A Generic Matrix Manipulator

Dylan Killingbeck

Department of Computer Science,
Brock University,
St. Catharines, Ontario, Canada, L2S 3A1
`dk10qt@brocku.ca`

Abstract. In this paper we describe a generic matrix manipulator system that performs operations on matrices in a flexible way using a graphical user interface. A user defines allowable data entries called a basis, as well as n-ary operations defined on basis elements. These operations can be used in multiple ways to define operations on matrices. A basis and n-ary operations can be entered into the system by various ways including predefined, Java datatypes, JavaScript, and various XML formats defining certain mathematical structures.

Keywords: Allegory, Matrix Manipulation, Semiring, Sup-Semiring

1 Introduction

It is well known that matrices can be used in several key areas of science to provide meaning to data. Often these matrices can be manipulated to provide a solution, based upon the data that they hold and the operations defined between the coefficients. Relations can be represented as graphs, or more specifically as a matrices that defines the relationships between nodes and vertices [7, p. 6]. Using this method of representation an observer can verify a qualitative relationship between elements associated within the relation, and then preform further study using relation algebraic definitions and constructions. For example matrices can be used to represent a relation given as a graph and determine if a Hamiltonian cycle exists. Matrices in generalized linear algebra can also be used to represent quantitative information, and similarly this can provide further information about the meaning of the data. To provide an example, a matrix can represent an interconnected network by which the probability for success of a message traveling between two nodes is represented by a value on the unit interval.

Between the matrix representations of qualitative and quantitative information, it is complex to find meaning, and as such a generic system with user specified parameters is required to further study their behavior. A generic system that is able to flexibly manipulate matrices by user defined operations lifted from the coefficients to matrices, with the inclusion of a user defined data sets (or basis) will allow for an observer to reason between matrices more easily. This generic system will perform similar to the RelView system (see [1]) through a graphical user interface, however it will expand further upon matrix operations, and representations. Primarily, this generic system will focus on automatically inferencing types of operations, and checking the results to ensure compatibility, expanding on RelViews limitations. The remainder of this paper will discuss the mathematical preliminaries, and finally the implementation of this generic matrix manipulator system in detail.

2 Mathematical Preliminary

This section will define the basic definitions of concepts and properties associated with the quantitative and qualitative aspect of matrices. We will start with the quantitative aspect by introducing semirings as a very general approach to linear algebra.

2.1 Semirings

We want to provide a structure that capture the quantitative and qualitative information as discussed in the introduction. We summarize the theory presented in [5] and start with the definition of a semiring.

Definition 1. *$\langle R$,+,$*$,0_R,$1_R\rangle$ denotes a semiring if:*

1. *R is a commutative monoid, i.e. we have:*
   (a) *x+0_R=x for all x∈R ,* *(Identity)*
   (b) *x+(y+z)=(x+y)+z for all x,y,z∈R ,* *(Associativity)*
   (c) *x+y=y+x for all x,y∈R ,* *(Commutativity)*
2. *R is a monoid i.e., we have:*
   (a) *x$*$$1_R$= $1_R$$*$x = x for all x∈R ,* *(Identity)*
   (b) *x$*$(y$*$z)=(x$*$y)$*$z for all x,y,z∈R ,* *(Associativity)*
3. *Multiplication will distribute over addition, from both the left and the right:*
   (a) *x$*$(y+z) = x$*$y+x$*$z ,* *(Left Distributivity)*
   (b) *(x+y)$*$z = x$*$z+y$*$z ,* *(Right Distributivity)*
4. *0_R is the annihilator for multiplication over R, meaning:*
   (a) *$0_R$$*$x=$0_R$=$0_R$$*$x for all x∈R ,* *(Annihilator Law)*

A commutative semiring is a semiring where the monoid $\langle R, *, 1_R\rangle$ is commutative. An additively idempotent semiring is a semiring so that $x + x = x$ for all $x \in R$, similarly a multiplicatively idempotent semiring is semiring so that $x * x = x^2 = x$ for all $x \in R$. We will denote the set of multiplicative idempotent elements of a semiring R by $I(R)$.

Semirings are widely used in theoretical computer science, for example in parsing formal languages [3], or if a semiring is commutative and idempotent (multiplicatively) then it forms a lower semilattice with respect to multiplication. A generalization of this property is stated in the first theorem below [5].

Theorem 1. *Let $\langle R, +, *, 0, 1\rangle$ be a commutative semiring. Then $\langle I(R), *, 0, 1\rangle$ is a lower semilattice with least element 0 and greatest element 1.*

2.2 Allegory

An allegory is the generalization of the category of binary relations between two sets. A morphism R from source A and target B is denoted as $R : A \to B$, from category $\mathcal{R}$, with all the possible morphisms denoted as $\mathcal{R}[A, B]$ [2]. Composition is denoted by ;, for example $R; S$, which reads first R and then S. $\mathbb{I}_A$ denotes the identity morphism for an object A.

Definition 2. *A category $\mathcal{R}$ is an allegory if:*

1. *The class of morphisms $\mathcal{R}[A, B]$ form a lower semilattice, with meet denoted by $\sqcap$ and the induced ordering by $\sqsubseteq$. Elements within this class are called relations.*
2. *For all relations Q, there is a converse such that R:A→B and S:B→C the following holds: $(Q; S)^{\smile} = S^{\smile}; Q^{\smile}$, and $(Q^{\smile})^{\smile} = Q$.*
3. *For all relations Q: A→B, R, S:B→C, then Q;(R⊓S) ⊑ Q;R⊓Q;S.*
4. *For all relations Q:A→B, R:B→C and S:A→C, the modular law Q;R ⊓ S ⊑ Q;(R ⊓ $Q^{\smile}$;S) holds.*

Finally $\mathcal{R}[A, B]$ is a distributive allegory if $\mathcal{R}[A, B]$ is a distributive lattice with join $\sqcup$ and least element $\perp\!\!\!\perp_{AB}$, satisfying the additional properties:

5. *$Q; \perp\!\!\!\perp_{BC} = \perp\!\!\!\perp_{AC}$ for all relations $Q : A \to B$,*
6. *$Q; (R \sqcup S) = Q; R \sqcup Q; S$ for all relations $Q : A \to B$, $R, S : B \to C$.*

Rel, the category of binary relations between sets as well as the category L-Rel of L-valued relations between sets form a distributive allegory.

2.3 Matrices over Semirings

Matrices of size $m \times n$ will form over a semiring of equal size, induced by addition, denoted by $+$, for example if R is a semiring denoted by $\langle R, +, *, 0_R, 1_R \rangle$ and M is an $m \times n$ matrix, then $M = [a_{ij}]_{mn}$ where coefficients a_{ij} are elements from R. Regular matrix addition and multiplication is respectively defined by:

$$[a_{ij}]_{mn}+[b_{ij}]_{mn} = [a_{ij}+b_{ij}]_{mn}, \quad \text{and} \quad [a_{ij}]_{mn} * [b_{jk}]_{np} = [\sum_{j=1}^{n} a_{ij} * b_{jk}]_{mp}$$

Similarly we can define the transpose of a matrix, and the Hadamard product of two equal sized matrices, respectively defined by:

$$[a_{ij}]_{mn}^{\smile} = [a_{ji}]_{mn}, \quad \text{and} \quad [a_{ij}]_{mn} \cdot [b_{ij}]_{mn} = [a_{ij} * b_{ij}]_{mn}$$

0_R is defined as the zero matrix $[0]_{mn}$, and 1_R is defined at $[1]_{mn}$. $M + [0]_{mn} = M$, and $M * [0]_{mn} = [0]_{mn}$. Additionally as + is commutative then $M + M' = M' + M$, and as + is associative $M + (M'+M'') = (M + M') + M''$. A matrix that consists of only multiplicatively idempotent elements as coefficients is idempotent with respect to the Hadamard product.

It is a well-known fact that matrices over a lower semilattice form an allegory which leads to the following theorem.

Theorem 2. *Consider the category of matrices with coefficients from a commutative semiring. Then the subcategory of idempotent matrices with respect to the Hadamard product forms an allegory.*

2.4 Sup-Semiring

Recall idempotent elements from a semiring form a lower-semilattice, alternatives matrices with idempotent coefficients form an allegory. We will use these concepts to formulate a new structure, to derive a distributive lattice, and a distributive allegory.

Definition 3. *$\langle D, +, *, \sqcup, 0_D, 1_D \rangle$ denotes a sup-semiring if:*

1. *$\langle D, +, *, 0_D, 1_D \rangle$ is a commutative semiring.*
2. *$\langle D, \sqcup \rangle$ is a commutative semigroup, then*
   (a) *$x \sqcup (y \sqcup z) = (x \sqcup y) \sqcup z$ for all x,y,z∈D,* *(Associativity)*
   (b) *$x \sqcup y = y \sqcup x$ for all x,y∈D,* *(Commutativity)*
3. *$(x \sqcup y) * (x \sqcup y) = x \sqcup y$ for all x,y∈D,* *(Relative Idempotency)*
4. *$x * (x \sqcup y) = x$ for all x,y∈D,* *(Absorption)*
5. *if $x^2 = x$, then $x \sqcup (x * y) = x$ for all x,y∈D,* *(Relative Absorption)*
6. *if $x^2 = x, y^2 = y$ and $z^2 = z$,*
   then $x * (y \sqcup z) = x * y \sqcup x * z$ for all x,y,z∈D. *(Relative Distributivity)*

In the case of a sup-semiring we are able to strengthen Theorem 1:

Theorem 3. *Let $\langle D, +, *, \sqcup, 0, 1 \rangle$ be a sup-semiring. Then the structure $\langle I(D), *, \sqcup, 0, 1 \rangle$ is a distributive lattice.*

Similarly, in the case of a sup-semiring, we are able to strengthen Theorem 2 as follows.

Theorem 4. *Consider the category of matrices with coefficients from a sup-semiring. Then the subcategory of idempotent matrices with respect to the Hadamard product forms a distributive allegory.*

3 Brief Description of the System

As already mentioned in the introduction, a generic matrix manipulator system will be required to carry out various operations and provide meaning to these combined structures. This section will outline the various features, implementations and user interactions provided by the system. The matrix manipulator system is constructed using the Java environment as it offers flexibility to the developer, which can be passed on to the user. This system has previously been started by Milene Santos Teixeira as a project [6]. During this project the basic matrix operations such as matrix multiplication, addition etc have been implemented as methods that use operations on the coefficients of the matrix as parameters. The coefficients and their operations are loaded from user-defined files in XML format. Currently the system is further developed by the author as part of his MSc. thesis.

3.1 User Input

The heart of the system relies on input provided by a user. A user must supply information about two components of the system. The first, a basis to outline coefficients (elements), as well as operations between these coefficients. Secondly, operations between matrices must also be defined. Once these components are defined, a user can supply commands to the system to manipulate matrices. The manipulation possibilities includes:

1. Performing operations on coefficients (elements) within a matrix
2. Performing operations between matrices
3. Storing results, recalling results through executing equations.

Essentially a user will use the graphical user interface to facilitate the input of the basis component and any desirable operations.

Once information has been loaded into the system, the user is free to manipulate the constructed environment within the system as desired, through the graphical user interface, as well as inputting commands through an input field. This input field will accept only valid input based on stored matrices (variables), and the user defined operations between matrices. In order to facilitate such input a flexible parser must be constructed at runtime to parse user input, based on the environment. JParsec, an implementation of Haskell Parsec, is used as it is a two stage parser that provides the flexibility required to parse user input at runtime, given operators with specified operator priority [8].

3.2 User Defined Matrix

Matrices represented in this generic system can be of various forms to provide flexibility. A user may specify the values of the coefficients that correspond specifically to the source (row) and the target (column) values. To be even more general, a matrix can consists of only 1 object type A, and having morphisms that such that F is a morphism, then $F : A \to A$. This type of matrix is convenient to represent Boolean matrices, matrices with real coefficients, or even matrices with integer coefficients. To be more specific, matrices within the system can also represent homogeneous relations, or heterogeneous relations. To provide an example, the following figure below demonstrates a relation with sources along the rows, and targets along the columns.

$$\mathrm{M} = \begin{array}{c|cccc} & A & B & A & A \\ A & 1 & 3 & 2 & 1 \\ B & 2 & 4 & 3 & 1 \end{array}$$

Fig. 1. Typical matrix with sources A and B and targets A,B,A and A

Source and target objects, along with their morphisms must produce a result that is defined and allowable in the system, which is enforced by the user defined basis.

3.3 User Defined Basis

As previously outlined, a user defines a basis which specifies the allowable environment values and n-ary operations between the elements (coefficients). For example, if the basis is defined as the set $\mathbb{N}$, then possible operations could be addition, multiplication, etc. as defined for natural number. Operations defined within the basis must be closed on the given environment. In other words, the operations must accept arbitrary values from then environment as arguments and must return a value within the environment as a result. To make the system as generic as possible, these operations can be loaded into the system through various ways:

1. Java datatypes, i.e., int, double, boolean...
2. XML formats, i.e., lattice structures, graphs...

Below is a typical explicit example, whereby the user defined basis is supplied by .xml files. The first .xml file defines the allowable objects, and data types for the various source and target morphisms (integer in this case):

```
<?xml version="1.0" encoding="UTF-8"?>

<basis name="Example␣Basis" type="explicit">
    <objects type="String">
        A,B
    </objects>
    <morphisms type="Integer" symmetric="true">
        <morSet source="A" target="A">1,2</morSet>
        <morSet source="A" target="B">1,2,3</morSet>
        <morSet source="B" target="B">1,2,3,4,5</morSet>
    </morphisms>
</basis>
```

Above we can see that this basis is defined for only A and B objects, and is explicitly defined. Morphism maps are defined for each source and target object as per above, that is to say with a source A and a target A, the resulting value can only be an integer, with a value of 1 or 2. Since this explicit morphism is defined as symmetric, a source A and a target B is the same as a source B and a target A.

In addition to the user supplied basis .xml file, the user must provide operation for the matrix coefficients, as mentioned above. A typical user defined operation in .xml format is supplied below:

```
<?xml version="1.0" encoding="UTF-8"?>

<unaryOP name="My␣Op" basis="Example␣Basis"
                              code="+" notation="postfix">
    <objectMap type="explicit">
        (A,A),(B,B)
    </objectMap>
    <morphismMap type="explicit" symmetric="true">
        <morMap source="A" target="A">
            (1,2),(2,1)
        </morMap>
        <morMap source="A" target="B">
            (1,2),(2,1),(3,3)
        </morMap>
        <morMap source="B" target="B">
            (1,2),(2,4),(3,2),(4,5),(5,1)
        </morMap>
    </morphismMap>
</unaryOP>
```

Above we can see that the operation is a unary operator, with a name, code, notation type and corresponding basis. The notation maps the first value of the tuple, to the given second value in the tuple. For example, if the source is A and the target is A and the value is 1, the resulting value of the $+$ operation is 2.

To make the system more convenient, various mathematical structures can be automatically generated by the user. For example, if the user provides a Hasse diagram, then it can be used to generate the corresponding matrix representation, and basis.

3.4 User Defined Operations

Recall, a user is able to specify operations between matrices. These operations have an arbitrary number of parameters, hence the operations can theoretically be n-ary operators, keeping the concept of being generic. These operations between matrices are comprised of the operations defined within the basis. Furthermore, there are two ways that these n-ary operators can be evaluated on matrice. First, operations can be evaluated component wise as defined above, for example matrix addition defined for elements in $M(2, \mathbb{R})$ (the set of all 2×2 matrices with real coefficients), or more specifically the Hadamard product is component wise multiplication. Secondly, operations between two matrices can be defined similar to regular matrix multiplication, as defined above. This type of operation requires two binary operations, the first to provide a result between two coefficients, and the second combines the results of the first operation. An example of this can be observed through regular matrix multiplication in $M(2, \mathbb{R})$, where regular multiplication is the first operation, and the second operation is addition.

Operations between matrices must rely on the underlying operations defined in the basis. This is to ensure that the operations between matrices produce results containing elements defined by the basis. For example, using regular addition defined for matrices with real coefficients, implies that the addition between coefficients is also defined within the basis. These operations can therefore be described as higher order functions, or functions built by using previously defined functions.

Similar to a user defined basis, a user may define these operations between matrices through various methods to increase convenience and usability. A user can define an operation by several methods including:

1. XML formats, i.e. explicitly defined.
2. JavaScript, i.e., user defined functions parsed from JavaScript.
3. Java built-in operations, i.e. addition, multiplication, min, max...
4. Special mathematical structures such as lattices, additive cyclic groups $\mathbb{Z}_n^+$ modulo an integer n, or multiplicative monoids/groups $\mathbb{Z}_n^*$ modulo an integer n.

A user defined operation between matrices must require a symbol to represent the operation for user command input, this symbol is required as part of the operation definition. To further enhance the flexibility of the system, operations can either be in prefix, infix or postfix format.

Finally, operations defined by this system do not require special properties such as associativity or commutativity. The system is more general then required to investigate matrices over semirings or sup-semirings, and in fact any arbitrary matrices can be used, with corresponding user provided operations.

3.5 Graphical User Interface

Another important component of the matrix manipulator system is the graphical user interface (GUI). The GUI allows the user to have a visual representation of the data set they are working with, and also input commands into the system. Overall, the matrix manipulator system will have a comparable interface molded after the RelView system [1]. The GUI is written using Java Swing components to provide a familiar interface, rich component listing as well as reduce developer complexity. The user is able to modify the working space to modify the representations of matrices, for example they can modify the font, size and spacing of coefficients within the matrix. The GUI will provide feedback to the user in the form of a visual display, and also provide feedback be denoting working conditions such as the operations defined, or basis loaded.

4 Conclusion

In order to investigate and simplify matrix manipulation, the creation of a formal system is required. This paper has outlined the requirements, and otherwise desirable features as needed to facilitate such a system. The more flexible the generic matrix manipulator system is, the more use it will be to those requiring such a tool to investigate structures. In closing, the system is currently in development as part of the author's MSc. thesis.

References

1. Berghammer R.: Relview System. `http://www.informatik.uni-kiel.de/~progsys/relview/`: Christian-Albrechts-University of Kiel (2012).

2. Freyd P., Scedrov A.: Categories, Allegories. North-Holland (1990).
3. Goodman J.: Semiring Parsing. Computational Linguistics, 25(4), 573-605 (1999).
4. Hebisch U., & Weinert H.: Semirings. World Scientific (1998).
5. Killingbeck D., Santos Teixeira M., Winter M.: Relations among Matrices over a Semiring. In Kahl W., Oliveira J.N., Winter M. (Eds.): Relational and Algebraic Methods in Computer Science (RAMiCS 15), LNCS 9348, 96-113 (2015).
6. Santos Teixeira M.: A Generic Matrix Manipulator. Undergraduate Project (COSC 3P99), Brock University (2014).
7. Schmidt G., Ströhlein T.: Relations And Graphs. Springer (1993).
8. Yu B.: JParsec. `https://github.com/jparsec/jparsec/`: GitHub (2014).