# Matrix

Steven Obua

February 20, 2021

**theory** *Matrix*
**imports** *Main HOL−Library.Lattice-Algebras*
**begin**

**type-synonym** $'a$ *infmatrix* = *nat* ⇒ *nat* ⇒ $'a$

**definition** *nonzero-positions* :: (*nat* ⇒ *nat* ⇒ $'a$::*zero*) ⇒ (*nat* × *nat*) *set* **where**
*nonzero-positions A* = {*pos*. *A* (*fst pos*) (*snd pos*) ~= *0*}

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

**typedef** (**overloaded**) $'a$ *matrix* = *matrix* :: (*nat* ⇒ *nat* ⇒ $'a$::*zero*) *set*
⟨*proof*⟩

**declare** *Rep-matrix-inverse*[*simp*]

**lemma** *finite-nonzero-positions* : *finite* (*nonzero-positions* (*Rep-matrix A*))
⟨*proof*⟩

**definition** *nrows* :: ($'a$::*zero*) *matrix* ⇒ *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* ⇒ *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* ≤ *m*
**shows** (*Rep-matrix A m n*) = *0*
⟨*proof*⟩

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

**definition** *transpose-matrix* :: ($'a$::*zero*) *matrix* ⇒ $'a$ *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*
⟨*proof*⟩

**lemma** *transpose-infmatrix*: *transpose-infmatrix* (% *j i*. *P j i*) = (% *j i*. *P i j*)
⟨*proof*⟩

**lemma** *transpose-infmatrix-closed*[*simp*]: *Rep-matrix* (*Abs-matrix* (*transpose-infmatrix* (*Rep-matrix x*))) = *transpose-infmatrix* (*Rep-matrix x*)
⟨*proof*⟩

**lemma** *infmatrixforward*: (*x*::′*a infmatrix*) = *y* ⟹ ∀ *a b*. *x a b* = *y a b* ⟨*proof*⟩

**lemma** *transpose-infmatrix-inject*: (*transpose-infmatrix A* = *transpose-infmatrix* *B*) = (*A* = *B*)
⟨*proof*⟩

**lemma** *transpose-matrix-inject*: (*transpose-matrix A* = *transpose-matrix B*) = (*A* = *B*)
⟨*proof*⟩

**lemma** *transpose-matrix*[*simp*]: *Rep-matrix*(*transpose-matrix A*) *j i* = *Rep-matrix* *A i j*
⟨*proof*⟩

**lemma** *transpose-transpose-id*[*simp*]: *transpose-matrix* (*transpose-matrix A*) = *A*
⟨*proof*⟩

**lemma** *nrows-transpose*[*simp*]: *nrows* (*transpose-matrix A*) = *ncols A*
⟨*proof*⟩

**lemma** *ncols-transpose*[*simp*]: *ncols* (*transpose-matrix A*) = *nrows A*
⟨*proof*⟩

**lemma** *ncols*: *ncols A* <= *n* ⟹ *Rep-matrix A m n* = *0*
⟨*proof*⟩

**lemma** *ncols-le*: (*ncols A* <= *n*) = (∀*j i*. *n* <= *i* ⟶ (*Rep-matrix A j i*) = *0*)
(**is** - = *?st*)
⟨*proof*⟩

**lemma** *less-ncols*: (*n* < *ncols A*) = (∃*j i*. *n* <= *i* & (*Rep-matrix A j i*) ≠ *0*)
⟨*proof*⟩

**lemma** *le-ncols*: (*n* <= *ncols A*) = (∀ *m*. (∀ *j i*. *m* <= *i* ⟶ (*Rep-matrix A j i*)

$= 0) \longrightarrow n <= m)$
$\langle proof \rangle$

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

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

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

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

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

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

**lemma** *finite-natarray2*: $finite\ \{(x,\ y).\ x < (m::nat)\ \&\ y < (n::nat)\}$
$\langle proof \rangle$

**lemma** *RepAbs-matrix*:
 **assumes** *aem*: $\exists m.\ \forall j\ i.\ m <= j \longrightarrow x\ j\ i = 0$ (**is** *?em*) **and** *aen*:$\exists n.\ \forall j\ i.\ (n <= i \longrightarrow x\ j\ i = 0)$ (**is** *?en*)
 **shows** $(Rep\text{-}matrix\ (Abs\text{-}matrix\ x)) = x$
$\langle proof \rangle$

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

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

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

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

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

⟨*proof*⟩

**lemma** *expand-combine-infmatrix*[*simp*]: *combine-infmatrix f A B j i* = *f* (*A j i*) (*B j i*)
⟨*proof*⟩

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

**definition** *associative* :: $('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow$ *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 situtation: Assume that we have sets $A$ and $B$ with $B \subset A$ and an abstraction $u : A \to B$. This abstraction has to fulfill $u(b) = b$ for all $b \in B$, but is arbitrary otherwise. Each function $f : A \times A \to A$ now induces a function $f' : B \times B \to B$ 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*:
*commutative f* $\Longrightarrow$ *commutative* (*combine-infmatrix f*)
⟨*proof*⟩

**lemma** *combine-matrix-commute*:
*commutative f* $\Longrightarrow$ *commutative* (*combine-matrix f*)
⟨*proof*⟩

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* $\Longrightarrow$ *nonzero-positions* (*combine-infmatrix f A B*) $\subseteq$ (*nonzero-positions A*) $\cup$ (*nonzero-positions B*)
⟨*proof*⟩

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

**lemma** *combine-infmatrix-closed* [*simp*]:
$f\ 0\ 0 = 0 \Longrightarrow$ *Rep-matrix* (*Abs-matrix* (*combine-infmatrix* $f$ (*Rep-matrix* $A$) (*Rep-matrix* $B$))) = *combine-infmatrix* $f$ (*Rep-matrix* $A$) (*Rep-matrix* $B$)
⟨*proof*⟩

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*]: $f\ 0 = 0 \Longrightarrow$ *nonzero-positions* (*apply-infmatrix* $f$ $A$) $\subseteq$ *nonzero-positions* $A$
⟨*proof*⟩

**lemma** *apply-infmatrix-closed* [*simp*]:
$f\ 0 = 0 \Longrightarrow$ *Rep-matrix* (*Abs-matrix* (*apply-infmatrix* $f$ (*Rep-matrix* $A$))) = *apply-infmatrix* $f$ (*Rep-matrix* $A$)
⟨*proof*⟩

**lemma** *combine-infmatrix-assoc*[*simp*]: $f\ 0\ 0 = 0 \Longrightarrow$ *associative* $f \Longrightarrow$ *associative* (*combine-infmatrix* $f$)
⟨*proof*⟩

**lemma** *comb*: $f = g \Longrightarrow x = y \Longrightarrow f\ x = g\ y$
⟨*proof*⟩

**lemma** *combine-matrix-assoc*: $f\ 0\ 0 = 0 \Longrightarrow$ *associative* $f \Longrightarrow$ *associative* (*combine-matrix* $f$)
⟨*proof*⟩

**lemma** *Rep-apply-matrix*[*simp*]: $f\ 0 = 0 \Longrightarrow$ *Rep-matrix* (*apply-matrix* $f$ $A$) $j\ i$ = $f$ (*Rep-matrix* $A\ j\ i$)
⟨*proof*⟩

**lemma** *Rep-combine-matrix*[*simp*]: $f\ 0\ 0 = 0 \Longrightarrow$ *Rep-matrix* (*combine-matrix* $f$ $A$ $B$) $j\ i = f$ (*Rep-matrix* $A\ j\ i$) (*Rep-matrix* $B\ j\ i$)
⟨*proof*⟩

**lemma** *combine-nrows-max*: $f\ 0\ 0 = 0 \Longrightarrow$ *nrows* (*combine-matrix* $f$ $A$ $B$) <= *max* (*nrows* $A$) (*nrows* $B$)
⟨*proof*⟩

**lemma** *combine-ncols-max*: $f\ 0\ 0 = 0 \Longrightarrow$ *ncols* (*combine-matrix* $f$ $A$ $B$) <= *max* (*ncols* $A$) (*ncols* $B$)
⟨*proof*⟩

**lemma** *combine-nrows*: $f\ 0\ 0 = 0 \Longrightarrow$ *nrows* $A$ <= $q \Longrightarrow$ *nrows* $B$ <= $q \Longrightarrow$ *nrows*(*combine-matrix* $f$ $A$ $B$) <= $q$
⟨*proof*⟩

**lemma** *combine-ncols*: $f\ 0\ 0 = 0 \Longrightarrow$ *ncols* $A$ <= $q \Longrightarrow$ *ncols* $B$ <= $q \Longrightarrow$ *ncols*(*combine-matrix* $f$ $A$ $B$) <= $q$

⟨*proof*⟩

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

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

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

**primrec** *foldseq* :: $('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow (nat \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a$
**where**
*foldseq f s* $0 = s\ 0$
| *foldseq f s* $(Suc\ n) = f\ (s\ 0)\ (foldseq\ f\ (\%\ k.\ s(Suc\ k))\ n)$

**primrec** *foldseq-transposed* :: $('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow (nat \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a$
**where**
*foldseq-transposed f s* $0 = s\ 0$
| *foldseq-transposed f s* $(Suc\ n) = f\ (foldseq\text{-}transposed\ f\ s\ n)\ (s\ (Suc\ n))$

**lemma** *foldseq-assoc* : *associative f* $\Longrightarrow$ *foldseq f* = *foldseq-transposed f*
⟨*proof*⟩

**lemma** *foldseq-distr*: $[\![associative\ f;\ commutative\ f]\!] \Longrightarrow foldseq\ f\ (\%\ k.\ f\ (u\ k)\ (v\ k))\ n = f\ (foldseq\ f\ u\ n)\ (foldseq\ f\ v\ n)$
⟨*proof*⟩

**theorem** $[\![associative\ f;\ associative\ g;\ \forall\, a\ b\ c\ d.\ g\ (f\ a\ b)\ (f\ c\ d) = f\ (g\ a\ c)\ (g\ b\ d);\ \exists\, x\ y.\ (f\ x) \neq (f\ y);\ \exists\, x\ y.\ (g\ x) \neq (g\ y);\ f\ x\ x = x;\ g\ x\ x = x]\!] \Longrightarrow f{=}g \mid (\forall\, y.\ f\ y\ x = y) \mid (\forall\, y.\ g\ y\ x = y)$
⟨*proof*⟩

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

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

**lemma** *foldseq-tail*:
**assumes** $M <= N$
**shows** *foldseq f S N* $= foldseq\ f\ (\%\ k.\ (if\ k < M\ then\ (S\ k)\ else\ (foldseq\ f\ (\%\ k.\ S(k{+}M))\ (N{-}M))))\ M$
⟨*proof*⟩

**lemma** *foldseq-zerotail*:
**assumes**
*fz*: $f\ 0\ 0 = 0$
**and** *sz*: $\forall\, i.\ \ n <= i \longrightarrow s\ i = 0$
**and** *nm*: $n <= m$
**shows**
$foldseq\ f\ s\ n = foldseq\ f\ s\ m$
$\langle proof \rangle$

**lemma** *foldseq-zerotail2*:
**assumes** $\forall\, x.\ f\ x\ 0 = x$
**and** $\forall\, i.\ n < i \longrightarrow s\ i = 0$
**and** *nm*: $n <= m$
**shows** $foldseq\ f\ s\ n = foldseq\ f\ s\ m$
$\langle proof \rangle$

**lemma** *foldseq-zerostart*:
$\forall\, x.\ f\ 0\ (f\ 0\ x) = f\ 0\ x \Longrightarrow \forall\, i.\ i <= n \longrightarrow s\ i = 0 \Longrightarrow foldseq\ f\ s\ (Suc\ n) = f\ 0\ (s\ (Suc\ n))$
$\langle proof \rangle$

**lemma** *foldseq-zerostart2*:
$\forall\, x.\ f\ 0\ x = x \Longrightarrow \forall\, i.\ i < n \longrightarrow s\ i = 0 \Longrightarrow foldseq\ f\ s\ n = s\ n$
$\langle proof \rangle$

**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** $foldseq\ f\ s\ n = (if\ \ (j <= n)\ then\ (s\ j)\ else\ 0)$
$\langle proof \rangle$

**lemma** *foldseq-distr-unary*:
**assumes** !! $a\ b.\ g\ (f\ a\ b) = f\ (g\ a)\ (g\ b)$
**shows** $g(foldseq\ f\ s\ n) = foldseq\ f\ (\%\ x.\ g(s\ x))\ n$
$\langle proof \rangle$

**definition** *mult-matrix-n* :: $nat \Rightarrow (('a::zero) \Rightarrow ('b::zero) \Rightarrow ('c::zero)) \Rightarrow ('c \Rightarrow 'c \Rightarrow 'c) \Rightarrow 'a\ matrix \Rightarrow 'b\ matrix \Rightarrow 'c\ matrix$ **where**
$mult\text{-}matrix\text{-}n\ n\ fmul\ fadd\ A\ B == Abs\text{-}matrix(\%\ j\ i.\ foldseq\ fadd\ (\%\ k.\ fmul\ (Rep\text{-}matrix\ A\ j\ k)\ (Rep\text{-}matrix\ B\ k\ i))\ n)$

**definition** *mult-matrix* :: $(('a::zero) \Rightarrow ('b::zero) \Rightarrow ('c::zero)) \Rightarrow ('c \Rightarrow 'c \Rightarrow 'c) \Rightarrow 'a\ matrix \Rightarrow 'b\ matrix \Rightarrow 'c\ matrix$ **where**
$mult\text{-}matrix\ fmul\ fadd\ A\ B == mult\text{-}matrix\text{-}n\ (max\ (ncols\ A)\ (nrows\ B))\ fmul\ fadd\ A\ B$

**lemma** *mult-matrix-n*:
**assumes** $ncols\ A \leq\ n$ (**is** $?An$) $nrows\ B \leq n$ (**is** $?Bn$) $fadd\ 0\ 0 = 0\ fmul\ 0\ 0$

$= 0$
**shows** $c$:*mult-matrix fmul fadd A B* $=$ *mult-matrix-n n fmul fadd A B*
⟨*proof*⟩

**lemma** *mult-matrix-nm*:
**assumes** *ncols A* <= *n nrows B* <= *n ncols A* <= *m nrows B* <= *m fadd 0 0* $=$ *0 fmul 0 0* $=$ *0*
**shows** *mult-matrix-n n fmul fadd A B* $=$ *mult-matrix-n m fmul fadd A B*
⟨*proof*⟩

**definition** *r-distributive* :: $('a \Rightarrow 'b \Rightarrow 'b) \Rightarrow ('b \Rightarrow 'b \Rightarrow 'b) \Rightarrow$ *bool* **where**
*r-distributive fmul fadd* == ∀ *a u v*. *fmul a* (*fadd u v*) = *fadd* (*fmul a u*) (*fmul a v*)

**definition** *l-distributive* :: $('a \Rightarrow 'b \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow$ *bool* **where**
*l-distributive fmul fadd* == ∀ *a u v*. *fmul* (*fadd u v*) *a* = *fadd* (*fmul u a*) (*fmul v a*)

**definition** *distributive* :: $('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow$ *bool* **where**
*distributive fmul fadd* == *l-distributive fmul fadd* & *r-distributive fmul fadd*

**lemma** *max1*: !! *a x y*. (*a*::*nat*) <= *x* ⟹ *a* <= *max x y* ⟨*proof*⟩
**lemma** *max2*: !! *b x y*. (*b*::*nat*) <= *y* ⟹ *b* <= *max x y* ⟨*proof*⟩

**lemma** *r-distributive-matrix*:
**assumes**
*r-distributive fmul fadd*
*associative fadd*
*commutative fadd*
*fadd 0 0* = *0*
∀ *a*. *fmul a 0* = *0*
∀ *a*. *fmul 0 a* = *0*
**shows** *r-distributive* (*mult-matrix fmul fadd*) (*combine-matrix fadd*)
⟨*proof*⟩

**lemma** *l-distributive-matrix*:
**assumes**
*l-distributive fmul fadd*
*associative fadd*
*commutative fadd*
*fadd 0 0* = *0*
∀ *a*. *fmul a 0* = *0*
∀ *a*. *fmul 0 a* = *0*
**shows** *l-distributive* (*mult-matrix fmul fadd*) (*combine-matrix fadd*)
⟨*proof*⟩

**instantiation** *matrix* :: (*zero*) *zero*
**begin**

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

**instance** ⟨*proof*⟩

**end**

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

**lemma** *zero-matrix-def-nrows*[*simp*]: $nrows\ 0 = 0$
⟨*proof*⟩

**lemma** *zero-matrix-def-ncols*[*simp*]: $ncols\ 0 = 0$
⟨*proof*⟩

**lemma** *combine-matrix-zero-l-neutral*: $zero\text{-}l\text{-}neutral\ f \Longrightarrow zero\text{-}l\text{-}neutral\ (combine\text{-}matrix\ f)$
⟨*proof*⟩

**lemma** *combine-matrix-zero-r-neutral*: $zero\text{-}r\text{-}neutral\ f \Longrightarrow zero\text{-}r\text{-}neutral\ (combine\text{-}matrix\ f)$
⟨*proof*⟩

**lemma** *mult-matrix-zero-closed*: $[\![fadd\ 0\ 0 = 0;\ zero\text{-}closed\ fmul]\!] \Longrightarrow zero\text{-}closed\ (mult\text{-}matrix\ fmul\ fadd)$
⟨*proof*⟩

**lemma** *mult-matrix-n-zero-right*[*simp*]: $[\![fadd\ 0\ 0 = 0;\ \forall a.\ fmul\ a\ 0 = 0]\!] \Longrightarrow mult\text{-}matrix\text{-}n\ n\ fmul\ fadd\ A\ 0 = 0$
⟨*proof*⟩

**lemma** *mult-matrix-n-zero-left*[*simp*]: $[\![fadd\ 0\ 0 = 0;\ \forall a.\ fmul\ 0\ a = 0]\!] \Longrightarrow mult\text{-}matrix\text{-}n\ n\ fmul\ fadd\ 0\ A = 0$
⟨*proof*⟩

**lemma** *mult-matrix-zero-left*[*simp*]: $[\![fadd\ 0\ 0 = 0;\ \forall a.\ fmul\ 0\ a = 0]\!] \Longrightarrow mult\text{-}matrix\ fmul\ fadd\ 0\ A = 0$
⟨*proof*⟩

**lemma** *mult-matrix-zero-right*[*simp*]: $[\![fadd\ 0\ 0 = 0;\ \forall a.\ fmul\ a\ 0 = 0]\!] \Longrightarrow mult\text{-}matrix\ fmul\ fadd\ A\ 0 = 0$
⟨*proof*⟩

**lemma** *apply-matrix-zero*[*simp*]: $f\ 0 = 0 \Longrightarrow apply\text{-}matrix\ f\ 0 = 0$
⟨*proof*⟩

**lemma** *combine-matrix-zero*: $f\ 0\ 0 = 0 \Longrightarrow combine\text{-}matrix\ f\ 0\ 0 = 0$
⟨*proof*⟩

**lemma** *transpose-matrix-zero*[*simp*]: *transpose-matrix 0 = 0*
⟨*proof*⟩

**lemma** *apply-zero-matrix-def*[*simp*]: *apply-matrix* (% *x. 0*) *A = 0*
⟨*proof*⟩

**definition** *singleton-matrix* :: *nat* ⇒ *nat* ⇒ ('*a*::*zero*) ⇒ '*a matrix* **where**
*singleton-matrix j i a* == *Abs-matrix*(% *m n. if j = m & i = n then a else 0*)

**definition** *move-matrix* :: ('*a*::*zero*) *matrix* ⇒ *int* ⇒ *int* ⇒ '*a matrix* **where**
*move-matrix A y x* == *Abs-matrix*(% *j i. if* (((*int j*)−*y*) < *0*) | (((*int i*)−*x*) < *0*) *then 0 else Rep-matrix A* (*nat* ((*int j*)−*y*)) (*nat* ((*int i*)−*x*)))

**definition** *take-rows* :: ('*a*::*zero*) *matrix* ⇒ *nat* ⇒ '*a matrix* **where**
*take-rows A r* == *Abs-matrix*(% *j i. if* (*j* < *r*) *then* (*Rep-matrix A j i*) *else 0*)

**definition** *take-columns* :: ('*a*::*zero*) *matrix* ⇒ *nat* ⇒ '*a matrix* **where**
*take-columns A c* == *Abs-matrix*(% *j i. if* (*i* < *c*) *then* (*Rep-matrix A j i*) *else 0*)

**definition** *column-of-matrix* :: ('*a*::*zero*) *matrix* ⇒ *nat* ⇒ '*a matrix* **where**
*column-of-matrix A n* == *take-columns* (*move-matrix A 0* (− *int n*)) *1*

**definition** *row-of-matrix* :: ('*a*::*zero*) *matrix* ⇒ *nat* ⇒ '*a 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 & i = n then e else 0*)
⟨*proof*⟩

**lemma** *apply-singleton-matrix*[*simp*]: *f 0 = 0* ⟹ *apply-matrix f* (*singleton-matrix j i x*) = (*singleton-matrix j i* (*f x*))
⟨*proof*⟩

**lemma** *singleton-matrix-zero*[*simp*]: *singleton-matrix j i 0 = 0*
⟨*proof*⟩

**lemma** *nrows-singleton*[*simp*]: *nrows*(*singleton-matrix j i e*) = (*if e = 0 then 0 else Suc j*)
⟨*proof*⟩

**lemma** *ncols-singleton*[*simp*]: *ncols*(*singleton-matrix j i e*) = (*if e = 0 then 0 else Suc i*)
⟨*proof*⟩

**lemma** *combine-singleton*: *f 0 0 = 0* ⟹ *combine-matrix f* (*singleton-matrix j i a*) (*singleton-matrix j i b*) = *singleton-matrix j i* (*f a b*)
⟨*proof*⟩

**lemma** *transpose-singleton*[*simp*]: *transpose-matrix* (*singleton-matrix j i a*) = *singleton-matrix i j a*
⟨*proof*⟩

**lemma** *Rep-move-matrix*[*simp*]:
*Rep-matrix* (*move-matrix A y x*) *j i* =
(*if* (((*int j*)−*y*) < *0*) | (((*int i*)−*x*) < *0*) *then 0 else Rep-matrix A* (*nat*((*int j*)−*y*)) (*nat*((*int i*)−*x*)))
⟨*proof*⟩

**lemma** *move-matrix-0-0*[*simp*]: *move-matrix A 0 0* = *A*
⟨*proof*⟩

**lemma** *move-matrix-ortho*: *move-matrix A j i* = *move-matrix* (*move-matrix A j 0*) *0 i*
⟨*proof*⟩

**lemma** *transpose-move-matrix*[*simp*]:
*transpose-matrix* (*move-matrix A x y*) = *move-matrix* (*transpose-matrix A*) *y x*
⟨*proof*⟩

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

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

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

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

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

**lemma** *column-of-matrix*: *ncols A* <= *n* $\Longrightarrow$ *column-of-matrix A n* = *0*
⟨*proof*⟩

**lemma** *row-of-matrix*: *nrows A* <= *n* $\Longrightarrow$ *row-of-matrix A n* = *0*

$\langle proof \rangle$

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

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

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

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

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

**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**

*foldseq g* (% *j*. *foldseq f* (*A j*) *n*) *m* = *foldseq f* (% *j*. *foldseq g* ((*transpose-infmatrix A*) *j*) *m*) *n*
⟨*proof*⟩

**lemma** *mult-n-nrows*:
**assumes**
$\forall$ *a*. *fmul 0 a* = *0*
$\forall$ *a*. *fmul a 0* = *0*
*fadd 0 0* = *0*
**shows** *nrows* (*mult-matrix-n n fmul fadd A B*) $\leq$ *nrows A*
⟨*proof*⟩

**lemma** *mult-n-ncols*:
**assumes**
$\forall$ *a*. *fmul 0 a* = *0*
$\forall$ *a*. *fmul a 0* = *0*
*fadd 0 0* = *0*
**shows** *ncols* (*mult-matrix-n n fmul fadd A B*) $\leq$ *ncols B*
⟨*proof*⟩

**lemma** *mult-nrows*:
**assumes**
$\forall$ *a*. *fmul 0 a* = *0*
$\forall$ *a*. *fmul a 0* = *0*
*fadd 0 0* = *0*
**shows** *nrows* (*mult-matrix fmul fadd A B*) $\leq$ *nrows A*
⟨*proof*⟩

**lemma** *mult-ncols*:
**assumes**
$\forall$ *a*. *fmul 0 a* = *0*
$\forall$ *a*. *fmul a 0* = *0*
*fadd 0 0* = *0*
**shows** *ncols* (*mult-matrix fmul fadd A B*) $\leq$ *ncols B*
⟨*proof*⟩

**lemma** *nrows-move-matrix-le*: *nrows* (*move-matrix A j i*) <= *nat*((*int* (*nrows A*)) + *j*)
⟨*proof*⟩

**lemma** *ncols-move-matrix-le*: *ncols* (*move-matrix A j i*) <= *nat*((*int* (*ncols A*)) + *i*)
⟨*proof*⟩

**lemma** *mult-matrix-assoc*:
**assumes**
$\forall$ *a*. *fmul1 0 a* = *0*
$\forall$ *a*. *fmul1 a 0* = *0*
$\forall$ *a*. *fmul2 0 a* = *0*

$\forall a.$ *fmul2 a 0* $=$ *0*
*fadd1 0 0* $=$ *0*
*fadd2 0 0* $=$ *0*
$\forall a\ b\ c\ d.$ *fadd2* (*fadd1 a b*) (*fadd1 c d*) $=$ *fadd1* (*fadd2 a c*) (*fadd2 b d*)
*associative fadd1*
*associative fadd2*
$\forall a\ b\ c.$ *fmul2* (*fmul1 a b*) *c* $=$ *fmul1 a* (*fmul2 b c*)
$\forall a\ b\ c.$ *fmul2* (*fadd1 a b*) *c* $=$ *fadd1* (*fmul2 a c*) (*fmul2 b c*)
$\forall a\ b\ c.$ *fmul1 c* (*fadd2 a b*) $=$ *fadd2* (*fmul1 c a*) (*fmul1 c b*)
**shows** *mult-matrix fmul2 fadd2* (*mult-matrix fmul1 fadd1 A B*) *C* $=$ *mult-matrix fmul1 fadd1 A* (*mult-matrix fmul2 fadd2 B C*)
⟨*proof*⟩

**lemma**
**assumes**
$\forall a.$ *fmul1 0 a* $=$ *0*
$\forall a.$ *fmul1 a 0* $=$ *0*
$\forall a.$ *fmul2 0 a* $=$ *0*
$\forall a.$ *fmul2 a 0* $=$ *0*
*fadd1 0 0* $=$ *0*
*fadd2 0 0* $=$ *0*
$\forall a\ b\ c\ d.$ *fadd2* (*fadd1 a b*) (*fadd1 c d*) $=$ *fadd1* (*fadd2 a c*) (*fadd2 b d*)
*associative fadd1*
*associative fadd2*
$\forall a\ b\ c.$ *fmul2* (*fmul1 a b*) *c* $=$ *fmul1 a* (*fmul2 b c*)
$\forall a\ b\ c.$ *fmul2* (*fadd1 a b*) *c* $=$ *fadd1* (*fmul2 a c*) (*fmul2 b c*)
$\forall a\ b\ c.$ *fmul1 c* (*fadd2 a b*) $=$ *fadd2* (*fmul1 c a*) (*fmul1 c b*)
**shows**
(*mult-matrix fmul1 fadd1 A*) *o* (*mult-matrix fmul2 fadd2 B*) $=$ *mult-matrix fmul2 fadd2* (*mult-matrix fmul1 fadd1 A B*)
⟨*proof*⟩

**lemma** *mult-matrix-assoc-simple*:
**assumes**
$\forall a.$ *fmul 0 a* $=$ *0*
$\forall a.$ *fmul a 0* $=$ *0*
*fadd 0 0* $=$ *0*
*associative fadd*
*commutative fadd*
*associative fmul*
*distributive fmul fadd*
**shows** *mult-matrix fmul fadd* (*mult-matrix fmul fadd A B*) *C* $=$ *mult-matrix fmul fadd A* (*mult-matrix fmul fadd B C*)
⟨*proof*⟩

**lemma** *transpose-apply-matrix*: *f 0* $=$ *0* $\Longrightarrow$ *transpose-matrix* (*apply-matrix f A*) $=$ *apply-matrix f* (*transpose-matrix A*)
⟨*proof*⟩

**lemma** *transpose-combine-matrix*: $f\ 0\ 0 = 0 \Longrightarrow$ *transpose-matrix* (*combine-matrix* $f\ A\ B$) = *combine-matrix* $f$ (*transpose-matrix* $A$) (*transpose-matrix* $B$)
⟨*proof*⟩

**lemma** *Rep-mult-matrix*:
  **assumes**
  $\forall a.$ *fmul* $0\ a = 0$
  $\forall a.$ *fmul* $a\ 0 = 0$
  *fadd* $0\ 0 = 0$
  **shows**
  *Rep-matrix*(*mult-matrix fmul fadd* $A\ B$) $j\ i$ =
  *foldseq fadd* (% $k$. *fmul* (*Rep-matrix* $A\ j\ k$) (*Rep-matrix* $B\ k\ i$)) (*max* (*ncols* $A$) (*nrows* $B$))
⟨*proof*⟩

**lemma** *transpose-mult-matrix*:
  **assumes**
  $\forall a.$ *fmul* $0\ a = 0$
  $\forall a.$ *fmul* $a\ 0 = 0$
  *fadd* $0\ 0 = 0$
  $\forall x\ y.$ *fmul* $y\ x$ = *fmul* $x\ y$
  **shows**
  *transpose-matrix* (*mult-matrix fmul fadd* $A\ B$) = *mult-matrix fmul fadd* (*transpose-matrix* $B$) (*transpose-matrix* $A$)
  ⟨*proof*⟩

**lemma** *column-transpose-matrix*: *column-of-matrix* (*transpose-matrix* $A$) $n$ = *transpose-matrix* (*row-of-matrix* $A\ n$)
⟨*proof*⟩

**lemma** *take-columns-transpose-matrix*: *take-columns* (*transpose-matrix* $A$) $n$ = *transpose-matrix* (*take-rows* $A\ n$)
⟨*proof*⟩

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

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

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

**instance** ⟨*proof*⟩

**end**

**instance** *matrix* :: ({*zero*, *order*}) *order*
⟨*proof*⟩

**lemma** *le-apply-matrix*:
**assumes**
$f\ 0 = 0$
$\forall x\ y.\ x <= y \longrightarrow f\ x <= f\ y$
$(a::('a::\{ord,\ zero\})\ matrix) <= b$
**shows**
$apply\text{-}matrix\ f\ a <= apply\text{-}matrix\ f\ b$
⟨*proof*⟩

**lemma** *le-combine-matrix*:
**assumes**
$f\ 0\ 0 = 0$
$\forall a\ b\ c\ d.\ a <= b\ \&\ c <= d \longrightarrow f\ a\ c <= f\ b\ d$
$A <= B$
$C <= D$
**shows**
$combine\text{-}matrix\ f\ A\ C <= combine\text{-}matrix\ f\ B\ D$
⟨*proof*⟩

**lemma** *le-left-combine-matrix*:
**assumes**
$f\ 0\ 0 = 0$
$\forall a\ b\ c.\ a <= b \longrightarrow f\ c\ a <= f\ c\ b$
$A <= B$
**shows**
$combine\text{-}matrix\ f\ C\ A <= combine\text{-}matrix\ f\ C\ B$
⟨*proof*⟩

**lemma** *le-right-combine-matrix*:
**assumes**
$f\ 0\ 0 = 0$
$\forall a\ b\ c.\ a <= b \longrightarrow f\ a\ c <= f\ b\ c$
$A <= B$
**shows**
$combine\text{-}matrix\ f\ A\ C <= combine\text{-}matrix\ f\ B\ C$
⟨*proof*⟩

**lemma** *le-transpose-matrix*: $(A <= B) = (transpose\text{-}matrix\ A <= transpose\text{-}matrix\ B)$
⟨*proof*⟩

**lemma** *le-foldseq*:
**assumes**
$\forall a\ b\ c\ d\ .\ a <= b\ \&\ c <= d \longrightarrow f\ a\ c <= f\ b\ d$
$\forall i.\ i <= n \longrightarrow s\ i <= t\ i$
**shows**
$foldseq\ f\ s\ n <= foldseq\ f\ t\ n$
⟨*proof*⟩

**lemma** *le-left-mult*:
  **assumes**
  $\forall a\ b\ c\ d.\ a <= b\ \&\ c <= d \longrightarrow fadd\ a\ c <= fadd\ b\ d$
  $\forall c\ a\ b.\quad 0 <= c\ \&\ a <= b \longrightarrow fmul\ c\ a <= fmul\ c\ b$
  $\forall a.\ fmul\ 0\ a = 0$
  $\forall a.\ fmul\ a\ 0 = 0$
  $fadd\ 0\ 0 = 0$
  $0 <= C$
  $A <= B$
  **shows**
  $mult\text{-}matrix\ fmul\ fadd\ C\ A <= mult\text{-}matrix\ fmul\ fadd\ C\ B$
  ⟨*proof*⟩

**lemma** *le-right-mult*:
  **assumes**
  $\forall a\ b\ c\ d.\ a <= b\ \&\ c <= d \longrightarrow fadd\ a\ c <= fadd\ b\ d$
  $\forall c\ a\ b.\ 0 <= c\ \&\ a <= b \longrightarrow fmul\ a\ c <= fmul\ b\ c$
  $\forall a.\ fmul\ 0\ a = 0$
  $\forall a.\ fmul\ a\ 0 = 0$
  $fadd\ 0\ 0 = 0$
  $0 <= C$
  $A <= B$
  **shows**
  $mult\text{-}matrix\ fmul\ fadd\ A\ C <= mult\text{-}matrix\ fmul\ fadd\ B\ C$
  ⟨*proof*⟩

**lemma** *spec2*: $\forall j\ i.\ P\ j\ i \Longrightarrow P\ j\ i$ ⟨*proof*⟩
**lemma** *neg-imp*: $(\neg\ Q \longrightarrow \neg\ P) \Longrightarrow P \longrightarrow Q$ ⟨*proof*⟩

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

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

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

**lemma** *move-matrix-le-zero*[*simp*]: $0 <= j \Longrightarrow 0 <= i \Longrightarrow (move\text{-}matrix\ A\ j\ i <= 0) = (A <= (0::('a::\{order,zero\})\ matrix))$
  ⟨*proof*⟩

**lemma** *move-matrix-zero-le*[*simp*]: $0 <= j \Longrightarrow 0 <= i \Longrightarrow (0 <= move\text{-}matrix\ A\ j\ i) = ((0::('a::\{order,zero\})\ matrix) <= A)$
  ⟨*proof*⟩

**lemma** *move-matrix-le-move-matrix-iff* [*simp*]: $0 <= j \Longrightarrow 0 <= i \Longrightarrow$ (*move-matrix* $A$ $j$ $i$ <= *move-matrix* $B$ $j$ $i$) = ($A$ <= ($B$::('*a*::{*order*,*zero*}) *matrix*))
⟨*proof*⟩

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

**definition** *inf* = *combine-matrix inf*

**definition** *sup* = *combine-matrix sup*

**instance**
⟨*proof*⟩

**end**

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

**definition**
*plus-matrix-def*: $A + B$ = *combine-matrix* (+) $A$ $B$

**instance** ⟨*proof*⟩

**end**

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

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

**instance** ⟨*proof*⟩

**end**

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

**definition**
*diff-matrix-def*: $A - B$ = *combine-matrix* (−) $A$ $B$

**instance** ⟨*proof*⟩

**end**

**instantiation** *matrix* :: ({*plus*, *times*, *zero*}) *times*
**begin**

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

**instance** ⟨*proof*⟩

**end**

**instantiation** *matrix* :: ({*lattice*, *uminus*, *zero*}) *abs*
**begin**

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

**instance** ⟨*proof*⟩

**end**

**instance** *matrix* :: (*monoid-add*) *monoid-add*
⟨*proof*⟩

**instance** *matrix* :: (*comm-monoid-add*) *comm-monoid-add*
⟨*proof*⟩

**instance** *matrix* :: (*group-add*) *group-add*
⟨*proof*⟩

**instance** *matrix* :: (*ab-group-add*) *ab-group-add*
⟨*proof*⟩

**instance** *matrix* :: (*ordered-ab-group-add*) *ordered-ab-group-add*
⟨*proof*⟩

**instance** *matrix* :: (*lattice-ab-group-add*) *semilattice-inf-ab-group-add* ⟨*proof*⟩
**instance** *matrix* :: (*lattice-ab-group-add*) *semilattice-sup-ab-group-add* ⟨*proof*⟩

**instance** *matrix* :: (*semiring-0*) *semiring-0*
⟨*proof*⟩

**instance** *matrix* :: (*ring*) *ring* ⟨*proof*⟩

**instance** *matrix* :: (*ordered-ring*) *ordered-ring*
⟨*proof*⟩

**instance** *matrix* :: (*lattice-ring*) *lattice-ring*
⟨*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* (+) (% *k*. (*Rep-matrix a j k*) ∗ (*Rep-matrix b k i*)) (*max* (*ncols a*) (*nrows b*))
⟨*proof*⟩

**lemma** *apply-matrix-add*: ∀ *x y*. *f* (*x*+*y*) = (*f x*) + (*f y*) ⟹ *f 0* = (*0*::'*a*)
⟹ *apply-matrix f* ((*a*::('*a*::*monoid-add*) *matrix*) + *b*) = (*apply-matrix f a*) + (*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*) <= *nrows A*
⟨*proof*⟩

**lemma** *ncols-mult*: *ncols* ((*A*::('*a*::*semiring-0*) *matrix*) ∗ *B*) <= *ncols B*
⟨*proof*⟩

**definition**
*one-matrix* :: *nat* ⇒ ('*a*::{*zero*,*one*}) *matrix* **where**
*one-matrix n* = *Abs-matrix* (% *j i*. *if j* = *i* & *j* < *n then 1 else 0*)

**lemma** *Rep-one-matrix*[*simp*]: *Rep-matrix* (*one-matrix n*) *j i* = (*if* (*j* = *i* & *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*]: *ncols A* <= *n* ⟹ (*A*::('*a*::{*semiring-1*}) *matrix*) ∗ (*one-matrix n*) = *A*
⟨*proof*⟩

**lemma** *one-matrix-mult-left*[*simp*]: *nrows A* <= *n* ⟹ (*one-matrix n*) ∗ *A* = (*A*::('*a*::*semiring-1*) *matrix*)
⟨*proof*⟩

**lemma** *transpose-matrix-mult*: *transpose-matrix* ((*A*::('*a*::*comm-ring*) *matrix*)∗*B*) = (*transpose-matrix B*) ∗ (*transpose-matrix A*)
⟨*proof*⟩

**lemma** *transpose-matrix-add*: *transpose-matrix* $((A::('a::monoid\text{-}add)\ matrix)+B)$ $=$ *transpose-matrix* $A$ $+$ *transpose-matrix* $B$
⟨*proof*⟩

**lemma** *transpose-matrix-diff*: *transpose-matrix* $((A::('a::group\text{-}add)\ matrix)-B)$ $=$ *transpose-matrix* $A$ $-$ *transpose-matrix* $B$
⟨*proof*⟩

**lemma** *transpose-matrix-minus*: *transpose-matrix* $(-(A::('a::group\text{-}add)\ matrix))$ $= -$ *transpose-matrix* $(A::'a\ matrix)$
⟨*proof*⟩

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

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

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

**lemma** *right-inverse-matrix-dim*: *right-inverse-matrix* $A$ $X$ $\Longrightarrow$ $nrows\ A = ncols\ X$
⟨*proof*⟩

**lemma** *left-inverse-matrix-dim*: *left-inverse-matrix* $A$ $Y$ $\Longrightarrow$ $ncols\ A = nrows\ Y$
⟨*proof*⟩

**lemma** *left-right-inverse-matrix-unique*:
**assumes** *left-inverse-matrix* $A$ $Y$ *right-inverse-matrix* $A$ $X$
**shows** $X = Y$
⟨*proof*⟩

**lemma** *inverse-matrix-inject*: ⟦ *inverse-matrix* $A$ $X$; *inverse-matrix* $A$ $Y$ ⟧ $\Longrightarrow$ $X = Y$
⟨*proof*⟩

**lemma** *one-matrix-inverse*: *inverse-matrix* (*one-matrix* $n$) (*one-matrix* $n$)
⟨*proof*⟩

**lemma** *zero-imp-mult-zero*: $(a::'a::semiring\text{-}0) = 0 \mid b = 0 \Longrightarrow a * b = 0$
⟨*proof*⟩

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

⟨*proof*⟩

**lemma** *add-nrows*: *nrows* (*A*::(*'a*::*monoid-add*) *matrix*) <= *u* ⟹ *nrows B* <= *u* ⟹ *nrows* (*A* + *B*) <= *u*
⟨*proof*⟩

**lemma** *move-matrix-row-mult*: *move-matrix* ((*A*::(*'a*::*semiring-0*) *matrix*) ∗ *B*) *j* *0* = (*move-matrix A j 0*) ∗ *B*
⟨*proof*⟩

**lemma** *move-matrix-col-mult*: *move-matrix* ((*A*::(*'a*::*semiring-0*) *matrix*) ∗ *B*) *0* *i* = *A* ∗ (*move-matrix B 0 i*)
⟨*proof*⟩

**lemma** *move-matrix-add*: ((*move-matrix* (*A* + *B*) *j i*)::((*'a*::*monoid-add*) *matrix*)) = (*move-matrix A j i*) + (*move-matrix B j i*)
⟨*proof*⟩

**lemma** *move-matrix-mult*: *move-matrix* ((*A*::(*'a*::*semiring-0*) *matrix*)∗*B*) *j i* = (*move-matrix A j 0*) ∗ (*move-matrix B 0 i*)
⟨*proof*⟩

**definition** *scalar-mult* :: (*'a*::*ring*) ⇒ *'a matrix* ⇒ *'a matrix* **where**
  *scalar-mult a m* == *apply-matrix* ((∗) *a*) *m*

**lemma** *scalar-mult-zero*[*simp*]: *scalar-mult y 0* = *0*
⟨*proof*⟩

**lemma** *scalar-mult-add*: *scalar-mult y* (*a*+*b*) = (*scalar-mult y a*) + (*scalar-mult y b*)
⟨*proof*⟩

**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*⟩

**lemmas** [*code*] = *sparse-row-vector-empty* [*symmetric*]

**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*))
⟨*proof*⟩

**lemma** *sparse-row-vector-cons*[*simp*]:
*sparse-row-vector* (*a* # *arr*) = (*singleton-matrix 0* (*fst a*) (*snd a*)) + (*sparse-row-vector arr*)
⟨*proof*⟩

**lemma** *sparse-row-vector-append*[*simp*]:
*sparse-row-vector* (*a* @ *b*) = (*sparse-row-vector a*) + (*sparse-row-vector b*)
⟨*proof*⟩

**lemma** *nrows-spvec*[*simp*]: *nrows* (*sparse-row-vector x*) <= (*Suc 0*)
⟨*proof*⟩

**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*
⟨*proof*⟩

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

⟨proof⟩

**primrec** *sorted-spvec* :: *'a spvec* ⇒ *bool*
**where**
*sorted-spvec* [] = *True*
| *sorted-spvec-step*: *sorted-spvec* (*a*#*as*) = (*case as of* [] ⇒ *True* | *b*#*bs* ⇒ ((*fst a* < *fst b*) & (*sorted-spvec as*)))

**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*
⟨proof⟩

**lemma** *sorted-spvec-cons1*: *sorted-spvec* (*a*#*as*) ⟹ *sorted-spvec as*
⟨proof⟩

**lemma** *sorted-spvec-cons2*: *sorted-spvec* (*a*#*b*#*t*) ⟹ *sorted-spvec* (*a*#*t*)
⟨proof⟩

**lemma** *sorted-spvec-cons3*: *sorted-spvec*(*a*#*b*#*t*) ⟹ *fst a* < *fst b*
⟨proof⟩

**lemma** *sorted-sparse-row-vector-zero*[*rule-format*]: *m* <= *n* ⟹ *sorted-spvec* ((*n*,*a*)#*arr*) ⟶ *Rep-matrix* (*sparse-row-vector arr*) *j m* = *0*
⟨proof⟩

**lemma** *sorted-sparse-row-matrix-zero*[*rule-format*]: *m* <= *n* ⟹ *sorted-spvec* ((*n*,*a*)#*arr*) ⟶ *Rep-matrix* (*sparse-row-matrix arr*) *m j* = *0*
⟨proof⟩

**primrec** *minus-spvec* :: (*'a*::*ab-group-add*) *spvec* ⇒ *'a spvec*
**where**
*minus-spvec* [] = []
| *minus-spvec* (*a*#*as*) = (*fst a*, −(*snd a*))#(*minus-spvec as*)

**primrec** *abs-spvec* :: (*'a*::*lattice-ab-group-add-abs*) *spvec* ⇒ *'a spvec*
**where**
*abs-spvec* [] = []
| *abs-spvec* (*a*#*as*) = (*fst a*, |*snd a*|)#(*abs-spvec as*)

**lemma** *sparse-row-vector-minus*:
*sparse-row-vector* (*minus-spvec v*) = − (*sparse-row-vector v*)
⟨proof⟩

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

**lemma** *sparse-row-vector-abs*:
*sorted-spvec* ($v$ :: $'a$::*lattice-ring spvec*) $\Longrightarrow$ *sparse-row-vector* (*abs-spvec* $v$) = |*sparse-row-vector* $v$|
⟨*proof*⟩

**lemma** *sorted-spvec-minus-spvec*:
*sorted-spvec* $v$ $\Longrightarrow$ *sorted-spvec* (*minus-spvec* $v$)
⟨*proof*⟩

**lemma** *sorted-spvec-abs-spvec*:
*sorted-spvec* $v$ $\Longrightarrow$ *sorted-spvec* (*abs-spvec* $v$)
⟨*proof*⟩

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

**lemma** *smult-spvec-empty*[*simp*]: *smult-spvec* $y$ [] = []
⟨*proof*⟩

**lemma** *smult-spvec-cons*: *smult-spvec* $y$ ($a$#*arr*) = (*fst* $a$, $y$ $*$ (*snd* $a$)) # (*smult-spvec* $y$ *arr*)
⟨*proof*⟩

**fun** *addmult-spvec* :: ($'a$::*ring*) ⇒ $'a$ *spvec* ⇒ $'a$ *spvec* ⇒ $'a$ *spvec*
**where**
*addmult-spvec y arr* [] = *arr*
| *addmult-spvec y* [] *brr* = *smult-spvec y brr*
| *addmult-spvec y* ((*i*,*a*)#*arr*) ((*j*,*b*)#*brr*) = (
*if i* < *j then* ((*i*,*a*)#(*addmult-spvec y arr* ((*j*,*b*)#*brr*)))
*else* (*if* (*j* < *i*) *then* ((*j*, *y* $*$ *b*)#(*addmult-spvec y* ((*i*,*a*)#*arr*) *brr*))
*else* ((*i*, *a* + *y*$*$*b*)#(*addmult-spvec y arr brr*))))

**lemma** *addmult-spvec-empty1*[*simp*]: *addmult-spvec y* [] *a* = *smult-spvec y a*
⟨*proof*⟩

**lemma** *addmult-spvec-empty2*[*simp*]: *addmult-spvec y a* [] = *a*
⟨*proof*⟩

**lemma** *sparse-row-vector-map*: ($\forall$ $x$ $y$. $f$ ($x$+$y$) = ($f$ $x$) + ($f$ $y$)) $\Longrightarrow$ ($f$::$'a$⇒($'a$::*lattice-ring*)) *0* = *0* $\Longrightarrow$
*sparse-row-vector* (*map* (% $x$. (*fst* $x$, $f$ (*snd* $x$))) $a$) = *apply-matrix* $f$ (*sparse-row-vector* $a$)
⟨*proof*⟩

**lemma** *sparse-row-vector-smult*: *sparse-row-vector* (*smult-spvec y a*) = *scalar-mult*

$y$ (*sparse-row-vector* $a$)
⟨*proof*⟩

**lemma** *sparse-row-vector-addmult-spvec*: *sparse-row-vector* (*addmult-spvec* ($y$::$'a$::*lattice-ring*) $a$ $b$) =
(*sparse-row-vector* $a$) + (*scalar-mult* $y$ (*sparse-row-vector* $b$))
⟨*proof*⟩

**lemma** *sorted-smult-spvec*: *sorted-spvec* $a \Longrightarrow$ *sorted-spvec* (*smult-spvec* $y$ $a$)
⟨*proof*⟩

**lemma** *sorted-spvec-addmult-spvec-helper*: ⟦*sorted-spvec* (*addmult-spvec* $y$ (($a$, $b$) # *arr*) *brr*); *aa* < $a$; *sorted-spvec* (($a$, $b$) # *arr*);
*sorted-spvec* ((*aa*, *ba*) # *brr*)⟧ $\Longrightarrow$ *sorted-spvec* ((*aa*, $y$ ∗ *ba*) # *addmult-spvec* $y$ (($a$, $b$) # *arr*) *brr*)
⟨*proof*⟩

**lemma** *sorted-spvec-addmult-spvec-helper2*:
⟦*sorted-spvec* (*addmult-spvec* $y$ *arr* ((*aa*, *ba*) # *brr*)); $a$ < *aa*; *sorted-spvec* (($a$, $b$) # *arr*); *sorted-spvec* ((*aa*, *ba*) # *brr*)⟧
$\Longrightarrow$ *sorted-spvec* (($a$, $b$) # *addmult-spvec* $y$ *arr* ((*aa*, *ba*) # *brr*))
⟨*proof*⟩

**lemma** *sorted-spvec-addmult-spvec-helper3*[*rule-format*]:
*sorted-spvec* (*addmult-spvec* $y$ *arr* *brr*) ⟶ *sorted-spvec* ((*aa*, $b$) # *arr*) ⟶ *sorted-spvec* ((*aa*, *ba*) # *brr*)
⟶ *sorted-spvec* ((*aa*, $b$ + $y$ ∗ *ba*) # (*addmult-spvec* $y$ *arr* *brr*))
⟨*proof*⟩

**lemma** *sorted-addmult-spvec*: *sorted-spvec* $a \Longrightarrow$ *sorted-spvec* $b \Longrightarrow$ *sorted-spvec* (*addmult-spvec* $y$ $a$ $b$)
⟨*proof*⟩

**fun** *mult-spvec-spmat* :: ($'a$::*lattice-ring*) *spvec* ⇒ $'a$ *spvec* ⇒ $'a$ *spmat* ⇒ $'a$ *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*[*rule-format*]: *sorted-spvec* ($a$::($'a$::*lattice-ring*) *spvec*) ⟶ *sorted-spvec* $B$ ⟶
*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*[*rule-format*]:

*sorted-spvec* (*c*::(′*a*::*lattice-ring*) *spvec*) ⟶ *sorted-spmat B* ⟶ *sorted-spvec* (*mult-spvec-spmat c a B*)
⟨*proof*⟩

**primrec** *mult-spmat* :: (′*a*::*lattice-ring*) *spmat* ⇒ ′*a spmat* ⇒ ′*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* ⟹ *sorted-spvec B* ⟹
*sparse-row-matrix* (*mult-spmat A B*) = (*sparse-row-matrix A*) ∗ (*sparse-row-matrix B*)
⟨*proof*⟩

**lemma** *sorted-spvec-mult-spmat*[*rule-format*]:
*sorted-spvec* (*A*::(′*a*::*lattice-ring*) *spmat*) ⟶ *sorted-spvec* (*mult-spmat A B*)
⟨*proof*⟩

**lemma** *sorted-spmat-mult-spmat*:
*sorted-spmat* (*B*::(′*a*::*lattice-ring*) *spmat*) ⟹ *sorted-spmat* (*mult-spmat A B*)
⟨*proof*⟩

**fun** *add-spvec* :: (′*a*::*lattice-ab-group-add*) *spvec* ⇒ ′*a spvec* ⇒ ′*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*]: *add-spvec* [] *a* = *a*
⟨*proof*⟩

**lemma** *sparse-row-vector-add*: *sparse-row-vector* (*add-spvec a b*) = (*sparse-row-vector a*) + (*sparse-row-vector b*)
⟨*proof*⟩

**fun** *add-spmat* :: (′*a*::*lattice-ab-group-add*) *spmat* ⇒ ′*a spmat* ⇒ ′*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*]: *add-spmat as* [] = *as*
⟨*proof*⟩

**lemma** *sparse-row-add-spmat*: *sparse-row-matrix* (*add-spmat A B*) = (*sparse-row-matrix A*) + (*sparse-row-matrix B*)
⟨*proof*⟩

**lemmas** [*code*] = *sparse-row-add-spmat* [*symmetric*]
**lemmas** [*code*] = *sparse-row-vector-add* [*symmetric*]

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

**lemma** *sorted-add-spvec-helper*: *add-spvec arr brr* = $(ab, bb)$ # *list* $\Longrightarrow$ $((arr \neq [] \;\&\; ab = fst\ (hd\ arr)) \mid (brr \neq [] \;\&\; ab = fst\ (hd\ brr)))$
⟨*proof*⟩

**lemma** *sorted-add-spmat-helper*: *add-spmat arr brr* = $(ab, bb)$ # *list* $\Longrightarrow$ $((arr \neq [] \;\&\; ab = fst\ (hd\ arr)) \mid (brr \neq [] \;\&\; ab = fst\ (hd\ brr)))$
⟨*proof*⟩

**lemma** *add-spvec-commute*: *add-spvec a b* = *add-spvec b a*
⟨*proof*⟩

**lemma** *add-spmat-commute*: *add-spmat a b* = *add-spmat b a*
⟨*proof*⟩

**lemma** *sorted-add-spvec-helper2*: *add-spvec* $((a,b)\#arr)$ *brr* = $(ab, bb)$ # *list* $\Longrightarrow$ $aa < a \Longrightarrow$ *sorted-spvec* $((aa, ba)$ # *brr*$)$ $\Longrightarrow aa < ab$
⟨*proof*⟩

**lemma** *sorted-add-spmat-helper2*: *add-spmat* $((a,b)\#arr)$ *brr* = $(ab, bb)$ # *list* $\Longrightarrow aa < a \Longrightarrow$ *sorted-spvec* $((aa, ba)$ # *brr*$)$ $\Longrightarrow aa < ab$
⟨*proof*⟩

**lemma** *sorted-spvec-add-spvec*[*rule-format*]: *sorted-spvec a* $\longrightarrow$ *sorted-spvec b* $\longrightarrow$ *sorted-spvec* (*add-spvec a b*)
⟨*proof*⟩

**lemma** *sorted-spvec-add-spmat*[*rule-format*]: *sorted-spvec A* $\longrightarrow$ *sorted-spvec B* $\longrightarrow$ *sorted-spvec* (*add-spmat A B*)
⟨*proof*⟩

**lemma** *sorted-spmat-add-spmat*[*rule-format*]: *sorted-spmat A* $\Longrightarrow$ *sorted-spmat B* $\Longrightarrow$ *sorted-spmat* (*add-spmat A B*)
⟨*proof*⟩

**fun** *le-spvec* :: (*'a*::*lattice-ab-group-add*) *spvec* ⇒ *'a spvec* ⇒ *bool*
**where**

*le-spvec* [] [] = *True*
| *le-spvec* ((-,*a*)#*as*) [] = (*a* <= *0* & *le-spvec as* [])
| *le-spvec* [] ((-,*b*)#*bs*) = (*0* <= *b* & *le-spvec* [] *bs*)
| *le-spvec* ((*i*,*a*)#*as*) ((*j*,*b*)#*bs*) = (
*if* (*i* < *j*) *then a* <= *0* & *le-spvec as* ((*j*,*b*)#*bs*)
*else if* (*j* < *i*) *then 0* <= *b* & *le-spvec* ((*i*,*a*)#*as*) *bs*
*else a* <= *b* & *le-spvec as bs*)

**fun** *le-spmat* :: (*'a*::*lattice-ab-group-add*) *spmat* ⇒ *'a spmat* ⇒ *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* ⇒ *'a matrix* ⇒ *bool* **where**
*disj-matrices A B* $\longleftrightarrow$
($\forall$ *j i*. (*Rep-matrix A j i* $\neq$ *0*) $\longrightarrow$ (*Rep-matrix B j i* = *0*)) & ($\forall$ *j i*. (*Rep-matrix B j i* $\neq$ *0*) $\longrightarrow$ (*Rep-matrix A j i* = *0*))

**declare** [[*simp-depth-limit* = *6*]]

**lemma** *disj-matrices-contr1*: *disj-matrices A B* $\Longrightarrow$ *Rep-matrix A j i* $\neq$ *0* $\Longrightarrow$ *Rep-matrix B j i* = *0*
⟨*proof*⟩

**lemma** *disj-matrices-contr2*: *disj-matrices A B* $\Longrightarrow$ *Rep-matrix B j i* $\neq$ *0* $\Longrightarrow$ *Rep-matrix A j i* = *0*
⟨*proof*⟩

**lemma** *disj-matrices-add*: *disj-matrices A B* $\Longrightarrow$ *disj-matrices C D* $\Longrightarrow$ *disj-matrices A D* $\Longrightarrow$ *disj-matrices B C* $\Longrightarrow$

$(A + B <= C + D) = (A <= C \ \& \ B <= (D::('a::lattice\text{-}ab\text{-}group\text{-}add)\ matrix))$
⟨*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* $\Longrightarrow$
$(A + B <= 0) = (A <= 0 \ \& \ (B::('a::lattice\text{-}ab\text{-}group\text{-}add)\ matrix) <= 0)$
⟨*proof*⟩

**lemma** *disj-matrices-add-zero-le*: *disj-matrices A B* $\Longrightarrow$
$(0 <= A + B) = (0 <= A \ \& \ 0 <= (B::('a::lattice\text{-}ab\text{-}group\text{-}add)\ matrix))$
⟨*proof*⟩

**lemma** *disj-matrices-add-x-le*: *disj-matrices A B* $\Longrightarrow$ *disj-matrices B C* $\Longrightarrow$
$(A <= B + C) = (A <= C \ \& \ 0 <= (B::('a::lattice\text{-}ab\text{-}group\text{-}add)\ matrix))$
⟨*proof*⟩

**lemma** *disj-matrices-add-le-x*: *disj-matrices A B* $\Longrightarrow$ *disj-matrices B C* $\Longrightarrow$
$(B + A <= C) = (A <= C \ \& \ (B::('a::lattice\text{-}ab\text{-}group\text{-}add)\ matrix) <= 0)$
⟨*proof*⟩

**lemma** *disj-sparse-row-singleton*: $i <= j \Longrightarrow$ *sorted-spvec*$((j,y)\#v) \Longrightarrow$ *disj-matrices* (*sparse-row-vector v*) (*singleton-matrix 0 i x*)
⟨*proof*⟩

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

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

**lemma** *disj-singleton-matrices* [*simp*]: *disj-matrices* (*singleton-matrix j i x*) (*singleton-matrix u v y*) $= (j \neq u \mid i \neq v \mid x = 0 \mid y = 0)$
⟨*proof*⟩

**lemma** *disj-move-sparse-vec-mat* [*simplified disj-matrices-commute*]:
$j <= a \Longrightarrow$ *sorted-spvec*$((a,c)\#as) \Longrightarrow$ *disj-matrices* (*move-matrix* (*sparse-row-vector b*) (*int j*) *i*) (*sparse-row-matrix as*)
⟨*proof*⟩

**lemma** *disj-move-sparse-row-vector-twice*:
  $j \neq u \Longrightarrow$ *disj-matrices* (*move-matrix* (*sparse-row-vector* $a$) $j$ $i$) (*move-matrix* (*sparse-row-vector* $b$) $u$ $v$)
  ⟨*proof*⟩

**lemma** *le-spvec-iff-sparse-row-le*[*rule-format*]: (*sorted-spvec* $a$) $\longrightarrow$ (*sorted-spvec* $b$) $\longrightarrow$ (*le-spvec* $a$ $b$) = (*sparse-row-vector* $a$ <= *sparse-row-vector* $b$)
  ⟨*proof*⟩

**lemma** *le-spvec-empty2-sparse-row*[*rule-format*]: *sorted-spvec* $b$ $\longrightarrow$ *le-spvec* $b$ [] = (*sparse-row-vector* $b$ <= *0*)
  ⟨*proof*⟩

**lemma** *le-spvec-empty1-sparse-row*[*rule-format*]: (*sorted-spvec* $b$) $\longrightarrow$ (*le-spvec* [] $b$ = (*0* <= *sparse-row-vector* $b$))
  ⟨*proof*⟩

**lemma** *le-spmat-iff-sparse-row-le*[*rule-format*]: (*sorted-spvec* $A$) $\longrightarrow$ (*sorted-spmat* $A$) $\longrightarrow$ (*sorted-spvec* $B$) $\longrightarrow$ (*sorted-spmat* $B$) $\longrightarrow$
  *le-spmat* $A$ $B$ = (*sparse-row-matrix* $A$ <= *sparse-row-matrix* $B$)
  ⟨*proof*⟩

**declare** [[*simp-depth-limit* = *999*]]

**primrec** *abs-spmat* :: ('*a*::*lattice-ring*) *spmat* ⇒ '*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* ⇒ '*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*: $x \neq 0 \Longrightarrow$ *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*: (¬ *True*) = *False* ⟨*proof*⟩
**lemma** *bool2*: (¬ *False*) = *True* ⟨*proof*⟩
**lemma** *bool3*: ((*P*::*bool*) ∧ *True*) = *P* ⟨*proof*⟩
**lemma** *bool4*: (*True* ∧ (*P*::*bool*)) = *P* ⟨*proof*⟩
**lemma** *bool5*: ((*P*::*bool*) ∧ *False*) = *False* ⟨*proof*⟩
**lemma** *bool6*: (*False* ∧ (*P*::*bool*)) = *False* ⟨*proof*⟩
**lemma** *bool7*: ((*P*::*bool*) ∨ *True*) = *True* ⟨*proof*⟩

**lemma** *bool8*: (*True* ∨ (*P*::*bool*)) = *True* ⟨*proof*⟩
**lemma** *bool9*: ((*P*::*bool*) ∨ *False*) = *P* ⟨*proof*⟩
**lemma** *bool10*: (*False* ∨ (*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* => *x* | *False* => *y*)
⟨*proof*⟩

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

**primrec** *nprt-spvec* :: (′*a*::{*lattice-ab-group-add*}) *spvec* ⇒ ′*a spvec*
**where**
*nprt-spvec* [] = []
| *nprt-spvec* (*a*#*as*) = (*fst a*, *nprt* (*snd a*)) # (*nprt-spvec as*)

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

**primrec** *nprt-spmat* :: (′*a*::{*lattice-ab-group-add*}) *spmat* ⇒ ′*a spmat*
**where**
*nprt-spmat* [] = []
| *nprt-spmat* (*a*#*as*) = (*fst a*, *nprt-spvec* (*snd a*))#(*nprt-spmat as*)

**lemma** *pprt-add*: *disj-matrices A* (*B*::(-::*lattice-ring*) *matrix*) ⟹ *pprt* (*A*+*B*) = *pprt A* + *pprt B*
⟨*proof*⟩

**lemma** *nprt-add*: *disj-matrices A* (*B*::(-::*lattice-ring*) *matrix*) ⟹ *nprt* (*A*+*B*) = *nprt A* + *nprt B*
⟨*proof*⟩

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

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

**lemma** *less-imp-le*: *a* < *b* ⟹ *a* <= (*b*::-::*order*) ⟨*proof*⟩

**lemma** *sparse-row-vector-pprt*: *sorted-spvec* (*v* :: ′*a*::*lattice-ring spvec*) ⟹ *sparse-row-vector* (*pprt-spvec v*) = *pprt* (*sparse-row-vector v*)

$\langle proof \rangle$

**lemma** *sparse-row-vector-nprt*: *sorted-spvec* ($v$ :: $'a$::*lattice-ring spvec*) $\Longrightarrow$ *sparse-row-vector* (*nprt-spvec* $v$) = *nprt* (*sparse-row-vector* $v$)
$\langle proof \rangle$

**lemma** *pprt-move-matrix*: *pprt* (*move-matrix* ($A$::($'a$::*lattice-ring*) *matrix*) $j$ $i$) = *move-matrix* (*pprt* $A$) $j$ $i$
$\langle proof \rangle$

**lemma** *nprt-move-matrix*: *nprt* (*move-matrix* ($A$::($'a$::*lattice-ring*) *matrix*) $j$ $i$) = *move-matrix* (*nprt* $A$) $j$ $i$
$\langle proof \rangle$

**lemma** *sparse-row-matrix-pprt*: *sorted-spvec* ($m$ :: $'a$::*lattice-ring spmat*) $\Longrightarrow$ *sorted-spmat* $m$ $\Longrightarrow$ *sparse-row-matrix* (*pprt-spmat* $m$) = *pprt* (*sparse-row-matrix* $m$)
$\langle proof \rangle$

**lemma** *sparse-row-matrix-nprt*: *sorted-spvec* ($m$ :: $'a$::*lattice-ring spmat*) $\Longrightarrow$ *sorted-spmat* $m$ $\Longrightarrow$ *sparse-row-matrix* (*nprt-spmat* $m$) = *nprt* (*sparse-row-matrix* $m$)
$\langle proof \rangle$

**lemma** *sorted-pprt-spvec*: *sorted-spvec* $v$ $\Longrightarrow$ *sorted-spvec* (*pprt-spvec* $v$)
$\langle proof \rangle$

**lemma** *sorted-nprt-spvec*: *sorted-spvec* $v$ $\Longrightarrow$ *sorted-spvec* (*nprt-spvec* $v$)
$\langle proof \rangle$

**lemma** *sorted-spvec-pprt-spmat*: *sorted-spvec* $m$ $\Longrightarrow$ *sorted-spvec* (*pprt-spmat* $m$)
$\langle proof \rangle$

**lemma** *sorted-spvec-nprt-spmat*: *sorted-spvec* $m$ $\Longrightarrow$ *sorted-spvec* (*nprt-spmat* $m$)
$\langle proof \rangle$

**lemma** *sorted-spmat-pprt-spmat*: *sorted-spmat* $m$ $\Longrightarrow$ *sorted-spmat* (*pprt-spmat* $m$)
$\langle proof \rangle$

**lemma** *sorted-spmat-nprt-spmat*: *sorted-spmat* $m$ $\Longrightarrow$ *sorted-spmat* (*nprt-spmat* $m$)
$\langle proof \rangle$

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

*s1*) (*nprt-spmat r1*))))

**lemmas** *sparse-row-matrix-op-simps* =
*sorted-sparse-matrix-imp-spmat sorted-sparse-matrix-imp-spvec*
*sparse-row-add-spmat sorted-spvec-add-spmat sorted-spmat-add-spmat*
*sparse-row-diff-spmat sorted-spvec-diff-spmat sorted-spmat-diff-spmat*
*sparse-row-matrix-minus sorted-spvec-minus-spmat sorted-spmat-minus-spmat*
*sparse-row-mult-spmat sorted-spvec-mult-spmat sorted-spmat-mult-spmat*
*sparse-row-matrix-abs sorted-spvec-abs-spmat sorted-spmat-abs-spmat*
*le-spmat-iff-sparse-row-le*
*sparse-row-matrix-pprt sorted-spvec-pprt-spmat sorted-spmat-pprt-spmat*
*sparse-row-matrix-nprt sorted-spvec-nprt-spmat 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*
*nprt-spmat.simps nprt-spvec.simps*
*mult-est-spmat-def*

**end**

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

**lemma** *le-add-right-mono*:
**assumes**
$a <= b + (c::'a::ordered\text{-}ab\text{-}group\text{-}add)$
$c <= d$
**shows** $a <= b + d$
⟨*proof*⟩

**lemma** *linprog-dual-estimate*:
**assumes**
$A * x \leq (b::'a::lattice\text{-}ring)$
$0 \leq y$
$|A - A'| \leq \delta\text{-}A$
$b \leq b'$

$|c - c'| \leq \delta\text{-}c$
$|x| \leq r$
**shows**
$c * x \leq y * b' + (y * \delta\text{-}A + |y * A' - c'| + \delta\text{-}c) * r$
⟨*proof*⟩

**lemma** *le-ge-imp-abs-diff-1*:
**assumes**
*A1 <= (A::'a::lattice-ring)*
*A <= A2*
**shows** *|A−A1| <= A2−A1*
⟨*proof*⟩

**lemma** *mult-le-prts*:
**assumes**
*a1 <= (a::'a::lattice-ring)*
*a <= a2*
*b1 <= b*
*b <= b2*
**shows**
*a * b <= pprt a2 * pprt b2 + pprt a1 * nprt b2 + nprt a2 * pprt b1 + nprt a1 * nprt b1*
⟨*proof*⟩

**lemma** *mult-le-dual-prts*:
**assumes**
$A * x \leq (b::'a::lattice\text{-}ring)$
$0 \leq y$
$A1 \leq A$
$A \leq A2$
$c1 \leq c$
$c \leq c2$
$r1 \leq x$
$x \leq r2$
**shows**
*c * x ≤ y * b + (let s1 = c1 − y * A2; s2 = c2 − y * A1 in pprt s2 * pprt r2 + pprt s1 * nprt r2 + nprt s2 * pprt r1 + nprt s1 * nprt 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* $x$ = ($\exists$ ($u$::*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*]: *int-of-real* (*real-of-int* $x$) = $x$
⟨*proof*⟩

**lemma** *real-int-of-real*[*simp*]: *real-is-int* $x$ $\Longrightarrow$ *real-of-int* (*int-of-real* $x$) = $x$
⟨*proof*⟩

**lemma** *real-is-int-add-int-of-real*: *real-is-int* $a$ $\Longrightarrow$ *real-is-int* $b$ $\Longrightarrow$ (*int-of-real* ($a$+$b$)) = (*int-of-real* $a$) + (*int-of-real* $b$)
⟨*proof*⟩

**lemma** *real-is-int-add*[*simp*]: *real-is-int* $a$ $\Longrightarrow$ *real-is-int* $b$ $\Longrightarrow$ *real-is-int* ($a$+$b$)
⟨*proof*⟩

**lemma** *int-of-real-sub*: *real-is-int* $a$ $\Longrightarrow$ *real-is-int* $b$ $\Longrightarrow$ (*int-of-real* ($a$−$b$)) = (*int-of-real* $a$) − (*int-of-real* $b$)
⟨*proof*⟩

**lemma** *real-is-int-sub*[*simp*]: *real-is-int* $a$ $\Longrightarrow$ *real-is-int* $b$ $\Longrightarrow$ *real-is-int* ($a$−$b$)
⟨*proof*⟩

**lemma** *real-is-int-rep*: *real-is-int* $x$ $\Longrightarrow$ $\exists$!($a$::*int*). *real-of-int* $a$ = $x$
⟨*proof*⟩

**lemma** *int-of-real-mult*:
  **assumes** *real-is-int* $a$ *real-is-int* $b$
  **shows** (*int-of-real* ($a$∗$b$)) = (*int-of-real* $a$) ∗ (*int-of-real* $b$)
  ⟨*proof*⟩

**lemma** *real-is-int-mult*[*simp*]: *real-is-int* $a$ $\Longrightarrow$ *real-is-int* $b$ $\Longrightarrow$ *real-is-int* ($a$∗$b$)
⟨*proof*⟩

**lemma** *real-is-int-0*[*simp*]: *real-is-int* (*0*::*real*)
⟨*proof*⟩

**lemma** *real-is-int-1*[*simp*]: *real-is-int* (*1*::*real*)
⟨*proof*⟩

**lemma** *real-is-int-n1*: *real-is-int* ($-$*1*::*real*)
⟨*proof*⟩

**lemma** *real-is-int-numeral*[*simp*]: *real-is-int* (*numeral x*)
⟨*proof*⟩

**lemma** *real-is-int-neg-numeral*[*simp*]: *real-is-int* ($-$ *numeral x*)
⟨*proof*⟩

**lemma** *int-of-real-0*[*simp*]: *int-of-real* (*0*::*real*) = (*0*::*int*)
⟨*proof*⟩

**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*: $a \neq 0 \Longrightarrow a \neq -1 \Longrightarrow |(a::int)\ div\ 2| < |a|$
⟨*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*

$\langle proof \rangle$

**lemma** *int-pow-1*: $(a::int)\,\hat{}(Numeral1) = a$
$\langle proof \rangle$

**lemma** *one-eq-Numeral1-nring*: $(1::'a::numeral) = Numeral1$
$\langle proof \rangle$

**lemma** *one-eq-Numeral1-nat*: $(1::nat) = Numeral1$
$\langle proof \rangle$

**lemma** *zpower-Pls*: $(z::int)\,\hat{}0 = Numeral1$
$\langle proof \rangle$

**lemma** *fst-cong*: $a=a' \Longrightarrow fst\ (a,b) = fst\ (a',b)$
$\langle proof \rangle$

**lemma** *snd-cong*: $b=b' \Longrightarrow snd\ (a,b) = snd\ (a,b')$
$\langle proof \rangle$

**lemma** *lift-bool*: $x \Longrightarrow x=True$
$\langle proof \rangle$

**lemma** *nlift-bool*: $\sim x \Longrightarrow x=False$
$\langle proof \rangle$

**lemma** *not-false-eq-true*: $(\sim\ False) = True$ $\langle proof \rangle$

**lemma** *not-true-eq-false*: $(\sim\ True) = False$ $\langle proof \rangle$

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

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

**lemma** *float-add-l0*: $float\ (0,\ e) + x = x$
$\langle proof \rangle$

**lemma** *float-add-r0*: $x + float\ (0,\ e) = x$
$\langle proof \rangle$

**lemma** *float-add*:
$float\ (a1,\ e1) + float\ (a2,\ e2) =$
$(if\ e1<=e2\ then\ float\ (a1+a2*2\,\hat{}(nat(e2-e1)),\ e1)\ else\ float\ (a1*2\,\hat{}(nat\ (e1-e2))+a2,$
$e2))$
$\langle proof \rangle$

**lemma** *float-mult-l0*: $float\ (0,\ e) * x = float\ (0,\ 0)$

⟨proof⟩

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

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

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

**lemma** *zero-le-float*:
$(0 <= float\ (a,b)) = (0 <= a)$
⟨proof⟩

**lemma** *float-le-zero*:
$(float\ (a,b) <= 0) = (a <= 0)$
⟨proof⟩

**lemma** *float-abs*:
$|float\ (a,b)| = (if\ 0 <= a\ then\ (float\ (a,b))\ else\ (float\ (-a,b)))$
⟨proof⟩

**lemma** *float-zero*:
$float\ (0,\ b) = 0$
⟨proof⟩

**lemma** *float-pprt*:
$pprt\ (float\ (a,\ b)) = (if\ 0 <= a\ then\ (float\ (a,b))\ else\ (float\ (0,\ b)))$
⟨proof⟩

**lemma** *float-nprt*:
$nprt\ (float\ (a,\ b)) = (if\ 0 <= a\ then\ (float\ (0,b))\ else\ (float\ (a,\ b)))$
⟨proof⟩

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

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

**lemma** *lbound*: $lbound\ x \leq x$
⟨proof⟩

**lemma** *ubound*: $x \leq ubound\ x$
⟨proof⟩

**lemma** *pprt-lbound*: *pprt* (*lbound* $x$) = *float* (*0*, *0*)
⟨*proof*⟩

**lemma** *nprt-ubound*: *nprt* (*ubound* $x$) = *float* (*0*, *0*)
⟨*proof*⟩

**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 nprt-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*

⟨*ML*⟩

**end**

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

⟨*ML*⟩

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

**lemma** *Trueprop-eq-eq*: *Trueprop* $X$ == ($X$ == *True*) ⟨*proof*⟩
**lemma** *meta-eq-trivial*: $x == y \Longrightarrow x == y$ ⟨*proof*⟩
**lemma** *meta-eq-imp-eq*: $x == y \Longrightarrow x = y$ ⟨*proof*⟩
**lemma** *eq-trivial*: $x = y \Longrightarrow x = y$ ⟨*proof*⟩
**lemma** *bool-to-true*: $x$ :: *bool* $\Longrightarrow x ==$ *True* ⟨*proof*⟩
**lemma** *transmeta-1*: $x = y \Longrightarrow y == z \Longrightarrow x = z$ ⟨*proof*⟩
**lemma** *transmeta-2*: $x == y \Longrightarrow y = z \Longrightarrow x = z$ ⟨*proof*⟩
**lemma** *transmeta-3*: $x == y \Longrightarrow y == z \Longrightarrow x = z$ ⟨*proof*⟩

**lemma** *If-True*: *If True* = ($\lambda$ $x$ $y$. $x$) ⟨*proof*⟩
**lemma** *If-False*: *If False* = ($\lambda$ $x$ $y$. $y$) ⟨*proof*⟩

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

**lemma** *bool1*: ($\neg$ *True*) = *False* ⟨*proof*⟩

**lemma** *bool2*: $(\neg\ False) = True$ ⟨*proof*⟩
**lemma** *bool3*: $(P \wedge True) = P$ ⟨*proof*⟩
**lemma** *bool4*: $(True \wedge P) = P$ ⟨*proof*⟩
**lemma** *bool5*: $(P \wedge False) = False$ ⟨*proof*⟩
**lemma** *bool6*: $(False \wedge P) = False$ ⟨*proof*⟩
**lemma** *bool7*: $(P \vee True) = True$ ⟨*proof*⟩
**lemma** *bool8*: $(True \vee P) = True$ ⟨*proof*⟩
**lemma** *bool9*: $(P \vee False) = P$ ⟨*proof*⟩
**lemma** *bool10*: $(False \vee P) = P$ ⟨*proof*⟩
**lemma** *bool11*: $(True \longrightarrow P) = P$ ⟨*proof*⟩
**lemma** *bool12*: $(P \longrightarrow True) = True$ ⟨*proof*⟩
**lemma** *bool13*: $(True \longrightarrow P) = P$ ⟨*proof*⟩
**lemma** *bool14*: $(P \longrightarrow False) = (\neg\ P)$ ⟨*proof*⟩
**lemma** *bool15*: $(False \longrightarrow P) = True$ ⟨*proof*⟩
**lemma** *bool16*: $(False = False) = True$ ⟨*proof*⟩
**lemma** *bool17*: $(True = True) = True$ ⟨*proof*⟩
**lemma** *bool18*: $(False = True) = False$ ⟨*proof*⟩
**lemma** *bool19*: $(True = False) = False$ ⟨*proof*⟩

**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* $(x,y) = x$ ⟨*proof*⟩
**lemma** *compute-snd*: *snd* $(x,y) = y$ ⟨*proof*⟩
**lemma** *compute-pair-eq*: $((a,\ b) = (c,\ d)) = (a = c \wedge b = d)$ ⟨*proof*⟩

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

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

**lemma** *compute-the*: *the* $(Some\ x) = x$ ⟨*proof*⟩
**lemma** *compute-None-Some-eq*: $(None = Some\ x) = False$ ⟨*proof*⟩
**lemma** *compute-Some-None-eq*: $(Some\ x = None) = False$ ⟨*proof*⟩
**lemma** *compute-None-None-eq*: $(None = None) = True$ ⟨*proof*⟩
**lemma** *compute-Some-Some-eq*: $(Some\ x = Some\ y) = (x = y)$ ⟨*proof*⟩

**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*)
  ⟨*proof*⟩

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

⟨proof⟩

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

**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* $xs$)
⟨proof⟩

**lemma** *length-nil*: *length* [] = *0*
⟨proof⟩

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

**definition** *case-list-compute* :: $'b$ *list* $\Rightarrow$ $'a$ $\Rightarrow$ ($'b$ $\Rightarrow$ $'b$ *list* $\Rightarrow$ $'a$) $\Rightarrow$ $'a$
**where** *case-list-compute* $l$ $a$ $f$ = *case-list* $a$ $f$ $l$

**lemma** *case-list-compute*: *case-list* = ($\lambda$ ($a$::$'a$) $f$ ($l$::$'b$ *list*). *case-list-compute* $l$ $a$ $f$)
⟨proof⟩

**lemma** *case-list-compute-empty*: *case-list-compute* ([]::$'b$ *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*: (($x$#$xs$) ! $n$) = (*if* $n$ = *0* *then* $x$ *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 Divides.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-eq fst-conv snd-conv numeral-One*
*case-prod-beta rel-simps Divides.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*) (*nprt-spmat r2*))
  (*add-spmat* (*mult-spmat* (*nprt-spmat s2*) (*pprt-spmat r1*)) (*mult-spmat* (*nprt-spmat s1*) (*nprt-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*

$A * x \leq$ *sparse-row-matrix* (*b*::('*a*::*lattice-ring*) *spmat*)
**shows**
$c * x \leq$ *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**