A Formalization of Elementary Linear Algebra: Part II

David M. Russinoff

david@russinoff.com

This is the second installment of an exposition of an ACL2 formalization of elementary linear algebra. It extends the results of Part I, which covers the algebra of matrices over a commutative ring, but focuses on aspects of the theory that apply only to matrices over a field: elementary row reduction and its application to the computation of matrix inverses and the solution of simultaneous systems of linear equations.

1 Introduction

This is the second installment of an exposition of an ACL2 formalization of elementary linear algebra. Part I [6], which is also included in this workshop, covers the algebra of matrices over a commutative ring with unity and their determinants. In this sequel, we focus on aspects of the theory of matrices that apply only to matrices over a field, i.e., depend on the existence of a multiplicative inverse operator. These include row reduction and its application to matrix invertibility and the solution of systems of linear equations. In an anticipated Part III, all of these results will be applied to the study of abstract vector spaces and linear transformations.

The proof scripts supporting both papers reside in the same directory, books/projects/linear/. As described in [6], the abstract definition of a ring is formalized in the file ring.lisp by a set of constrained encapsulated functions: a predicate rp that recognizes ring elements, the binary addition and multiplication operations r+ and r*, the corresponding identity constants r0 and r1, and the additive inverse operator r-. The notion of a field is similarly defined by an encapsulation in the file field.lisp, in which these functions are renamed fp, f+, f*, etc.:

```
(encapsulate (((fp *) => *)
                                                   ;field element recognizer
               ((f + * *) \Rightarrow *) ((f * * *) \Rightarrow *); addition and multiplication
               ((f0) \Rightarrow *) ((f1) \Rightarrow *)
                                                   ;identities
               ((f - *) \Rightarrow *) ((f / *) \Rightarrow *))
                                                   ;inverses
  (local (defun fp (x) (rationalp x)))
  (local (defun f+ (x y) (+ x y)))
  (local (defun f* (x y) (* x y)))
  (local (defun f0 () 0))
  (local (defun f1 () 1))
  (local (defun f- (x) (- x)))
  (local (defun f/ (x) (/ x)))
  ;; Closure:
  . . .
  ;; Multiplicative inverse:
  (defthm fpf/
    (implies (and (fp x) (not (equal x (f0)))) (fp (f/x))))
  (defthm f*inv
    (implies (and (fp x) (not (equal x (f0)))) (equal (f* x (f/ x)) (f1))))
```

© D.M. Russinoff This work is licensed under the Creative Commons Attribution License. The only other difference between the two encapsulations is the inclusion here of the multiplicative inverse f/along with two constraining axioms, appended to the constraints adapted from the ring axioms.

Informally we shall refer to the field F that is characterized by this encapsulation. When our intention is clear, the identity elements (f0) and (f1) will be abbreviated as 0 and 1. Clearly, all properties of rings hold for fields as well. Thus, all definitions and theorems that appear in ring.lisp, rmat.lisp, and rdet.lisp have analogs in the corresponding files field.lisp, fmat.lisp, and fdet.lisp. In particular, the predicate flistnp recognizes a list of specified length of elements of F, called an *flist*; fmatp recognizes a matrix over F; and fdet computes its determinant. In principle, all results in the latter set of files could be derived by functional instantiation from the corresponding events in the former, but we found it more expedient to reproduce the proofs, simply by selectively replacing occurrences of the character r with f. An additional file, reduction.lisp, contains the results reported in this paper. In describing these results, we assume the reader is familiar with Part I and is aware of the renaming convention.

In Section 2, we define the notion of a reduced row-echelon matrix and develop a procedure that converts an arbitrary matrix to reduced row-echelon form. An equivalent procedure, based on matrix multiplication, is also defined. This leads to a criterion for invertibility of a square matrix and a method for computing inverses. A second method of matrix inversion, based on determinants and the classical adjoint, is derived from the results of Part I.

Section 3 addresses the solution of systems of linear equations, mainly as an application of row reduction. We derive algorithmic tests for solvability and uniqueness of the solution, as well as a formula that computes the solution in the uniquely solvable case. For the special case of an invertible square coefficient matrix, we prove Cramer's Rule, an alternative formula based on determinants. In the general solvable case, we show that the solution set is infinite and establish a test that identifies solutions. In Part III, this will lead to a formula that generates a basis for the solution space of a homogeneous system of equations.

All of these results, which are stated and proved in the context of an abstract field, may be applied to any concrete field of interest through functional instantiation. Eventually, we plan to apply the theory to algebraic number fields. Of course, in their abstract formulation based on constrained functions, the definitions are not executable. For the immediate purposes of illustration and testing, however, all functions defined in field.lisp, fmat.lisp, fdet.lisp, and reduction.lisp have been adapted to the field of rational numbers as executable functions, which are listed in the file rational.lisp.

2 Row Reduction

2.1 Reduced Row-Echelon Form

A reduced row-echelon matrix may be characterized as follows:

- (1) Every all-zero row is preceded by every nonzero row;
- (2) The first nonzero entry of each nonzero row is 1, and every other entry in the same column is 0;
- (3) The column of the leading 1 in the ith nonzero row is an increasing function of i.

The formalization of this definition requires several auxiliary functions. First, we define the index of the leading nonzero entry of a nonzero row r:

```
(defun first-nonzero (r)
  (if (consp r)
```

In the following, we assume that a is an $m \times n$ matrix. Starting with row k, where $0 \le k \le m$, find the row of a with nonzero entry of least index, or return NIL if all rows beyond the first k are 0:

Given j < n and k < m, check that (entry k j a) = 1 and that all other entries in column k are 0:

Given $k \leq m$, check that the first k rows of a form a reduced row-echelon matrix:

Finally, check that a is a reduced row-echelon matrix:

(defund row-echelon-p (a) (row-echelon-p-aux a (len a) (len a)))

2.2 Conversion to Reduced Row-Echelon Form

We shall develop a procedure that converts an arbitrary $m \times n$ matrix a to reduced row-echelon form by a sequence of *elementary row operations* of three types:

(1) Multiply row k by a scalar c:

(defund ero1 (a c k) (replace-row a k (flist-scalar-mul c (nth k a))))

(2) Add a scalar multiple of row j to row k, where $j \neq k$:

```
(defund ero2 (a c j k)
  (replace-row a k (flist-add (flist-scalar-mul c (nth j a)) (nth k a))))
```

(3) Interchange rows i and j, where $i \neq j$:

```
(defund ero3 (a j k) (replace-row (replace-row a k (nth j a)) j (nth k a)))
```

Under the assumption that (entry k j a) = 1, the following function applies ero2 to clear all other entries in column j by adding the appropriate multiple of row k to each of the other rows:

Assume the first k rows of a are in reduced row-echelon form, i.e., (row-echelon-p-aux a m k) = T, where k < m, and that i = (row-with-nonzero-at-least-index a m k) \neq NIL. Let j = (first-nonzero (nth i a)). The following function performs the next step of the reduction, producing a matrix a' satisfying (row-echelon-p-aux a' m (1+ k)):

The function row-reduce converts a to a reduced row-echelon matrix, using an auxiliary function that completes the conversion under the assumption (row-echelon-p-aux a m k), where $0 \le k \le m$:

```
(defund row-reduce (a) (row-reduce-aux a (len a) 0))
```

The following confirms that this procedure produces the desired result:

We also note that row reduction does not alter a reduced row-echelon matrix:

As an example, consider the following 4×5 matrix (a0):

DM !>(defun a0 () '((0 -3 -6 4 9) (-1 -2 -1 3 1) (-2 -3 0 3 -1) (1 4 5 -9 -7)))

In the first step in the row reduction of (a0), row 1 is divided by its leading nonzero entry, -1, and interchanged with row 0. The other entries in column 0 are then cleared:

```
DM !>(row-reduce-step (a0) 4 0 1 0)
((1 2 1 -3 -1)
(0 -3 -6 4 9)
(0 1 2 -3 -3)
(0 2 4 -6 -6))
```

Row reduction of (a0) requires three executions of row-reduce-step:

```
DM !>(row-reduce (a0))
((1 0 -3 0 5)
(0 1 2 0 -3)
(0 0 0 1 0)
(0 0 0 0 0))
We define the row rank of a to be the number of nonzero rows of (row-reduce a):
(defun num-nonzero-rows (a)
  (if (consp a)
        (if (flistOp (car a)) 0 (1+ (num-nonzero-rows (cdr a))))
        0))
(defun row-rank (a) (num-nonzero-rows (row-reduce a)))
```

Note that (row-reduce (a0)) has 3 nonzero rows:

DM !>(row-rank (a0)) 3

Obviously, the row rank of an $m \times n$ matrix cannot exceed m:

Nor can the row rank exceed n. To see this, consider the list of indices of the leading 1s of the nonzero rows of a reduced row-echelon matrix a:

```
(defun lead-inds (a)
 (if (and (consp a) (not (flistOp (car a))))
      (cons (first-nonzero (car a)) (lead-inds (cdr a)))
      ()))
DM !>(lead-inds (row-reduce (a0)))
      (0 1 3)
```

Clearly, the length of (lead-inds a) is the number of nonzero rows of a. Furthermore, (lead-inds a) is a strictly increasing sublist of (ninit t). It follows that (len (lead-inds a)) \leq n. Consequently, the row rank of a is bounded by n:

We also note that if (row-rank a) = n, then (lead-inds a) is an increasing sublist of (ninit n) of length n, which implies that the two lists are equal:

Along with row reduction, there is an obvious analogous notion of *column reduction* and a corresponding definiton of the *column rank* of a matrix, which may alternatively be defined as the row rank of its transpose. As we shall show in Part III in the context of vector spaces, the row and column ranks of a matrix are always equal.

2.3 Row Reduction as Matrix Multiplication

Once we have identified the sequence of operations required to derive the reduced row-echelon form of an $m \times n$ matrix a, an alternative derivation may be achieved by applying the same operations to the $m \times m$ identity matrix and right-multiplying the result by a. To this end, a row operation is encoded as a list of length 3 or 4; the first member indicates the operation type (1, 2, or 3 as listed in the preceding subsection), and the others are the parameters of the operation. The following predicate characterizes an encoding of a row operation on a matrix of m rows:

The function apply-row-op applies an encoded row operation to a matrix:

```
(defund apply-row-op (op a)
  (case (car op)
  ;(apply-row-op (list 1 c k) a) = (ero1 a c k)
    (1 (ero1 a (cadr op) (caddr op)))
  ;(apply-row-op (list 2 c j k) a) = (ero2 a c j k)
    (2 (ero2 a (cadr op) (caddr op) (cadddr op)))
  ;(apply-row-op (list 3 j k) a) = (ero3 a j k)
    (3 (ero3 a (cadr op) (caddr op)))))
```

A list of row operations is identified in the obvious way:

The function apply-row-ops applies a list of operations in sequence from left to right:

```
(defun apply-row-ops (ops a)
  (if (consp ops)
        (apply-row-ops (cdr ops) (apply-row-op (car ops) a))
        a))
```

By examining the definitions of row-reduce and its auxiliary functions, we construct the list of encodings of the operations that reduce a matrix a to reduced row-echelon form. The next four functions encode the lists of operations performed by clear-column, row-reduce-step, row-reduce-aux, and row-reduce, respectively:

```
(defun clear-column-ops (a k j m)
  (if (zp m) ()
   (if (= k (1 - m)))
        (clear-column-ops a k j (1- m))
      (cons (list 2 (f- (nth j (nth (1- m) a))) k (1- m))
            (clear-column-ops (ero2 a (f- (nth j (nth (1- m) a))) k (1- m))
                              k j (1- m)))))
(defund row-reduce-step-ops (a m k i j)
  (cons (list 1 (f/ (nth j (nth i a))) i)
        (cons (list 3 i k)
              (clear-column-ops (ero3 (ero1 a (f/ (nth j (nth i a))) i) i k)
                                k j m))))
(defun row-reduce-aux-ops (a m k)
  (let* ((i (row-with-nonzero-at-least-index a m k))
         (j (and i (first-nonzero (nth i a)))))
   (if (and (natp k) (natp m) (< k m) i)
        (append (row-reduce-step-ops a m k i j)
                (row-reduce-aux-ops (row-reduce-step a m k i j) m (1+ k)))
(defund row-reduce-ops (a) (row-reduce-aux-ops a (len a) 0))
```

The correctness of this encoding procedure is confirmed by the following:

Returning to the example of Subsection 2.2, we find that the first step in the row reduction of (a0) involves five elementary operations:

```
DM !>(row-reduce-step-ops (a0) 4 0 1 0)
((1 -1 1) (3 1 0) (2 -1 0 3) (2 2 0 2) (2 0 0 1))
DM !>(apply-row-ops '((1 -1 1) (3 1 0) (2 -1 0 3) (2 2 0 2) (2 0 0 1)) (a0))
((1 2 1 -3 -1)
(0 -3 -6 4 9)
(0 1 2 -3 -3)
(0 2 4 -6 -6))
```

The reader may wish to compute (row-reduce-ops (a0)), a list of length 15, and check that the lemma apply-row-reduce-ops holds in this case.

The $m \times m$ *elementary matrix* corresponding to a row operation is defined to be the result of applying the operation to the $m \times m$ identity matrix:

(defund elem-mat (op m) (apply-row-op op (id-fmat m)))

Application of a row operation is equivalent to left multiplication by the corresponding elementary matrix:

The product of the list of elementary matrices associated with the row reduction of a matrix is computed recursively by the function row-reduce-mat:

```
(defund row-ops-mat (ops m)
  (if (consp ops)
        (fmat* (row-ops-mat (cdr ops) m) (elem-mat (car ops) m))
        (id-fmat m)))
(defund row-reduce-mat (a) (row-ops-mat (row-reduce-ops a) (len a)))
```

It follows from elem-mat-row-op by induction that applying a sequence ops of row operations to a is equivalent to multiplication of a by (row-ops-mat ops m):

In particular, by apply-row-reduce-ops, row reduction of a is equivalent to multiplication by (row-reduce-mat a):

In our example, the product of the 15 elementary matrices corresponding to (row-reduce-ops (a0)) is

```
DM !>(row-reduce-mat (a0))
(( 3/5 -3/5 -1/5 0)
(-3/5 8/5 -4/5 0)
(-1/5 6/5 -3/5 0)
( 0 5 -2 1))
```

The conclusion of the lemma row-ops-mat-row-reduce may be readily verified for this case.

2.4 Invertibility

In this subsection, we focus on square matrices. Given an $n \times n$ matrix a, we seek an *inverse* of a, i.e., an $n \times n$ matrix b such that

(fmat* a b) = (fmat* b a) = (id-fmat n).

If such a matrix exists, then it is unique in the strong sense that it is the only left or right inverse of a. For example, if (fmat * c a) = (id-fmat n), then

```
c = (fmat* c (id-fmat n))
= (fmat* c (fmat* a b))
= (fmat* (fmat* c a) b))
= (fmat* (id-fmat n) b))
= b,
```

and the same conclusion similarly follows from the assumption (fmat* a c) = (id-fmat n). Thus, we have

Every elementary matix has an inverse:

Consequently, every product of elementary matrices has an inverse:

We shall show that a has an inverse iff (row-rank a) = n and that in this case, the inverse of a is (row-reduce-mat a). Thus, we define

(defund invertiblep (a n) (= (row-rank a) n))

and

(defund inverse-mat (a) (row-reduce-mat a))

First we note that as a consequence of lead-inds-ninit, if (invertiblep a n), then (row-reduce a) = (id-fmat n):

Now let

```
p = (inverse-mat a) = (row-reduce-mat a) = (row-ops-mat (row-reduce-ops a) n),
q = (row-ops-mat (invert-row-ops (row-reduce-ops a)) n),
```

and

r = (fmat * p a) = (row-reduce a).

By invert-row-ops-mat, (fmat* p q) = (fmat* q p) = (id-fmat n). Suppose (row-rank r) = n. By row-echelon-p-id-fmat, (fmat* p a) = r = (id-fmat n), and by inverse-unique, a = q. Thus, (invertiblep a n) is a sufficient condition for the existence of an inverse:

To prove the necessity of (invertiblep a n), suppose (fmatp b n n) and (fmat* a b) = (id-fmat n). Then

If (invertiblep a n) = NIL, then the last row of r is zero, and the same must be true of (id-fmat n), a contradiction.

We note several consequences of the preceding results. First, an invertible matrix is the inverse of its inverse:

Cancellation laws hold for invertible matrices, e.g.,

A matrix product is invertible iff each factor is invertible:

Finally, we shall show that a is invertible iff its determinant is nonzero. First note that if a has inverse b and (fdet a) = 0, then by fdet-multiplicative,

(fdet (id-fmat n) n) = (fdet (fmat* a b)) = (f* (fdet a) (fdet b)) = 0,

a contradiction. Thus,

On the other hand, assume (fdet a n) $\neq 0$. By fmat*-adjoint-fmat,

```
(fmat* a (adjoint-fmat a n)) = (fmat-scalar-mul (fdet a n) (id-fmat n)),
```

which implies

```
(fmat* a (fmat-scalar-mul (f/ (fdet a n)) (adjoint-fmat a n)))
= (fmat-scalar-mul (f/ (fdet a n)) (fmat* a (adjoint-fmat a n)))
= (fmat-scalar-mul (f/ (fdet a n)) (fmat-scalar-mul (fdet a n) (id-fmat n)))
= (id-fmat n),
```

and by invertiblep-necessary, a is invertible. This also establishes an alternative method for computing the inverse:

3 Simultaneous Systems of Linear equations

Let a be an m×n matrix with (entry i j a) = $a_{i,j}$ for $0 \le i < m$ and $0 \le j < n$, and let $b = (b_0 \dots b_{m-1})$ be an flist of length m. We seek an flist $x = (x_0 \dots x_{n-1})$ of length n such that for $0 \le i < m$,

 $a_{\mathtt{i},0}x_0 + \ldots + a_{\mathtt{i},\mathtt{n}-1}x_{\mathtt{n}-1} = b_{\mathtt{i}}.$

We shall refer to a as the *coefficient matrix* of this system of m linear equations in n unknowns. To express the system as a matrix equation, we define the *column matrix* corresponding to a given flist:

(defund col-mat (x) (transpose-mat (list x)))

The above equations are naturally expressed by the matrix equation in the following definition:

(defund solutionp (x a b) (equal (fmat* a (col-mat x)) (col-mat b)))

Let bc = (col-mat b), xc = (col-mat x), p = (row-reduce-mat a), ar = (fmat* p a), and br = (fmat* p bc). Left-multiplying the above equation by p yields the equivalent equation

$$(fmat* ar xc) = br.$$
 (1)

Thus, we have

Our objective, therefore, is to compute an $n \times 1$ column matrix xc that solves Equation (1), in which ar is an $m \times n$ reduced row-echelon matrix and br is an $m \times 1$ column matrix.

Let q = (num-nonzero-rows ar) = (row-rank a). We shall show that the existence of a solution to this equation is determined by whether the last m - q entries of br are all 0. This is true iff the following search returns NIL:

```
(defun find-nonzero (br q m)
 (if (and (natp q) (natp m) (< q m))
    (if (= (entry (1- m) 0 br) (f0))
        (find-nonzero br q (1- m))
        (1- m))
    ()))
```

Thus, we define

Suppose first that (find-nonzero br q m) = $k \neq NIL$, so that (solvablep a b) = NIL. Then (row k ar) = (flistn0 n) and (entry k 0 br) $\neq 0$. It follows that (entry k 0 (fmat* ar xc)) \neq (nth k 0 br), and hence (fmat* ar xc) \neq br. Combining this with reduce-linear-equations, we conclude that the system of equations has no solution:

Thus, we may assume (solvablep a b) = T. As a first step toward the solution, consider the matrices aq and bq consisting of the first q rows of ar and br, respectively, computed by the following:

```
(defun first-rows (q a)
 (if (zp q) ()
      (cons (car a) (first-rows (1- q) (cdr a)))))
```

It is easily shown that aq is a reduced row-echelon $q \times n$ matrix of row rank q and that (fmat* ar xc) = br iff (fmat* aq xc) = bq. Our objective, therefore, is to solve the equation (fmat* aq xc) = bq.

3.1 Uniquely Solvable Case

By row-rank<=n, $q \le n$. We first consider the case q = n. By row-echelon-p-id-fmat, aq = (id-fmat n) and (fmat* aq xc) = bq iff xc = bq. Combining this observation with first-rows-linear-equations and reduce-linear-equations, we conclude that there exists a unique solution in this case:;

Our results on cofactor expansion lead to an alternative method of solving a system of n linear equations in n unknowns in the case of a unique solution, known as Cramer's rule. Suppose m = n = q, so that a is an invertible $n \times n$ matrix. Our objective is to compute, as a function of a and b, for each i < n, the ith component (nth i x) of the unique x such that

We refer to the analogs of the results of [6, Sec. 5] that appear in fdet.lisp. In particular, we shall substitute a' = (replace-row (transpose-mat a) i b) for a in fdot-cofactor-fmat-row-fdet. Clearly, (row i a') = b. By cofactor-fmat-transpose,

and by fdet-transpose,

Thus, the substitution yields the following:

```
(fdot b (row i (adjoint-fmat a n))) = (fdet (replace-col a i b) n)).
```

Multiplying Equation (2) by (adjoint-fmat a n) yields

```
(fmat* (adjoint-fmat a n) (fmat* a xc)) = (fmat* (adjoint-fmat a n) bc).
```

But

```
(fmat* (adjoint-fmat a n) (fmat* a xc))
= (fmat* (fmat* (adjoint-fmat a n) a) xc)
= (fmat* (flist-scalar-mul (fdet a n) (id-fmat n)) xc)
= (flist-scalar-mul (fdet a n) (fmat* (id-fmat n) xc))
= (flist-scalar-mul (fdet a n) xc),
```

and hence

```
(flist-scalar-mul (fdet a n) xc) = (fmat* (adjoint-fmat a n) bc).
```

Equating the entries of these matrices in row i and column 0, we have

which yields Cramer's rule:

3.2 General Solvable Case

In the remainder of this section, we treat the general case (solvablep a b) = T with arbitrary q = (row-rank a) \leq n. The desired equation (fmat* aq xc) = bq holds iff for $0 \leq i < q$,

(nth i (fmat* aq xc)) = (nth i bq)

or equivalently,

$$(fdot (row i aq) x) = (car (nth i bq)).$$
 (3)

We shall split the dot product (fdot (nth i aq) x) into two sums, corresponding to the list (leadinds aq) of leading indices and the list of remaining indices, which we call the *free indices*:

```
(defund free-inds (a n) (set-difference-equal (ninit n) (lead-inds a)))
```

In general, given a sublist inds of (ninit n) and two flists r and x of length n, the following function extracts and sums the terms of the dot product of r and x that correspond to the indices inds:

```
(defun fdot-select (inds r x)
 (if (consp inds)
      (f+ (f* (nth (car inds) r) (nth (car inds) x))
            (fdot-select (cdr inds) r x))
      (f0)))
```

In particular, (fdot (row i aq) x) may be expressed as the following sum:

```
(f+ (fdot-select (lead-inds aq) (row i aq) x)
      (fdot-select (free-inds aq n) (row i aq) x))))).
```

Now since (row i aq) has a 1 at index (nth i (lead-inds aq)) and a 0 at all other lead indices, the first of these two sums reduces to the single term (nth (nth i (lead-inds aq)) x), and hence Equation (3) may be expressed as

```
(nth (nth i (lead-inds aq)) x)
= (f+ (car (nth i bq))
                                 (f- (fdot-select (free-inds aq n) (row i aq) x))).
```

Thus, x is a solution of our system of equations iff this condition holds for all i < q. This is checked recursively by the following function:

This provides a test to be applied to a candidate solution:

If q = (len (lead-inds aq)) = n, then (free-inds aq n) = NIL, the equation

(nth (nth i l) x) = (f + (car (nth i bq)) (f - (fdot-select f (nth i aq) x)))

reduces to

(nth i x) = (car (nth i bq),

(solution-test-aux x aq bq l f q) reduces to $x = (col \ 0 \ bq)$, and the last theorem reduces to the earlier result linear-equations-unique-solution-case.

Otherwise, $(free-inds aq n) \neq NIL$ and the components of x corresponding to the indices in (lead-inds aq) are determined by the components corresponding to (free-inds aq n), which are unconstrained. Thus, there is a single solution corresponding to every assignment of values to the latter set of components, and hence infinitely many solutions. We shall revisit this result in Part III, where we show that in the homogeneous case, b = (flistn0 n), the solutions form a vector space of dimension n - q. A basis for this solution space will be provided by a formula derived from the function solution-test.

4 Future Work

This formalization of linear algebra is a work in progress. In Part I, we developed the algebra of matrices over a commutative ring with unity and the theory of determinants. In this sequel, we have restricted our attention to matrices over a field in order to address the process of row reduction and its applications. To allow our results to be applied to an arbitrary ring or field, we have characterized each by an encapsulated set of constrained functions.

There is progress to report on a planned Part III, which begins with another encapsulation that formalizes the notion of an abstract finite-dimensional vector space over the field F. The constrained functions of this encapsulation naturally include a predicate that recognizes vectors in the space, the operations of vector addition and scalar multiplication, and the constant 0 vector. Two additional functions embody the requirement of finite dimensionality: (1) a constant list of vectors of unspecified length that serves as a canonical basis, and (2) a function that returns the coordinates of a given vector with respect to this basis. Thus, whenever we define a concrete vector space, we are obligated to identify a basis for it. This establishes a tight connection between vector spaces and matrices: a list of vectors may be identified by the matrix of coordinates of its members. As an unexpected application of row reduction, this connection provides an algorithmic definition of the basic notion of linear independence without use of quantifiers: a list of vectors is linearly independent iff the row rank of its coordinate matrix is the length of the list.

The reader may have noticed that the definition of the basic notion of row equivalence is omitted from our treatment of row reduction. Recall that two matrices are said to be row equivalent if one may be derived from the other by a sequence of elementary row operations. This definition could be formalized in ACL2 using the support for existential quantification provided by defun-sk, and the properties of an equivalence relation could be derived from the results of Section 2. We could also prove the important theorem that distinct reduced row-echelon matrices cannot be row equivalent. However, since its most expedient proof is based on vector spaces (in particular, the row space of a matrix), this result is postponed to Part III. Note that it provides an alternative definition of row equivalence that avoids quantification: two matrices a and b are row equivalent iff (row-reduce a) = (row-reduce b). Since we prefer this algorithmic formulation, the entire topic is deferred to Part III.

Other topics to be addressed in the sequel include linear transformations and diagonalization. As discussed in Part I, a factor in our decision to develop the algebra of matrices over an arbitrary commutative ring rather than a field is that this allows us to define the characteristic polynomial of a square matrix over the field F as the determinant of a certain matrix over the polynomial ring F[t]. A related objective is the proof of the Cayley-Hamilton Theorem (every square matrix over a commutative ring is a root of its own characteristic polynomial), which has wide-ranging applications in other areas of mathematics.

This project is part of a broader effort in the formalization of algebra, which began with group theory [3, 4, 5] and will continue beyond linear algebra. Our next targeted area of investigation will be Galois theory, which we hope eventually to apply to the study of algebraic number fields. These intended applications guided our choices of formalization schemes for the basic algebraic structures. Since we are interested in infinite rings and fields, the encapsulation approach seems to be the only viable representation scheme for these structures provided by the ACL2 logic. The disadvantages of not being able to refer to such a structure as an ACL2 object are obvious. On the other hand, since the groups of primary interest are finite, our investigation of group theory is limited to the finite case. Under this restriction, a group is conveniently represented as an object characterized by a predicate defined as an ACL2 function. There are, however, infinite groups of interest, which are not accommodated by our theory. For example, the general linear group of invertible matrices over a field, which is in general an infinite structure, would otherwise have provided an interesting example and another connection between group theory and linear algebra.

Although we have no immediate plans to apply this theory beyond the realm of pure mathematics, its potential utility in the formal verification of hardware and software applications is limitless. Linear algebra is central to the rapidly advancing fields of machine learning and neural networks [1], providing essential tools for data representation and manipulation. Along with finite group theory, it is also important to a variety of cryptographic algorithms [2]. It is our hope that ACL2 users who are interested in pursuing such applications may find our results useful.

References

- [1] Sahar Halim (2020): *Application of Linear Algebra in Machine Learning*. International Research Journal of Engineering and Technology 7(2).
- [2] Yuling Qian (2023): *Application of Modern Algebra in Cryptography*. Theoretical and Natural Science 10(1), doi:10.54254/2753-8818/10/20230304.
- [3] David M. Russinoff (2022): A Formalization of Finite Froup Theory. In: ACL2 2022: 17th International Workshop on the ACL2 Theorem Prover and its Applications, Austin, Texas, doi:10.4204/EPTCS.359.10.
- [4] David M. Russinoff (2023): A Formalization of Finite Froup Theory: Part II. In: ACL2 2023: 18th International Workshop on the ACL2 Theorem Prover and its Applications, Austin, Texas, doi:10.4204/EPTCS.393.4.
- [5] David M. Russinoff (2023): A Formalization of Finite Froup Theory: Part III. In: ACL2 2023: 18th International Workshop on the ACL2 Theorem Prover and its Applications, Austin, Texas, doi:10.4204/EPTCS.393.5.
- [6] David M. Russinoff (2025): A Formalization of Elementary Linear Algebra: Part I. In: ACL2 2025: 19th International Workshop on the ACL2 Theorem Prover and its Applications, Austin, Texas.