Matrices in ACL2

Joe Hendrix
University of Illinois at Urbana-Champaign
jhendrix@uiuc.edu

July 5, 2003

1 Introduction

This paper describes some initial work on a formalization of matrices in ACL2. The current work is focused on creating an executable implementation that is simple to mechanically reason with and complete enough to be capable of analyzing real problems in linear algebra. A number of basic operations have been defined including matrix addition, transposition and multiplication by a scalar, a vector, and another matrix. These operations have an extensive library of over 200 theorems to make reasoning about these operations more convenient and automatic. These theorems include the axioms for rings and other rewrite rules that attempt to change matrix expressions into a canonical form. Unlike the work in [2] where the focus was mainly on implementing Strassens algorithm for square matrices where the number of rows and columns is a power of 2, this operations are for arbitrary $m \times n$ matrices.

This paper is intended to be a brief introduction to the matrix library. It is not intended as a reference, and for detailed questions there is no substitute for consulting the source code of the books directly. Hopefully after reading this paper, navigating those books will be easier.

2 Data Representation

In the context of this paper, *vector* refers to a mathematical vector and not to the Common Lisp data type. It is represented as a proper list of numbers. Operations defined on vectors include vector addition, subtraction, multiplication by a scalar, and dot product. The book `vectors` defines these operations, includes a number of theorems, and defines a predicate `mvectorp` for recognizing vectors. It is named `mvectorp` to avoid conflicting with the Common Lisp predicate `vectorp`.

A *matrix* is represented as a list of vectors where each vector represents a row and must have the same number of elements. For technical reasons, `Nil` is considered a valid matrix known as the *empty matrix*, however atoms other

than `Nil` are not considered matrices. Each row vector in a non-empty matrix must contain at least one element.

3 Basic Operations

Before defining the interesting operations about matrices, it turns out to be helpful to define a core set of primitive operations for recognizing, constructing, and destructing matrices. These core operations are defined in terms of basic Lisp operations, but their internal definitions are disabled so that proofs about higher level operations such as matrix multiplication can avoid dealing with the internal representation of matrices. The book `mdefuns` contains the function definitions and performs guard verification. The book `mdefthms` defines the theorems used for theorem proving before disabling the implementation details. The basic operations are summarized below:

Operation	Description
`(matrixp m)`	Predicate that accepts if `m` is a valid matrix.
`(matrix-emptyp m)`	Predicate that accepts if `m` is an atom.
`(empty-matrix)`	Returns `nil`.
`(row-count m)`	Number of rows in the matrix `m` or 0 if `m` is the empty matrix.
`(col-count m)`	Number of columns in the matrix `m` or 0 if `m` is the empty matrix.
`(row-car m)`	Top row of the matrix `m`.
`(row-cdr m)`	Matrix formed by removing the top row of `m`.
`(row-cons v m)`	Matrix whose top row is the vector `v` and whose remaining rows are the rows of the matrix `m`.
`(col-car m)`	Leftmost column of the matrix `m`.
`(col-cdr m)`	Matrix formed by removing the leftmost column of the matrix `m`.
`(col-cons v m)`	Returns the matrix whose leftmost column is the vector `v` and whose remaining columns are the columns of the matrix `m`.

Example 1 Sample Matrix:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$ can be formed from

```
(row-cons `(1 2 3)
          `((4 5 6)
            (7 8 9)))
```

It may seem surprising that `row-count` and `col-count` are included as basic operations. Although they could be defined in terms of the other operations and (and logical definitions using the other operators are created in `mdefthms`), the guard for `row-cons` requires that if `m` is a non-empty matrix, the length of the vector `v` must equal the column count of the matrix `m`, and for `col-cons`,

the length of `v` must equal the row count of `m`. If `m` is the empty matrix, `v` can be any vector containing at least one element.
Since the original definition of each operator is disabled, `mdefthms` contains a logical definition for each basic operation. These logical definitions are mutually recursive and would not terminate if executed, so they cannot be admitted using `defun`, but they can be defined using `defthm` and used for reasoning purposes. In addition, `mdefthms` contains theorems for when the result of these operations is a valid matrix, rewrite rules to simplify terms containing these operators, and a proof about the `acl2-count` of `row-cdr` and `col-cdr` to allow inductive definitions using those operations. In total, there are over 60 different theorems about matrices in this book, and not all of them are enabled by default, because this could lead to infinite loops. The default strategy is to simplify where possible, but otherwise move row operations to the outside and column operations to the inside. This may not always be desirable, so users interested in implementing their own matrix operations using these primitives are encouraged to study the contents of `mdefthms`.

4 Defined Operations

All other operations are defined in terms of the basic operations listed above. For modularity, the operations are broken up into several different books, but can all be imported by including the book `matrices`. The operators currently defined include:

Operation	Description
`(row i m)`	Vector for `i`th row in `m`.
`(col j m)`	Vector for `j`th column in `m`.
`(mentry i j m)`	Entry at row `i` and column `j` in `m`.
`(mzero r c)`	Zero matrix with `r` rows and `c` columns.
`(mid n)`	Identity matrix with `n` rows and columns.
`(m+ m n)`	Adds the matrices `m` and `n`.
`(m- m)`	Multiplies the matrix `m` by -1
`(m- m n)`	Subtracts the matrix `n` from `m`.
`(sm* c m)`	Multiplies each entry in `m` by the scalar `c`.
`(row* v m)`	Returns vector containing the dot product of `v` and each row in `m`.
`(col* v m)`	Returns a vector containing the dot product of `v` and each column in `m`.
`(m* m n)`	Multiplies the matrices `m` and `n`.
`(mtrans m)`	Transpose of `m`.

5 Theorems

Since it is advisable to keep conditions simple on rewrite rules to make them easier to apply, some of the ACL2 theorems may contain fewer conditions than the

mathematical definition would allow. This can make theorem proving simpler, but if it is important that applications do not depend at all on the particular details of the ACL2 representation, guard checking should be performed on all functions using the matrix library. The guards for each operation indicate the proper usage of the functions.
The theorems are often proven by induction on the number of rows of an argument, and case-splitting on whether a matrix is the empty matrix is an often needed tactic. In initially proving the results, forcing the conditions was heavily used, but as the theorems have been simplified that technique was phased out and forcing is no longer required. The theorems are too numerous to mention each specifically in this paper, but they can be broadly classified into a number of different categories.

5.1 Predicate Theorems

Predicate theorems concern the result types of defined operations and when the result is a valid matrix, empty matrix, vector, or number.

Example 2 Addition matrix predicate:

```
(defthm matrixp-m+
  (implies (matrixp m)
           (matrixp (m+ m n))))
```

Note that `n` does not have to be a matrix for `(m+ m n)` to be a matrix.

5.2 Size Theorems

Size theorems concern the number of rows, number of columns, or length of operation results. These are essential conditions for many rewrite rules and are frequently used in backchaining.

Example 3 Column count of addition:

```
(defthm col-count-m+
  (implies (matrixp m)
           (equal (col-count (m+ m n))
                  (col-count m))))
```

5.3 Alternative Definitions

For efficiency reasons, operations are usually defined using `row-cons`, `row-car`, `row-cdr`. These functions operate in constant time whereas the column variants take time proportional to the number of rows in the matrix. However it turns out to be useful, particularly when dealing with transpose, to also have logical definitions of operations using the column constructors and destructors.

Example 4 Addition by columns:

```
(defthm m+-by-col-def
  (implies (and (matrixp m) (matrixp n))
           (equal (m+ m n)
                  (if (matrix-emptyp m)
                      (empty-matrix)
                    (col-cons (v+ (col-car m) (col-car n))
                              (m+ (col-cdr m)
                                  (col-cdr n))))))
  :rule-classes :definition)
```

5.4 Entry Values

Although the higher level operations are the preferred mechanism for manipulating matrices, it may be necessary to prove properties about the contents of matrices. For this, properties about the row, column, and entry values for each operation are defined.

Example 5 Entry of addition:

```
(defthm entry-m+
  (implies (and (matrixp m) (matrixp n))
           (equal (mentry r c (m+ m n))
                  (if (and (< (nfix r) (row-count m))
                           (< (nfix c) (col-count m)))
                      (+ (mentry r c m) (mentry r c n))
                    nil))))
```

5.5 Zero and Identity Matrices

Zero and identity matrices are identity elements of addition and multiplication, and transposition. Theorems have been defined to eliminate operations on these matrices when possible.

Example 6 Transpose of zero:

```
(defthm mtrans-zero
  (equal (mtrans (mzero r c))
         (mzero c r)))
```

5.6 Simplification properties

Simplification properties refer to a broad assortment of properties involved in transforming matrix expressions into a canonical form. This includes operations involving a single operation as well as relations among the different operations.

Example 7 Successive multiplication by a scalar:

```
(defthm sm*-sm*
  (implies (matrixp m)
           (equal (sm* a (sm* b m))
                  (sm* (* a b) m))))
```

Example 8 Transpose of matrix multiplication:

```
(defthm mtrans-m*
  (implies (m*-guard m n)
           (equal (mtrans (m* m n))
                  (m* (mtrans n) (mtrans m))))
```

These theorems generally try to convert a term involving higher level operations into a canonical form where addition operations are moved to the outside, followed by multiplication by a scalar, matrix by matrix multiplication, and with transpose on the inside.

6 Conclusion and Future Work

Future plans include adding operations for inverting matrices and determinates using Gaussian elimination. Except for its lack of matrix inversion, this library could be used for proving the results in [1], but using executable definitions and proven theorems rather than axioms. Linear algebra is a rich and interesting subject with a wide range of practical applications. I hope that this work can serve as the basis for reasoning about such problems in ACL2.

7 Acknowledgements

The original idea to work on this came while I was attending a class at UIUC taught by Grigore Roşu [1]. Many of the theorems about matrices were taken directly from axioms used in some of his work with Kalman filters. In addition, I must thank Miguel Palomino for reviewing a draft copy, and Francisco Palomo-Lozano, J Moore, and Matt Kaufmann for their encouragement to write this paper.

References

[1] Leustean Laurentiu and Grigore Roşu. Certifying Kalman filters. Technical report, RIACS 03.02, January 2003.

[2] F. Palomo-Lozano, I. Medina-Bulo, and J. A. Alonso-Jimenez. Certification of matrix multiplication algorithms: Strassen's algorithm in acl2. In *TPHOLS*, 2001.