REALM - A SYSTEM TO MANIPULATE RELATIONS

By

Zafor Ahmed

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
AT
BROCK UNIVERSITY
ST. CATHARINES, ONTARIO
DECEMBER 2009

Table of Contents

Abstract

Given a heterogeneous relation algebra $\mathcal{R}$, it is well known that the algebra of matrices with coefficient from $\mathcal{R}$ is relation algebra with relational sums that is not necessarily finite. When a relational product exists or the point axiom is given, we can represent the relation algebra by concrete binary relations between sets, which means the algebra may be seen as an algebra of Boolean matrices. However, it is not possible to represent every relation algebra. It is well known that the smallest relation algebra that is not representable has only 16 elements. Such an algebra can not be put in a Boolean matrix form.[15]

In [15, 16] it was shown that every relation algebra $\mathcal{R}$ with relational sums and sub-objects is equivalent to an algebra of matrices over a suitable basis. This basis is given by the integral objects of $\mathcal{R}$, and is, compared to $\mathcal{R}$, much smaller.

Aim of my thesis is to develop a system called ReAlM - Relation Algebra Manipulator - that is capable of visualizing computations in arbitrary relation algebras using the matrix approach.

Chapter 1

ReAlM - A System to Manipulate Relations

1.1 Introduction

The calculus of relation algebra traces back its origin all the way to the second half of the last century with the pioneering work on binary relations of G. Boole, A. de Morgan, C.S. Peirce, and E. Schröder. Tarski and his co-workers also have been contributed significantly toward the present day axiomatic development [13]. Relation algebra is a fruitful base for describing fundamental concepts such as graphs, combinatorics, orders, lattices and games in mathematics as well as relational databases and program correctness and verification in computer science. This has been widely accepted by mathematicians and computer scientists over the past two decades.

Under certain circumstances, i.e. relational products exist or the point axiom is given, a relation algebra may be represented by concrete relations between sets [8, 14]. Any concrete relation can be visualized by a Boolean matrix. The rows of the matrix correspond to elements of the source and the columns to the elements of the target of the relation. A 'true' (or '1') entry in the i-th row and j-th column indicates that

the elements i and j are in relation. Similarly, a 'false' (or '0') entry means that the corresponding elements are not in relation.

Even though the set of operations in a relation algebra is quite large, most of those operations can be defined in terms of a small set of basic operations. This set of basic operations may include operations such as union, intersection and composition. Furthermore, these operations can easily be implemented on Boolean matrices in a similar way to the operations on linear maps in linear algebra. It is also possible to graphically represent concrete relations over finite sets using directed graphs in a computer system. With the currently available programming technologies and interfacing techniques, it is possible to build a computer system capable of computing relational programs and relations.

One such system, which is already available with the capability of calculating relational programs, is RELVIEW. This system is written in C programming language and runs under the X Windows System and makes full use of the graphical user interface. The first version of the RELVIEW was written at the University of the German Forces Munich and later redesigned and extended at Kiel University. RELVIEW can be used to solve many different tasks while working with relational algebra, concrete relations, relations based on discrete structures and relational programs. For further details and some examples using RELVIEW in applications we refer to [2, 3].

As known, not every relation algebra is representable [9] and therefore is not an algebra of Boolean matrices. The currently available RELVIEW system works with

Boolean matrices, and can, therefore, be used to work only within the class of representable relation algebras. In [16] it was shown that in every relation algebra $\mathcal{R}$ with relational sums and subobjects it is possible to characterize a full subalgebra $\mathcal{B}$, called the basis of $\mathcal{R}$, such that the matrix algebra $\mathcal{B}^+$ with the coefficients from $\mathcal{B}$ is equivalent to $\mathcal{R}$. The objects of $\mathcal{B}$ are the integral objects of $\mathcal{R}$. Integral objects are defined similar to integral domains in algebra, i.e. the endorelations of an integral objects permit no zero divisors. This property can equivalently be characterized by the fact that the identity morphism is an atom.

In my thesis, I want to develop a system called ReAlM (Relation Algebra Manipulator) similar to RELVIEW but based on the theory above. With the ReAlM system it would be possible to work with arbitrary heterogeneous relation algebras. The high interactive nature of ReAlM allows the users to formulate the proof of relational theorems and to visualize the computations.

1.2 Heterogeneous Relation Algebras

Relation algebras were originally introduced as homogeneous structures, i.e. every relation was supposed to be defined on the same global universe U [13, 14]. A variation of this theory has evolved where relations are considered as heterogeneous or rectangular from the beginning, i.e. relations where the normal case is that they are relations between two different sets. In this chapter, we provide the basic definitions of the theory of heterogeneous relation algebras [5, 11, 12]. We assume that the reader is familiar with the basic notions of category and lattice theory. For any notion not defined here we refer to [1, 4, 7, 10].

Definition 1.2.1. *A (heterogeneous abstract) relation algebra is a locally small category $\mathcal{R}$ consisting of a class $Obj_{\mathcal{R}}$ of objects and a set $\mathcal{R}$ [A,B] of morphisms for all $A, B \in Obj_{\mathcal{R}}$. The morphisms are usually called relations. Composition is denoted by ";" and identities are denoted by $\mathbb{I} \in \mathcal{R}[A, A]$. In addition, there is a totally defined unary operation $\smile_{AB}: \mathcal{R}[A, B] \to \mathcal{R}[B, A]$ between the set of morphisms, called conversion. The operations satisfy the following rules:*

1. *Every set $\mathcal{R}$[A,B] carries the structure of a complete atomic boolean algebra with operations $\sqcup_{A,B}, \sqcap_{A,B}, -_{A,B}$, order relation $\sqsubseteq$, zero element $\perp\!\!\!\perp_{A,B}$, universal element $\top_{A,B}$.*

2. *For all relations Q, R and S the Schröder equivalence holds:*

   $Q; R \sqsubseteq S \Leftrightarrow Q^{\smile}; \overline{S} \sqsubseteq \overline{R} \Leftrightarrow \overline{S}; R^{\smile} \sqsubseteq \overline{Q}$.

The category Rel with sets as objects and binary relations as morphisms is the standard example of a heterogeneous relation algebra. As already mentioned in the introduction, relations in that category can be visualized by Boolean matrices.
The Schröder equivalences can be easily memorized by the following: converse the first (or second), then complement and permute the other two.
In the following example, we will be using kinship or family relation and demonstrate how Schröder equivalences can be applied.

Example 1.2.1 Let the following relations be given:

1. B for the relation "is **B**rother of",

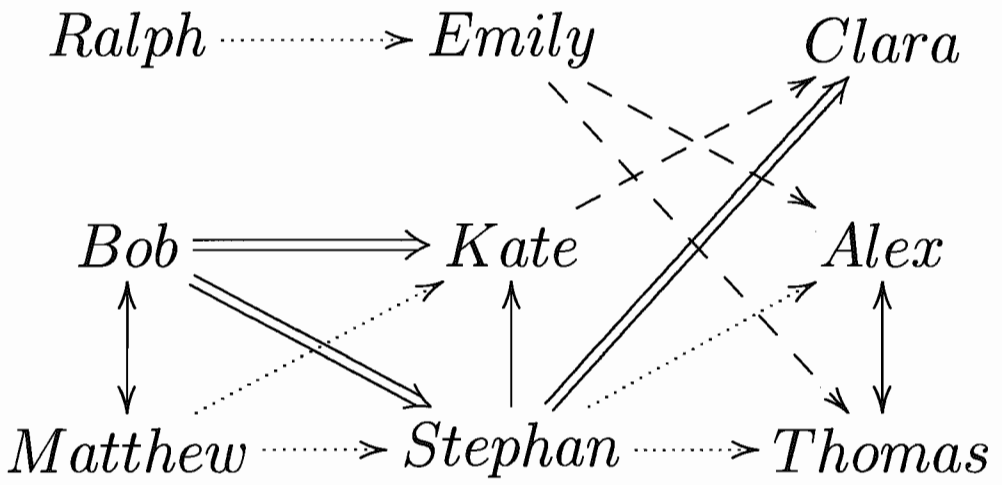
2. F for the relation "is **F**ather of",

3. M for the relation "is **M**other of".

In addition, let G denote the relation "is **G**odfather of". Then, $P := F \sqcup M$ means "is **P**arent of" and B ; P means "is **U**ncle of". Now, let's assume that uncles in our family traditionally become godfathers. So we have $B; P \sqsubseteq G$. If we apply Schröder equivalence to the family relation above, we will see that both formula $B^{\smile}; \overline{G} \sqsubseteq \overline{P}$ and $\overline{G}; P^{\smile} \sqsubseteq \overline{B}$ holds.

$B^{\smile}; \overline{G} \sqsubseteq \overline{P}$ can be read as : If a family member x has a brother z, who is not a godfather of y, then x can not be a parent of y. The latter statement must be true because, otherwise, contrary to family tradition, x would be an uncle of y without being y's godfather.

We want to illustrate this situation even further using a concrete family and their relationships visualized by the following diagram and Boolean matrcies.



Legend :

Brother $\longrightarrow$

Father $\cdots\cdots\cdots\cdots\cdots\rightarrow$

Mother $- - - - - \rightarrow$

Uncle $(B;P)$ $\Longrightarrow$

$F =$

	R	B	M	E	K	S	C	A	T
R	0	0	0	1	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0
M	0	0	0	0	1	1	0	0	0
E	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	1	1
C	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0

$M =$

	R	B	M	E	K	S	C	A	T
R	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0	1	1
K	0	0	0	0	0	0	1	0	0
S	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0

$B;P =$

	R	B	M	E	K	S	C	A	T
R	0	0	0	0	0	0	0	0	0
B	0	0	0	0	1	1	0	0	0
M	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	1	0	0
C	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0

$B =$

	R	B	M	E	K	S	C	A	T
R	0	0	0	0	0	0	0	0	0
B	0	0	1	0	0	0	0	0	0
M	0	1	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0
S	0	0	0	0	1	0	0	0	0
C	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	0	0	1
T	0	0	0	0	0	0	0	1	0

$G =$

	R	B	M	E	K	S	C	A	T
R	0	0	0	0	0	0	0	0	0
B	0	0	0	0	1	1	0	0	0
M	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	1	0	0
C	0	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0

$B^{\smile}; \overline{G} =$

	R	B	M	E	K	S	C	A	T
R	0	0	0	0	0	0	0	0	0
B	1	1	1	1	1	1	1	1	1
M	1	1	1	1	0	0	1	1	1
E	0	0	0	0	0	0	0	0	0
K	1	1	1	1	1	1	0	1	1
S	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0
A	1	1	1	1	1	1	1	1	1
T	1	1	1	1	1	1	1	1	1

Now, we would like to use the relation matrices and try to explain the Schröder equivalence using relations between certain family members.

For example, Bob is a godfather of Stephan. This follows from the fact that Bob is a brother of Matthew and Matthew is the father of Stephan, i.e. Bob is an uncle of Stephan, and the fact that according to the tradition an uncle becomes a godfather, $B; P \sqsubseteq G$ as a formula.

As a second example we have that Bob is not a brother of Stephan. In fact, Bob is not a godfather of Thomas who is a son of Stephan. According to the formula $\overline{G}; P^{\smile} \sqsubseteq \overline{B}$, which is equivalent to $B; P \sqsubseteq G$ using the Schröder equivalences, we obtain that Bob is not a brother of Stephan.

As a last example, we know that Bob is not a parent of Stephan. Using the third inclusion $B^{\smile}; \overline{G} \sqsubseteq \overline{P}$ obtained from the Schröder equivalences this follows from the fact that Bob has a brother Matthew who is not a godfather of Stephan. ♦

1.3 Matrix Algebras

Given a heterogeneous relation algebra $\mathcal{R}$, we can define an algebra of matrices $\mathcal{R}^+$ with coefficients from $\mathcal{R}$.

Definition 1.3.1. Let $\mathcal{R}$ be a relation algebra. The algebra $\mathcal{R}^+$ of matrices with coefficients from $\mathcal{R}$ is defined by:

1. *The class of objects of* $\mathcal{R}^+$ *is the collection of all functions from an arbitrary set* I *to* Obj_R.

2. *For every pair* $f : I \to Obj_{\mathcal{R}}$, $g : J \to Obj_{\mathcal{R}}$ *of objects from* $\mathcal{R}^+$, *the set of morphisms* $\mathcal{R}^+[f,g]$ *is the set of all functions* $R : I \times J \to Mor_{\mathcal{R}}$ *such that* $R(i,j) \in \mathcal{R}[f(i),g(j)]$ *holds.*

3. *For* $R \in \mathcal{R}^+[f,g]$ *and* $S \in \mathcal{R}^+[g,h]$ *composition is defined by*
   $(R;S)(i,k) := \bigsqcup_{j \in J} R(i,j);\ S(j,k)$.

4. *For* $R \in \mathcal{R}^+[f,g]$ *conversion and negation is defined by*
   $R^{\smile}(j,i) := (R(i,j))^{\smile}$ *and* $\overline{R}(i,j) := \overline{R(i,j)}$

5. *For* $R, S \in \mathcal{R}^+[f,g]$ *union and intersection are defined by*
   $(R \sqcup S)(i,j) := R(i,j) \sqcup S(i,j)$
   and
   $(R \sqcap S)(i,j) := R(i,j) \sqcap S(i,j)$

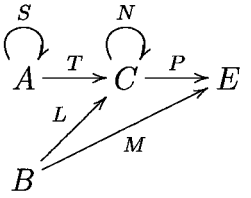
6. *Identity, empty and universal elements are defined by :*
   $$\mathbb{I}_f(i_1,i_2) := \begin{cases} \mathbb{I}_f(i_1) & : \ i_1 = i_2 \\ \perp\!\!\!\perp_{f(i_1)f(i_2)} & : \ i_1 \neq i_2 \end{cases}$$
   $\perp\!\!\!\perp_{f,g}(i,j) := \perp\!\!\!\perp_{f(i)g(j)}$, $\top_{f,g}(i,j) := \top_{f(i)g(j)}$.

In the following examples we want to demonstrate the definition above.

Example 1.3.1 [Objects and Relations]

$$\begin{array}{ccccc} f & \longrightarrow & g & \longrightarrow & h \\ I \to Obj_{\mathcal{R}} & & J \to Obj_{\mathcal{R}} & & K \to Obj_{\mathcal{R}} \end{array}$$

In the diagram above, f, g and h are the objects of $\mathcal{R}^+$ which are themselves (infinite) lists of objects of relation algebra $\mathcal{R}$. If that list of objects is finite, we write the object as a list. e.g. as [A,B,A,C]. Now, we can visualize the concept as follows: Assume that we are given the following relations from a relation algebra $\mathcal{R}$ (besides the constant relations $\perp\!\!\!\perp, \top, \mathbb{I}$) :



Then we can define a relation Q in $\mathcal{R}^+$ with source [A,B,C] and target [A,C,E] using the coefficients from $\mathcal{R}$ above:

$$Q = \begin{array}{c|ccc} & \mathbf{A} & \mathbf{C} & \mathbf{E} \\ \hline \mathbf{A} & S & T & \perp\!\!\!\perp \\ \mathbf{B} & \perp\!\!\!\perp & L & M \\ \mathbf{C} & \perp\!\!\!\perp & N & P \end{array} \blacklozenge$$

Example 1.3.2 [Composition]

Let $\mathcal{R}$ be the relation algebra with one object A and the following 4 relations:

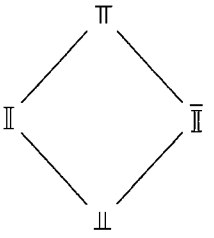
1. $\top$ for the universal relation,

2. $\perp\!\!\!\perp$ for the empty relation,

3. $\mathbb{I}$ for the identity relation,

4. $\overline{\mathbb{I}}$ for the complement of the identity.

The order structure of the relations above is given by the following diagram:



The composition table for those relations is:

;	$\perp\!\!\!\perp$	$\mathbb{I}$	$\overline{\mathbb{I}}$	$\top\!\!\!\top$
$\perp\!\!\!\perp$	$\perp\!\!\!\perp$	$\perp\!\!\!\perp$	$\perp\!\!\!\perp$	$\perp\!\!\!\perp$
$\mathbb{I}$	$\perp\!\!\!\perp$	$\mathbb{I}$	$\overline{\mathbb{I}}$	$\top\!\!\!\top$
$\overline{\mathbb{I}}$	$\perp\!\!\!\perp$	$\overline{\mathbb{I}}$	$\mathbb{I}$ or $\top\!\!\!\top$	$\top\!\!\!\top$
$\top\!\!\!\top$	$\perp\!\!\!\perp$	$\top\!\!\!\top$	$\top\!\!\!\top$	$\top\!\!\!\top$

As we can see from the table above, that composition of identity complement with itself could be the $\mathbb{I}$ or $\top\!\!\!\top$ (universal relation), which means we have actually specified two relation algebras.

Let $R = \begin{bmatrix} \perp\!\!\!\perp & \top\!\!\!\top & \mathbb{I} \\ \overline{\mathbb{I}} & \top\!\!\!\top & \perp\!\!\!\perp \end{bmatrix}$ and $S = \begin{bmatrix} \perp\!\!\!\perp \\ \mathbb{I} \\ \overline{\mathbb{I}} \end{bmatrix}$. Then $R;S = \begin{bmatrix} \perp\!\!\!\perp \cup \top\!\!\!\top \cup \overline{\mathbb{I}} \\ \perp\!\!\!\perp \cup \top\!\!\!\top \cup \perp\!\!\!\perp \end{bmatrix} = \begin{bmatrix} \top\!\!\!\top \\ \top\!\!\!\top \end{bmatrix}$.

Let $R = \begin{bmatrix} \bot\!\!\bot & \mathbb{I} & \overline{\mathbb{I}} \\ \overline{\mathbb{I}} & \top & \bot\!\!\bot \end{bmatrix}$ and S $= \begin{bmatrix} \top \\ \mathbb{I} \\ \overline{\mathbb{I}} \end{bmatrix}$. Then $R;S = \begin{bmatrix} \bot\!\!\bot \cup \mathbb{I} \cup \overline{\mathbb{I}};\overline{\mathbb{I}} \\ \top \cup \top \cup \bot\!\!\bot \end{bmatrix} = \begin{bmatrix} \mathbb{I} \text{ or } \top \\ \top \end{bmatrix}$. ♦

In the second computation we can see that the composition of $\overline{\mathbb{I}}$ and $\overline{\mathbb{I}}$ could result in identity or $\top$ depending of the composition table in $\mathcal{R}$.

Example 1.3.3 [Conversion and Complement]

If S is a matrix with source $[A, B]$ and target $[B, A]$ given by $\begin{bmatrix} \bot\!\!\bot & \mathbb{I} \\ \mathbb{I} & \top \end{bmatrix}$ where A and B are two different objects with existing relations, then $S^{\smile}$ would be:

$$S^{\smile} = \begin{bmatrix} (\bot\!\!\bot)^{\smile} & (\mathbb{I})^{\smile} \\ (\mathbb{I})^{\smile} & (\top)^{\smile} \end{bmatrix} = \begin{bmatrix} \bot\!\!\bot & \mathbb{I} \\ \mathbb{I} & \top \end{bmatrix}.$$

The complement of S can be computed as $\overline{R}(i,j) := \overline{R(i,j)}$

$$\overline{S} = \begin{bmatrix} \overline{\bot\!\!\bot} & \overline{\mathbb{I}} \\ \overline{\mathbb{I}} & \overline{\top} \end{bmatrix} = \begin{bmatrix} \top & \overline{\mathbb{I}} \\ \overline{\mathbb{I}} & \bot\!\!\bot \end{bmatrix}. \blacklozenge$$

Example 1.3.4 [Union and Intersection]

If we have a matrix X $= \begin{bmatrix} \top & \mathbb{I} \\ \bot\!\!\bot & \bot\!\!\bot \end{bmatrix}$ and matrix Y $= \begin{bmatrix} \bot\!\!\bot & \overline{\mathbb{I}} \\ \top & \top \end{bmatrix}$,

then $X \sqcap Y = \begin{bmatrix} \bot\!\!\bot & \bot\!\!\bot \\ \bot\!\!\bot & \bot\!\!\bot \end{bmatrix}$ and $X \sqcup Y = \begin{bmatrix} \top & \top \\ \top & \top \end{bmatrix}$. ♦

Example 1.3.5 [Identity, Empty and Universal relation]

For the object $[A, B]$ we obtain the identity, the empty and the universal relation as $\begin{bmatrix} \mathbb{I} & \perp\!\!\!\perp \\ \perp\!\!\!\perp & \mathbb{I} \end{bmatrix}$, $\begin{bmatrix} \perp\!\!\!\perp & \perp\!\!\!\perp \\ \perp\!\!\!\perp & \perp\!\!\!\perp \end{bmatrix}$ and $\begin{bmatrix} \top\!\!\!\top & \top\!\!\!\top \\ \top\!\!\!\top & \top\!\!\!\top \end{bmatrix}$ respectively. Notice that the coefficients of the previous matrices denote relations between (or on) different objects. For example, the identity at position $(1,1)$ of the first matrix is the identity on A whereas the one at position $(2,2)$ is defined on B. ♦

1.4 Integral Objects and the Basis of $\mathcal{R}$

As we have stated in the introduction, it is possible to characterize a full subalgebra $\mathcal{B}_{\mathcal{R}}$ which is the basis of $\mathcal{R}$. This basis $\mathcal{B}_{\mathcal{R}}$ is constructed by using the integral objects of the relation algebra $\mathcal{R}$. An integral object is defined similar to an integral domain in abstract algebra [6] as an object that does not have zero devisors. It turns out that in a relation algebra this can equivalently be characterized by the property that the identity is an atom. Normally, the basis $\mathcal{B}_{\mathcal{R}}$ is much smaller than the original relation algebra R, i.e. $\mathcal{B}_{\mathcal{R}}$ and R are not isomorphic. In this section, we will look at the characteristics of the integral objects which are the building blocks of the basis $\mathcal{B}_{\mathcal{R}}$ for the relation algebra $\mathcal{R}$.

Definition 1.4.1. *An object A of a relation algebra is called integral* iff $\perp\!\!\!\perp_{AA} \neq \top\!\!\!\top_{AA}$ *and for all $Q, R \in R[A, A]$ the equation $Q;R = \perp\!\!\!\perp_{AA}$ implies either $Q = \perp\!\!\!\perp_{AA}$ or $R = \perp\!\!\!\perp_{AA}$. The basis of $\mathcal{R}$ is the full subcategory induced by the integral objects of $\mathcal{R}$.*

Suppose we have a basis that has one object A and two relations 0,1 on A with 0 being the zero or empty relation and 1 being the identity and universal relation. This basis can be seen as the Boolean values, 0=false, 1=true. In this sense, the algebra

of Boolean matrices is the matrix algebra over this particular basis.

Example 1.4.1 If Q=$\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$ and R =$\begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$ then we get Q;R =$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$.

Let us assume that the source (and the target) [A,A] of Q and R are integral objects. According to the definition of integral objects, either Q or R must be equal to zero. But this is not the case here. As we can see both $Q \neq \perp\!\!\!\perp$ and $R \neq \perp\!\!\!\perp$ therefore the object [A,A] is not integral. ♦

Lemma 1.4.2. *The following properties are equivalent:*

1. *A is an integral object.*
2. *Every non-empty relation $R : A \to A$ is total, i.e. $\mathbb{I}_A \subseteq R; R^{\smile}$.*
3. *$\mathbb{I}_A$ is an atom.*

A proof can be found in [3].

Being integral depends on the relations that are available.

Example 1.4.3 In the preceding example the object $[A, A]$ was not integral. In total there are 2^4 = 16 relations with source and target $[A, A]$. If we consider fewer

relations, i.e. a different relation algebra that contains only $\perp\!\!\!\perp$, $\top$, $\mathbb{I}$ and $\bar{\mathbb{I}}$, then $[A, A]$ becomes integral.

For example, $\top_A ; \perp\!\!\!\perp_A$ is $\perp\!\!\!\perp_{AA}$, when $\perp\!\!\!\perp_A$ is $\perp\!\!\!\perp_A$, which satisfies the definition. ♦

1.5 Subobjects

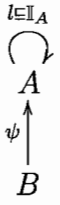
In order to show that the matrix algebra over the basis of a relation algebra $\mathcal{R}$ is equivalent to $\mathcal{R}$ we need the concepts of subobjects and relational sums. In this section we want to start with subobjects [5, 12].

Definition 1.5.1. *Let $\ell \in \mathcal{R}[A, A]$ be a partial identity, i.e. $l \sqsubseteq \mathbb{I}_A$. An object B together with a relation $\psi \in \mathcal{R}[B, A]$ is called a subobject of A induced by ℓ iff*

$$\psi; \psi^{\smile} = \mathbb{I}_B, \psi^{\smile}; \psi = \ell.$$

A relation algebra has subobjects iff for each partial identity, a subobject exists.

We can visualize the definition from the following diagram:



Example 1.5.1 Here we would like to use the diagram above to formulate a concrete example to explain the concept of subobjects. Let's assume that we have a relation A given by a set of colours and that A has partial identity.

$A = \{red, green, blue, yellow, violet, orange, white, black\}$

$\ell = \{(red, red), (green, green), (black, black)\}$

$B = \{red, green, black\}$

$\psi : B \to A$

	Red	**Green**	**Blue**	**Yellow**	**Violet**	**Orange**	**White**	**Black**
Red	1	0	0	0	0	0	0	0
Green	0	1	0	0	0	0	0	0
Blue	0	0	0	0	0	0	0	0
Yellow	0	0	0	0	0	0	0	0
Violet	0	0	0	0	0	0	0	0
Orange	0	0	0	0	0	0	0	0
White	0	0	0	0	0	0	0	0
Black	0	0	0	0	0	0	0	1

Now, set $B = \{red, green, black\}$ and $\psi : B \to A$

	Red	**Green**	**Blue**	**Yellow**	**Violet**	**Orange**	**White**	**Black**
Red	1	0	0	0	0	0	0	0
Green	0	1	0	0	0	0	0	0
Black	0	0	0	0	0	0	0	1

In the preceding example, if we take i as an atom then we get an integral object. For example, we may take $\ell_i = \{(red, red)\}$.

	Red	**Green**	**Blue**	**Yellow**	**Violet**	**Orange**	**White**	**Black**
Red	1	0	0	0	0	0	0	0
Green	0	0	0	0	0	0	0	0
Blue	0	0	0	0	0	0	0	0
Yellow	0	0	0	0	0	0	0	0
Violet	0	0	0	0	0	0	0	0
Orange	0	0	0	0	0	0	0	0
White	0	0	0	0	0	0	0	0
Black	0	0	0	0	0	0	0	0

. ♦

1.6 Relational Sum

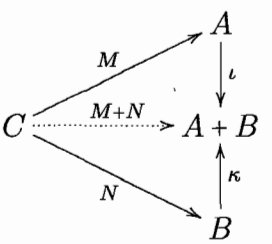
The relational sum corresponds to a disjoint union of sets. This construction actually establishes a categorical product in a relation algebra [5, 12]. In this section we define sums and provide some examples. We also define some derived operations on relations called sum, co-sum and disjoint union.

Definition 1.6.1. *Let A and B be objects of a relation algebra $\mathcal{R}$. An object $A + B$ together with two relations $\iota : A \longrightarrow A + B$ and $\kappa : B \longrightarrow A + B$ is called a relational sum of A and B iff*

1. $\iota ; \iota^{\smile} = \mathbb{I}_A$
2. $\kappa ; \kappa^{\smile} = \mathbb{I}_B$
3. $\iota ; \kappa^{\smile} = \mathbb{1}_{AB}$
4. $(\iota^{\smile} ; \iota) \sqcup (\kappa^{\smile} ; \kappa) = \mathbb{I}_{A+B}$

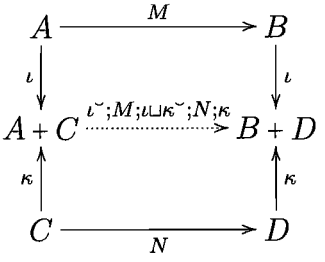
$\mathcal{R}$ is said to have binary (or finite) sums if a relational sum exists for every pair of objects.

If $M : C \to A$ and $N : C \to B$ are arbitrary relations, then the sum $M + N$ is defined by $M ; \iota \sqcup N ; \kappa$. This construction is visualized in the following diagram.



The co-sum of $M : A \to C$ and $N : B \to C$ is simply defined as the converse of the sum of $M^\smile$ and $N^\smile$, i.e. it is the expression $\iota^\smile; M \sqcup \kappa^\smile; N$.

If $M : A \to B$ and $N : C \to D$ are arbitrary relations, then the disjoint union is defined by $\iota^\smile; M; \iota \sqcup \kappa^\smile; N; \kappa$. This construction is visualized in the following diagram.



Example 1.6.1 We can demonstrate the construction of sum and disjoint union using concrete Boolean matrices. Let Q=$\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$ and R =$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, then as a result of sum we get $Q + R = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$.

Disjoint union using the A and B above would look like: $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$. ♦

A generalization of binary sums to arbitrary (not necessarily finite) sums is straightforward. Since this concept is not used in this thesis we refer for further details to [15, 16]. Consequently, we say that a relation algebra has arbitrary sums if a relational

sum exists for every set of objects.

1.7 Theorem

Now that we have defined and explained relational sums, subobjects and integral objects $\mathcal{B}$, we are ready to present the main theorem upon which this paper is based.

Theorem 1.7.1. *Let $\mathcal{R}$ be a relation algebra with arbitrary relational sums and subobjects, and $\mathcal{B}$ be the basis of the $\mathcal{R}$. Then $\mathcal{R}$ and $\mathcal{B}^+$ are equivalent.*

This theorem states that $\mathcal{R}$ and $\mathcal{B}^+$ are equivalent in the sense of category theory [1, 10]. The two structures, in general, are not isomorphic since the functor from $\mathcal{R}$ to $\mathcal{B}^+$ may identify isomorphic objects.

1.8 Splitting

An important additional construction in relation algebras is splittings. They combine subobjects as defined earlier and the process of computing equivalence classes of an equivalence relation into one basic concept [5, 12].

Definition 1.8.1. *Let $Q : A \to A$ be a partial equivalence relation, i.e. $Q^{\smile} = Q$ and $Q;Q = Q$. An object B together with a relation $S : B \to A$ is called a splitting of Q (or S splits Q) iff $S;S^{\smile} = \mathbb{I}_B$ and $S^{\smile};S = Q$.*

We can explain the splitting with the following sequences of examples:

Example 1.8.1 Let A be the set of persons where

$$A = \{Charles, James, David, Chris, Keith, Derek, Barb\}$$

and let Q be the partial equivalence on persons where two persons are equivalent if their age is in the same decade. Lets also assume that ages of Charles and James are 25 and 29 respectively, ages of David, Chris and Keith are 36, 38 and 29 respectively, age of Derek is 45 and age of Barb is 55. Consequently, Q is a homogenous relation on A, i.e. $Q : A \to A$, with equivalence classes of persons with age in the same decade. As a Boolean matrix we get:

	Charles	**James**	**David**	**Chris**	**Keith**	**Derek**	**Barb**
Charles	1	1	0	0	0	0	0
James	1	1	0	0	0	0	0
David	0	0	1	1	1	0	0
Chris	0	0	1	1	1	0	0
Keith	0	0	1	1	1	0	0
Derek	0	0	0	0	0	1	0
Barb	0	0	0	0	0	0	1

If we denote the set of equivalence classes with B then, the splitting R of Q is a relation from the set B of equivalence classes of Q to A. Using Q we get the following 4 equivalence classes [Charles],[David],[Derek] and [Barb]. R relates each equivalence class of B with the elements of A. As a Boolean matrix we obtain R as:

	Charles	**James**	**David**	**Chris**	**Keith**	**Derek**	**Barb**
[Charles]	1	1	0	0	0	0	0
[James]	0	0	1	1	1	0	0
[David]	0	0	0	0	0	1	0
[Chris]	0	0	0	0	0	0	1

Now, if we compute $R; R^{\smile}$ we get the following matrix which is the identity of B:

	[Charles]	[James]	[David]	[Chris]
[Charles]	1	0	0	0
[James]	0	1	0	0
[David]	0	0	1	0
[Chris]	0	0	0	1

And if we compute $R^{\smile}; R$ we get the following relation which is Q.

	Charles	**James**	**David**	**Chris**	**Keith**	**Derek**	**Barb**
Charles	1	1	0	0	0	0	0
James	1	1	0	0	0	0	0
David	0	0	1	1	1	0	0
Chris	0	0	1	1	1	0	0
Keith	0	0	1	1	1	0	0
Derek	0	0	0	0	0	1	0
Barb	0	0	0	0	0	0	1

R satisfies both properties of the definition above so that R is indeed a splitting of Q in this example. ♦

Example 1.8.2 As a second example we would like to use a partial identity of Q, i.e. a subset of the equivalence relation Q. Let A={1,2,3,4,5,6,7,8,9,10} and Q be the partial identity generating the subset of prime numbers. In this case there will

be only one number in each equivalence class at most. As a Boolean matrix we get:

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0	0	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	1	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	1	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0

The splitting R of Q relates every (non empty) equivalence class B to its elements. In this example every equivalence class has at most one element which are [2], [3], [5], and [7] and some elements such as 1, 4, 6, 8, 9, and 10 do not belong to any equivalence class. If we represent the splitting R as a matrix, we get the following:

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
[2]	0	1	0	0	0	0	0	0	0	0
[3]	0	0	1	0	0	0	0	0	0	0
[5]	0	0	0	0	1	0	0	0	0	0
[7]	0	0	0	0	0	0	1	0	0	0

This example also shows that the concept of subobjects is just a special case of a splitting. ♦

Example 1.8.3 As a final example we would like to combine both situations,

i.e. equivalence classes with one or more elements and elements that do not belong to any class.

Let A be again the set of persons where

$$A = \{Charles, James, David, Chris, Keith, Derek, Barb\}.$$

Now, let Q be the partial equivalence relation on those persons having studied the same topic at the university. In this example, let's assume Charles and Chris did not go to university so that they did not study any topic. Let's also assume that James and Barb studied computer science and David and Derek studied History. Keith studied Physics. We obtain the following Boolean matrix for Q:

	Charles	**James**	**David**	**Chris**	**Keith**	**Derek**	**Barb**
Charles	0	0	0	0	0	0	0
James	0	1	0	0	0	0	1
David	0	0	1	0	0	1	0
Chris	0	0	0	0	0	0	0
Keith	0	0	0	0	1	0	0
Derek	0	0	1	0	0	1	0
Barb	0	1	0	0	0	0	1

Let B be given by the set of (existing) equivalence classes for this example. There are exactly three such equivalence classes, namely: [James],[David] and [Keith]. Since James and Barb studied Computer Science, Derek and David studied History and Keith studied Physics. So the elements of each equivalence class are related to each other within that class by Q. Now, Let R be the splitting of Q which should relate each of the equivalence classes to its elements ie. $R : B_Q \to A$. Then we will have R:

	Charles	James	David	Chris	Keith	Derek	Barb
[James]	0	1	0	0	0	0	1
[David]	0	0	1	0	0	1	0
[Keith]	0	0	0	0	1	0	0

. ♦

Chapter 2

ReAlM - Design Specifications

This section of the ReAlM documentation contains the technical details of the system, the design description, platform and system requirements. The Design Overview Section provides a high level overview of the system and the Detail Design Specification Section describes the major components of ReAlM in more detail.

2.1 Scope

Scope of this document is to provide a detail description of the ReAlM system. The system architecture, the software components, the user interface, and the basis files syntax are described in the remaining sections of this chapter.

2.2 Java Platform

The Java platform targeted here is JDK 1.6 mainly because of its support of generic types which made it much easier to implement `RelAlg`, `FiniteRelAlg`, `MatrixAlg` as well as `Matrix`.

2.3 Naming Convension

In general, to name the package, class, attributes and methods we have followed standard Java naming convension. In case of class names, first alphabet of each word has been capitalized and spaces between the words have been removed. Camel case is used in naming the methods and attributes.

2.4 Design Overview

In the system ReAlm , the abstract class `RelAlg` implements a heterogeneous relation algebra. The class has two generic type parameters O and M for the type of objects and the type of the relations. It has several accessor methods for performing binary and unary operations between objects. It defines abstract methods for performing standard operations and the abstract methods `source` and `target` for getting the source and target objects of a relation.

The class `FiniteRelAlg` extends from the `RelAlg` and is itself an abstract class. This class contains inherited abstract accessor methods from `RelAlg` to get relations between two objects of the relation algebra, a hashmap of all relations for each pair of objects in the algebra and two abstract methods for retrieving the list of objects and the list of relations from the basis.

The class `MatrixAlg` is a sub-class of `RelAlg`. This class is not a sub-class of `FiniteRelAlg` because the set of objects of a matrix algebra is not finite. This class provides implementation of all inherited abstract methods from class `RelAlg` specific to matrices. This class also provides implementation of operations like sum, co-sum, disjoint union.

The class `Basis` represents a basis of a matrix algebra. In order to be able to

handle such a structure on a computer this class is a sub-class of `FiniteRelAlg`. This class is initialized by reading / parsing the XML file with a set of objects, relations between them as well as operations available.

2.5 Component Details

2.5.1 Basis File Structure

A potential basis for the relation algebras in ReAlM is stored in XML format for ease of representation extensibility purpose. Basis files including their associated schema definition (.xsd) are stored in "Basis" subdirectory of the project. The XML schema file is also used to validate the contents of a basis file.

A basis file starts with the root element "FiniteRelAlg" where the name of the basis is specified in the attribute `name`. The comment element `<comment>...</comment>` can be used to provide a brief description of the basis.

The `relations` tag is used to specify the number of relations between two objects. An example might be like

```
<relations source="A" target="A" number="2"/>.
```

This tag indicates that there are four relations between the objects `A` and `A`. These relations will be represented later in the system by the numbers 0,1,2, and 3.

The `identity`, `top`, and `bottom` tags are used to specify relations between objects using decimal values. Sample usage of these tags might be:

```
<identity object="A" relation="1"/>
<bottom source="A" target="A" relation="0"/>
<top source="A" target="A" relation="3"/>.
```

The `identity` tag indicates that identity relation of an object A is represented by value 1, the `bottom` tag indicates that bottom relation between A and A is represented by value 0 and the `top` tag indicates that top relation between A and A is represented by value 3.

Having defined the relations, we define operations between pair of objects for all combinations. Basis file only defines the standard relational operations. We use `union`, `intersection`, `composition`, `transposition` and `complement` tags to define the operation between objects. For example, `union` can be used in the following way to define a union operation between two objects A and B, i.e.

```
<union source="A" target= "B"> 0,1,1; 1,0,1; 0,0,0; 1,1,1 </union>.
```

The `union` tag uses ";" as a delimiter among different union operations. The above `union` tag specifies four different union operation. If we look at the first union operation, it states - union operation between relation "0" having source A and target B, and relation "1" having source A and target B would result in relation "1" having source A and target B.

For a detailed structure of the basis and XML schema file, we refer to the appendix A.

2.5.2 Loading and Saving

ReAlM starts with an empty state, ie. no basis and no relations. It provides options through its interface to load basis, relations and to save relations. The method `private void load(File fileName)` is used to load a previously serialized data as a HashMap into the variable `relationMap` object, then to add all relation names to

the display list on left of the Panel. While loading a saved relation list from file, it also verifies that the relation list indeed belongs to the currently selected basis, otherwise it generates an error message to the user for unsuccessful operation.

The method `public void initializeBasis(String filename)` method is used for loading the Basis file into the ReAlM system. It takes the name of the basis file as an argument and does not return any value. Once the basis file is parsed, the method checks if the basis is already loaded into the system. If the basis is not currently loaded, the method adds the basis into variable `r1Map` which is an HashMap storing all the loaded basis. This method also adds the name of the basis to current display list and updates the display information.

The method `private void save(File fileName)` is used to save the list of relations to a file so that it can be later loaded and reused. While saving a relation list, it also indicates the basis name to which this list belongs so that it can be verified during the load process at later time. The method generates an error message if the relation list is empty while the save command is executing and also handles unexpected fail exceptions.

2.5.3 XML Parsing

Since the basis information is stored as an XML file, XML processing is a significant part of ReAlM. Most of the XML parsing takes place in the class `Basis` where the basis gets initialized after parsing. This class defines several member variables to store the name, description, object list, relation list, relations between objects and operations between objects. Relations are stored in a HashMap where a `RelType` object is used as a key for the value of a `BasisMorphs` object. Operations are also

stored as HashMap.

The constructor `public Basis(Node node)` in the class `Basis` takes a single parameter which is the root of the XML document and then it traverses all the nested child elements. The method `getChildNodes` is called upon the argument node to get a complete list of the child nodes in the XML document. After that the program loops through list and tries to identify each element by performing a string match on the name attribute of those elements to initialize appropriate member variables. During this process the method `getLocalName()` is used to retrieve the name attribute of the node elements.

For parsing the XML file, ReAlM uses class `XMLReader<E>` from COSC3P40 package which uses the Dom parser for parsing the XML document. COSC3P40 package is available at the course website of "Advanced Object-Oriented Programming (COSC 3P40)" of Brock University and accessible to all students for use.

2.5.4 Operations

The private class `newMatrixHandler` is used to generate the standard relations such as identity relation, universal relation, empty relation and diversity relations. This private class is primarily responsible for error checking and input validation such as relation names, empty parameters and duplicate relation names. It also updates the display of the ReAlM interface according to the requested operation. Once a valid request has been made, this class calls the method `createMatrix` to handle the actual relation generation process. The method `createMatrix` based on the type of operation, uses the class `Basis` member variables and operations to generate the requested relation. Once the relation has been computed, the method adds the

relation to the relation list and updates the display of the list on the interface.

The private class `newMatrixInjectionHandler` handles the creation of non-standard relations using left injection and right injection. This private class plays the same role as the the class `newMatrixHandler` does above. On execution this class calls the method `createInjectedMatrix` with appropriate parameters for actual formation of the relations. The method `createInjectedMatrix` performs a general validation of the input and ensures the condition for injection are satisfied such as for right injection, source objects must correspond to target objects in last half and for left injection, source objects must correspond to target objects in the first half. If the requirements are met, this function then uses methods defined in the class `Basis` to formulate the injected relation.

The private class `operationMatrixHandler` is used to handle requests related to standard matrix operations which include union, intersection, complement, composition and conversion. Upon basic input validation and pre-requisites being checked, this class calls the method `performStdOperation` with appropriate parameters to perform the actual operation. The Method `performStdOperation` calls appropriate function from the class `Basis` based on the type of operation is requested. All functions take either one or two relations as an input and return one relation as an output. If the operation is successful, the new relation is added to the relation list. If the operation did not succeed, this method produces meaning full error messages for the users.

In the class hierarchy of ReAlM, we have the class `RelAlg` at the top level. This is an abstract class representing behavior of general relation algebra. It has a specialized

version which is called `FiniteRelAlg` outlining the behavior of a finite relation algebra. This class is also abstract. The class `MatrixAlg` which extends the `FiniteRelAlg` is a concrete implementation of the class `FiniteRelAlg` representing a matrix algebra in ReAlM system. To perform any operation on relation matrices, ReAlM uses an instance of the class `MatrixAlg` to call functions implemented in it which are inherited from its parent abstract classes. Most functions take one or two matrices as an input and produce one matrix as an output. Functions in this class use the input matrices to call appropriate functions defined in class `Matrix`. The class `Matrix` provides the detailed implementation of relational operations such as union, intersection, composition, conversion, complement, sum, co-sum and disjoint union for matrices. Inputs for these methods are usually a relation matrix and/or an operation object, which are implemented in the class `RelAlg`. Methods that handle the binary and unary tests on the relations in ReAlM are also defined in the class `MatrixAlg`. This class also implements the method `split` which computes the splitting of a given relation of the algebra.

2.5.5 Interface and Display

ReAlM has several methods for generating the easy to use and interactive interface. Most of these methods are located in class `applicationInterface`. The public constructor `applicationInterface()` generates the overall display of the ReAlM which uses gridbag layout to divide the display in to various panels such as relations lists display, basis information display, operation tabs, new relation generation tabs etc. Tabbed display for generating new relations section is handled by the method `buildNewMatrixPanel` and tabbed display for various operations on the relations is

handled by the method `buildOperationPanel`.

The other major component of the ReAlM interface is the center panel where a relation is displayed and altered interactively. The method `drawMatrix` generates a Panel which contains the display of a relation. Each cell of the relation matrix is constructed using an instance of a private class called `square`. This private class uses standard graphics methods `drawRect` and `drawString` in its `paintComponent` method to draw the relation symbols on the `JPanel`. The class `square` also has a mouse listener that adds appropriate pop-up menu that should be displayed to user when a given cell of the relation matrix is clicked from the display panel.

2.6 Component Details

Class ReAlM	This class contains the static main method where the execution begins. This class instantiate the ApplicationInterface.java which generates the GUI for ReAlM system.
Attributes	None
Constructor	None
Methods	public static void main(String[] args)

Class RelAlg < O, M >	This is an abstract class which represents the relation algebra. This class defines a set of abstract methods specific to relation algebra.
Attributes	None
Constructor	None

Methods	public abstract O source(M r);
	public abstract O target(M r);
	public abstract M getIdentityRelation(O src);
	public abstract M getUniversalRelation(O src, O trg);
	public abstract M getEmptyRelation(O src, O trg);
	public abstract M union(M r, M s);
	public abstract M intersection(M r, M s);
	public abstract M composition(M r, M s);
	public abstract M complement(M r);
	public abstract M transposition(M r);
	public abstract M split(M r);
	public BinOperation< M > getUnionOp()
	public BinOperation< M > getIntersectionOp()
	public BinOperation < M > getCompositionOp()
	public UnarOperation< M > getComplementOp() // returns unary operation

Class FiniteRelAlg	This is a subclass of RelAlg which is also an abstract class. This class defines methods related to finite relation algebra.
Attributes	private Map< $RelType < O >, List < M >>$ objectRelationMap
Constructor	None

Methods	public abstract List< O > getObjList();
	public abstract List< M > getRelList();
	public abstract M getIdentityRelation(O src);
	public abstract M getUniversalRelation(O src, O trg);
	public abstract M getEmptyRelation(O src, O trg);
	public abstract M getRelation(O src, O trg, O type);
	public abstract List< M > getRelation(O src, O trg);
	public abstract Map< $RelType < O >, List < M >>$ getObjectRelationMap();
	public List< M > getRelList(O source, O target);

Class MatrixAlg	This class represents the Matrix algebra and provides all the necessary methods for matrix operations.
Attributes	private FiniteRelAlg< O, M > alg;
Constructor	public MatrixAlg(FiniteRelAlg< O, M > rel)
Methods	public List< O > source(Matrix< M > mat) public List< O > target(Matrix< M > mat) public Matrix< M > getEmptyRelation(List< O > src,List< O > trg) public Matrix< M > getIdentityRelation(List< O > src) public Matrix< M > getUniversalRelation(List< O > src,List< O > trg) public Matrix< M > union(Matrix< M > m1, Matrix< M > m2) public Matrix< M > intersection(Matrix< M > m1, Matrix< M > m2) public Matrix< M > complement(Matrix< M > m1) public Matrix< M > composition(Matrix< M > m1, Matrix< M > m2) public Matrix< M > transposition(Matrix< M > m1) public Matrix< M > sum(Matrix< M > m1, Matrix< M > m2) public Matrix< M > coSum(Matrix< M > m1, Matrix< M > m2) public Matrix< M > disjointUnion(Matrix< M > m1, Matrix< M > m2, O abR[], O abC[]) public boolean binaryEqual(Matrix< M > m1, Matrix< M > m2) public boolean unaryTest(Matrix< M > m1,Matrix< M > orig, String typ) public Matrix< M > split(Matrix< M > m1)

Class Matrix	This is a Matrix object in the ReAlM system. It provides related accessor methods to manipulate the matrix and also provides methods to perform on the elements of matrices.

Attributes	private int i; // ROW length
	private int j; //column length
	private String source[]; // source objects of Matrix
	private String target[]; // target objects of Matrix
	private X[][] mat;

Constructor	public Matrix(int i, int j)
Methods	public int getRowDim() public int getColDim() public X[][] getMat() public X get(int x, int y) public void set(int x, int y, X v) public Matrix< X > addBy(Matrix< X > m, BinOperation< X > op) public Matrix< X > complementBy(UnarOperation< X > op) public Matrix< X > composeBy(Matrix< X > m, BinOperation< X > op, BinOperation< X > op2) public Matrix< X > transposeBy(UnarOperation< X > op) public Matrix< X > sumBy(Matrix< X > m) public Matrix< X > coSumBy(Matrix< X > m) public Matrix< X > disjointUnionBy(Matrix< X > m) public String toString()

Class BoolAlg	This is a subclass of FiniteRelAlg and represents Boolean algebra. It provides methods related to Boolean algebra which operated on Boolean values 1 and 0.
Attributes	private List< *String* > objList; private List< *Boolean* > relList; private Map¡RelType< *String* >, *List* < *Boolean* >> objectRelationMap
Constructor	public BoolAlg(String s)
Methods	public Boolean getIdentityRelation(String src) public Boolean getUniversalRelation(String src, String trg) public Boolean getEmptyRelation(String src, String trg) public Boolean getRelation(String src, String trg, String type) public List< *Boolean* > getRelation(String src, String trg) public Map< *RelType* < *String* >, *List* < *Boolean* >> getObjectRelationMap() // returns the relationMAP between objects public List< *String* > getObjList() public List< *Boolean* > getRelList() public String source(Boolean b) public String target(Boolean b) public Boolean union(Boolean b1, Boolean b2) public Boolean transposition(Boolean b1) public Boolean composition(Boolean b1, Boolean b2) public Boolean intersection(Boolean b1, Boolean b2) public Boolean complement(Boolean b1) public Boolean split(Boolean r)

Class Basis	This class represents the basis in the ReAlM system. It contains the objects of the algebra, all the relations between the objects and operations among them. This class is instantiated by parsing the basis information stored in an xml file.
Attributes	private String basisName; private String basisDescription; private List< *String* > objList; private List< *BasisMorphs* > relList; private Map < *RelType* < *String* >, *BasisMorphs* > identity; private Map < *RelType* < *String* >, *BasisMorphs* > top; private Map < *RelType* < *String* >, *BasisMorphs* > bottom; private Map< *Pair* < *BasisMorphs*, *BasisMorphs* > , *BasisMorphs* >union; private Map < *Pair* < *BasisMorphs*, *BasisMorphs* > , *BasisMorphs* >intersection; private Map < *Pair* < *BasisMorphs*, *BasisMorphs* > , *BasisMorphs* >composition; private Map < *BasisMorphs*, *BasisMorphs* >transposition; private Map < *BasisMorphs*, *BasisMorphs* >complement; private Map< *RelType* < *String* >, *List* < *BasisMorphs* >> objectRelationMap; private RelType < *String* > tempRel;
Constructor	public Basis(Node node) public Basis()
Methods	public static Basis load(String fileName) public void parseObjects(String objects) public void parseOperation(String src, String trg,String ops, String opType) public void parseCompOperation(String src, String intm, String trg,String ops, String opType) public BasisMorphs getIdentityRelation(String src) public BasisMorphs getUniversalRelation(String src, String trg) public BasisMorphs getEmptyRelation(String src, String trg) public BasisMorphs getRelation(String src, String trg, String type) public List¡BasisMorphs¿ getRelation(String src, String trg) public String getBasisName()

Methods	public String getBasisDescription() public Map< *RelType* < *String* >, *List* < *BasisMorphs* >> getObjectRelationMap() public List< *String* > getObjList() public List< *BasisMorphs* > getRelList() public int getIdentity(String src, String trg) public int getTop(String src, String trg) public int getBottom(String src, String trg) public Map< *RelType* < *String* >, *BasisMorphs* > getBottomMap() public Map < *RelType* < *String* >, *BasisMorphs* > getTopMap() public String source(BasisMorphs r) public String target(BasisMorphs r) public BasisMorphs split(BasisMorphs r) public BasisMorphs union(BasisMorphs r1, BasisMorphs r2) public BasisMorphs intersection(BasisMorphs r1, BasisMorphs r2) public BasisMorphs complement(BasisMorphs r) public BasisMorphs transposition(BasisMorphs r) public BasisMorphs composition(BasisMorphs r1,BasisMorphs r2)

Class ApplicationInterface	This class is responsible for generating the main display panel as well as creating instances of other classes that generates parts of the display panel in ReAlM. This class also provides the handler functions for various events.

Attributes	private FiniteRelAlg< *String*, *BasisMorphs* > r1;
	private JComboBox comboBoxBasis;
	private ArrayList< *String* > basisList;
	private MatrixAlg< *String*, *BasisMorphs*, *Basis* > r2;
	private Map< *String*, *Matrix* > relationMap; // HASH MAP for MATRIX
	private Map< *String*, *Map* < *String*, *Matrix* >> basisMap;
	private Map< *String*, *MatrixAlg* > basisInstanceMap ;
	private Map< *String*, *FiniteRelAlg* > r1Map;

Constructor	public applicationInterface()
Methods	private void addComponents(Component component, int row, int column,int width, int height) private JPanel buildCenterPanel(String layoutTitle, JList parsed) private JTabbedPane buildOperationPanel(String layoutTitle) private void performStdOperation(String outpt, String in1, String in2) private void performRelOperation(String outpt, String in1, String in2) private JTabbedPane buildNewMatrixPanel(String layoutTitle) private void createInjectedMatrix(String rw, String cl, String type, String name) private void createMatrix(String rw, String cl, String type, String name) private JPanel drawMatrix(Matrix< *BasisMorphs* > mat) private void load(File fileName) private void save(File fileName) private void performDelete() public void initializeBasis(String filename) public boolean getEqualTest(String rel1, String rel2) public boolean getIncludeTest(String rel1, String rel2) public boolean getUnary(String rel1, String type) private boolean checkIdentity(Matrix< *BasisMorphs* > mat1) public void testRelationSplit(String rel, String newRel) throws NoSplitException

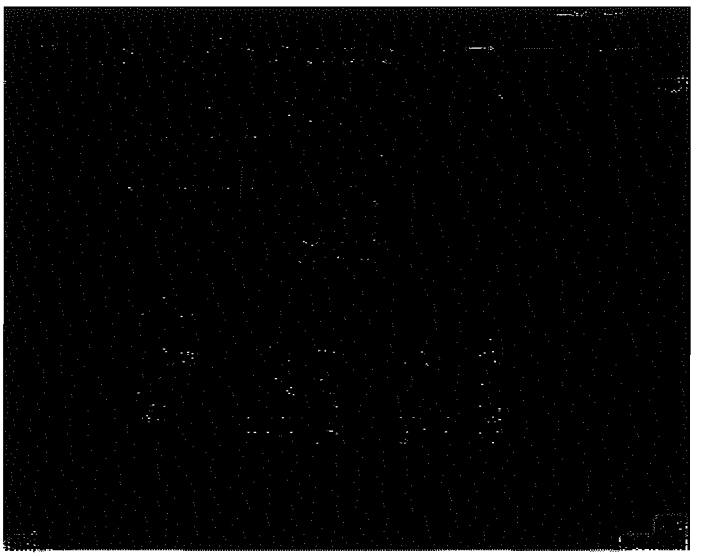
Chapter 3

ReAlM - Users Manual

3.1 System Initialization

Loading a basis

Initially, when the system starts up, ReAlM has no basis loaded. Therefore, first and only allowable action after the system starts up is to load a basis. The user is allowed to load a basis from the local file system, mapped network drive, and storage media.

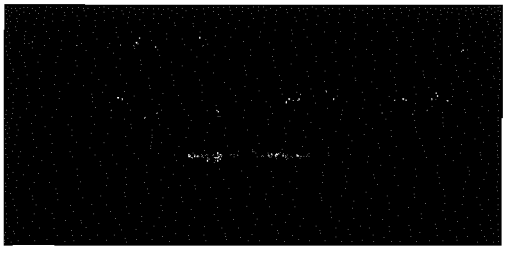


The "Load Basis" button is located at the top right corner of the screen with a image of a floppy disk. Hovering the mouse over it, a tooltip appears showing the

message "Load Basis from File System". Clicking on this button opens up a file browser window. To load the basis into the ReAlM system, the user should browse to the appropriate location and select the desired basis file which must be an XML file and then click open button on the file browser. It will initiate the basis loading process.

Switching Basis

The "Change Basis" drop-down menu is located immediate to the "Load Basis" button. It is possible to load several basis into ReAlM to work with different algebras. The drop-down menu labelled as "Change Basis", located at top-right corner of the interface is used for this purpose. If multiple basises are loaded into the system, clicking on the drop-down arrow will show the list of currently loaded basises in the system. A user can choose any of the available basis from the list. Once selected, if the selection is different from currently working basis, it will prompt the user with a pop-up dialog to confirm this action.

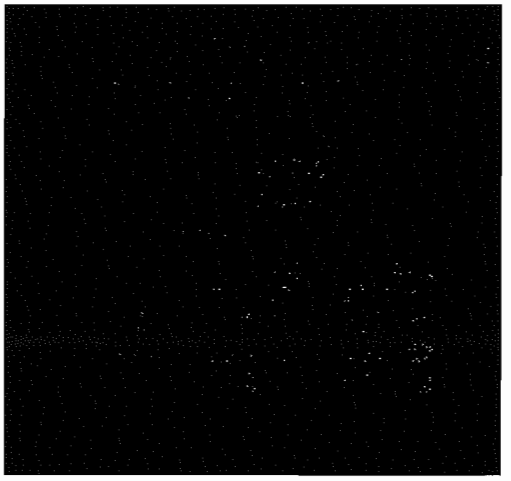


If the basis is changed without saving the current relation list, the relations will be lost. If the user selects "Yes", then ReAlM will discard the relation list and clean up the display on main window as well as basis information window which is located on the right side. The system will then update the basis information window with the information from the newly selected basis as well as the name of the basis will appear on the "Change basis" down-down selection menu. If the user chooses "No" on the

popup dialogue box, the system will keep the currently working basis without taking any further action.

Basis Information Window

Immediately bellow the Basis Load/Basis Change section, located is the "Basis Information Window". This leftmost section of the ReAlM interface shows general information about current working basis in the system. It displays the name, description, available objects, number of relations between those objects, and constants of the basis.



When the user switches to a different basis, this window is refreshed with the information from newly selected basis.

3.2 Load / Save and Delete Relations to Relation List

Relation List Window/View Relation Properties

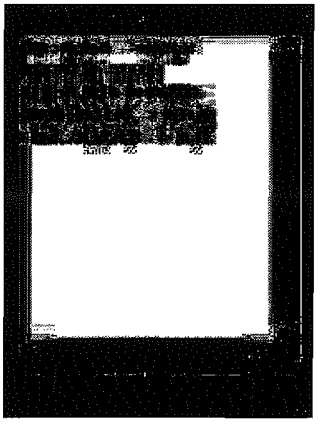
The List box located on leftmost side of the interface marked "Relation List" is used to display the list of relations currently used in the system. A relation appears in this

area in one of the two ways:

1. An action performed by the user on the system which generates a new relation.

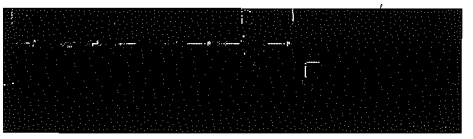
2. Loading a previously saved relation list from a file.

The display structure on the window is - Name of the relation followed by a colon then all the source objects followed by a right arrow and them all the target objects of that relation. Any given relation can be deleted from this list simply by selecting the relation and clicking the delete button bellow. Once a relation is selected from this list, the matrix representation of the relation appears on the main display area of the system.



Loading a Relations List

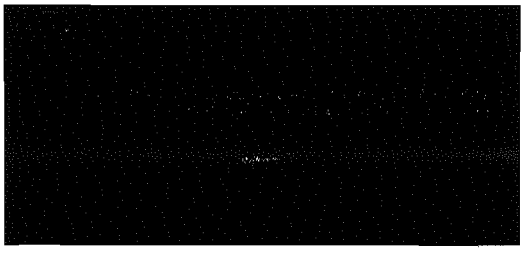
In ReAlM, it allows the user to save a working relation list in the event where a user wants to switch basis or to exit the system. This saved relation list can later be loaded into the system. To load a previously saved relation list into the system, the "Load Relations" button is used which is located immediately bellow the "Relation List" window. It is the first of the three buttons. Hovering the mouse over the button shows a tooltip message "Load a previously saved relation list".



Once clicked, it will open a file browser thus giving the user the option to select the desired file. After the relation list file is selected and opened, the system parses the file and loads all existing relations found in that file. A list of relations will be displayed in the "Relation List" window and we can examine the properties by clicking on them.

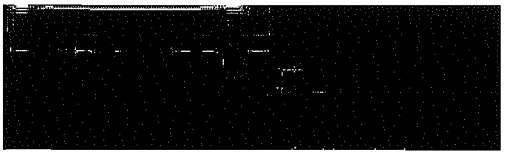
Preconditions

A file containing a relation list also has the information about the basis used to define the relations within the list. If the current basis of the system is different from that basis, the loading process will fail and the following error message will be displayed:



Saving Relation List

Bellow the "Relation List", second button, with image where an arrow is pointing to the floppy disk, is used to save a relation list to file. Mouse over action shows a tooltip message "Save current relation list to a file".

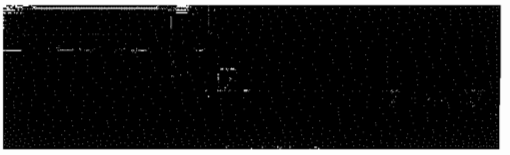


Clicking on this button will present the user a file save dialogue box where the user is required to type a name for the file and choose a location in the file system to save

the file containing relations. This file is saved with a ".rel" extension. This option prevents the user from losing all the relations in the event of a basis change or system exit.

Delete Relations

The rightmost button, with an image of a red cross bellow the "relation list" window, is the "Delete Relations" button.

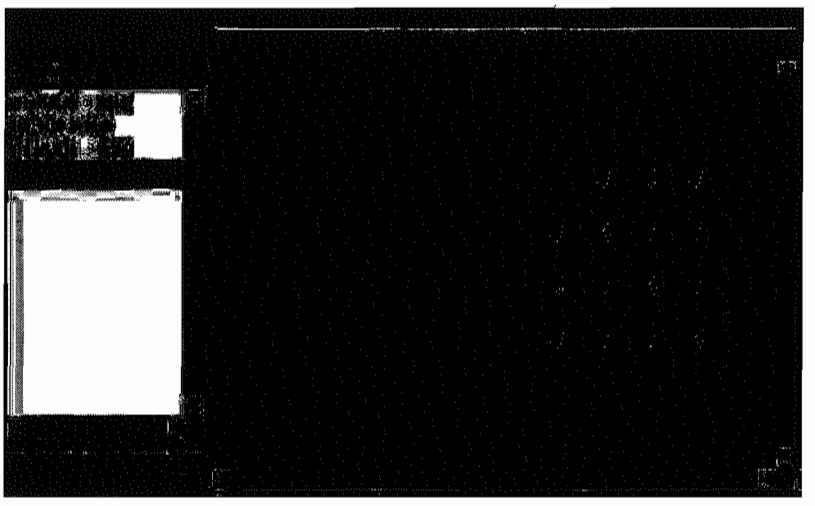


To delete a desired relation from the current relation list, the user must select the relation that needs to be deleted and then click the "Delete Relation" button. On a mouse over action on this button shows the tooltip message "Delete Selected Relation from current list".

3.3 Displaying and Modifying a Relation

Visually Display a Relation

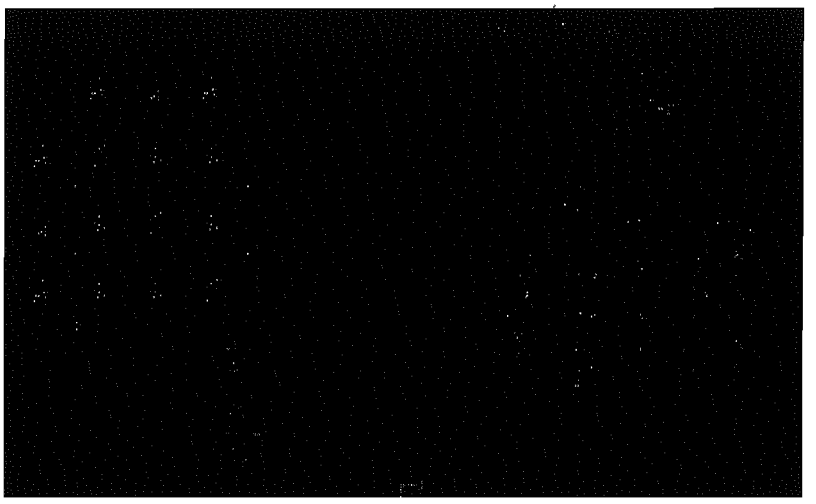
The main display area of ReAlM, the rectangular area central to the system, is initially empty. This display panel is used to present a visual representation of any relation in the system. To display a relation, the user needs to select the desired relation from relation list located on the left side of the interface. Once clicked, the relation matrix is presented on the display window.



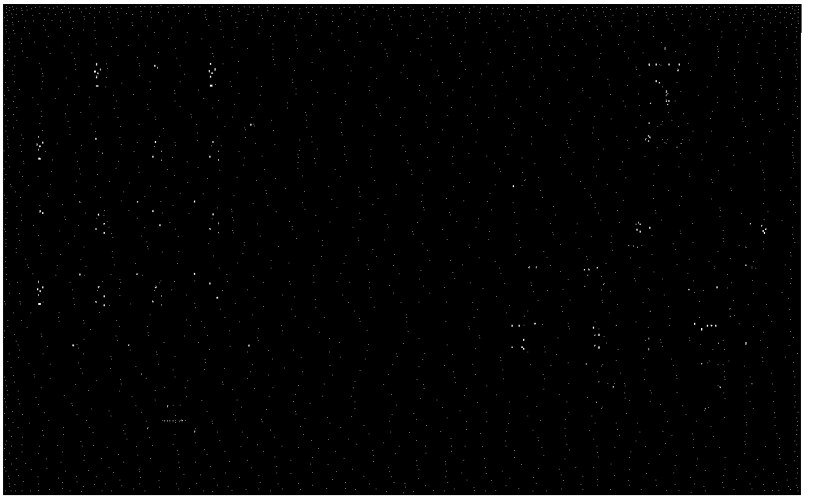
Source objects of a relation are displayed as row label of the left and target objects are displayed as column label on the top of the matrix. This main display area is scrollable vertically and horizontally in the event that when a relation requires bigger display area then the normally visible rectangular area. If a different relation is selected while a relation is displayed on this area, this center panel is re-drawn with the newly selected relation.

Editing/Modifying a Relation visually

All cells of the relation matrix, which are displayed on the center display panel, are clickable. This feature allows the user to change the properties of a relation visually and in an interactive way. Once a cell on the relation matrix is clicked, it presents the user a popup display menu close to the cell that has been clicked. This popup menu shows all the available relations between source and target objects corresponding to that cell.



As an example, from the basis description on the right, it can be seen that there are four relations with source and target A. When the last cell on the diagonal is clicked where source and target both are A, the popup menu allows the user to choose from four relations for that cell, i.e. from 0,1,2 and 3. One can choose any of the 4 relations from the menu and it will change the relation in the system.



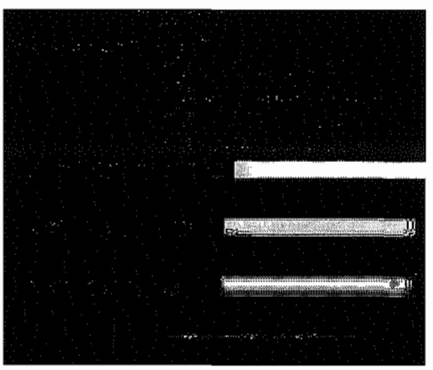
In the second example above the user wants to modify the second cell of the last row. The source of a relation in this cell has to be A, and its target has to be B. The current

basis has two relations between those objects; the relations 0 and 1. Therefore the system allows to select from exactly those two relations.

3.4 Creating Relations

Creating a Standard relation

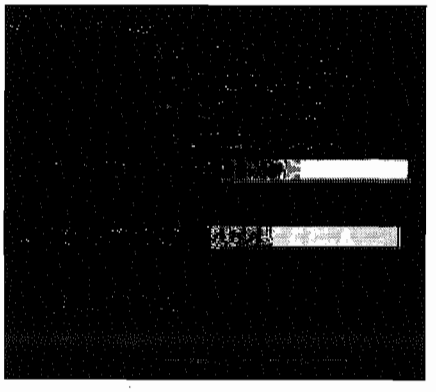
Once a basis is loaded into the system, the user has the option to create constants using the objects of that basis. Under the standard relations, we can create identity, universal, empty and diversity identity relations. Options to create a relation are located at the bottom right corner of the system's interface. To create any of the standard relations, we have to use the first tab labelled as "Standard". Bellow we briefly describe each type of action required to create relations using the panel.



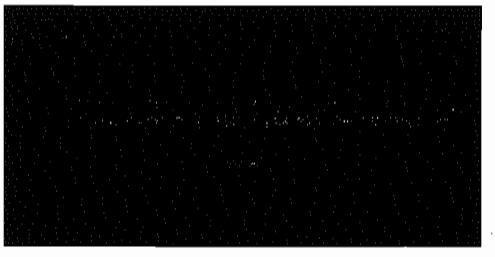
Source or Target objects must be comma separated and from the current basis. Any target name or source name that is not from current basis will produce an error message such as:

Identity Relation

To create an identity relation, the user needs to choose the radio button option labelled as "Identity Relation". Then he has to provide a name for this new relation in the text box "New Relation Name" and a list of objects from current basis in the text box named "Source Names". Relation name is case sensitive. So the user needs to type it exactly as it appears on the relation list display. Once these required parameters are provided, clicking on the "Create Relation" will create the relation and add it to the relation list of the system.



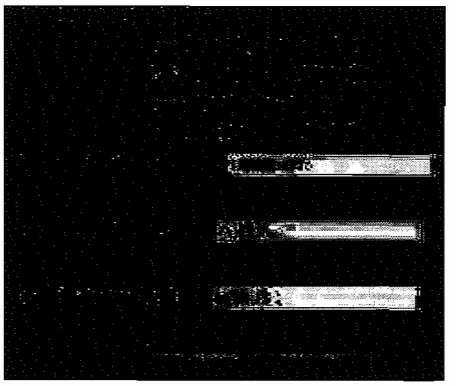
Since the identity relation has same source and target, once this option is selected - the input box for target names is disabled. All three parameters are required for this operation and missing any of these will display an error message.



Universal Relation:

Creating an universal relation is similar to creating an identity relation. However in

this case, source and target objects can be different. So the user needs to provide both source names and target names in a separate input box. Also the user needs to select appropriate radio option. Any missing argument will produce the error message previously shown.



Empty Relation

Creating an empty relation is similar to creating a universal relation. The user needs to select "Empty Relation" from the radio button options and then provide a name, source names and target names since they can be different.

Diverse Identity Relation

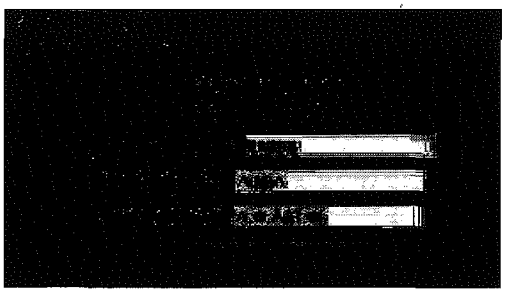
Creating a diverse identity relation is similar to creating an identity relation. For a diverse identity relation, the user only needs to provide the source names since the source and target are the same for this kind of relation.

Creating Relations related to Relational Sums

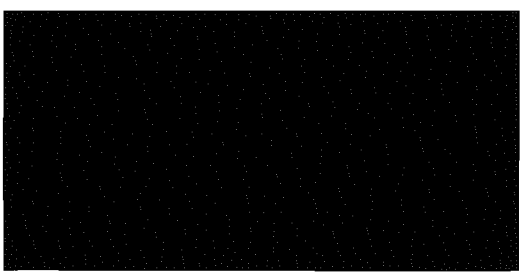
Non standard relations are created using the input form located under the tab "Rel. Sum" or relational sum.

Left Injection

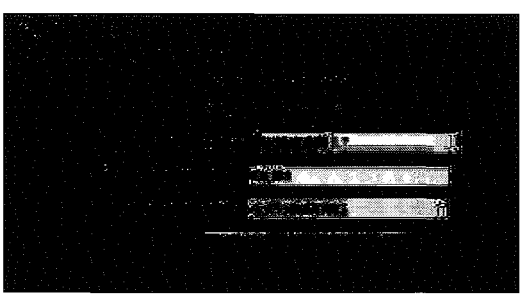
To create a left injection, the user first needs to select the radio option marked "Left Injection" and then provide the name of new relation, a left and a right component.



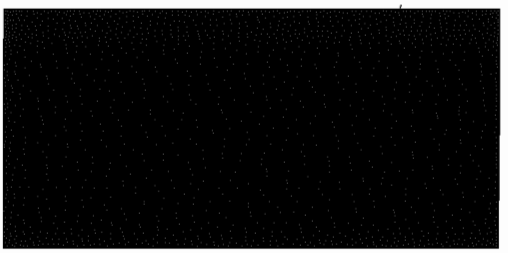
Suppose the left component is A,B,A and the right component is B,B. The source of the left injection is the left component A,B,A (for the right injection the right component) and the target is the concatenation of left and right components in that order A,B,A,B,B. The matrix is



Right Injection To create a right injection, the user first needs to select the radio option marked "Right Injection"and then provide the name of new relation, a left and a right component.



Suppose the left component is A, B, A and the right component is B, B. The source of the right injection is the left component B,B and the target is the concatenation of right and left components in that order A,B,A,B,B. The matrix is

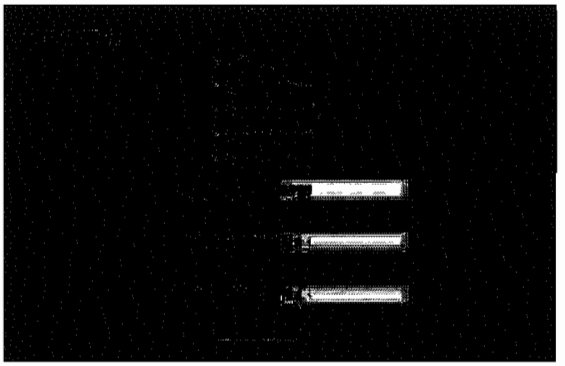


3.5 Standard Operations

Standard operations are located in the bottom right part of the application, immediately bellow the main display area, on the first tab marked "Standard Operation" provides the facility to perform standard operations on relations. Among the standard operations, ReAlM has: union, intersection, complement, composition, and conversion.

Union

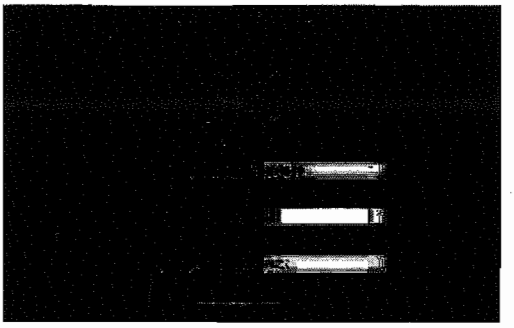
To perform an union operation, the user needs to select the radio option marked "Union" and then provide a name for the resulting relation and two relations as the first and second input of the operation. Both inputs must be names of relations that are already defined in the system. After providing all the parameters clicking on "Execute" button will perform the union operation and add the newly created relation to relation list.



For union operation, both relations must have identical source and target objects. If the source and target do not match, the operation will fail and the user is notified of this fact by a pop up message.

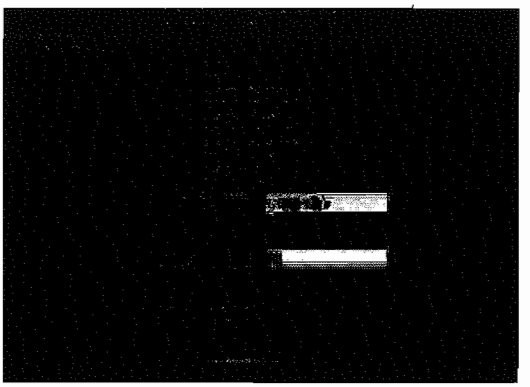
Intersection

To perform an intersection operation, the user needs to select radio option marked "Intersection". Required parameters, actions and constraints are identical to the union operation.



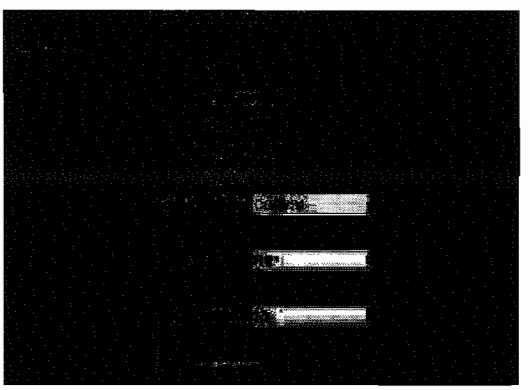
Complement

To perform a complement operation, the user needs to select radio option marked "Complement". This operation is similar to the union and the intersection except that it takes only one relation as its input.



Composition

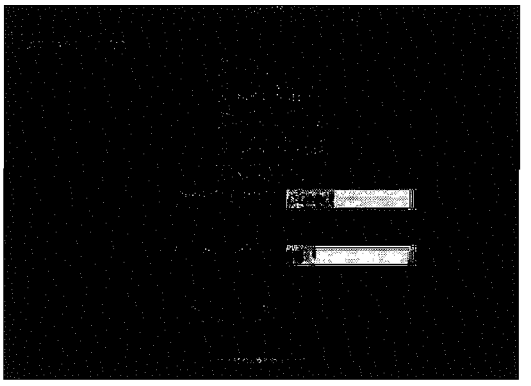
To perform a composition operation, the user needs to select radio option marked "Composition". This operation is similar to the union and intersection described above.



In case of composition, target objects of first relation must be identical to the source objects of the second relation. If this condition is not satisfied the operation will fail and a warning is shown to the user.

Conversion

To perform a conversion operation, the user needs to select radio option marked "Conversion". Everything else is identical to the complement operation.

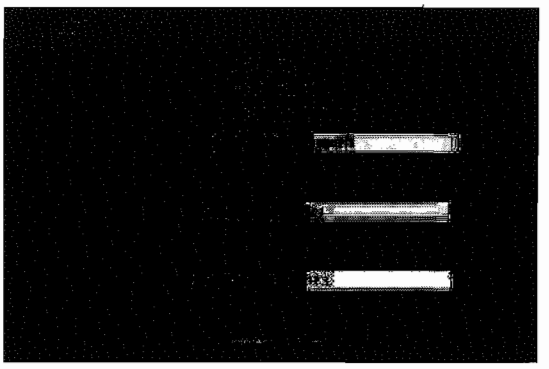


3.6 Relational Sum

Operations of relational sum are performed using the tab marked as "Rel. Sum" bellow the main display area of the relations. In this tab we can perform three different operations; sum, co-sum and disjoint union.

Sum

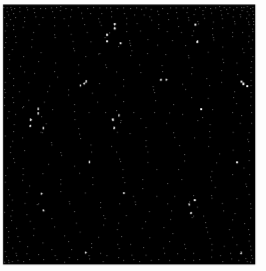
To perform a sum operation, the user needs to select the radio option marked "Sum" and then provide a name for the newly created relation as a result of the operation, left and right input which both have to be names of relation already defined in the system. After providing all the parameters, clicking on the execute button will create the new relation and add it to relation list of the current basis.



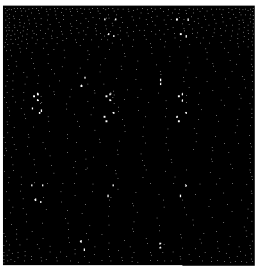
For the "Sum" operation both left input and right input relations must have identical source objects, but the targets could be different. If the sources are not same, the operation will fail and user is notified of the event.

Example

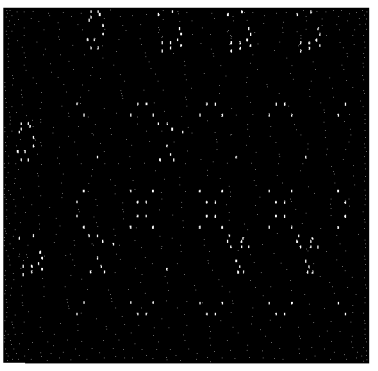
If the left input R1 is :



And the right input R3 is:

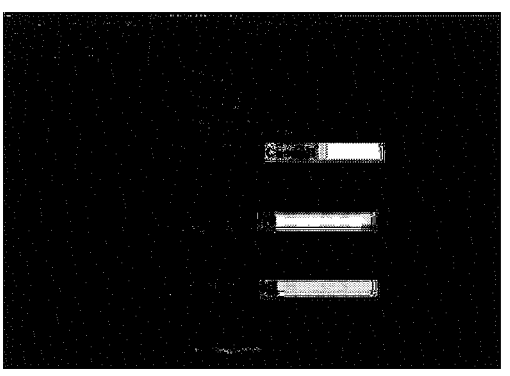


Then the result of the Sum would be :

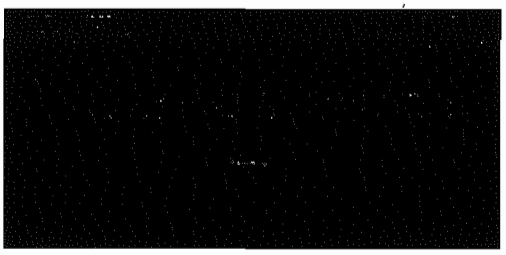


Co-Sum

To perform a co-sum operation, the user needs to select the radio option marked "Co-sum" and rest of the input requirements are identical to that in creating the sum operation described above.

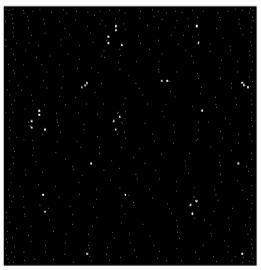


For the "Co-sum" operation both left input and right input relations must have identical target objects, but the sources could be different. If the targets are not same, the operation will fail and user is notified of the event.

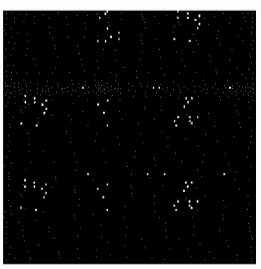


Example

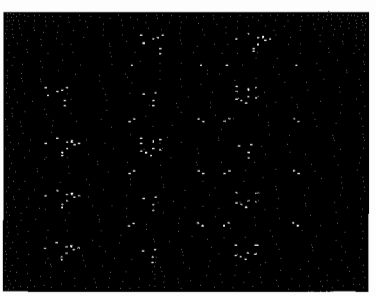
If the left input R1 is :



And the right input R4 is:



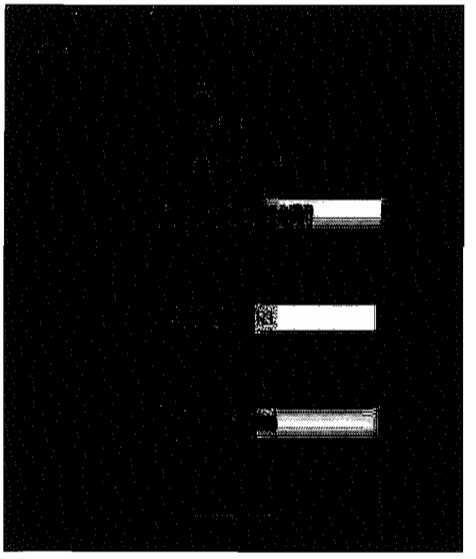
Then the result of the Co-sum would be :



Disjoint Union

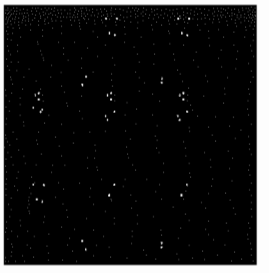
To perform a disjoint-union operation, the user needs to select the radio option marked

"Disjoint-Union". All other input requirements are similar to those in the sum operation described above.

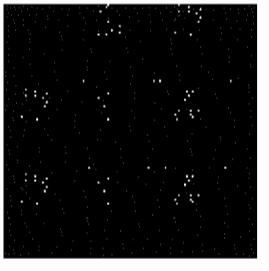


Example

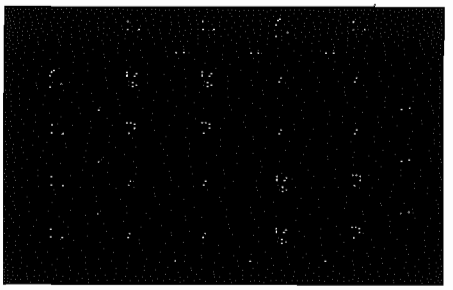
If the left input R3 is :



And the right input R4 is:



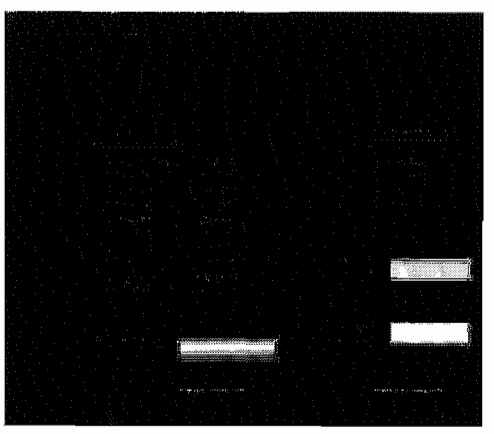
Then the result of the disjoint would be :



3.7 Test Operations

This feature of the ReAlM allows the user to perform unary and binary tests on the available relations of the system. Unary test includes : Equal bottom, equal top, univalent, total, injective, surjective, symmetric, transitive, reflexive, idempotent. Binary test includes: included and equal.

For unary test, only one relation is required. The user needs to enter the relation name in the box marked as "Unary Relation" and click "Test Unary". ReAlM will then test the specified relation and put check marks in the checkboxes for the properties that holds for this relation.

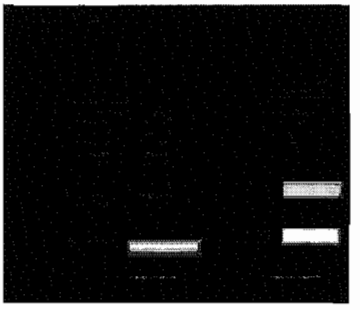


To perform a binary test, the user needs to enter two relation names, one in the "Relation 1" and the other in "Relation 2" then click "Test Binary". ReAlM will put

a check mark in "included" if the second relation is included in the first one and it will put a check mark in the "equal" if both relations are equal.

3.8 Splitting

Splitting Tab of the operations sections allows the user to compute the splitting of a relation from the relation list. To compute the splitting, the basis must contain splitting.



To perform splitting operation, the user is required to enter the name of the relation for which the splitting will be computed and the name of the target relation which will be used to store calculated result from the operation in the fields that are labeled as "Input Relation" and "Output Relation" respectively.

Chapter 4

Appendix A

4.1 Schema File

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

< xs : schema xmlns : xs = "http : //www.w3.org/2001/XMLSchema" elementFormDefault =

"qualified" >

< xs : element name = "FiniteRelAlg" >

< xs : complexType >

< xs : sequence >

< xs : element ref = "comment" / >

< xs : element ref = "objects" / >

< xs : element ref = "relations" maxOccurs = "unbounded" / >

< xs : element ref = "identity" maxOccurs = "unbounded" / >

< xs : element ref = "bottom" maxOccurs = "unbounded" / >

< xs : element ref = "top" maxOccurs = "unbounded" / >

< xs : element ref = "union" maxOccurs = "unbounded" / >

< xs : element ref = "intersection" maxOccurs = "unbounded" / >

< xs : element ref = "composition" maxOccurs = "unbounded" / >

<xs:element ref="transposition" maxOccurs="unbounded"/>

<xs:element ref="complement" maxOccurs="unbounded"/>

</xs:sequence>

<xs:attribute name="name" use="required"/>

</xs:complexType>

</xs:element

<xs:element name="comment" type="xs:string"/>

<xs:element name="objects"/>

<xs:element name="relations">

<xs:complexType>

<xs:attribute name="number" use="required" type="xs:positiveInteger"/>

<xs:attribute name="source" use="required" type="xs:NCName"/>

<xs:attribute name="target" use="required" type="xs:NCName"/>

</xs:complexType>

</xs:element

<xs:element name="identity">

<xs:complexType>

<xs:attribute name="object" use="required" type="xs:NCName"/>

<xs:attribute name="relation" use="required" type="xs:nonNegativeInteger"/>

</xs:complexType>

</xs:element

<xs:element name="bottom">

<xs:complexType>

$< xs : attribute\ name = "relation"\ use = "required"\ type = "xs : nonNegativeInteger" / >$

$< xs : attribute\ name = "source"\ use = "required"\ type = "xs : NCName" / >$

$< xs : attribute\ name = "target"\ use = "required"\ type = "xs : NCName" / >$

$< /xs : complexType >$

$< /xs : element$

$< xs : element\ name = "top" >$

$< xs : complexType >$

$< xs : attribute\ name = "relation"\ use = "required"\ type = "xs : nonNegativeInteger" / >$

$< xs : attribute\ name = "source"\ use = "required"\ type = "xs : NCName" / >$

$< xs : attribute\ name = "target"\ use = "required"\ type = "xs : NCName" / >$

$< /xs : complexType >$

$< /xs : element$

$< xs : element\ name = "union" >$

$< xs : complexType\ mixed = "true" >$

$< xs : attribute\ name = "source"\ use = "required"\ type = "xs : NCName" / >$

$< xs : attribute\ name = "target"\ use = "required"\ type = "xs : NCName" / >$

$< /xs : complexType >$

$< /xs : element$

$< xs : element\ name = "intersection" >$

$< xs : complexType\ mixed = "true" >$

$< xs : attribute\ name = "source"\ use = "required"\ type = "xs : NCName" / >$

$< xs : attribute\ name = "target"\ use = "required"\ type = "xs : NCName" / >$

</xs:complexType>

</xs:element

<xs:element name="composition">

<xs:complexType mixed="true">

<xs:attribute name="intermediate" use="required" type="xs:NCName"/>

<xs:attribute name="source" use="required" type="xs:NCName"/>

<xs:attribute name="target" use="required" type="xs:NCName"/>

</xs:complexType>

</xs:element

<xs:element name="transposition">

<xs:complexType mixed="true">

<xs:attribute name="source" use="required" type="xs:NCName"/>

<xs:attribute name="target" use="required" type="xs:NCName"/>

</xs:complexType>

</xs:element

<xs:element name="complement">

<xs:complexType mixed="true">

<xs:attribute name="source" use="required" type="xs:NCName"/>

<xs:attribute name="target" use="required" type="xs:NCName"/>

</xs:complexType>

</xs:element

</xs:schema>

4.2 Sample Basis File

<?xml version = "1.0" encoding = "utf − 8"? >

< FiniteRelAlg name = "twoObjectBasis" >

< comment > Firstexample. < /comment >

< objects > A, B < /objects >

< relations source = "A" target = "A" number = "4" / >

< relations source = "A" target = "B" number = "2" / >

< relations source = "B" target = "B" number = "4" / >

< relations source = "B" target = "A" number = "2" / >

< identity object = "A" relation = "1" / >

< identity object = "B" relation = "1" / >

< bottom source = "A" target = "A" relation = "0" / >

< bottom source = "A" target = "B" relation = "0" / >

< bottom source = "B" target = "A" relation = "0" / >

< bottom source = "B" target = "B" relation = "0" / >

< top source = "A" target = "A" relation = "3" / >

< top source = "A" target = "B" relation = "1" / >

< top source = "B" target = "A" relation = "1" / >

< top source = "B" target = "B" relation = "3" / >

< union source = "A" target = "B" >

0, 1, 1;

1, 0, 1;

0, 0, 0;

1, 1, 1

$< /union >$

$< union\ source = "B"\ target = "A" >$

$0, 1, 1;$

$1, 0, 1;$

$0, 0, 0;$

$1, 1, 1$

$< /union >$

$< union\ source = "A"\ target = "A" >$

$0, 0, 0;$

$0, 1, 1;$

$0, 2, 2;$

$0, 3, 3;$

$1, 0, 1;$

$1, 1, 1;$

$1, 2, 3;$

$1, 3, 3;$

$2, 0, 0;$

$2, 1, 3;$

$2, 2, 2;$

$2, 3, 3;$

$3, 0, 3;$

$3, 1, 3;$

$3, 2, 3;$

$3, 3, 3$

</ union >

< union source = "B" target = "B" >

0, 0, 0;

0, 1, 1;

0, 2, 2;

0, 3, 3;

1, 0, 1;

1, 1, 1;

1, 2, 3;

1, 3, 3;

2, 0, 0;

2, 1, 3;

2, 2, 2;

2, 3, 3;

3, 0, 3;

3, 1, 3;

3, 2, 3;

3, 3, 3

</ union >

< intersection source = "A" target = "B" >

0, 1, 0;

1, 0, 0;

0, 0, 0;

1, 1, 1

$< /intersection >$

$< intersection\ source = "B"\ target = "A" >$

$0, 1, 0;$

$1, 0, 0;$

$0, 0, 0;$

$1, 1, 1$

$< /intersection >$

$< intersection\ source = "A"\ target = "A" >$

$0, 0, 0;$

$0, 1, 0;$

$0, 2, 0;$

$0, 3, 0;$

$1, 0, 0;$

$1, 1, 1;$

$1, 2, 0;$

$1, 3, 1;$

$2, 0, 0;$

$2, 1, 0;$

$2, 2, 2;$

$2, 3, 2;$

$3, 0, 0;$

$3, 1, 1;$

$3, 2, 2;$

$3, 3, 3$

$</intersection>$

$<intersection\ source="B"\ target="B">$

$0,0,0;$

$0,1,0;$

$0,2,0;$

$0,3,0;$

$1,0,0;$

$1,1,1;$

$1,2,0;$

$1,3,1;$

$2,0,0;$

$2,1,0;$

$2,2,2;$

$2,3,2;$

$3,0,0;$

$3,1,1;$

$3,2,2;$

$3,3,3$

$</intersection>$

$<composition\ source="A"\ intermediate="B"\ target="A">$

$0,0,0;$

$0,1,0;$

$1,0,0;$

$1,1,1$

$< /composition >$

$< composition\ source = "B"\ intermediate = "A"\ target = "B" >$

$0, 0, 0;$

$0, 1, 0;$

$1, 0, 0;$

$1, 1, 1$

$< /composition >$

$< composition\ source = "A"\ intermediate = "A"\ target = "A" >$

$0, 0, 0;$

$0, 1, 0;$

$0, 2, 0;$

$0, 3, 0;$

$1, 0, 0;$

$1, 1, 1;$

$1, 2, 2;$

$1, 3, 3;$

$2, 0, 0;$

$2, 1, 2;$

$2, 2, 1;$

$2, 3, 3;$

$3, 0, 0;$

$3, 1, 3;$

$3, 2, 3;$

$3, 3, 3$

$</composition>$

$<composition\ source="A"\ intermediate="A"\ target="B">$

$0,0,0;$

$0,1,0;$

$1,0,0;$

$1,1,1$

$</composition>$

$<composition\ source="A"\ intermediate="B"\ target="B">$

$0,0,0;$

$0,1,0;$

$1,0,0;$

$1,1,1$

$</composition>$

$<composition\ source="B"\ intermediate="A"\ target="A">$

$0,0,0;$

$0,1,0;$

$1,0,0;$

$1,1,1$

$</composition>$

$<composition\ source="B"\ intermediate="B"\ target="A">$

$0,0,0;$

$0,1,0;$

$1,0,0;$

$1,1,1$

</composition>

<composition source="B" intermediate="B" target="B">

0,0,0;

0,1,0;

0,2,0;

0,3,0;

1,0,0;

1,1,1;

1,2,2;

1,3,3;

2,0,0;

2,1,2;

2,2,1;

2,3,3;

3,0,0;

3,1,3;

3,2,3;

3,3,3

</composition>

<transposition source="A" target="B">

0,0;

1,1

</transposition>

<transposition source="B" target="A">

0, 0;

1, 1

< /*transposition* >

< *transposition source* = "*A*" *target* = "*A*" >

0, 0;

1, 1;

2, 2;

3, 3

< /*transposition* >

< *transposition source* = "*B*" *target* = "*B*" >

0, 0;

1, 1;

2, 2;

3, 3

< /*transposition* >

< *complement source* = "*A*" *target* = "*B*" >

0, 1;

1, 0

< /*complement* >

< *complement source* = "*B*" *target* = "*A*" >

0, 1;

1, 0

< /*complement* >

< *complement source* = "*A*" *target* = "*A*" >

0, 1;

1, 2;

2, 1;

3, 0

< /*complement* >

< *complement source* = "*B*" *target* = "*B*" >

0, 1;

1, 2;

2, 1;

3, 0

< /*complement* >

< /*FiniteRelAlg* >

Bibliography

[1] Asperti A., Longo G.: Categories, Types and Structures. The MIT Press, Cambridge, Massachusetts, London, England (1991).

[2] Berghammer R., Leoniuk B., Milanese U.: Implementation of relational algebra using binary decision diagrams. Proc. 6th International Conference RelMiCS 2001 and 1st Workshop of COST Action 274 TARSKI, LNCS 2561, 241-257 (2002)

[3] Berghammer R.: Applying relation algebra and RelView to solve problems on orders and lattices. Acta Informatica 45 (3), 211-236 (2008)

[4] Birkhoff G.: Lattice Theory. American Mathematical Society Colloquium Publications Vol. XXV, 3rd edition (1968).

[5] Freyd P., Scedrov A.: Categories, Allegories. North-Holland (1990).

[6] Gallian J.A.: Contemporary Abstract Algebra, 7th Edition. Brooks/Cole Cengage Learning (2009)

[7] Grätzer, G.: General Lattice Theory. Birkhäuser, 2nd edition (1998).

[8] Maddux R.D.: Some sufficient conditions for the representability of relation algebras. Algebra Universalis 8, 162-172 (1978)

[9] McKenzie R,: Representations of integral relation algebras. Michigan Mathematical Journal 17 (1970)

[10] Mac Lane S.: Categories for the Working Mathematician, 2nd edition. Springer (1998)

[11] Schmidt G., Ströhlein T.: Relationen und Graphen. Springer (1989); English version: Relations and Graphs. Discrete Mathematics for Computer Scientists, EATCS Monographs on Theoret. Comput. Sci., Springer (1993).

[12] Schmidt G., Hattensperger C., Winter M.: Heterogeneous Relation Algebras. *In:* Brink C., Kahl W., Schmidt G. (eds.), Relational Methods in Computer Science, Advances in Computing Science, Springer Vienna (1997).

[13] Tarski, A.: On the calculus of relations, J. Symbolic Logic 6, 73-89 (1941).

[14] Tarski, A. and Givant, S.: A Formalization of Set Theory without Variables. Amer. Math. Soc. Colloq. Publ. 41 (1987).

[15] Winter, M.: Pseudo Representation Theorem for various Categories of Relations. TAC Theory and Applications of Categories 7(2), 23-37 (2000)

[16] Winter, M.: Relation Algebras are Matrix Algebras over a suitable Basis. University of the Federal Armed Forces Munich, Report Nr. 1998-05 (1998)