-Suc-0 [symmetric]
  Suc-numeral natfac.simps

lemmas number-norm = numeral-One[symmetric]

lemmas compute-numberarith =
  arith-simps rel-simps number-norm

lemmas compute-num-conversions =
  of-nat-numeral of-nat-0
  nat-numeral nat-0 nat-neg-numeral
  of-int-numeral of-int-neg-numeral of-int-0

lemmas zpowerarith = power-numeral-even power-numeral-odd zpower-Pls int-pow-1

lemmas compute-div-mod = div-0 mod-0 div-by-0 mod-by-0 div-by-1 mod-by-1
  one-div-numeral one-mod-numeral minus-one-div-numeral minus-one-mod-numeral

```

*one-div-minus-numeral one-mod-minus-numeral*  
*numeral-div-numeral numeral-mod-numeral minus-numeral-div-numeral minus-numeral-mod-numeral*  
*numeral-div-minus-numeral numeral-mod-minus-numeral*  
*div-minus-minus mod-minus-minus Parity.adjust-div-eq of-bool-eq one-neq-zero*  
*numeral-neq-zero neg-equal-0-iff-equal arith-simps arith-special divmod-trivial*  
*divmod-steps divmod-cancel divmod-step-def fst-conv snd-conv numeral-One*  
*case-prod-beta rel-simps Parity.adjust-mod-def div-minus1-right mod-minus1-right*  
*minus-minus numeral-times-numeral mult-zero-right mult-1-right*

**lemma** *even-0-int*: *even (0::int) = True*  
*<proof>*

**lemma** *even-One-int*: *even (numeral Num.One :: int) = False*  
*<proof>*

**lemma** *even-Bit0-int*: *even (numeral (Num.Bit0 x) :: int) = True*  
*<proof>*

**lemma** *even-Bit1-int*: *even (numeral (Num.Bit1 x) :: int) = False*  
*<proof>*

**lemmas** *compute-even = even-0-int even-One-int even-Bit0-int even-Bit1-int*

**lemmas** *compute-numeral = compute-if compute-let compute-pair compute-bool*  
*compute-natarith compute-numberarith max-def min-def*  
*compute-num-conversions zpowerarith compute-div-mod compute-even*

**end**

**theory** *Cplex*  
**imports** *SparseMatrix LP ComputeFloat ComputeNumeral*  
**begin**

*<ML>*

**lemma** *spm-mult-le-dual-prts*:  
**assumes**  
*sorted-sparse-matrix A1*  
*sorted-sparse-matrix A2*  
*sorted-sparse-matrix c1*  
*sorted-sparse-matrix c2*  
*sorted-sparse-matrix y*  
*sorted-sparse-matrix r1*  
*sorted-sparse-matrix r2*  
*sorted-spvec b*

```

le-spmat [] y
sparse-row-matrix A1 ≤ A
A ≤ sparse-row-matrix A2
sparse-row-matrix c1 ≤ c
c ≤ sparse-row-matrix c2
sparse-row-matrix r1 ≤ x
x ≤ sparse-row-matrix r2
A * x ≤ sparse-row-matrix (b::('a::lattice-ring) spmat)
shows
c * x ≤ sparse-row-matrix (add-spmat (mult-spmat y b)
  (let s1 = diff-spmat c1 (mult-spmat y A2); s2 = diff-spmat c2 (mult-spmat y
A1) in
    add-spmat (mult-spmat (pprt-spmat s2) (pprt-spmat r2)) (add-spmat (mult-spmat
(pprt-spmat s1) (nprrt-spmat r2))
      (add-spmat (mult-spmat (nprrt-spmat s2) (pprt-spmat r1)) (mult-spmat (nprrt-spmat
s1) (nprrt-spmat r1))))))
  ⟨proof⟩

```

**lemma** *spm-mult-le-dual-prts-no-let:*

```

assumes
sorted-sparse-matrix A1
sorted-sparse-matrix A2
sorted-sparse-matrix c1
sorted-sparse-matrix c2
sorted-sparse-matrix y
sorted-sparse-matrix r1
sorted-sparse-matrix r2
sorted-spvec b
le-spmat [] y
sparse-row-matrix A1 ≤ A
A ≤ sparse-row-matrix A2
sparse-row-matrix c1 ≤ c
c ≤ sparse-row-matrix c2
sparse-row-matrix r1 ≤ x
x ≤ sparse-row-matrix r2
A * x ≤ sparse-row-matrix (b::('a::lattice-ring) spmat)
shows
c * x ≤ sparse-row-matrix (add-spmat (mult-spmat y b)
  (mult-est-spmat r1 r2 (diff-spmat c1 (mult-spmat y A2)) (diff-spmat c2 (mult-spmat
y A1))))
  ⟨proof⟩

```

⟨ML⟩

**end**