University of economics in Prague
Faculty of Informatics and Statistics
Department of Information and Knowledge Engineering

PROLOG: MATRIX CALCULATOR

January 3, 2017

Nguyen Viet Bach
nvbach91@outlook.com

Abstract

This paper is a seminar work documentation for subject 4IZ617 - Logika a sémantika. It deals with the implementation of a text-based SWI-Prolog program that can perform several linear matrix operations.

1 Introduction

The objective of this paper is to implement a simple matrix calculator in Prolog. This program is executable in the SWI-Prolog console and answers to queries on matrix operations within the environment of SWI-Prolog.

1.1 Limitations

The matrix calculator can only perform several operations on matrices listed as follows:

- Addition
- Subtraction
- Scalar multiplication
- Scalar division
- Multiplication of two matrices
- Transposition
- Finding determinant of a square matrix of size 3x3 or smaller
- Finding the main diagonal of the matrix
- Trace of a square matrix

2 Used libraries

This project makes use of one single SWI-Prolog library **clpfd** [3] . Constrain Logic Programming over Finite Domains is an instance of general Constrain Logic Programming scheme, extending logic programming with reasoning over specialized domains. This library clpfd contains the predicate `transpose/2` which is used to transpose matrices.

3 Usage

To use this program, first the user has to load the source code file `prolog-matrix-calculator.pl` into SWI-Prolog by *consulting* it. After the file has been sucessfully consulted, the user can start querying for matrix operations. The syntax of queries are listed as follows:

```
% matrix addition
?- [[1,2,3],[1,2,3],[1,2,3]] plus [[1,2,3],[1,2,3],[1,2,3]].

% matrix substraction
?- [[1,2,3],[1,2,3],[1,2,3]] minus [[1,2,3],[1,2,3],[1,2,3]].

% matrix multiplication by constant
?- [[4,6,8],[2,4,6],[4,6,8]] multiply_c 2.

% matrix division by constant
?- [[4,6,8],[2,4,6],[4,6,8]] divide_c 2.

% matrix multiplication
?- [[1,2,3],[1,2,3],[1,2,3]] multiply [[1,2,3],[1,2,3],[1,2,3]].

% matrix transposition
?- transpose [[1,2,3],[1,2,3],[1,2,3]].

% finding determinant
?- determinant [[1,2,3],[1,-2,3],[4,5,-6]].
```

```
% finding diagonal
?- diagonal [[1,2],[-2,3]].

% trace
?- trace [[1,2],[-2,3]].

% variables operations are valid
?- A plus B.

% calculate arithmetic expressions up to 3 members
?- calc(A+B+C, R).
?- calc(A*B-C, R).
?- calc(A+B-C, R).
?- calc(A*Const/B, R).
?- calc(A+B/Const, R).
.
.
.
```

The result for queries are printed in the console.

4 Implementation

This section focuses on the implementation of the matrix calculator in Prolog.

4.1 Utilities

Several generic predicates are defined to help implement the logic of this program such as `isNumericMatrix` for input checking. These utility predicates are listed below:

`isNumericMatrices(+ListOfMatrices)`	validates a list of matrices
`isNumericMatrix(+Matrix)`	validates one matrix
`isNumericList(+List)`	validates one row in a matrix
`nMatrixRows(+Matrix,-NRows)`	finds vertical dimension of the matrix
`nMatrixColss(+Matrix,-NCols)`	finds horizontal dimension of the matrix
`isSquareMatrix(+Matrix)`	truw if the matrix is square
`numericMultiply(+N1,+N2,-Product)`	Prolog Goal for use with `maplist`
`numericDivide(+N1,+N2,-Quotient)`	Prolog Goal for use with `maplist`
`sumList(+L,-S)`	computes the sum of all elements in a list
`printMatrix(+M)`	pretty print matrix in the console
`isSquareMatrix(+M)`	checks for square matrix

4.2 Operators

Prolog operators are defined to improve the readability of source code. By defining custom operators, one can create a very intuitive interface that can help users to easily understand the usage of the program. Operators can be defined using the built-in predicate `op/3`. In this project, there are several custom operators defined to use with matrix operations. These operators are:

`plus`	matrix addition
`minus`	matrix subtraction
`multiply_c`	scalar multiplication by constant
`divide_c`	scalar division by constant
`multiply`	multiplication between two matrices
`transpose`	matrix transposition
`determinant`	finding determinant of given matrix
`diagonal`	finding main diagonal of given matrix
`trace`	finding trace of given matrix

The syntax of these operators can be seen in section 3.

4.3 Matrix in Prolog

Matrices are usually referred to as rectangular arrays of numbers. In other words, matrices are tables of numbers. This means the numbers are arranged in rows and columns, which are called *dimensions*. A matrix of dimensions 2x3 has 2 rows and 3 columns.

Like many other programming languages, matrices can be represented by arrays of arrays. In Prolog, arrays are called *lists*. Therefore, a matrix is a list of lists of numbers.

4.4 Matrix operations

There are many operations that can be performed over matrices. Given specific rules and methods, matrices can be added together, subtracted from each other, multiplied and so on. The implementation of these operations are written in Prolog using rules and predicates.

4.4.1 Addition and subtraction

The algorithm for addition and subtraction of matrices of the same size is straightforward. The pairs of elements at the same positions in both matrices are added together and the results are projected to a final matrix. For example, let $C = A \pm B$, and then

$$C_{ij} = A_{ij} \pm B_{ij}$$

SWI-Prolog built-int predicate `maplist/4` [4] is used to apply these operations on all elements of the matrix. This predicate is true if Goal (operation template) can successfully be applied on all elements of the given three Lists.

4.4.2 Scalar multiplication and scalar division

Scalar multiplication and scalar division are also very simple. Each element in the matrix is multiplied or divided by the given scalar or constant [2]. Similarly to addition and subtration, scalar multiplication and scalar division can be implemented using `maplist`. But in this project, a more naive algorithm based on list manipulations is used where each row of the matrix is scalar multiplied or divided by the constant and incrementally added to the final list of lists because scalar multiplication and scalar addition give a matrix if the same size.

4.4.3 Transposition

Matrix transposition in linear algebra means to find a transpose of a given matrix. The transpose of matrix A is another matrix A^T where the rows and columns are *swapped*. A^T can also be described as the reflection of A over its main diagonal. Formally:

$$[A^T]_{ij} = [A]_{ji}$$

In this project, the predicate `transpose/2` is used to transpose matrices. This predicate is available in the `clpfd` library of SWI-Prolog.

4.4.4 Multiplication

The multiplication of two matrices is a *non-commutative* operation and it requires a particular properties of both matrices. The number of columns of the first matrix must match the number of columns of the second matrix. Therefore, if A is an $n \times m$ matrix and B is an $m \times p$ matrix, then their matrix product AB is an $n \times p$ matrix. Formally:

$$(AB)_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj}$$

where $(AB)_{ij}$ is the dot product of row i of matrix A and column j of matrix B. The implemented algorithm first checks for the required dimensions of the two input matrices A and B. The second matrix is transposed, then the multiplication is computed using `maplist/3`. Under the hood, dot product of pairs of vectors are computed to find each element $(AB)_{ij}$ of the final matrix AB.

4.4.5 Determinant

The algorithm for finding the determinant of a 2×2 matrix is simple using the criss-cross multiplication and then subtract two numbers.

$$det(A_{2\times 2}) = [A]_{11}[A]_{22} - [A]_{12}[A]_{21}$$

Finding determinant of matrices of size 3×3 is still easy.

$$\begin{aligned} &det(A_{3\times 3}) = \\ &[A]_{11} * ([A]_{22} * [A]_{33} - [A]_{23} * [A]_{32}) - \\ &[A]_{12} * ([A]_{21} * [A]_{33} - [A]_{23} * [A]_{31}) + \\ &[A]_{13} * ([A]_{21} * [A]_{32} - [A]_{22} * [A]_{31}). \end{aligned}$$

But for bigger matrices, the process is more complicated. There are several ways of finding determinants of matrices, for example using Gaussian elimination on the matrix and get the product of the numbers on the main diagonal. Another algorithm is *cofactor expansion* [1]. This algorithm uses *minors* and *cofactors* to find the determinant.

Let $det(A)$ be the determinant of matrix A. A minor matrix $M_{i,j}$ is the square matrix formed by deleting i^{th} row and j^{th} column from a larger square matrix A and $[A]_{ij}$ is the element where i^{th} row and j^{th} column intersect. In the following example, $[A]_{ij} = c$.

$$\begin{pmatrix} \cancel{a} & \cancel{b} & \cancel{c} & \cancel{d} \\ e & f & \cancel{g} & h \\ i & j & \cancel{k} & l \\ m & n & \cancel{o} & p \end{pmatrix} \rightarrow \begin{pmatrix} e & f & h \\ i & j & l \\ m & n & p \end{pmatrix}$$

The cofactor $C_{i,j} = (-1)^{i+j}[A]_{i,j}$ and the determinant of matrix A is:

$$det(A) = \sum_{i=1}^{n} C_{i,j} det(M_{i,j}).$$

By all means, this is a recursive function. Recursion will occur until the algorithm reaches the computation of the determinant of a square matrix of size 2×2 which is trivial. The implementation of finding determinant in SWI-Prolog has the same approach.

References

[1] Stapel, E. Minors and cofactors: Expanding along a row. Available at http://www.purplemath.com/modules/minors.htm. Cit. [Jan 2 2017].

[2] Stapel, E. Scalar and matrix multiplication. Available at http://www.purplemath.com/modules/mtrxmult.htm. Cit. [Jan 2 2017].

[3] SWI-Prolog. Constrain logic programming over finite domains. Available at http://www.swi-prolog.org/man/clpfd.html. Cit. [Jan 2 2017].

[4] SWI-Prolog. maplist/2. Available at http://www.swi-prolog.org/pldoc/man?predicate=maplist/2. Cit. [Jan 2 2017].

Appendix

- Source code `prolog-matrix-calculator.pl` can be found under `src` folder.
- To test the project, query for `-? test.`