

Executable Matrix Operations on Matrices of Arbitrary Dimensions

Christian Sternagel and René Thiemann

December 14, 2021

Abstract

We provide the operations of matrix addition, multiplication, transposition, and matrix comparisons as executable functions over ordered semirings. Moreover, it is proven that strongly normalizing (monotone) orders can be lifted to strongly normalizing (monotone) orders over matrices.

We further show that the standard semirings over the naturals, integers, and rationals, as well as the arctic semirings satisfy the axioms that are required by our matrix theory.

Our formalization was performed as part of the `IsaFoR/CeTA`-system [3]¹ which contains several termination techniques. The provided theories have been essential to formalize matrix-interpretations [1] and arctic interpretations [2]. A short description of this formalization can be found in [4].

Contents

1	Utility Functions and Lemmas	2
1.1	Miscellaneous	2
1.2	A connection between class based semirings and set based semirings	5
2	Basic Operations on Matrices	6
2.1	types and well-formedness of vectors / matrices	6
2.2	definitions / algorithms	6
2.3	algorithms preserve dimensions	8
2.4	properties of algorithms which do not depend on properties of type of matrix	10
2.5	lemmas requiring properties of plus, times,	14
2.6	Connection to HOL-Algebra	18

¹<http://cl-informatik.uibk.ac.at/software/ceta>

1 Utility Functions and Lemmas

```
theory Utility
imports Main
begin
```

1.1 Miscellaneous

```
lemma ballI2[Pure.intro]:
  assumes  $\bigwedge x y. (x, y) \in A \implies P x y$ 
  shows  $\forall (x, y) \in A. P x y$ 
  <proof>
```

```
lemma infinite-imp-elem:  $\neg \text{finite } A \implies \exists x. x \in A$ 
  <proof>
```

```
lemma infinite-imp-many-elems:
  infinite A  $\implies \exists xs. \text{set } xs \subseteq A \wedge \text{length } xs = n \wedge \text{distinct } xs$ 
  <proof>
```

```
lemma inf-pigeonhole-principle:
  assumes  $\forall k :: \text{nat}. \exists i < n :: \text{nat}. f k i$ 
  shows  $\exists i < n. \forall k. \exists k' \geq k. f k' i$ 
  <proof>
```

```
lemma map-upt-Suc:  $\text{map } f [0 ..< \text{Suc } n] = f 0 \# \text{map } (\lambda i. f (\text{Suc } i)) [0 ..< n]$ 
  <proof>
```

```
lemma map-upt-add:  $\text{map } f [0 ..< n + m] = \text{map } f [0 ..< n] @ \text{map } (\lambda i. f (i + n)) [0 ..< m]$ 
  <proof>
```

```
lemma map-upt-split: assumes  $i: i < n$ 
  shows  $\text{map } f [0 ..< n] = \text{map } f [0 ..< i] @ f i \# \text{map } (\lambda j. f (j + \text{Suc } i)) [0 ..< n - \text{Suc } i]$ 
  <proof>
```

```
lemma all-Suc-conv:
   $(\forall i < \text{Suc } n. P i) \longleftrightarrow P 0 \wedge (\forall i < n. P (\text{Suc } i))$  (is ?l = ?r)
  <proof>
```

```
lemma ex-Suc-conv:
   $(\exists i < \text{Suc } n. P i) \longleftrightarrow P 0 \vee (\exists i < n. P (\text{Suc } i))$  (is ?l = ?r)
  <proof>
```

```
fun sorted-list-subset :: 'a :: linorder list  $\Rightarrow$  'a list  $\Rightarrow$  'a option where
  sorted-list-subset (a # as) (b # bs) =
    (if a = b then sorted-list-subset as (b # bs)
```

else if $a > b$ then *sorted-list-subset* ($a \# as$) bs
 else *Some a*)
 | *sorted-list-subset* [] - = *None*
 | *sorted-list-subset* ($a \# -$) [] = *Some a*

lemma *sorted-list-subset*:

assumes *sorted as and sorted bs*

shows (*sorted-list-subset as bs* = *None*) = (*set as* \subseteq *set bs*)

<proof>

lemma *zip-nth-conv*: $length\ xs = length\ ys \implies zip\ xs\ ys = map\ (\lambda\ i.\ (xs\ !\ i,\ ys\ !\ i))\ [0\ ..<\ length\ ys]$

<proof>

lemma *nth-map-conv*:

assumes $length\ xs = length\ ys$

and $\forall i < length\ xs.\ f\ (xs\ !\ i) = g\ (ys\ !\ i)$

shows $map\ f\ xs = map\ g\ ys$

<proof>

lemma *sum-list-0*: $\llbracket \bigwedge x.\ x \in set\ xs \implies x = 0 \rrbracket \implies sum-list\ xs = 0$

<proof>

lemma *foldr-foldr-concat*: $foldr\ (foldr\ f)\ m\ a = foldr\ f\ (concat\ m)\ a$

<proof>

lemma *sum-list-double-concat*:

fixes $f :: 'b \Rightarrow 'c \Rightarrow 'a :: comm-monoid-add$ **and** $g\ as\ bs$

shows $sum-list\ (concat\ (map\ (\lambda\ i.\ map\ (\lambda\ j.\ f\ i\ j + g\ i\ j)\ as)\ bs))$

$= sum-list\ (concat\ (map\ (\lambda\ i.\ map\ (\lambda\ j.\ f\ i\ j)\ as)\ bs)) +$

$sum-list\ (concat\ (map\ (\lambda\ i.\ map\ (\lambda\ j.\ g\ i\ j)\ as)\ bs))$

<proof>

fun *max-list* :: $nat\ list \Rightarrow nat$ **where**

$max-list\ [] = 0$

| $max-list\ (x \# xs) = max\ x\ (max-list\ xs)$

lemma *max-list*: $x \in set\ xs \implies x \leq max-list\ xs$

<proof>

lemma *max-list-mem*: $xs \neq [] \implies max-list\ xs \in set\ xs$

<proof>

lemma *max-list-set*: $max-list\ xs = (if\ set\ xs = \{\}\ then\ 0\ else\ (THE\ x.\ x \in set\ xs \wedge (\forall\ y \in set\ xs.\ y \leq x)))$

<proof>

lemma *max-list-eq-set*: $set\ xs = set\ ys \implies max-list\ xs = max-list\ ys$

<proof>

lemma *all-less-two*: $(\forall i < \text{Suc } (\text{Suc } 0). P i) = (P 0 \wedge P (\text{Suc } 0))$ (**is** ?l = ?r)
 <proof>

Induction over a finite set of natural numbers.

lemma *bound-nat-induct*[consumes 1]:
assumes $n \in \{l..u\}$ **and** $P l$ **and** $\bigwedge n. \llbracket P n; n \in \{l..<u\} \rrbracket \implies P (\text{Suc } n)$
shows $P n$
 <proof>

end

theory *Ordered-Semiring*

imports

HOL-Algebra.Ring

Abstract-Rewriting.SN-Orders

begin

record *'a ordered-semiring* = *'a ring* +
geq :: *'a* \Rightarrow *'a* \Rightarrow *bool* (**infix** \succeq_1 50)
gt :: *'a* \Rightarrow *'a* \Rightarrow *bool* (**infix** \succ_1 50)
max :: *'a* \Rightarrow *'a* \Rightarrow *'a* (*Max*)

lemmas *ordered-semiring-record-simps* = *ring-record-simps ordered-semiring.simps*

locale *ordered-semiring* = *semiring* +

assumes *compat*: $\llbracket s \succeq (t :: 'a); t \succ u; s \in \text{carrier } R; t \in \text{carrier } R; u \in \text{carrier } R \rrbracket \implies s \succ u$

and *compat2*: $\llbracket s \succ (t :: 'a); t \succeq u; s \in \text{carrier } R; t \in \text{carrier } R; u \in \text{carrier } R \rrbracket \implies s \succ u$

and *plus-left-mono*: $\llbracket x \succeq y; x \in \text{carrier } R; y \in \text{carrier } R; z \in \text{carrier } R \rrbracket \implies x \oplus z \succeq y \oplus z$

and *times-left-mono*: $\llbracket z \succeq \mathbf{0}; x \succeq y; x \in \text{carrier } R; y \in \text{carrier } R; z \in \text{carrier } R \rrbracket \implies x \otimes z \succeq y \otimes z$

and *times-right-mono*: $\llbracket x \succeq \mathbf{0}; y \succeq z; x \in \text{carrier } R; y \in \text{carrier } R; z \in \text{carrier } R \rrbracket \implies x \otimes y \succeq x \otimes z$

and *geq-refl*: $x \in \text{carrier } R \implies x \succeq x$

and *geq-trans*[*trans*]: $\llbracket x \succeq y; y \succeq z; x \in \text{carrier } R; y \in \text{carrier } R; z \in \text{carrier } R \rrbracket \implies x \succeq z$

and *gt-trans*[*trans*]: $\llbracket x \succ y; y \succ z; x \in \text{carrier } R; y \in \text{carrier } R; z \in \text{carrier } R \rrbracket \implies x \succ z$

and *gt-imp-ge*: $x \succ y \implies x \in \text{carrier } R \implies y \in \text{carrier } R \implies x \succeq y$

and *max-comm*: $x \in \text{carrier } R \implies y \in \text{carrier } R \implies \text{Max } x y = \text{Max } y x$

and *max-ge*: $x \in \text{carrier } R \implies y \in \text{carrier } R \implies \text{Max } x y \succeq x$

and *max-id*: $x \in \text{carrier } R \implies y \in \text{carrier } R \implies x \succeq y \implies \text{Max } x y = x$

and *max-mono*: $x \succeq y \implies x \in \text{carrier } R \implies y \in \text{carrier } R \implies z \in \text{carrier } R \implies \text{Max } z x \succeq \text{Max } z y$

and *wf-max*[*simp*, *intro*]: $x \in \text{carrier } R \implies y \in \text{carrier } R \implies \text{Max } x \ y \in \text{carrier } R$
and *one-geq-zero*: $1 \succeq 0$
begin
lemma *max-ge-right*: **assumes** $x: x \in \text{carrier } R$ **and** $y: y \in \text{carrier } R$ **shows** $\text{Max } x \ y \succeq y$
 ⟨*proof*⟩

lemma *wf-max0*: $x \in \text{carrier } R \implies \text{Max } 0 \ x \in \text{carrier } R$ ⟨*proof*⟩

lemma *max0-id-pos*: **assumes** $x: x \succeq 0$ **and** *wf*: $x \in \text{carrier } R$
 shows $\text{Max } 0 \ x = x$ ⟨*proof*⟩
end
hide-const (**open**) *gt geq max*

1.2 A connection between class based semirings and set based semirings

definition *class-semiring* :: $'a \text{ itself} \Rightarrow 'b \Rightarrow ('a :: \{\text{plus, times, one, zero}\}, 'b) \text{ ring-scheme}$
where
 class-semiring - $b \equiv (\mid \text{carrier} = \text{UNIV}, \text{mult} = (*), \text{one} = 1, \text{zero} = 0, \text{add} = (+), \dots = b)$

lemma *class-semiring: semiring* (*class-semiring* ($\text{TYPE}('a :: \text{ordered-semiring-1})$) b)
 ⟨*proof*⟩

definition *class-ordered-semiring* :: $'a \text{ itself} \Rightarrow ('a :: \text{ordered-semiring-1} \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow 'b \Rightarrow ('a, 'b) \text{ ordered-semiring-scheme}$ **where**
 class-ordered-semiring $a \text{ gt } b \equiv \text{class-semiring } a \ (\mid$
 ordered-semiring.geq = (\geq) ,
 gt = *gt*,
 max = *max*,
 $\dots = b)$

lemma *class-ordered-semiring: assumes order-pair* (*gt* :: $('a :: \text{ordered-semiring-1} \Rightarrow 'a \Rightarrow \text{bool})$) d
 shows *ordered-semiring*
 (*class-ordered-semiring* ($\text{TYPE}('a)$) *gt* b)
 (**is** *ordered-semiring* ? R)
 ⟨*proof*⟩

lemma (**in** *one-mono-ordered-semiring-1*) *class-ordered-semiring:*
 ordered-semiring
 (*class-ordered-semiring* ($\text{TYPE}('a)$) (\succ) b)
 ⟨*proof*⟩

lemma (**in** *both-mono-ordered-semiring-1*) *class-ordered-semiring:*
 ordered-semiring

```

    (class-ordered-semiring (TYPE('a)) (>) b)
  <proof>

```

end

2 Basic Operations on Matrices

theory *Matrix-Legacy*

imports

Utility

Ordered-Semiring

begin

This theory is marked as legacy, since there is a better implementation of matrices available in `../Jordan_Normal_Form/Matrix.thy`. That formalization is more abstract, more complete in terms of operations, and it still provides an efficient implementation.

This theory provides the operations of matrix addition, multiplication, and transposition as executable functions. Most properties are proven via pointwise equality of matrices.

2.1 types and well-formedness of vectors / matrices

type-synonym *'a vec* = *'a list*

type-synonym *'a mat* = *'a vec list*

definition *vec* :: *nat* \Rightarrow *'x vec* \Rightarrow *bool*

where *vec* *n* *x* = (*length* *x* = *n*)

definition *mat* :: *nat* \Rightarrow *nat* \Rightarrow *'a mat* \Rightarrow *bool* **where**

mat *nr* *nc* *m* = (*length* *m* = *nc* \wedge *Ball* (*set* *m*) (*vec* *nr*))

2.2 definitions / algorithms

note that these algorithms are generic in all basic definitions / operations like 0 (ze) 1 (on) addition (pl) multiplication (ti) and in the dimension(s) of the matrix/vector. Hence, many of these algorithms require these definitions/operations/sizes as arguments. All indices start from 0.

definition *vec0I* :: *'a* \Rightarrow *nat* \Rightarrow *'a vec* **where**

vec0I *ze* *n* = *replicate* *n* *ze*

definition *mat0I* :: *'a* \Rightarrow *nat* \Rightarrow *nat* \Rightarrow *'a mat* **where**

$mat0I\ ze\ nr\ nc = replicate\ nc\ (vec0I\ ze\ nr)$

definition $vec1I :: 'a \Rightarrow 'a \Rightarrow nat \Rightarrow nat \Rightarrow 'a\ vec$
where $vec1I\ ze\ on\ n\ i \equiv replicate\ i\ ze\ @\ on\ \# \ replicate\ (n - 1 - i)\ ze$

definition $mat1I :: 'a \Rightarrow 'a \Rightarrow nat \Rightarrow 'a\ mat$
where $mat1I\ ze\ on\ n \equiv map\ (vec1I\ ze\ on\ n)\ [0 ..< n]$

definition $vec-plusI :: ('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow 'a\ vec \Rightarrow 'a\ vec \Rightarrow 'a\ vec$ **where**
 $vec-plusI\ pl\ v\ w = map\ (\lambda\ xy.\ pl\ (fst\ xy)\ (snd\ xy))\ (zip\ v\ w)$

definition $mat-plusI :: ('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow 'a\ mat \Rightarrow 'a\ mat \Rightarrow 'a\ mat$
where $mat-plusI\ pl\ m1\ m2 = map\ (\lambda\ uv.\ vec-plusI\ pl\ (fst\ uv)\ (snd\ uv))\ (zip\ m1\ m2)$

definition $scalar-prodI :: 'a \Rightarrow ('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow 'a\ vec \Rightarrow 'a\ vec \Rightarrow 'a$
where
 $scalar-prodI\ ze\ pl\ ti\ v\ w = foldr\ (\lambda\ (x,y)\ s.\ pl\ (ti\ x\ y)\ s)\ (zip\ v\ w)\ ze$

definition $row :: 'a\ mat \Rightarrow nat \Rightarrow 'a\ vec$
where $row\ m\ i \equiv map\ (\lambda\ w.\ w\ !\ i)\ m$

definition $col :: 'a\ mat \Rightarrow nat \Rightarrow 'a\ vec$
where $col\ m\ i \equiv m\ !\ i$

fun $transpose :: nat \Rightarrow 'a\ mat \Rightarrow 'a\ mat$
where $transpose\ nr\ [] = replicate\ nr\ []$
 $| transpose\ nr\ (v\ \# m) = map\ (\lambda\ (vi,mi).\ (vi\ \# mi))\ (zip\ v\ (transpose\ nr\ m))$

definition $matT-vec-multI :: 'a \Rightarrow ('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow 'a\ mat \Rightarrow 'a\ vec \Rightarrow 'a\ vec$
where $matT-vec-multI\ ze\ pl\ ti\ m\ v = map\ (\lambda\ w.\ scalar-prodI\ ze\ pl\ ti\ w\ v)\ m$

definition $mat-multI :: 'a \Rightarrow ('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow ('a \Rightarrow 'a \Rightarrow 'a) \Rightarrow nat \Rightarrow 'a\ mat \Rightarrow 'a\ mat \Rightarrow 'a\ mat$
where $mat-multI\ ze\ pl\ ti\ nr\ m1\ m2 \equiv map\ (matT-vec-multI\ ze\ pl\ ti\ (transpose\ nr\ m1))\ m2$

fun *mat-powI* :: 'a ⇒ 'a ⇒ ('a ⇒ 'a ⇒ 'a) ⇒ ('a ⇒ 'a ⇒ 'a) ⇒ nat ⇒ 'a mat
 ⇒ nat ⇒ 'a mat
where *mat-powI ze on pl ti n m 0* = *mat1I ze on n*
 | *mat-powI ze on pl ti n m (Suc i)* = *mat-multI ze pl ti n (mat-powI ze on pl ti n m i) m*

definition *sub-vec* :: nat ⇒ 'a vec ⇒ 'a vec
where *sub-vec* = *take*

definition *sub-mat* :: nat ⇒ nat ⇒ 'a mat ⇒ 'a mat
where *sub-mat nr nc m* = *map (sub-vec nr) (take nc m)*

definition *vec-map* :: ('a ⇒ 'a) ⇒ 'a vec ⇒ 'a vec
where *vec-map* = *map*

definition *mat-map* :: ('a ⇒ 'a) ⇒ 'a mat ⇒ 'a mat
where *mat-map f* = *map (vec-map f)*

2.3 algorithms preserve dimensions

lemma *vec0[simp,intro]*: *vec nr (vec0I ze nr)*
 ⟨*proof*⟩

lemma *replicate-prop*:
assumes *P x*
shows $\forall y \in \text{set } (\text{replicate } n \ x). \ P \ y$
 ⟨*proof*⟩

lemma *mat0[simp,intro]*: *mat nr nc (mat0I ze nr nc)*
 ⟨*proof*⟩

lemma *vec1[simp,intro]*: **assumes** $i < nr$ **shows** *vec nr (vec1I ze on nr i)*
 ⟨*proof*⟩

lemma *mat1[simp,intro]*: *mat nr nr (mat1I ze on nr)*
 ⟨*proof*⟩

lemma *vec-plus[simp,intro]*: $\llbracket \text{vec } nr \ u; \text{vec } nr \ v \rrbracket \implies \text{vec } nr \ (\text{vec-plusI } pl \ u \ v)$
 ⟨*proof*⟩

lemma *mat-plus[simp,intro]*: **assumes** *mat nr nc m1* **and** *mat nr nc m2* **shows**
mat nr nc (mat-plusI pl m1 m2)
 ⟨*proof*⟩

lemma *vec-map*[*simp,intro*]: $vec\ nr\ u \implies vec\ nr\ (vec\text{-map}\ f\ u)$
(*proof*)

lemma *mat-map*[*simp,intro*]: $mat\ nr\ nc\ m \implies mat\ nr\ nc\ (mat\text{-map}\ f\ m)$
(*proof*)

fun *vec-fold* :: ($'a \Rightarrow 'b \Rightarrow 'b$) $\Rightarrow 'a\ vec \Rightarrow 'b \Rightarrow 'b$
 where [*code-unfold*]: $vec\text{-fold}\ f = foldr\ f$

fun *mat-fold* :: ($'a \Rightarrow 'b \Rightarrow 'b$) $\Rightarrow 'a\ mat \Rightarrow 'b \Rightarrow 'b$
 where [*code-unfold*]: $mat\text{-fold}\ f = foldr\ (vec\text{-fold}\ f)$

lemma *concat-mat*: $mat\ nr\ nc\ m \implies$
 $concat\ m = [m\ !\ i\ !\ j. i \leftarrow [0\ ..<\ nc], j \leftarrow [0\ ..<\ nr]]$
(*proof*)

lemma *row*: **assumes** $mat\ nr\ nc\ m$
 and $i < nr$
 shows $vec\ nc\ (row\ m\ i)$
(*proof*)

lemma *col*: **assumes** $mat\ nr\ nc\ m$
 and $i < nc$
 shows $vec\ nr\ (col\ m\ i)$
(*proof*)

lemma *transpose*[*simp,intro*]: **assumes** $mat\ nr\ nc\ m$
 shows $mat\ nc\ nr\ (transpose\ nr\ m)$
(*proof*)

lemma *matT-vec-multI*: **assumes** $mat\ nr\ nc\ m$
 shows $vec\ nc\ (matT\text{-vec}\text{-multI}\ ze\ pl\ ti\ m\ v)$
(*proof*)

lemma *mat-mult*[*simp,intro*]: **assumes** $wf1: mat\ nr\ n\ m1$
 and $wf2: mat\ n\ nc\ m2$
 shows $mat\ nr\ nc\ (mat\text{-multI}\ ze\ pl\ ti\ nr\ m1\ m2)$
(*proof*)

lemma *mat-pow*[*simp,intro*]: **assumes** $mat\ n\ n\ m$
 shows $mat\ n\ n\ (mat\text{-powI}\ ze\ on\ pl\ ti\ n\ m\ i)$
(*proof*)

lemma *sub-vec*[*simp,intro*]: **assumes** $vec\ nr\ v$ **and** $sd \leq nr$
 shows $vec\ sd\ (sub\text{-vec}\ sd\ v)$
(*proof*)

lemma *sub-mat*[*simp,intro*]: **assumes** *wf*: *mat nr nc m* **and** *sr*: $sr \leq nr$ **and** *sc*:
 $sc \leq nc$
shows *mat sr sc (sub-mat sr sc m)*
 ⟨*proof*⟩

2.4 properties of algorithms which do not depend on properties of type of matrix

lemma *mat0-index*[*simp*]: **assumes** $i < nc$ **and** $j < nr$
shows *mat0I ze nr nc ! i ! j = ze*
 ⟨*proof*⟩

lemma *mat0-row*[*simp*]: **assumes** $i < nr$
shows *row (mat0I ze nr nc) i = vec0I ze nc*
 ⟨*proof*⟩

lemma *mat0-col*[*simp*]: **assumes** $i < nc$
shows *col (mat0I ze nr nc) i = vec0I ze nr*
 ⟨*proof*⟩

lemma *vec1-index*: **assumes** $j: j < n$
shows *vec1I ze on n i ! j = (if i = j then on else ze) (is - = ?r)*
 ⟨*proof*⟩

lemma *col-transpose-is-row*[*simp*]:
assumes *wf*: *mat nr nc m*
and *i*: $i < nr$
shows *col (transpose nr m) i = row m i*
 ⟨*proof*⟩

lemma *mat-col-eq*:
assumes *wf1*: *mat nr nc m1*
and *wf2*: *mat nr nc m2*
shows $(m1 = m2) = (\forall i < nc. col m1 i = col m2 i)$ (**is** ?l = ?r)
 ⟨*proof*⟩

lemma *mat-col-eqI*:
assumes *wf1*: *mat nr nc m1*
and *wf2*: *mat nr nc m2*
and *id*: $\bigwedge i. i < nc \implies col m1 i = col m2 i$
shows $m1 = m2$
 ⟨*proof*⟩

lemma *mat-eq*:
assumes *wf1*: *mat nr nc m1*
and *wf2*: *mat nr nc m2*

shows $(m1 = m2) = (\forall i < nc. \forall j < nr. m1 ! i ! j = m2 ! i ! j)$ (**is** ?l = ?r)
<proof>

lemma *mat-eqI*:

assumes *wf1*: *mat nr nc m1*

and *wf2*: *mat nr nc m2*

and *id*: $\bigwedge i j. i < nc \implies j < nr \implies m1 ! i ! j = m2 ! i ! j$

shows $m1 = m2$

<proof>

lemma *vec-eq*:

assumes *wf1*: *vec n v1*

and *wf2*: *vec n v2*

shows $(v1 = v2) = (\forall i < n. v1 ! i = v2 ! i)$ (**is** ?l = ?r)

<proof>

lemma *vec-eqI*:

assumes *wf1*: *vec n v1*

and *wf2*: *vec n v2*

and *id*: $\bigwedge i. i < n \implies v1 ! i = v2 ! i$

shows $v1 = v2$

<proof>

lemma *row-col*: **assumes** *mat nr nc m*

and $i < nr$ **and** $j < nc$

shows $row\ m\ i\ !\ j = col\ m\ j\ !\ i$

<proof>

lemma *col-index*: **assumes** *m*: *mat nr nc m*

and $i < nc$

shows $col\ m\ i = map\ (\lambda j. m\ !\ i\ !\ j)\ [0 ..< nr]$

<proof>

lemma *row-index*: **assumes** *m*: *mat nr nc m*

and $i < nr$

shows $row\ m\ i = map\ (\lambda j. m\ !\ j\ !\ i)\ [0 ..< nc]$

<proof>

lemma *mat-row-eq*:

assumes *wf1*: *mat nr nc m1*

and *wf2*: *mat nr nc m2*

shows $(m1 = m2) = (\forall i < nr. row\ m1\ i = row\ m2\ i)$ (**is** ?l = ?r)

<proof>

lemma *mat-row-eqI*:

assumes *wf1*: *mat nr nc m1*

and *wf2*: *mat nr nc m2*

and $id: \bigwedge i. i < nr \implies \text{row } m1 \ i = \text{row } m2 \ i$
shows $m1 = m2$
 $\langle proof \rangle$

lemma *row-transpose-is-col*[simp]: **assumes** $wf: \text{mat } nr \ nc \ m$
and $i: i < nc$
shows $\text{row } (\text{transpose } nr \ m) \ i = \text{col } m \ i$
 $\langle proof \rangle$

lemma *matT-vec-mult-to-scalar*:
assumes $\text{mat } nr \ nc \ m$
and $\text{vec } nr \ v$
and $i < nc$
shows $\text{matT-vec-multI } ze \ pl \ ti \ m \ v \ ! \ i = \text{scalar-prodI } ze \ pl \ ti \ (\text{col } m \ i) \ v$
 $\langle proof \rangle$

lemma *mat-vec-mult-index*:
assumes $wf: \text{mat } nr \ nc \ m$
and $wfV: \text{vec } nc \ v$
and $i: i < nr$
shows $\text{matT-vec-multI } ze \ pl \ ti \ (\text{transpose } nr \ m) \ v \ ! \ i = \text{scalar-prodI } ze \ pl \ ti \ (\text{row } m \ i) \ v$
 $\langle proof \rangle$

lemma *mat-mult-index*[simp] :
assumes $wf1: \text{mat } nr \ n \ m1$
and $wf2: \text{mat } n \ nc \ m2$
and $i: i < nr$
and $j: j < nc$
shows $\text{mat-multI } ze \ pl \ ti \ nr \ m1 \ m2 \ ! \ j \ ! \ i = \text{scalar-prodI } ze \ pl \ ti \ (\text{row } m1 \ i) \ (\text{col } m2 \ j)$
 $\langle proof \rangle$

lemma *col-mat-mult-index* :
assumes $wf1: \text{mat } nr \ n \ m1$
and $wf2: \text{mat } n \ nc \ m2$
and $j: j < nc$
shows $\text{col } (\text{mat-multI } ze \ pl \ ti \ nr \ m1 \ m2) \ j = \text{map } (\lambda i. \text{scalar-prodI } ze \ pl \ ti \ (\text{row } m1 \ i) \ (\text{col } m2 \ j)) \ [0 \ .. < nr] \ (\text{is } \text{col } ?l \ j = ?r)$
 $\langle proof \rangle$

lemma *row-mat-mult-index* :
assumes $wf1: \text{mat } nr \ n \ m1$
and $wf2: \text{mat } n \ nc \ m2$
and $i: i < nr$
shows $\text{row } (\text{mat-multI } ze \ pl \ ti \ nr \ m1 \ m2) \ i = \text{map } (\lambda j. \text{scalar-prodI } ze \ pl \ ti \ (\text{row } m1 \ i) \ (\text{col } m2 \ j)) \ [0 \ .. < nc] \ (\text{is } \text{row } ?l \ i = ?r)$
 $\langle proof \rangle$

lemma *scalar-prod-cons*:
scalar-prodI ze pl ti (a # as) (b # bs) = pl (ti a b) (scalar-prodI ze pl ti as bs)
 ⟨*proof*⟩

lemma *vec-plus-index[simp]*:
assumes *wf1: vec nr v1*
and *wf2: vec nr v2*
and *i: i < nr*
shows *vec-plusI pl v1 v2 ! i = pl (v1 ! i) (v2 ! i)*
 ⟨*proof*⟩

lemma *mat-map-index[simp]*: **assumes** *wf: mat nr nc m* **and** *i: i < nc* **and** *j: j < nr*
shows *mat-map f m ! i ! j = f (m ! i ! j)*
 ⟨*proof*⟩

lemma *mat-plus-index[simp]*:
assumes *wf1: mat nr nc m1*
and *wf2: mat nr nc m2*
and *i: i < nc*
and *j: j < nr*
shows *mat-plusI pl m1 m2 ! i ! j = pl (m1 ! i ! j) (m2 ! i ! j)*
 ⟨*proof*⟩

lemma *col-mat-plus*: **assumes** *wf1: mat nr nc m1*
and *wf2: mat nr nc m2*
and *i: i < nc*
shows *col (mat-plusI pl m1 m2) i = vec-plusI pl (col m1 i) (col m2 i)*
 ⟨*proof*⟩

lemma *transpose-index[simp]*: **assumes** *wf: mat nr nc m*
and *i: i < nr*
and *j: j < nc*
shows *transpose nr m ! i ! j = m ! j ! i*
 ⟨*proof*⟩

lemma *transpose-mat-plus*: **assumes** *wf: mat nr nc m1 mat nr nc m2*
shows *transpose nr (mat-plusI pl m1 m2) = mat-plusI pl (transpose nr m1)*
(transpose nr m2) (is ?l = ?r)
 ⟨*proof*⟩

lemma *row-mat-plus*: **assumes** *wf1: mat nr nc m1*
and *wf2: mat nr nc m2*
and *i: i < nr*
shows *row (mat-plusI pl m1 m2) i = vec-plusI pl (row m1 i) (row m2 i)*
 ⟨*proof*⟩

lemma *col-mat1*: **assumes** $i < nr$
shows $col (mat1I ze on nr) i = vec1I ze on nr i$
 $\langle proof \rangle$

lemma *mat1-index*: **assumes** $i: i < n$ **and** $j: j < n$
shows $mat1I ze on n ! i ! j = (if i = j then on else ze)$
 $\langle proof \rangle$

lemma *transpose-mat1*: $transpose nr (mat1I ze on nr) = (mat1I ze on nr)$ (**is ?l**
 $= ?r$)
 $\langle proof \rangle$

lemma *row-mat1*: **assumes** $i: i < nr$
shows $row (mat1I ze on nr) i = vec1I ze on nr i$
 $\langle proof \rangle$

lemma *sub-mat-index*:
assumes $wf: mat nr nc m$
and $sr: sr \leq nr$
and $sc: sc \leq nc$
and $j: j < sr$
and $i: i < sc$
shows $sub-mat sr sc m ! i ! j = m ! i ! j$
 $\langle proof \rangle$

2.5 lemmas requiring properties of plus, times, ...

context *plus*
begin

abbreviation $vec-plus :: 'a vec \Rightarrow 'a vec \Rightarrow 'a vec$
where $vec-plus \equiv vec-plusI plus$

abbreviation $mat-plus :: 'a mat \Rightarrow 'a mat \Rightarrow 'a mat$
where $mat-plus \equiv mat-plusI plus$
end

context *semigroup-add*
begin

lemma *vec-plus-assoc*: **assumes** $vec: vec nr u vec nr v vec nr w$
shows $vec-plus u (vec-plus v w) = vec-plus (vec-plus u v) w$
 $\langle proof \rangle$

lemma *mat-plus-assoc*: **assumes** $wf: mat nr nc m1 mat nr nc m2 mat nr nc m3$
shows $mat-plus m1 (mat-plus m2 m3) = mat-plus (mat-plus m1 m2) m3$ (**is ?l**
 $= ?r$)

<proof>
end

context *ab-semigroup-add*
begin
lemma *vec-plus-comm*: $vec\text{-plus } x \ y = vec\text{-plus } y \ x$
<proof>

lemma *mat-plus-comm*: $mat\text{-plus } m1 \ m2 = mat\text{-plus } m2 \ m1$
<proof>
end

context *zero*
begin
abbreviation *vec0* :: $nat \Rightarrow 'a \ vec$
where *vec0* $\equiv vec0I \ zero$

abbreviation *mat0* :: $nat \Rightarrow nat \Rightarrow 'a \ mat$
where *mat0* $\equiv mat0I \ zero$
end

context *monoid-add*
begin
lemma *vec0-plus[simp]*: **assumes** *vec nr u* **shows** $vec\text{-plus } (vec0 \ nr) \ u = u$
<proof>

lemma *plus-vec0[simp]*: **assumes** *vec nr u* **shows** $vec\text{-plus } u \ (vec0 \ nr) = u$
<proof>

lemma *plus-mat0[simp]*: **assumes** *wf: mat nr nc m* **shows** $mat\text{-plus } m \ (mat0 \ nr \ nc) = m$ (**is** *?l = ?r*)
<proof>

lemma *mat0-plus[simp]*: **assumes** *wf: mat nr nc m* **shows** $mat\text{-plus } (mat0 \ nr \ nc) \ m = m$ (**is** *?l = ?r*)
<proof>
end

context *semiring-0*
begin
abbreviation *scalar-prod* :: $'a \ vec \Rightarrow 'a \ vec \Rightarrow 'a$
where *scalar-prod* $\equiv scalar\text{-prodI } zero \ plus \ times$

abbreviation *mat-mult* :: $nat \Rightarrow 'a \ mat \Rightarrow 'a \ mat \Rightarrow 'a \ mat$
where *mat-mult* $\equiv mat\text{-multI } zero \ plus \ times$

lemma *scalar-prod*: $scalar\text{-prod } v1 \ v2 = sum\text{-list } (map \ (\lambda(x,y). x * y) \ (zip \ v1 \ v2))$
<proof>

lemma *scalar-prod-last*: **assumes** $\text{length } v1 = \text{length } v2$
shows $\text{scalar-prod } (v1 @ [x1]) (v2 @ [x2]) = x1 * x2 + \text{scalar-prod } v1 v2$
 $\langle \text{proof} \rangle$

lemma *scalar-product-assoc*:
assumes $wf_m: \text{mat } nr \ nc \ m$
and $wf_r: \text{vec } nr \ r$
and $wf_c: \text{vec } nc \ c$
shows $\text{scalar-prod } (\text{map } (\lambda k. \text{scalar-prod } r (\text{col } m \ k)) [0..<nc]) \ c = \text{scalar-prod } r (\text{map } (\lambda k. \text{scalar-prod } (\text{row } m \ k) \ c) [0..<nr])$
 $\langle \text{proof} \rangle$

lemma *mat-mult-assoc*:
assumes $wf1: \text{mat } nr \ n1 \ m1$
and $wf2: \text{mat } n1 \ n2 \ m2$
and $wf3: \text{mat } n2 \ nc \ m3$
shows $\text{mat-mult } nr (\text{mat-mult } nr \ m1 \ m2) \ m3 = \text{mat-mult } nr \ m1 (\text{mat-mult } n1 \ m2 \ m3)$ (**is** $?m12-3 = ?m1-23$)
 $\langle \text{proof} \rangle$

lemma *mat-mult-assoc-n*:
assumes $wf1: \text{mat } n \ n \ m1$
and $wf2: \text{mat } n \ n \ m2$
and $wf3: \text{mat } n \ n \ m3$
shows $\text{mat-mult } n (\text{mat-mult } n \ m1 \ m2) \ m3 = \text{mat-mult } n \ m1 (\text{mat-mult } n \ m2 \ m3)$
 $\langle \text{proof} \rangle$

lemma *scalar-left-zero*: $\text{scalar-prod } (\text{vec0 } nn) \ v = \text{zero}$
 $\langle \text{proof} \rangle$

lemma *scalar-right-zero*: $\text{scalar-prod } v (\text{vec0 } nn) = \text{zero}$
 $\langle \text{proof} \rangle$

lemma *mat0-mult-left*: **assumes** $wf: \text{mat } nc \ ncc \ m$
shows $\text{mat-mult } nr (\text{mat0 } nr \ nc) \ m = (\text{mat0 } nr \ ncc)$
 $\langle \text{proof} \rangle$

lemma *mat0-mult-right*: **assumes** $wf: \text{mat } nr \ nc \ m$
shows $\text{mat-mult } nr \ m (\text{mat0 } nc \ ncc) = (\text{mat0 } nr \ ncc)$
 $\langle \text{proof} \rangle$

lemma *scalar-vec-plus-distrib-right*:
assumes $wf1: \text{vec } nr \ u$
assumes $wf2: \text{vec } nr \ v$

assumes *wf3*: *vec nr w*
shows $\text{scalar-prod } u \ (\text{vec-plus } v \ w) = \text{plus } (\text{scalar-prod } u \ v) \ (\text{scalar-prod } u \ w)$
<proof>

lemma *scalar-vec-plus-distrib-left*:

assumes *wf1*: *vec nr u*
assumes *wf2*: *vec nr v*
assumes *wf3*: *vec nr w*
shows $\text{scalar-prod } (\text{vec-plus } u \ v) \ w = \text{plus } (\text{scalar-prod } u \ w) \ (\text{scalar-prod } v \ w)$
<proof>

lemma *mat-mult-plus-distrib-right*:

assumes *wf1*: *mat nr nc m1*
and *wf2*: *mat nc ncc m2*
and *wf3*: *mat nc ncc m3*
shows $\text{mat-mult } nr \ m1 \ (\text{mat-plus } m2 \ m3) = \text{mat-plus } (\text{mat-mult } nr \ m1 \ m2)$
 $(\text{mat-mult } nr \ m1 \ m3)$ (**is** $\text{mat-mult } nr \ m1 \ ?m23 = \text{mat-plus } ?m12 \ ?m13$)
<proof>

lemma *mat-mult-plus-distrib-left*:

assumes *wf1*: *mat nr nc m1*
and *wf2*: *mat nr nc m2*
and *wf3*: *mat nc ncc m3*
shows $\text{mat-mult } nr \ (\text{mat-plus } m1 \ m2) \ m3 = \text{mat-plus } (\text{mat-mult } nr \ m1 \ m3)$
 $(\text{mat-mult } nr \ m2 \ m3)$ (**is** $\text{mat-mult } nr \ ?m12 \ = \ \text{mat-plus } ?m13 \ ?m23$)
<proof>
end

context *semiring-1*

begin

abbreviation *vec1* :: *nat* \Rightarrow *nat* \Rightarrow 'a *vec*

where *vec1* \equiv *vec1I* *zero one*

abbreviation *mat1* :: *nat* \Rightarrow 'a *mat*

where *mat1* \equiv *mat1I* *zero one*

abbreviation *mat-pow* **where** *mat-pow* \equiv *mat-powI* (*0* :: 'a) *1* (+) (*)

lemma *scalar-left-one*: **assumes** *wf*: *vec nn v*

and *i*: *i* < *nn*

shows $\text{scalar-prod } (\text{vec1 } nn \ i) \ v = v \ ! \ i$

<proof>

lemma *scalar-right-one*: **assumes** *wf*: *vec nn v*

and *i*: *i* < *nn*

shows $\text{scalar-prod } v \ (\text{vec1 } nn \ i) = v \ ! \ i$

<proof>

lemma *mat1-mult-right*: **assumes** *wf: mat nr nc m*
shows *mat-mult nr m (mat1 nc) = m*
 ⟨*proof*⟩

lemma *mat1-mult-left*: **assumes** *wf: mat nr nc m*
shows *mat-mult nr (mat1 nr) m = m*
 ⟨*proof*⟩
end

declare *vec0[simp del] mat0[simp del] vec0-plus[simp del] plus-vec0[simp del] plus-mat0[simp del]*

2.6 Connection to HOL-Algebra

definition *mat-monoid* :: *nat* ⇒ *nat* ⇒ 'b ⇒ ((*'a* :: {*plus,zero*}) *mat,'b*) *monoid-scheme*
where

mat-monoid nr nc b ≡ (
carrier = *Collect (mat nr nc)*,
mult = *mat-plus*,
one = *mat0 nr nc*,
 ... = *b*)

definition *mat-ring* :: *nat* ⇒ 'b ⇒ ((*'a* :: *semiring-1*) *mat,'b*) *ring-scheme* **where**

mat-ring n b ≡ (
carrier = *Collect (mat n n)*,
mult = *mat-mult n*,
one = *mat1 n*,
zero = *mat0 n n*,
add = *mat-plus*,
 ... = *b*)

lemma *mat-monoid: monoid* (*mat-monoid nr nc b* :: ((*'a* :: *monoid-add*) *mat,'b*)*monoid-scheme*)
 ⟨*proof*⟩

lemma *mat-group: group* (*mat-monoid nr nc b* :: ((*'a* :: *group-add*) *mat,'b*)*monoid-scheme*)
 (**is** *group ?G*)
 ⟨*proof*⟩

lemma *mat-comm-monoid*:
comm-monoid (*mat-monoid nr nc b* :: ((*'a* :: *comm-monoid-add*) *mat,'b*)*monoid-scheme*)
 (**is** *comm-monoid ?G*)
 ⟨*proof*⟩

lemma *mat-comm-group*:
comm-group (*mat-monoid nr nc b* :: ((*'a* :: *ab-group-add*) *mat,'b*)*monoid-scheme*)

(**is comm-group** ?G)
<proof>

lemma *mat-abelian-monoid: abelian-monoid* (mat-ring n b :: (('a :: semiring-1)
mat,'b)ring-scheme)
<proof>

lemma *mat-abelian-group: abelian-group* (mat-ring n b :: (('a :: {ab-group-add,semiring-1})
mat,'b)ring-scheme)
(**is abelian-group** ?R)
<proof>

lemma *mat-semiring: semiring* (mat-ring n b :: (('a :: semiring-1) mat,'b)ring-scheme)
(**is semiring** ?R)
<proof>

lemma *mat-ring: ring* (mat-ring n b :: (('a :: ring-1) mat,'b)ring-scheme)
(**is ring** ?R)
<proof>

lemma *mat-pow-ring-pow: assumes* mat: mat n n (m :: ('a :: semiring-1)mat)
shows mat-pow n m k = m [∧]_{mat-ring n b k}
(**is** - = m [∧]_{?C k})
<proof>

end

References

- [1] J. Endrullis, J. Waldmann, and H. Zantema. Matrix Interpretations for Proving Termination of Term Rewriting. *Journal of Automated Reasoning*, 40(2-3):195–220, 2008.
- [2] A. Koprowski and J. Waldmann. Arctic termination ... below zero. In *Proc. RTA '08*, LNCS 5117, pages 202–216, 2008.
- [3] R. Thiemann and C. Sternagel. Certification of termination proofs using CeTA. In *Proc. TPHOLs '09*, LNCS 5674, pages 452–468, 2009.
- [4] R. Thiemann and C. Sternagel. Certified polynomial interpretations over matrices and over domains. In *Proc. WST '10*, 2010. To appear.