RelMDD-A Library for Manipulating Relations Based on MDDs

Atampore, Francis Kwesi

Department of Computer Science

Submitted in partial fulfillment
of the requirements for the degree of

Master of Science

Faculty of Mathematics and Science, Brock University
St. Catharines, Ontario

Abstract

Relation algebras is one of the state-of-the-art means used by mathematicians and computer scientists for solving very complex problems. As a result, a computer algebra system for relation algebras called RelView has been developed at Kiel University. RelView works within the standard model of relation algebras. On the other hand, relation algebras do have other models which may have different properties.

For example, in the standard model we always have $\mathbb{T};\mathbb{T}=\mathbb{T}$ (the composition of two (heterogeneous) universal relations yields a universal relation). This is not true in some non-standard models. Therefore, any example in RelView will always satisfy this property even though it is not true in general. On the other hand, it has been shown that every relation algebra with relational sums and subobjects can be seen as matrix algebra similar to the correspondence of binary relations between sets and Boolean matrices [49, 50].

The aim of my research is to develop a new system that works with both standard and non-standard models for arbitrary relations using multiple-valued decision diagrams (MDDs). This system will implement relations as matrix algebras. The proposed structure is a library written in C which can be imported by other languages such as Java or Haskell.

Acknowledgments

I would like to record my gratitude and acknowledgment to Dr. Michael Winter for his supervision, advice, and guidance from the very early stage of this research as well as giving me extraordinary experiences and directions through out the work. Above all and the most needed, he provided me unflinching encouragement and support in various ways as well as the many hours of relentless question and answer periods. His truly scientist intuition has made him as a constant oasis of ideas and passions in science, which exceptionally inspire and enrich my growth as a student, a researcher and a scientist want to be. I am indebted to him more than he knows.

I also thank the members of my graduate supervisory committee members - Dr. Sheridan Houghten and Dr. Ke Qiu for their guidance and support.

I will also like to thank Vanessa Boateng who took her time to read through my thesis.

F.A.K

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Relation algebras is one of the state-of-the-art means used by mathematicians and computer scientists for solving very complex problems. In fact, since the mid-1970's, relational methods have been the basis for many conceptual and methodological tools, and a very convenient means of dealing with fundamental concepts such as graphs, orders, lattices, games, databases, Petri nets, data types, and semantics in computer science and mathematics [5, 7, 44, 45].

Relation algebra has been used for analyzing and modeling various computer science problems such as program specification, heuristic approaches for program derivation and verification, automatic prover design, database and software decomposition, program fault tolerance, testing, data abstraction and information coding, and spatial reasoning.

Relation algebras can be represented by concrete relations between sets on the assumption that relational products exist or the point axiom is given. In this case, we can visualize relation algebras using a Boolean matrix such that the rows and columns of the matrix correspond to the elements of the source, and the elements of the target of the relation respectively. Since the entries of the matrix are Boolean values, 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.

Relation algebra is increasingly being used to solve problems because it has very small set of operations such as union, intersection, converse, complement and composition, which can easily and efficiently be implemented on finite carrier sets using data structures such as Boolean arrays, linked lists or decision diagrams [8].

As a result of this, several computer systems have been developed to facilitate the use of relations. RelView is one of such systems which visualizes relations as Boolean matrices using reduced ordered binary decision diagrams (ROBDDs). The use of ROBDD provides very efficient implementations of the operations on relations. This system is written in C programming language. RelView is an interactive and specific purpose computer algebra system with a graphical user interface for the manipulation and visualization of relation algebra and relational programming. The first versions were developed at the University of the German Forces Munich from 1988 to 1992. It was further rebuilt and extended by Kiel University since 1993. 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 [8].

However, RelView works within the standard model of relation algebra, i.e., binary relations between sets. On the other hand, relation algebras do have other models which may have different properties. That is, not all relation algebras can be represented as the algebra of Boolean matrices. For example, in the standard model the property $\mathbb{T};\mathbb{T}=\mathbb{T}$ (the composition of two (heterogeneous) universal relations equal a universal relation) is always true. But this is not true in some non-standard models. In [2] we list further examples such as:

> "Another example is given by the relationship between the power set of a disjoint union of two sets A and B and the product of the power set of A and the power set of B. In the standard model both constructions lead to isomorphic objects, while this might not be the case in certain non-standard models" [2], page 1.

In [49, 50], it was proved that every relation algebra $\mathcal{R}$ with relational sums and subobjects can be represented by matrices. In particular, 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 $\mathcal{R}$.

As a result of this, we can visualize an arbitrary relation algebras using matrices whose coefficients are not limited to Boolean values. Hence, we can associate all standard operations on relations to matrix operations. In order to implement relations using the matrix algebra approach and adopting a data structure similar to RelView, we have to use a more general version of BBD's called multiple valued decision diagrams (MDDs).

A multiple-valued decision diagram (MDD) is a natural extension of the reduced ordered binary decision diagrams (ROBDDs) to the multiple-valued case. It was introduced by Bryant in 1986 [17, 32]. MDDs are considered to be more efficient and require smaller memory size than the ROBDDs.

In my thesis, I want to develop a new system called RelMDD that works with both standard and non-standard models for any arbitrary relation using multiple-valued decision diagrams (MDDs). This system will implement relations as matrix algebras. The proposed structure is a library written in C which can be imported by other languages such as Java or Haskell. We will refer to [2, 3, 5, 8, 13, 17, 32, 39, 49] for definitions, terminology, notions, theorems as well as other theoretical aspects used in this thesis.

1.1 Outline

The outline of the subsequent chapters are as follows:

Chapter 2: In this chapter, we present the necessary background required for understanding this thesis, including algebraic preliminaries, concrete relation algebras and heterogeneous relation algebras. We shall introduce matrix algebra and state a theorem that relates heterogeneous relation algebras to matrix algebras. We will also recall various properties of relations including integral objects, subobjects and splittings.

Chapter 3: In this chapter, we will look at work done in this area. We shall review various systems developed for relations algebras and their implementations.

Chapter 4: This chapter describes the proposed system including its implementations, technical details, scope and how it operates.

Chapter 5: We will then conclude by summarizing what we have said in the preceding chapters and give suggestions for future work in this area.

Chapter 2

Relation-Algebraic Preliminaries

The Development of Relation Algebra

The basic theory and the calculus of binary relations was founded by Augustus De Morgan in 1860. However, it was further developed by C.S. Peirce around 1870. This theory was extended by Ernst Schröder in a very definitive, thorough and systematic approach around 1895. The interest in relation algebra was further awaken by Tarski in one of his article in 1941; noting that, "the calculus of relation deserves much more attention than it receives", having "the intrinsic charm and beauty which makes it a source of intellectual delight to all who become acquainted with it" [23, 48]. He then came up with certain axioms as the definition for a relation algebra. For more information on the history of relation algebras, see [23, 28].

Formerly, relation algebra was presented in its classical form in which relations were perceived to be quadratic or homogeneous, i.e., concrete relation over a given set. That is, relations were restricted to only one set on a fixed universe U. In recent years, a variant of the theory of relation has evolved to include relations between two or more different sets. This approach is called heterogeneous or rectangular relation algebras. Therefore, a homogeneous relation algebras can be seen as a heterogeneous relation algebras with a single universe [22, 23, 30, 39, 40]. Heterogeneous relation algebra uses category theory in order to distinguish the source and the target of a relation.

In this chapter, we will examine various properties and definitions of relation algebras. We will then introduce matrix algebra and a theorem that establishes 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 coefficients from $\mathcal{B}$ is equivalent to $\mathcal{R}$.

2.1 Concrete Relation Algebras

If we want to describe a relationship between elements of two sets A and B, we can use ordered pairs such that, the first element is taken from A and the second element is taken from B. Since this is a relation between two sets, it is called a binary relation (concrete relation). We often make statements in our everyday lives which illustrate the use of relations. Examples include “Lisza is the husband of Alan”, “Australia has a smaller population than China”, etc

Definition 1. *A concrete relation R between two sets A and B is a subset of the Cartesian product $A \times B$, where*

$$A \times B = \{(a, b) : a \in A, b \in B\}.$$

Example 1. *The relation “is a sister of”, is defined on a set of persons.*
The relation “is succeeded by”, defined on natural numbers can be given by
T:= {(0,1),(1,2),(2,3),(3,4),(4,5),...} $\subseteq \mathbb{N} \times \mathbb{N}$.

Example 2. *Let P be a set of people, C be a set of cars, and R be the relation describing which person drives which car(s).*
P = {John, Sarah, Peter, Mark}
C = {Mercedes, BMW, tricycle}
R = {(John, Mercedes), (Sarah, Mercedes),(Sarah, BMW), (Peter, tricycle)}

This means that John drives a Mercedes, Sarah drives a Mercedes and a BMW, Peter drives a tricycle, and Mark does not drive any of these vehicles. R can be visualized in the following diagram.

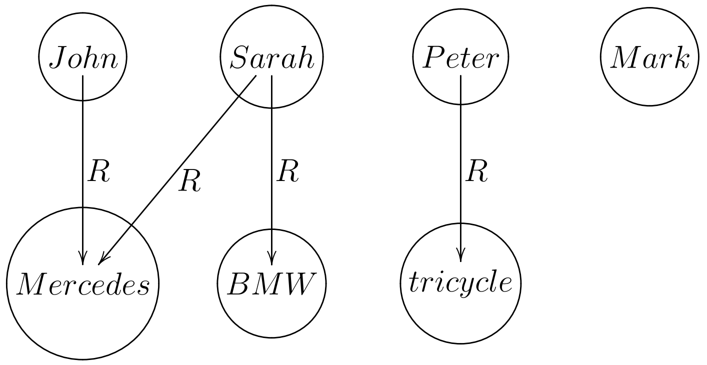
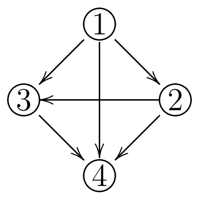


Figure 2.1: Relations between persons and cars: R = drives

Relations can be represented by tables, matrices or graphs when the set under consideration is finite.

Example 3. *Let $A = \{1,2,3,4\}$ and let R be a relation defined on A such that $R = \{(a,b) \mid a < b\}$, then the relation can be visualized as set given by:*
$R = \{(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)\}$.

R can also be visualized as a graph, a table or a boolean matrix as shown in the figures below:



R	1	2	3	4
1		x	x	x
2			x	x
3				x
4				

$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 1 & 1 \\ 2 & 0 & 0 & 1 & 1 \\ 3 & 0 & 0 & 0 & 1 \\ 4 & 0 & 0 & 0 & 0 \end{array}$$

Figure 2.2: Graph, Table and Boolean Matrix Representation of R

2.1.1 Basic Operations of Relation Algebras

The following defines the basic operations of relation algebras: Let A, B, C be sets, $R, T \subseteq A \times B$, and $S \subseteq B \times C$. Then we define:

1. Transpose or Converse: The converse of R is obtained by "turning around" all pairs of R, that is,
$$R^{\smile} = \{(x,y) \mid (y,x) \in R\} \subseteq B \times A$$

2. Complement: The complement of R consist of all ordered pairs of element of the universe that do not belong to R. That is,
$$\overline{R} = \{(x,y) \mid (x,y) \notin R\} \subseteq A \times B$$

3. Union:
$$R \sqcup T = \{(x,y) \mid (x,y) \in R \vee (x,y) \in T\} \subseteq A \times B$$

4. Intersection:
$$R \sqcap T = \{(x,y) \mid (x,y) \in R \wedge (x,y) \in T\} \subseteq A \times B$$

5. Composition:

$$R;S = \{(x,z) \mid \exists y \in B : (x,y) \in R \wedge (y,z) \in S\} \subseteq A \times C$$

6. Inclusion:

$$R \subseteq T \Leftrightarrow R \sqcap T = R, \text{ i.e., } R \subseteq T \Leftrightarrow \forall x,y : [(x,y) \in R \rightarrow (x,y) \in T]$$

7. Special relations include:

   - Empty or zero relation: $\perp\!\!\!\perp = \emptyset \subseteq A \times B$
   - Universal relation: $\top\!\!\!\top = A \times B$
   - Identity relation: $\mathbb{I} = \{(x,x) \mid x \in A\} \subseteq A \times A$

   For full description of the above operations see [5, 39, 29].

2.1.2 Product and the Schröder equivalences

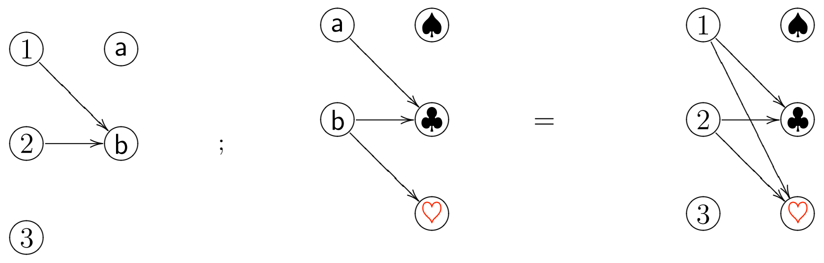
In this section, we want to illustrate the product of two relations and the interplay between the various operations defined in the previous sections:

Example 4. *Product or Composition of relation: In this example we want to illustrate the composition of two relations using all three representations, i.e., the set representation, the graph representation, and the Boolean matrix representations (example taken from [62]).*

Let $S = \{1, 2, 3\}$, $T = \{\mathsf{a}, \mathsf{b}\}$ and $U = \{\spadesuit, \clubsuit, \heartsuit\}$.

$$Q \quad ; \quad R \quad = \quad Q;R$$

$$\{(1,\mathsf{b}),(2,\mathsf{b})\} \quad ; \quad \{(\mathsf{a},\clubsuit),(\mathsf{b},\clubsuit),(\mathsf{b},\heartsuit)\} \quad = \quad \{(1,\clubsuit),(1,\heartsuit),(2,\clubsuit),(2,\heartsuit)\}$$



$$\begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \quad ; \quad \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \quad = \quad \begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

Composition of matrices is possible if sizes match.

Figure 2.3: Composition of two relations [62]

Proposition 2.1.1. *Schröder equivalences*
Given the relations Q, R, S, then, we have

$$Q; R \subseteq S \leftrightarrow Q^{\smile}; \bar{S} \subseteq \bar{R} \leftrightarrow \bar{S}; R^{\smile} \subseteq \bar{Q}$$

.

Example 5. *Schröder equivalences*

The interplay between composition, transposition, and complement, with respect to containment can be shown by the following example using the Schröder equivalence. The Schröder equivalence can be read as: converse the first (or second), then complement and permute the other two [23, 39, 48]. We shall illustrate how the Schröder equivalences can be applied using a kinship relation. This example is taken from [39]. The statement $Q^{\smile}; \overline{S} \subseteq_{BC} \overline{R}$ can be explicitly written as follows,
$\forall x, y : [\exists z : (x, z) \in Q^{\smile} \wedge (z, y) \in \overline{S} \subseteq_{BC} \rightarrow (x, y) \in \overline{R}]$
Suppose that B, F, M and G are relations defined as follows:

1. B = "is a Brother of ",

2. F = "is a Father of ",

3. M = “is a Mother of ”,

4. G = “is the Godfather of ”

Then, $P \coloneqq F \sqcup M$ means “is a Parent of ” and $B; P$ means “is an Uncle of ”. Now, let’s assume that uncles in our family tradition evolves into godfathers. So we have $B; P \subseteq G$. Now, by applying the Schröder’s equivalence to the family relation above, we have $B^{\smile}; \bar{G} \subseteq \bar{P}$ and $\bar{G}; P^{\smile} \subseteq \bar{B}$.
$B^{\smile}; \bar{G} \subseteq \bar{P}$ can easily 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, otherwise, contrary to family tradition, z would be an uncle of y without being y’s godfather. We can visualize this in the diagram below.

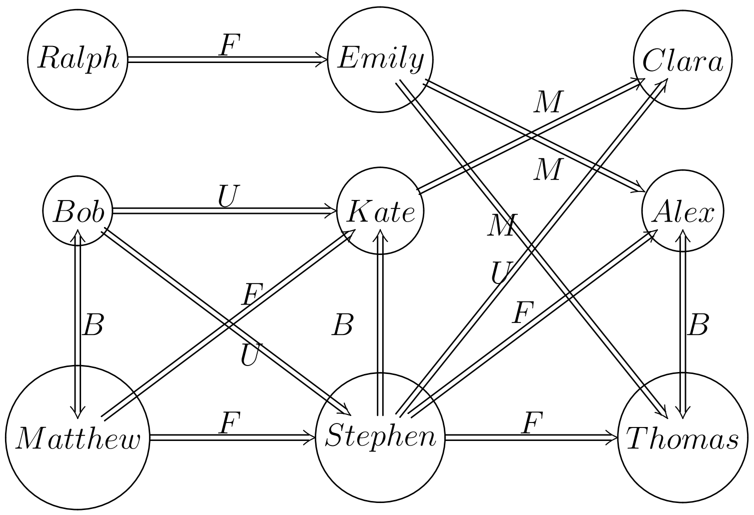


Figure 2.4: An instance of Schröder equivalence
F= Father, M = Mother, U = Uncle, B = Brother

The kinship relation above can be represented as a Boolean matrix as follows:

$$F = \begin{array}{c|ccccccccc} & R & B & M & E & K & S & C & A & T \\ \hline R & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ B & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ M & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ E & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ K & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ S & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ C & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ T & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

$$M = \begin{array}{c|ccccccccc} & R & B & M & E & K & S & C & A & T \\ \hline R & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ B & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ M & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ E & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ K & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ S & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ C & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ T & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

$$B;P = \begin{array}{c|ccccccccc} & R & B & M & E & K & S & C & A & T \\ \hline R & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ B & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ M & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ E & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ K & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ S & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ C & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ T & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Figure 2.5: Boolean Matrices for the kinship relation

$$B = \begin{array}{c} \\ R \\ B \\ M \\ E \\ K \\ S \\ C \\ A \\ T \end{array} \begin{array}{c} \begin{array}{ccccccccc} R & B & M & E & K & S & C & A & T \end{array} \\ \begin{pmatrix} 0&0&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&1&0 \end{pmatrix} \end{array} \qquad G = \begin{array}{c} \\ R \\ B \\ M \\ E \\ K \\ S \\ C \\ A \\ T \end{array} \begin{array}{c} \begin{array}{ccccccccc} R & B & M & E & K & S & C & A & T \end{array} \\ \begin{pmatrix} 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&1&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \end{pmatrix} \end{array}$$

$$B^{\smile};\bar{G} = \begin{array}{c} \\ R \\ B \\ M \\ E \\ K \\ S \\ C \\ A \\ T \end{array} \begin{array}{c} \begin{array}{ccccccccc} R & B & M & E & K & S & C & A & T \end{array} \\ \begin{pmatrix} 0&0&0&0&0&0&0&0&0 \\ 1&1&1&1&1&1&1&1&1 \\ 1&1&1&1&0&0&1&1&1 \\ 0&0&0&0&0&0&0&0&0 \\ 1&1&1&1&1&1&0&1&1 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 1&1&1&1&1&1&1&1&1 \\ 1&1&1&1&1&1&1&1&1 \end{pmatrix} \end{array}$$

Figure 2.6: Boolean Matrices for the kinship relation

Now, from the above, we can deduce that Bob is a godfather of Stephen. This is because Bob is a brother of Matthew and Matthew is the father of Stephen. In other words, Bob is an uncle of Stephen, and since an uncle becomes a godfather according to tradition we have $B;P \subseteq G$. Another scenario is the case where Bob is not a brother of Stephen, then Bob is not a godfather of Thomas who is a son of Stephen. Now, based on the formula $\bar{G};P^{\smile} \subseteq \bar{B}$, which is equivalent to $B;P \subseteq G$ using the Schröder equivalence, we obtain that Bob is not a brother of Stephen.

2.2 Category Theory

A variant of the theory of relation is called heterogeneous relation algebras. In this approach, relations have different source and target. This type of algebra is based on category theory, hence it suffices to review the basic concepts of category theory [1, 19, 55] before we proceed to heterogeneous relation algebra.

Definition 2. *A category* $\mathtt{C}$ *is a quintuple* $\mathtt{C} = (Obj_c, Mor_c, \mathtt{dom}, \mathtt{cod}, \text{“;”})$ *which consists of the following data:*

1. *A class of objects* Obj_c*, denoted by* $a, b...$ *or* $A, B...$
2. *A class of morphisms (arrows)* Mor_c*, denoted by* $f, g...$
3. *Each morphism* f *has a source object* (dom) A *called the domain of* f *and a target object* (cod) B *called the the codomain of* f*. In order to indicate that* f *has source* A *and target* B *we use the notion* $f : A \to B$.
4. *An operation “;” called composition assigning to each pair* f, g *of arrows with* $cod(f) = dom(g)$ *an arrow* $f;g$ *such that* $dom(f;g) = dom(f)$, $cod(f;g) = cod(g)$.
5. *An operation* $\mathbb{I}$ *assigning to each object* A *a morphism* $\mathbb{I}_A$ *(the identity of* A*) such that* $dom(\mathbb{I}_A) = cod(\mathbb{I}_A) = A$.

In addition, identity and composition must satisfy the following laws:

`Identity Law:` *For each morphism we have* $f : A \to B$, $f;\mathbb{I}_B = f$ *and* $\mathbb{I}_A;f = f$.

`Associativity Law:` *For morphisms* $f : A \to B$, $g : B \to C$, $h : C \to D$*, we have* $f;(g;h) = (f;g);h$

Given two objects A and B, the collection of all morphisms f such that $f : A \to B$ is denoted by $C[A, B]$. Also, composition of morphisms has to be read from left to right, i.e., $f;g$ means f, then g.

Example 6. *Examples of concrete categories*

1. A classic example is `Rel:` sets as objects, relations as morphisms.
2. `Vec:` vector spaces as objects, linear maps as morphisms.
3. `Group:` groups as objects, homomorphisms as morphisms.
4. `Top:` topological spaces as objects, continuous functions as morphisms.
5. `Diff:` smooth manifolds as objects, smooth maps as morphisms.
6. `Ring:` rings as objects, ring homomorphisms as morphisms.

2.2.1 Functor

A suitable function or transformation between two categories is called functor. This transformation maps objects and morphisms from one category to the objects and morphisms of another category.

Definition 3. *A functor $F : C \to D$ between categories C and D is a mapping of objects to objects $F_{obj} : Obj_C \to Obj_D$ and arrows to arrows $F_{Mor} : Mor_C \to Mor_D$, such that for each $f : A \to B$, $g : B \to C$ in C,*

1. $F_{Mor}(f) : F_{obj}(A) \to F_{obj}(B)$
2. $F_{Mor}(f; g) = F_{Mor}(f); F_{Mor}(g)$
3. $F_{Mor}(\mathbb{I}_A) = \mathbb{I}_{F_{obj}(A)}$

That is, F preserves domains and codomains, identity arrows, and composition. If for all $A, B \in Obj_C$ and every $i \in D[F(A), (B)]$, if there is $f \in C[A, B]$ such that $i = F(f)$, then we call F `full`. *Similarly, if for all $A, B \in Obj_C$ and for all $f, h \in D[A, B]$, $F(f) = F(h)$ implies $f = h$, then we call F* `faithful`.

A category R is said to be locally small if for all objects A and B in this category, $Mor_R[A, B]$ is a set (and not a class).

2.2.2 Dedekind Category

In this section, we want to consider a categorical structure on relation called Dedekind category. Under certain conditions Dedekind category may be seen as $\mathcal{L}$-fuzzy relations [20, 36, 37, 53]. $\mathcal{L}$-fuzzy relations are relations taking values from an arbitrary complete Brouwerian lattice [20, 36, 37, 53] instead of the unit interval [0,1] of real numbers. It is useful to consider it here since most of the examples in this thesis are based on this structure.

Definition 4. *A Dedekind category $\mathcal{R}_D$ is a locally small category satisfying the following:*

1. *For all objects A and B, the set $\mathcal{R}_D[A,B]$ is a complete distributive lattice. Meet, join, the induced ordering, the least and the greatest element are denoted by $\sqcup_{AB}$, $\sqcap_{AB}$, $\sqsubseteq_{AB}$, $\perp\!\!\!\perp_{AB}$, $\top\!\!\!\top_{AB}$ respectively.*

2. *There is a monotonic operation $\breve{}$ (called conversion) such that for all relations $Q\colon A \to B$ and $R : B \to C$ the following holds:*

$$(Q : R)\breve{} = R\breve{};Q\breve{}, \quad (Q\breve{})\breve{} = Q.$$

3. *For all relations $Q : A \to B$, $R : B \to C$ and $S : A \to C$ the modular law*

$$Q;R \sqcap S \sqsubseteq Q;(R \sqcap Q\breve{};S)$$

   holds.

4. *For all relations $R : B \to C$ and $S : A \to C$, there is a relation $S/R : A \to B$ (called the left residual of S and R) such that for all $Q : A \to B$ the following holds*

$$Q;R \sqsubseteq S \Leftrightarrow Q \sqsubseteq S/R$$

[53, 20, 36, 37]

From the definition above, it is important to note that the class of complete distributive lattices and the class of Heyting algebras are equivalent. If every hom-set (the collection of all morphisms) of $\mathcal{R}_D$ is an atomic lattice then, $\mathcal{R}_D$ is called atomic. Now, if every hom-set $\mathcal{R}_D[A,B]$ is a Boolean algebra then $\mathcal{R}_D$ is called a Schröder category.

2.3 Heterogeneous Relation Algebras

With the notion of objects and morphisms from the above, we shall now give a formal definition of heterogeneous relation algebras. Notice, that it can be shown that the notion of a heterogeneous relation algebra defined below is equivalent to an atomic Schröder category defined in the previous section [19, 22, 39, 40, 50, 54, 55].

Definition 5. *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 ";", identities are denoted by $\mathbb{I} \in \mathcal{R}[A,B]$, conversion $\smile_{AB}$: $\mathcal{R}[A,B] \to \mathcal{R}[B,A]$. The operations satisfy the following rules:*

1. *Every set $\mathcal{R}[A,B]$ carries the structure of a complete atomic boolean algebra with operations, $\sqcup_{AB}$, $\sqcap_{AB}$, $\overline{}_{AB}$, zero element $⫫_{AB}$, universal element $\top\!\!\!\top_{AB}$, and inclusion ordering $\subseteq_{AB}$.*

2. *The Schröder equivalences*

$$Q;R \subseteq_{AC} S \Longleftrightarrow Q^{\smile};\overline{S} \subseteq_{BC} \overline{R} \Longleftrightarrow \overline{S};R^{\smile} \subseteq_{AB} \overline{Q}$$

   holds for all relations $Q : A \to B$, $R : B \to C$ and $S : A \to C$.

In heterogeneous relation algebras, $\sqcup$, $\sqcap$ ";" are considered as partial operations because they cannot be applied to relations with different source or target. As we mentioned earlier on in this chapter, the definition of heterogeneous relation algebras is based on category theory. All the indices of elements and operations are usually omitted for brevity and can easily be reinvented. For each morphism $\mathcal{R}[A,A]$ where the source object is the same as the target object there is an identity, a zero and a universal element. However, there is only a zero and a universal relation per morphism set $\mathcal{R}[A,B]$ if $A \neq B$. A standard example is the full relation algebra `Rel`, where sets are objects and relations are morphisms.

In the following example, we want to illustrate the above definition. Figure 2.7 shows two Boolean algebras $\mathcal{L}_2$ and $\mathcal{L}_4$ with 2 and 4 elements respectively. Now, based on these lattices we can define a heterogeneous relation algebra $\mathcal{R}$ with two objects A and B. The relations from A to A and B to B are given by the lattice $\mathcal{L}_4$. Between A and B we use the lattice $\mathcal{L}_2$. The various relations and operations tables are shown in Table 2.1, Table 2.2, Table 2.3, Table 2.4 and Table 2.5. For simplicity, we will assume that the tables for composition for relations with source and target both equal to A or both equal to B are the same as join, i.e., $\sqcap$ =;, and that the of converse of an element x is defined as $x^{\smile} = x$. The other composition tables are given either explicitly in Table 2.5 or can be computed using the converse operation and Table 2.5 and/or by exchanging A and B. Clearly, there are two objects under consideration, i.e., A and B. Now, from Table 2.1 each object has an identity relation, for instance, the identity element of $A = 1$ and that of $B = 1$. Table 2.2 also defines the zero and universal elements between A and B. For example, the zero element and universal element with source as A and target as B is 0 and 1 respectively. Table 2.3 and Table 2.4 shows the meet and join operations respectively.

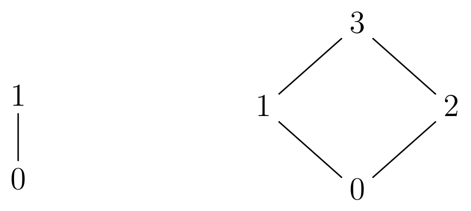


Figure 2.7: Boolean algebras $\mathcal{L}_2$ and $\mathcal{L}_4$

Identity	
A	1
B	1

Table 2.1: Identity relation

Bottom	
A, A	0
B, B	0
B, A	0
A, B	0

Top	
A, A	3
B, B	3
B, A	1
A, B	1

Table 2.2: Bottom and Top

A, B	0	1
0	0	1
1	1	1

B, A	0	1
0	0	1
1	1	1

A, A	0	1	2	3
0	0	1	2	3
1	1	1	3	3
2	2	3	2	3
3	3	3	3	3

B, B	0	1	2	3
0	0	1	2	3
1	1	1	3	3
2	2	3	2	3
3	3	3	3	3

Table 2.3: Join operation in the lattice algebra $\mathcal{R}$

A, B	0	1
0	0	0
1	0	1

B, A	0	1
0	0	0
1	0	1

B, B	0	1	2	3
0	0	0	0	0
1	0	1	0	1
2	0	0	2	2
3	0	1	2	3

A, A	0	1	2	3
0	0	0	0	0
1	0	1	0	1
2	0	0	2	2
3	0	1	2	3

Table 2.4: Meet operation in the Lattice algebra $\mathcal{R}$

$AB\backslash BB$	0	1	2	3
0	0	0	0	0
1	0	1	1	1

$AB\backslash BA$	0	1
0	0	0
0	0	1

Table 2.5: Composition operation in $\mathcal{R}$

We can visualize the relation described above in the following diagram.

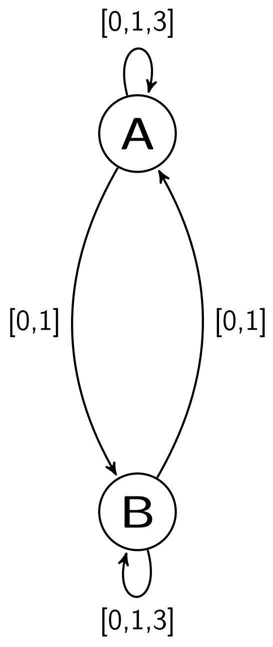


Figure 2.8: A two object relation
$[\perp\!\!\!\perp, \mathbb{I}, \top\!\!\!\top]$= [zero element, identity, universal relation], and $[\perp\!\!\!\perp\ \top\!\!\!\top]$= [zero element, universal relation]

Notice that we have $\top\!\!\!\top_{AB};\top\!\!\!\top_{BA}= 1;1 \neq 3 =\top\!\!\!\top_{AA}$ in $\mathcal{R}$.

2.3.1 Properties of relation algebras

In this section we will examine various properties of heterogeneous relation algebras.

Definition 6. *Let $T \in \mathcal{R}[A,B]$ be a relation, then we can define the following special relations as:*

1. *T is called univalent iff $T^{\smile};T \subseteq \mathbb{I}_B$*
2. *T is called total iff $\mathbb{I}_A \subseteq T;T^{\smile}$ or equivalent $T;\top\!\!\!\top_{BA}=\top\!\!\!\top_{AA}$*
3. *T is called map iff Q is univalent and total.*

We can visualize the above definition in the following diagram.

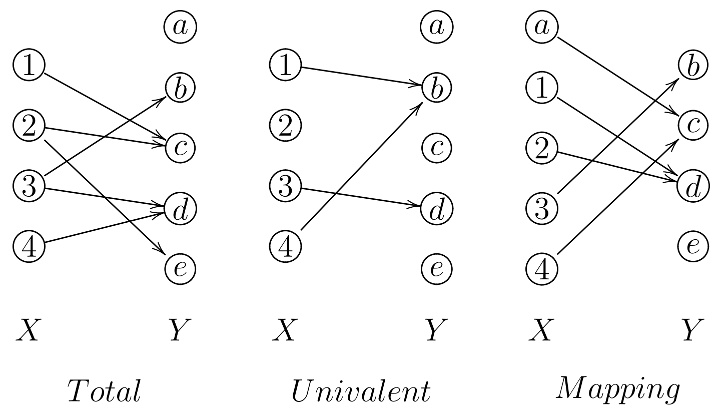


Figure 2.9: Total relation, Univalent relation and Mapping

Definition 7. *Homomorphisms*
Let $\mathcal{R}$ and $\mathcal{S}$ be relation algebras and $F : \mathcal{R} \to \mathcal{S}$ a functor, then F is called a homomorphism between relation algebra iff

1. $F(\sqcap S_i) = \underset{i \in I}{\sqcap} F(S_i)$,
2. $F(\overline{R}) = \overline{F(R)}$,
3. $F(R^{\smile}) = F(R)^{\smile}$

holds for all relations R, S, with $i \in I$.

2.3.2 Relational Sums

In `Rel`, the relational sum is given by the disjoint union of sets and the corresponding injection functions [6, 19, 40, 54].

Definition 8. *Let $\{A_i | i \in I\}$ be a set of objects indexed by a set I. An object $\sum_{i \in I} A_i$. together with relations $\iota_j \in \mathcal{R}[A_j, \sum_{i \in I} A_i]$ for all $j \in I$ is called a relational sum of $\{A_i | i \in I\}$ iff for all $i, j \in I$ with $i \neq j$ the following holds:*

1. $\iota_i ; \iota_i^{\smile} = \mathbb{I}_{A_i}$
2. $\iota_i ; \iota_j^{\smile} = \perp\!\!\!\perp_{A_i A_j}$
3. $\underset{i \in I}{\sqcup} \iota_i ; \iota_i^{\smile} = \mathbb{I}_{\sum_{i \in I} A_i}$

A relation algebra has relational sums if and only if for every set of objects a relational sum exists. Relational sums are unique (up to isomorphism) because they are simultaneously categorical product and co-product [50].

Suppose A and B are objects of a relation algebra, then the above definition corresponds to the usual definition of relational sums as follows. In the binary case we also write $\iota : A \to A + B$ and $\kappa : B \to A + B$ for the two injections. For two relations $R : A \to C$ and $S : B \to C$ we define the sum $R + S : A + B \to C$ by $R + S = \iota^{\smile}; R; \iota \sqcup \kappa^{\smile}; S; \kappa$. We now want to illustrate these constructions using Boolean matrices.

Example 7.

Suppose that R and S are represented by the following Boolean matrices.

$$R = \begin{array}{c|ccccc} & z_1 & z_2 & z_3 & z_4 & z_5 \\ \hline x_1 & 1 & 0 & 0 & 0 & 0 \\ x_2 & 0 & 0 & 1 & 0 & 0 \\ x_3 & 0 & 1 & 0 & 1 & 0 \end{array} \qquad S = \begin{array}{c|ccccc} & z_1 & z_2 & z_3 & z_4 & z_5 \\ \hline y_1 & 1 & 1 & 0 & 0 & 0 \\ y_2 & 0 & 1 & 1 & 0 & 0 \\ y_3 & 0 & 0 & 1 & 1 & 0 \\ y_4 & 0 & 0 & 0 & 1 & 1 \end{array}$$

Then the relational sum R+S is given by the following Boolean matrix

$$R + S = \begin{array}{c|ccccc} & z_1 & z_2 & z_3 & z_4 & z_5 \\ \hline x_1 & 1 & 0 & 0 & 0 & 0 \\ x_2 & 0 & 0 & 1 & 0 & 0 \\ x_3 & 0 & 1 & 0 & 1 & 0 \\ y_1 & 1 & 1 & 0 & 0 & 0 \\ y_2 & 0 & 1 & 1 & 0 & 0 \\ y_3 & 0 & 0 & 1 & 1 & 0 \\ y_4 & 0 & 0 & 0 & 1 & 1 \end{array}$$

Figure 2.10: Boolean matrix for direct product

2.4 Matrix Algebra

In the preceding sections we have used matrices to visualize relations in various forms. We will now give a formal definition of matrix algebras and how it can formally be used to represent relations. Given a heterogeneous relation algebra $\mathcal{R}$, an algebra $\mathcal{R}^+$ of matrices with coefficients from $\mathcal{R}$ is defined by:

Definition 9. *Let $\mathcal{R}$ be a relation algebra. The algebra $\mathcal{R}^+$ of the matrices with coefficients from $\mathcal{R}$ is defined by:*

- *The class of objects of $\mathcal{R}^+$ is the collection of all functions from an arbitrary set I to $Obj_{\mathcal{R}}$.*
- *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.*
- *For $R \in \mathcal{R}^+[f,g]$ and $S \in \mathcal{R}^+[g,h]$ composition is defined by $(R;S)(i,k) := \bigsqcup R(i,j); S(j,k)$.*
- *For $R \in \mathcal{R}^+[f,g]$ conversion and negation is defined by $R^{\smile}(j,i) := (R(i,j))^{\smile}$, $\overline{R}(i,j) = \overline{R(i,j)}$*
- *For $R, S \in \mathcal{R}^+[f,g]$ union and intersection is defined by $(R \sqcup S)(i,j) = R(i,j) \sqcup S(i,j)$, $(R \sqcap S)(i,j) = R(i,j) \sqcap S(i,j)$*
- *The identity, zero and universal elements are defined by*

$$\mathbb{I}_{f(i_1,i_2)} := \begin{cases} \perp\!\!\!\perp_{f(i_1)f(i_2)} & : i_1 \neq i_2 \\ \mathbb{I}_{f(i_1)} & : i_1 = i_2 \end{cases}$$

$$\perp\!\!\!\perp_{fg(i,j)} := \perp\!\!\!\perp_{f(i)g(j)}, \qquad \top\!\!\!\top_{fg(i,j)} := \top\!\!\!\top_{f(i)g(j)}$$

[50, 54].

The morphisms in matrix algebra are matrices indexed by objects from $\mathcal{R}$. Unlike the Boolean matrix where elements of the matrix were restricted to only Boolean values (0 or 1), here the elements could take values including real numbers. In order to understand the meaning of the definition above we will illustrate with several examples.

Given a finite list of objects [A, B] with a set of binary relations (morphisms) defined between each source and target, i.e., for each [A, A], [A ,B], [B, B], [B, A], we define a binary relation for how each element of a source object is mapped to the target object. We will use the lattice structure introduced in Figure 2.7 and its basis tables described in Table 2.1, Table 2.2, Table 2.3 and Table 2.4 of Section 2.3.

Example 8. *Identity, Zero and the Universal relations*

We can compute the zero and universal relations in $\mathcal{R}^+$ with source [A, B, A, B] and target [A, A, B, B] taking the coefficients from $\mathcal{R}$ and using Table 2.2 as our basis table by:

$$\begin{array}{c} \text{Zero} \\ A \\ A \\ B \\ B \end{array} \begin{array}{c} \begin{array}{cccc} A & B & A & B \end{array} \\ \begin{pmatrix} \perp\!\!\!\perp_{AA} & \perp\!\!\!\perp_{AB} & \perp\!\!\!\perp_{AA} & \perp\!\!\!\perp_{AB} \\ \perp\!\!\!\perp_{AA} & \perp\!\!\!\perp_{AB} & \perp\!\!\!\perp_{AA} & \perp\!\!\!\perp_{AB} \\ \perp\!\!\!\perp_{BA} & \perp\!\!\!\perp_{BB} & \perp\!\!\!\perp_{BA} & \perp\!\!\!\perp_{BB} \\ \perp\!\!\!\perp_{BA} & \perp\!\!\!\perp_{BB} & \perp\!\!\!\perp_{BA} & \perp\!\!\!\perp_{BB} \end{pmatrix} \end{array} = \begin{array}{c|cccc} & A & B & A & B \\ A & 0 & 0 & 0 & 0 \\ A & 0 & 0 & 0 & 0 \\ B & 0 & 0 & 0 & 0 \\ B & 0 & 0 & 0 & 0 \end{array}$$

$$\begin{array}{c} \text{Universal} \\ A \\ A \\ B \\ B \end{array} \begin{array}{c} \begin{array}{cccc} A & B & A & B \end{array} \\ \begin{pmatrix} \top\!\!\!\top_{AA} & \top\!\!\!\top_{AB} & \top\!\!\!\top_{AA} & \top\!\!\!\top_{AB} \\ \top\!\!\!\top_{AA} & \top\!\!\!\top_{AB} & \top\!\!\!\top_{AA} & \top\!\!\!\top_{AB} \\ \top\!\!\!\top_{BA} & \top\!\!\!\top_{BB} & \top\!\!\!\top_{BA} & \top\!\!\!\top_{BB} \\ \top\!\!\!\top_{BA} & \top\!\!\!\top_{BB} & \top\!\!\!\top_{BA} & \top\!\!\!\top_{BB} \end{pmatrix} \end{array} = \begin{array}{c|cccc} & A & B & A & B \\ A & 3 & 1 & 3 & 1 \\ A & 3 & 1 & 3 & 1 \\ B & 1 & 3 & 1 & 3 \\ B & 1 & 3 & 1 & 3 \end{array}$$

Figure 2.11: Zero and universal relation

Since in the case of identity relation the source objects must be equal to the target objects, we will consider the list [A, B, A, B] and using Table 2.2 as our basis table. We have:

$$\begin{array}{c} \text{Identity} \\ A \\ B \\ A \\ B \end{array} \begin{array}{c} \begin{array}{cccc} A & B & A & B \end{array} \\ \begin{pmatrix} \mathbb{I}_{AA} & \perp\!\!\!\perp_{AB} & \perp\!\!\!\perp_{AA} & \perp\!\!\!\perp_{AB} \\ \perp\!\!\!\perp_{BA} & \mathbb{I}_{BB} & \perp\!\!\!\perp_{BA} & \perp\!\!\!\perp_{BB} \\ \perp\!\!\!\perp_{AA} & \perp\!\!\!\perp_{AB} & \mathbb{I}_{AA} & \perp\!\!\!\perp_{AB} \\ \perp\!\!\!\perp_{BA} & \perp\!\!\!\perp_{BB} & \perp\!\!\!\perp_{BA} & \mathbb{I}_{BB} \end{pmatrix} \end{array} = \begin{array}{c|cccc} & A & B & A & B \\ A & 1 & 0 & 0 & 0 \\ B & 0 & 1 & 0 & 0 \\ A & 0 & 0 & 1 & 0 \\ B & 0 & 0 & 0 & 1 \end{array}$$

Example 9. *Intersection and Union*

In the case of union and intersection of two relations in $\mathcal{R}^+$ the list of source of objects (row of the matrix) of the two relations must be the same. Also, the list of target objects (columns of the matrix) of the first relation must be equal to that of the second relation. Now suppose we have two relations with [A, B, A, B] and [A, B, A, B] as the source and target objects respectively, then, we can compute the intersection in the algebra with coefficients from $\mathcal{R}$ using Table 2.4 as our basis relation. We have

$$\begin{array}{c|cccc} & A & B & A & B \\ \hline A & 1 & 0 & 3 & 0 \\ B & 1 & 3 & 0 & 1 \\ A & 3 & 1 & 0 & 0 \\ B & 0 & 1 & 0 & 3 \end{array} \sqcap \begin{array}{c|cccc} & A & B & A & B \\ \hline A & 2 & 1 & 1 & 1 \\ B & 0 & 0 & 0 & 2 \\ A & 3 & 1 & 3 & 0 \\ B & 1 & 2 & 1 & 1 \end{array} = \begin{array}{c|cccc} & A & B & A & B \\ \hline A & 0 & 0 & 1 & 0 \\ B & 0 & 0 & 0 & 0 \\ A & 3 & 1 & 0 & 0 \\ B & 0 & 0 & 0 & 1 \end{array}$$

Similarly, given that the list of source objects are [A, A, A, B] and the list of target objects are [B, A, A, B] and using Table 2.3 we have:

$$\begin{array}{c|cccc} & A & A & A & B \\ \hline B & 0 & 1 & 1 & 3 \\ A & 2 & 3 & 2 & 1 \\ A & 0 & 0 & 1 & 0 \\ B & 0 & 1 & 1 & 3 \end{array} \sqcup \begin{array}{c|cccc} & A & A & A & B \\ \hline B & 1 & 0 & 1 & 3 \\ A & 0 & 0 & 1 & 0 \\ A & 3 & 3 & 3 & 0 \\ B & 0 & 1 & 0 & 0 \end{array} = \begin{array}{c|cccc} & A & A & A & B \\ \hline B & 1 & 1 & 1 & 3 \\ A & 2 & 3 & 3 & 1 \\ A & 3 & 3 & 3 & 0 \\ B & 0 & 1 & 1 & 3 \end{array}$$

Example 10. *Multiplication or Composition*

We need to ensure that the target objects of the first relation is the same as the source of objects of the second relation. In Section 2.3 we assumed that the table for composition operation is the same as the table for intersection, hence we will use Table 2.3 for composition operation. Now, suppose we have two relations M and N such that M has [A, A, B, B] and [A, B, B, A] as source and target objects respectively, and N has [A, B, B, A] and [B, B, B, A] as source and target objects respectively, then, the composition is given by:

$$\begin{array}{c|cccc} & A & A & B & B \\ \hline A & 1 & 1 & 0 & 0 \\ B & 0 & 0 & 1 & 1 \\ B & 0 & 0 & 1 & 1 \\ A & 0 & 0 & 1 & 1 \end{array} \; ; \; \begin{array}{c|cccc} & B & B & B & A \\ \hline A & 0 & 0 & 1 & 1 \\ A & 0 & 0 & 1 & 1 \\ B & 1 & 0 & 1 & 1 \\ B & 0 & 0 & 1 & 1 \end{array} = \begin{array}{c|cccc} & B & B & B & A \\ \hline A & 0 & 0 & 0 & 0 \\ B & 0 & 0 & 0 & 0 \\ B & 0 & 0 & 0 & 0 \\ A & 0 & 0 & 0 & 0 \end{array}$$

Example 11. *Conversion*

Similarly, we will illustrate conversion by assuming that the converse of an element x is given by $x^{\smile} = x$. So suppose that T is a matrix with source [A, B, A, B] and target [A, A, B, B] given by:

$$T = \begin{array}{c|cccc} & A & B & A & B \\ \hline A & 1 & 1 & 2 & 0 \\ B & 0 & 3 & 1 & 2 \end{array}$$

Then Transpose of T is:

$$\begin{array}{c|cc} & A & B \\ \hline A & 1 & 0 \\ A & 1 & 3 \\ B & 2 & 1 \\ B & 0 & 2 \end{array}$$

We can summarize the basic properties of $\mathcal{R}^+$ in the following lemma. This lemma indicates that matrix algebras are relation algebras with relational sums.

Lemma 1. $\mathcal{R}^+$ *is a relation algebra that has relational sums [50].*

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

Every relation algebra $\mathcal{R}$ with relational sums and subobjects is equivalent to an algebra of matrices over a suitable basis. This basis $\mathcal{B}$ or coefficients are given by what we call integral objects of $\mathcal{R}$. Integral objects are characterized by the fact that their identity morphisms are atoms. It is clear that $\mathcal{B}$ cannot be isomorphic to $\mathcal{R}$ since $\mathcal{B}$ is normally smaller than $\mathcal{R}$. That is, $\mathcal{B}$ is a proper subalgebra of $\mathcal{R}$. In this section, we shall look at the definition and properties of integral objects which make them very important to consider [50].

Definition 10. *An object A of a relation algebra is called integral iff $\perp\!\!\!\perp_{AA} \neq \top\!\!\!\top_{AA}$ and for all $Q, R \in \mathcal{R}[A, A]$, the equation $Q; R = \perp\!\!\!\perp_{AA}$ implies either $Q = \perp\!\!\!\perp_{AA}$ or $R = \perp\!\!\!\perp_{AA}$.*

Example 12. *Suppose that the integral objects in **Rel** is given by the singleton sets and the basis of **Rel** is also given by all singleton sets with exactly two relations, $\emptyset$ and $\{(a, b) \mid a \in \{a\}, b \in \{b\}\}$. Then all objects in this basis are isomorphic, and the matrix algebra over this basis is the category of Boolean matrices.*

Example 13. *The following example shows that not all objects are integral objects. Let $Q, R \in [A, A]$ where A is a set with two elements. Now consider*

$$Q = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \quad ; \quad R = \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix} \quad = \quad \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

That is $Q; S = \perp\!\!\!\perp$. However, neither $Q = \perp\!\!\!\perp$ nor $S = \perp\!\!\!\perp$. Hence we can conclude that A is not an integral object according to the definition above.

Lemma 2. *The following properties are equivalent:*

1. *A is an integral object,*
2. *Every non-zero relation in $\mathcal{R}[A, A]$ is total,*
3. *$\mathbb{I}$ is an atom.*

The proof can be found in [50]. The following lemma summarizes the basic properties of relations for which the source or target object is integral.

Lemma 3. *Let B be an integral object, Then,*

1. *if $Q; R = \perp\!\!\!\perp_{AC}$ with $Q \in \mathcal{R}[A, B]$ and $R \in \mathcal{R}[B, C]$ then either $Q = \perp\!\!\!\perp_{AB}$ or $R = \perp\!\!\!\perp_{BC}$,*
2. *if $R \neq \perp\!\!\!\perp_{BC}$ then $R; \top\!\!\!\top_{CD} = \top\!\!\!\top_{BD}$ [50].*

Hence, from the above lemma we can deduce that all non-zero relations in $\mathcal{R}[B, C]$ are total if B is integral. Now, the following definitions states that the class of all integral objects forms the basis of $\mathcal{R}$.

Definition 11. *Let $\mathcal{R}$ be a relation algebra. The basis $\mathcal{B}_R$ of $\mathcal{R}$ is defined as the full subcategory given by the class of all integral objects.*

2.6 Splitting

The next construction is motivated by the following set-theoretic principle. If X is a partial equivalence relation, i.e., a relation that is symmetric and idempotent $X;X = X$, then each element is in at most one equivalence class. We can now consider the set of all those equivalence classes and the relation that relates an element with its class, if the element belongs to such a class. We start with the following definition [19, 40, 47].

Definition 12. *A relation $Q : A \to A$ is called a symmetric idempotent relation (partial equivalence), if and only if $Q^\smile = Q$ and $Q;Q = Q$.*

The following example illustrates the above definition. Let A be a set and an object in **Rel** defined by: $A =$ { Frank, Mick, Ken, Tom, Steve, Denis, Mark, Joan} such that $Q : A \to A$ is a relations on the set of persons which relates two people if they took the the same course or class at college defined by: {<Frank, Frank>, <Frank, Ken>, <Frank, Tom>, <Mick, Mick>, <Ken, Frank> <Ken, Ken>, <Ken, Tom>, <Tom, Frank>, <Tom, Ken>, <Tom, Tom>, <Denis, Mark>, <Denis, Denis>, <Mark, Mark>, <Mark, Denis> }. We can also represent Q as a Boolean matrix as in Figure 2.12.

$$\begin{array}{c} \\ Frank \\ Mick \\ Ken \\ Tom \\ Steve \\ Denis \\ Mark \\ Joan \end{array} \begin{array}{c} \begin{array}{cccccccc} Frank & Mick & Ken & Tom & Steve & Denis & Mark & Joan \end{array} \\ \left(\begin{array}{cccccccc} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right) \end{array}$$

Figure 2.12: Boolean Matrix of R

Q represents a partial equivalence relation which consists of 3 equivalence classes: {Frank, Ken, Tom},{Denis, Mark}, and {Mick} as well as elements which do not belong to any equivalence class, simply because they did not attend college or take the same course.

Definition 13. *Let $\mathcal{R}$ be a relation algebra, and $Q : A \to A$ be a partial equivalence relation. An object B together with a relation $R : B \to A$ is called a splitting of Q (or R splits Q) iff $R;R^{\smile} = \mathbb{I}$ and $R^{\smile};R = Q$.*

We shall illustrate this definition briefly using the persons and the partial equivalence relation $Q : A \to A$ from the preceding example. Let B = {{Frank, Ken, Tom},{Denis, Mark}, and {Mick} } be an object of the splitting of Q. Then, the splitting $R : B \to A$ is equivalent to the relation that maps every equivalence class to its elements. We can represent R as a Boolean matrix as follows:

$$\begin{array}{l} \\ \{Frank, Ken, Tom\} \\ \{Mick\} \\ \{Denis, Mark\} \end{array} \begin{array}{c} \begin{array}{cccccccc} Frank & Mick & Ken & Tom & Steve & Denis & Mark & Joan \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array}$$

Figure 2.13: Boolean Matrix of R (Splitting)

The Boolean matrix below verifies that $R;R^{\smile} = \mathbb{I}$, after computing $R;R^{\smile}$

$$\begin{array}{l} \\ \{Frank, Ken, Tom\} \\ \{Mick\} \\ \{Denis, Mark\} \end{array} \begin{array}{c} \begin{array}{ccc} \{Frank, Ken, Tom\} & \{Mick\} & \{Denis, Mark\} \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{array}$$

Figure 2.14: Boolean Matrix of $R;R^{\smile}$

And also we can verify that $R^{\smile};R = Q$. This is shown in the figure below.

$$\begin{array}{c|cccccccc}
 & Frank & Mick & Ken & Tom & Steve & Denis & Mark & Joan \\
\hline
Frank & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
Mick & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
Ken & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
Tom & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
Steve & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
Denis & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
Mark & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
Joan & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Figure 2.15: Boolean Matrix of $R^{\smile};R$

2.6.1 Subobjects or Subsets

In this subsection, we want to look at a special case of splittings. This will help us to state a theorem between a relation algebra and its basis. We can represent subsets in two different ways inside a relation algebra;

1. by vectors, i.e. a relation v such that $v = \mathbb{T};v$

2. by partial identities, i.e. a relation $\zeta \subseteq \mathbb{I}$.

The two properties above are equivalent and may be used to characterize subobjects [19, 40, 50]. Subobjects are defined as follows:

Definition 14. *Let $\zeta \in \mathcal{R}[A,A]$ be a partial identity. An object B together with a relation $\psi \in \mathcal{R}[B,A]$ is called subobject (or subsets) of A induced by ζ iff*

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

2. $\psi^{\smile};\psi = \zeta$

A relation algebra has subobjects iff for all partial identities a subobject exist. Notice that subobjects are special case of splittings since every partial identity is a partial equivalence relation.

As mentioned above, the basis is smaller than $\mathcal{R}$. Hence, the basis of $\mathcal{R}$ is never isomorphic or equivalent to $\mathcal{R}$. This makes integral objects and subobjects very important structure to consider. The theorem below throws more light on this.

Theorem 1. *Let $\mathcal{R}$ be a relation algebra with relational sums, and let $\mathcal{B}$ be the basis of $\mathcal{R}$. Then $\mathcal{B}$ is a proper subalgebra of $\mathcal{R}$.*

See [49] for proof.

2.7 A Pseudo Representation Theorem

From the above, we have been able to define relational sums, subobjects, integral objects, basis and their relationships. With all these we are now ready to state the main theorem. This theorem defines the relationship between relation algebras and matrix algebra [50].

Theorem 2. *Let $\mathcal{R}$ be a relation algebra with relational sums and subobjects and $\mathcal{B}$ be the basis of $\mathcal{R}$. Then $\mathcal{R}$ and $\mathcal{B}^+$ are equivalent [49].*

The theorem above states that relation algebras and matrix algebras are equivalent when dealing with categorical structures. However, this structure may not be isomorphic since the functors from $\mathcal{R}$ to $\mathcal{B}^+$ may identify isomorphic objects. See [49] for proof.

Chapter 3

Related Work

The increasing use of relations in several fields have resulted in the development of highly sophisticated computer tools to manipulate relations. For instance, researchers working with relations can only do computations by hand when the problem is relatively small but when the relation or problem at hand is very large then it becomes painful, tedious and time consuming to do. Thus, several computer automated systems have been developed to help curb this problem. In this chapter, we shall look at some systems that have been developed to manipulate relations [4, 9, 10, 54].

The following quotes list some computer algebra systems developed to assist in the manipulation and programming of relations including relation-algebraic theorem proving:

1. "RelView, developed in Kiel (formerly in Munich). RelView is an interactive tool for computer-supported manipulation of relations represented as Boolean matrices or directed graphs, especially for prototyping relational specifications and programs " [57].

2. "REALM, Computation and Visualization of Finite Relation Algebras. Brock University, 2006-2009 (Ahmed, Zafor, M.Sc. Thesis) " [54].

3. "Libra, developed in Adelaide. Libra is a relational programming language that explores the different values yielded by relations by back-tracking rather than parallel execution " [57].

4. "RATH, developed in Munich. RATH is a collection of Haskell modules that allow exploration of (finite) relation algebras and several

weaker structures such as categories, allegories, and Dedekind categories " [57].

5. "TituRel- Multilevel Relational Reference Language. A highly expressive multilevel relational reference language is proposed that covers most possibilities to use relations in practical applications. The language is designed to describe work in a heterogeneous setting. It originated from a Haskell-based system announced in [41], forerunners of which were [24, 25]. This language is intended to serve a variety of purposes. First, it shall allow to formulate all of the problems that have so far been tackled using relational methods providing full syntax- and type-control. Transformation of relational terms and formulae in the broadest sense shall be possible as well as interpretation in many forms. In the most simple way, boolean matrices will serve as an interpretation, but also non-representable models as with the Rath-system may be used. Proofs of relational formulae in the style of Ralf or in Rasiowa-Sikorski style are aimed at" [42].

6. "RALF, developed in Munich, currently not maintained. RALF is a relation-algebraic formula manipulation system and interactive proof checker. Its meta language is first-order predicate logic in calculational style. Proofs are manipulated via a graphical user interface: theorems are represented as trees and the subexpression to be transformed can be selected by mouse click " [57].

7. "RALL, developed in Munich, currently not maintained. RALL embeds the theory of abstract relation algebras in Isabelle " [57].

8. "CrocoPat, developed in Lausanne, Berkeley, and Cottbus. CrocoPat is a tool for simple and efficient relational computation, manipulating relations of any arity " [57].

9. "PCP - Point and Click Proofs, developed in Orange, California. The interactive "Point and Click Proof" (PCP) environment allows the investigation of algebraic theories, such as groups, rings, lattices, and others, including relation algebras " [57].

10. "RelAPS, developed in St. Catherines, Ontario RelAPS is an interactive system assisting in proving relation-algebraic theorems" [21].

In the rest of the chapter we shall review the implementation and application of the first four mentioned systems. For details on any of the above systems please refer to [42, 43, 57, 60].

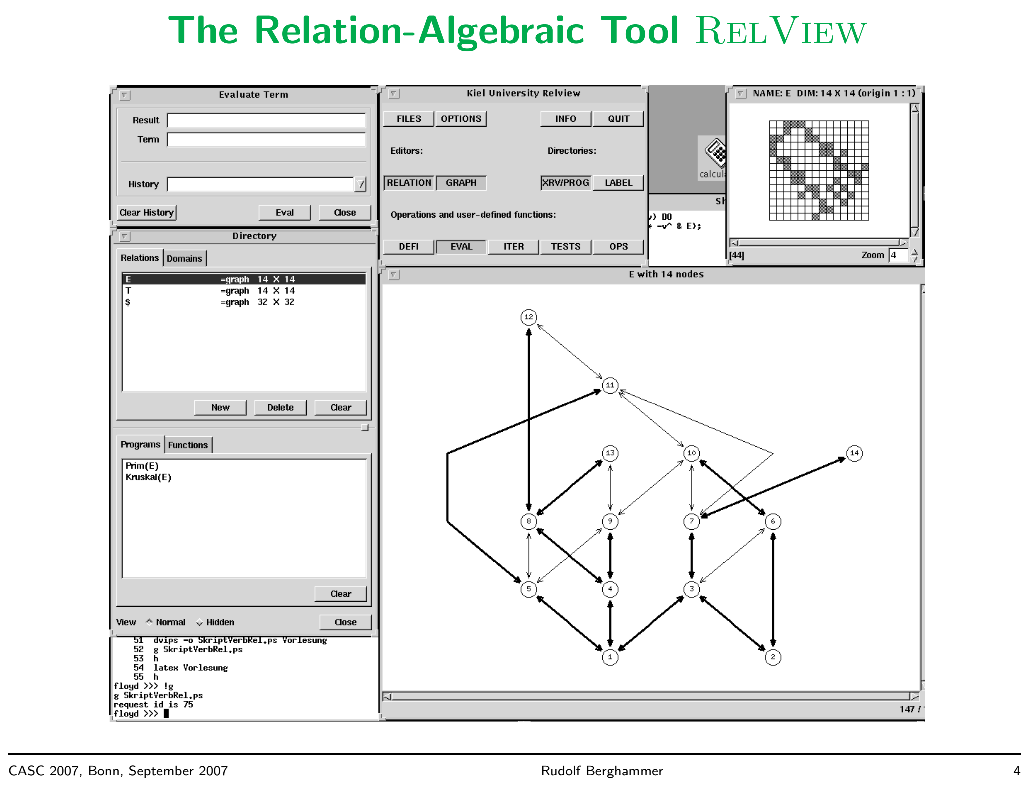
3.1 RelView- An OBDD-Based Computer Algebra System for Relations

RelView is an interactive and specific purpose computer algebra system with a graphical user interface for the manipulation and visualization of relation algebra and relational programming. The first versions were developed at the University of the German Forces Munich from 1988 to 1992. It was further rebuilt and extended by Kiel University since 1993.

In RelView, all data are represented as relations. It provides an interface for dealing with both homogeneous and heterogeneous relations. Homogeneous relations are represented as directed graphs, including sophisticated algorithms for drawing them nicely. Relations are represented in RelView as Boolean matrices. This second representation is very useful for visually editing and also for discovering various structural properties that are not evident from a representation of relations as directed graphs.

RelView is a system for set-theoretic relations, i.e., works within the standard model of relation algebras. This system was written in C programming language using a binary decision diagram package called CUDD [13, 46]. It runs on Sun SPARC workstations and INTEL-based Linux systems.

The RelView system is capable of manipulating very large relations including membership relation, inclusion relation and relations involving the whole set or the power set. It can manipulate several relations simultaneously as well as giving the user the freedom to manipulate and analyze them by pre-defined operations, tests and user-defined relational functions and relational programs. The figure below is a snapshot of the RelView system. For a detailed description please refer to [4, 9, 10, 54].



The Relation-Algebraic Tool RelView

```
E                    =graph  14 X 14
T                    =graph  14 X 14
$                    =graph  32 X 32
```

```
    51  dvips -o SkriptVerbRel.ps Vorlesung
    52  g SkriptVerbRel.ps
    53  h
    54  latex Vorlesung
    55  h
floyd >>> !g
g SkriptVerbRel.ps
request id is 75
floyd >>> 
```

CASC 2007, Bonn, September 2007 Rudolf Berghammer 4

Figure 3.1: pictorial view of RelView [58]

3.1.1 Applications of the RelView system

RelView assists mathematicians and computer scientists working with relation algebra, concrete relations, relation-based discrete structures, and relational programs [4, 9, 10, 11].

1. In the formulation and proving of relational theorems or relation algebraic reasoning, RelView can be used as an efficient tool to construct examples to demonstrate the validity of a theorem or to find counter examples to disprove a proposed property. The RelView system has been designed with an interactive interface which allows to add, change and remove relations and directed graphs, and makes it easier to apply functions and to reuse relational programs at every time within a working session.

2. Another important application domain of RelView is in relational program development. It helps in the implementation of relational algorithms using the programming language provided by the system.

3. It can also be used to accomplish various tasks in formal program development such as specification, testing, detection of loop invariants and other important properties necessary for correctness proofs, rapid prototyping, and improving efficiency.

4. RelView has a complete graphical user interface which makes it more suitable for teaching, visualization and animation. It is a very good means to demonstrate how certain algorithms work.

Some specific problems to which RelView has been applied include, graph-theoretic problems such as maximum cliques and the lattice-theoretic problems such as the cut completion. The results produced seem to be relatively good. For more information refer to [4, 9, 10, 11].

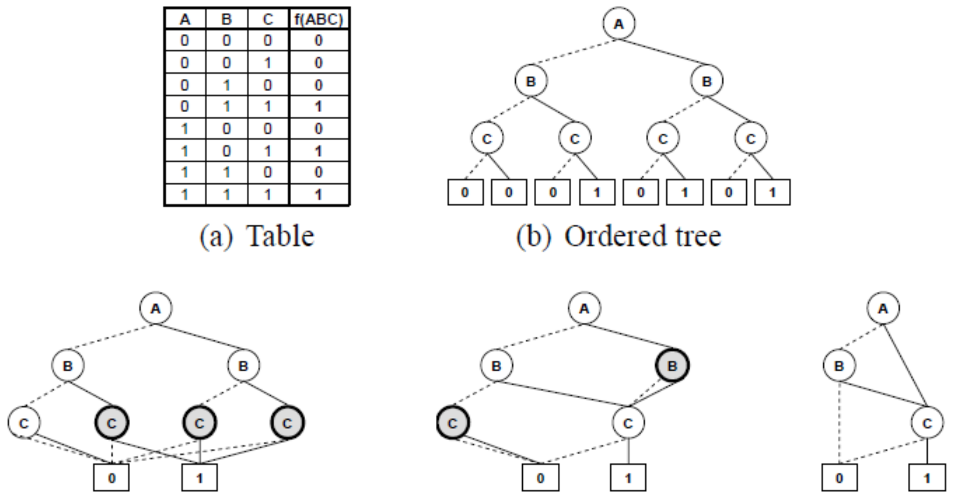
Initially RelView was implemented using Boolean arrays for storing and manipulating relations. As a result of this, its use in several applications was restricted to small input relations. RelView was redesigned to overcome this problem. A more efficient data structure called Binary Decision Diagram (BDD) was used this time. In this section we want to look at Binary Decision Diagrams and how RelView uses it to implement relations.

3.1.2 Binary Decision Diagrams

BDD has recently become one of the state of the art data structures used to manipulate very large Boolean functions. A BDD is a directed acyclic graph with one root and two leaf nodes. Each decision node is labeled by a Boolean variable and has two child nodes (*low child* and *high child*). A BDD is called ordered if different variables appear in the same order on all paths from the root node. A BDD is said to be reduced if there is no isomorphic subgraphs within the diagram, hence, the name Reduced Ordered Binary Decision Diagram (ROBDD).

We will illustrate the operations of BDDs by example. Let $f : \mathbb{B}^3 \to \mathbb{B}$ be a Boolean function defined in the disjunctive normal form as $f(A, B, C) = (\overline{A} \wedge B \wedge C) \vee (A \wedge C)$. The BDD for this function is shown in Figure 3.2. In the BDD diagram, 1 and 0 are represented by the solid lines and dotted lines respectively. Now, let us find the value of $f(A, B, C)$ given that $A = 1$, $B = 0$, $C = 1$. To compute this we have to follow corresponding directed arcs from the root node. From Figure 3.2 b the root node is the variable labeled A.

Now starting from the root node A with the given assignment $A = 1$, we take the 1-arc and traversing we encounter a node labeled with B variable and since the value of $B = 0$ we take the 0-arc. Traversing again we arrive at the node with a variable labeled C and since $C = 1$, we take the 1-arc and the leaf node of this arc is 1 which is the required result for evaluating $f(1, 0, 1)$.



A	B	C	f(ABC)
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

(a) Table (b) Ordered tree

(c) Isomorphic nodes (d) Redundant nodes (e) OBDD

Figure 3.2: Binary Decision Diagrams

The representation of a Boolean function by means of a BDD is not unique. However, when BDD variables are ordered and reduced in their canonical form, then the BDD representation is unique. Figure 3.2 (b), (c), (d), (e) are different BDD representations of the same Boolean function f. The BDD in Figure 3.2 (c) can be deduced from the BDD in (b) by applying reduction rules such as removing isomorphic subgraphs and elimination redundant nodes from the diagram. Also by continuous application of reduction rules the BDDs in Figure 3.2 (c), (d), (e) can be achieved.

Several BDD packages have been written to aid the manipulation of BDDs, notable among them is the CUDD package from the University of Colorado, Boulder (USA). This is the package used by RelView to implement relation algebra. For the detail description of this package refer to [13].

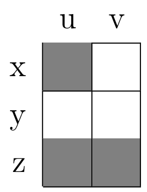
3.1.3 Implementation of Relations using ROBDDs in RelView

Reduced Ordered Binary Decision Diagram (ROBDD) is a data structure used by RelView to implement relations. In this representation RelView uses a binary encoding system in which the domain and range of relations are represented by the row (X) and column (Y) of a Boolean matrix respectively. We will illustrate how this is done in RelView with the following example (our example will be based on a finite set and a concrete relation).

Suppose that we have two sets $X = \{x, y, z\}$ and $Y = \{u, v\}$ and let $R : X \leftrightarrow Y$ be a relation defined by $R := \{(x, u), (z, u), (z, v)\}$ then R can be represented as a 3×2 Boolean matrix as:

$$\begin{array}{c|cc} & u & v \\ \hline x & 1 & 0 \\ y & 0 & 0 \\ z & 1 & 1 \end{array}$$

we can visualize this relation in RelView as:



Now let us define a binary encoding for X and Y as follows: $c_X : X \to \mathbb{B}^2$ such that $c_X(x) = 00$, $c_X(y) = 01$, $c_X(z) = 10$, and $c_Y : Y \to \mathbb{B}$ such that $c_Y(u) = 0$, $c_Y(v) = 1$. We can now represent this relation as a ternary partial Boolean function $f_R : \mathbb{B}^3 \to \mathbb{B}$. This Boolean function is depicted in the table below. x_1 and x_2 are the variables encoding X and y_1 is the variable encoding Y. In order to obtain a totally defined function, we will add 0's to where the function is undefined.

x_1	x_2	y_1	$f(x_1, x_2, x_3)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Table 3.1: Truth table: $f_R : \mathbb{B}^3 \to \mathbb{B}$

Base on this table we can derive a ROBDD for the function by using a fixed variable ordering such as $x_1 < x_2 < y_1$. This is shown in Figure 3.3.

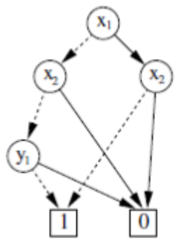


Figure 3.3: Binary Decision Diagram [5]

For any arbitrary values of m and n with the binary encoding $c_X : X \to \mathbb{B}^m$ and $c_Y : Y \to \mathbb{B}^n$ such that $m = \lceil \log |X| \rceil$ and $n = \lceil \log |Y| \rceil$, we can implement a relation $R : X \leftrightarrow Y$ by two sizes $|X|$ and $|Y|$ and the ROBDD of the totalized Boolean function $F_R : \mathbb{B}^{m+n} \to \mathbb{B}$, where $f_R(x_1, ..., x_m, y, ..., y_n) = 1$ if and only if the decodings $c_X^{-1}(x_1, ..., x_n)$ and $c_Y^{-1}(y_1, ..., y_n)$ are related through the relation R with variable ordering $x_1 < ... < x_m < y_1 ... < y_n$. We shall demonstrate this with the membership relation and one basic binary operation (composition). Let $\omega : X \leftrightarrow 2^X$ be a membership relation defined on the set $X = \{a, b, c, d\}$. This can be depicted in RelView as a Boolean 4×16 matrix:

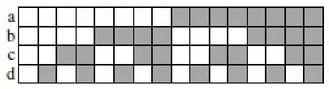


Figure 3.4: membership relation [5]

In order to represent this relation using an ROBDD we encode X and 2^X such that the variable x_1 and x_2 encode the 4 element of the domain of ω and the variable y_a, y_b, y_c and y_d encode the 16 elements of the range 2^X of ω. This is depicted by the figure below.

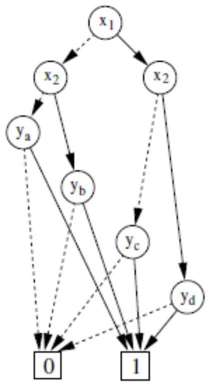


Figure 3.5: Binary Decision Diagram

Now let us see how composition of two relation is implemented in RelView. Suppose that X, Y, Z are finite sets with sizes $|X| = n$, $|Y| = m$ and $|Z| = k$ such that $R : X \leftrightarrow Y$ and $S : Y \leftrightarrow Z$ are two relations, then we can define the composition of R and S ($R;S : X \leftrightarrow Z$) in component-wise as $(RS)_{xz} \Leftrightarrow \exists y(R_{xy} \wedge Syz)$ for all $x \in X$ and $z \in Z$. Now let $l_n = \lceil \log n \rceil$, $l_m = \lceil \log m \rceil$, and $l_k = \lceil \log k \rceil$ and using the binary encodings $c_X : X \to \mathbb{B}^{l_n}$, $c_Y : Y \to \mathbb{B}^{l_m}$, $c_Z : Z \to \mathbb{B}^{l_k}$ we obtain the Boolean functions for R and S as $f_R : \mathbb{B}^{l_n+l_m} \to \mathbb{B}$ and $f_S : \mathbb{B}^{l_m+l_k} \to \mathbb{B}$ such that $f_R(r_1, ..., r_{l_n}, r_{l_n+1}, ..., r_{l_n+l_m}) = 1$ iff $C_X^{-1}(r_1, ..., r_{l_n})$ and $C_Y^{-1}(r_{l_n+1}, ..., r_{l_n+l_m})$ are related through R and $f_S(s_1, ..., s_{l_m}, s_{l_m+1}, ..., r_{l_m+l_k}) = 1$ iff $C_Y^{-1}(s_1, ..., r_{l_m})$ and $C_Z^{-1}(s_{l_m+1}, ..., s_{l_m+l_k})$ are also related through S. If we assume that $r_{l_n+1,...,r_{l_n+l_m}}$ with $s_1, ..., s_{l_m}$ are variables, then we can derive a general form of the composition of $R;S$ as $f_{RS} = \exists s_1 ... \exists s_{l_m}(f_R \wedge f_S)$ [5].

3.1.4 Limitation of RelView system for relation algebras

As pointed out earlier, relation algebra consist of standard and non-standard models. The standard model forms the concrete relation algebras. The basic underlying layer of RelView is the standard model, therefore it becomes impossible to implement or manipulate certain kind of arbitrary (heterogeneous) relation. For example, $\mathbb{T};\mathbb{T}=\mathbb{T}$ or $P(A) \times P(B) \cong P(A + B)$ is always true in RelView since it uses the standard model. However, these properties might not be true in some non-standard models.

3.2 REALM - A SYSTEM TO MANIPULATE RELATIONS

REALM is a more general interactive relation algebras manipulator that visualizes an arbitrary relation algebra (both concrete and abstract relation algebras) using matrix algebra approach. It was developed by Ahmed Zafor supervised by Professor Michael Winter at Brock University, Ontario Canada (2009). It implements a complete relation algebra and has a graphical user interface. It was written in Java using JDK 1.6 platform which supports several generic types. It has several abstract classes; for instance the class RelAlg has two generic type parameters O and M for the type objects and the type relations respectively. It has several accessors methods for performing binary and unary operations between objects. It defines abstract methods from standard operations and abstract methods such as source and target for getting the source and target of the relation. It also has a class that accepts basis files which contains all the relations. This class is initialized by reading or passing an XML file with a set of objects, relations between them as well as operations. It has several graphical user interfaces such as loading relations, saving relations, switching relations, deleting relations, etc. See [54] for detail description.

3.2.1 IMPLEMENTATION

REALM was implemented using the array data structure. Relations are stored in one or two dimensional arrays using hash tables for referencing. The underlying data structure of REALM limits its ability to manipulate and manage very large relations with respect to speed and memory as well as flexibility. This was the same problem faced by the first versions of RelView system. For more information see [54].

3.3 LIBRA: A Lazy Interpreter of Binary Relational Algebra

Another system that implements relations is called LIBRA. The following better explains what the LIBRA system is.

> "LIBRA is a general-purpose programming language based on the algebra of binary relations. It is an attempt to unify functional and logic programming, retaining the advantages of both, and avoiding some of

> the problems. It has all the features needed of a programming language, including a straightforward semantic interpretation. Since program specifications are easily expressed as relations, it offers a simple path from a specification to a program and from the program to its proof of correctness. The algebra of binary relations has several operators whose effects are like those of familiar procedural language constructs, for example, relational composition is analogous to sequential execution" [18].

See [59] for details description.

3.4 RATH - Relation Algebra Tools in Haskell

RATH is a tool for manipulating relations written in Haskell. The following quote gives a better descriptions of the system.

> "RATH-1 is a collection of Haskell modules that allow exploration of (finite) relation algebras and several weaker structures such as categories, allegories, and Dedekind categories by providing different means to construct and test such algebras. The kernel of our library is strictly conformant to the Haskell 98 standard, and can therefore be expected to be usable on future Haskell systems, too. For ease of use, we additionally provide a more elegant interface using non-standard extensions" [27].

Since our main focus is on RelView and the REALM systems we will not give details on the last four systems, however for details on these systems refer to [42, 43, 57, 60].

Chapter 4

RelMDD System

From the preceding chapters, we have been able to show how relations can be visualized using matrices, so base on this, we want to develop a system for relations using MDDs. RelMDD is an arbitrary relation algebra manipulator library written in C programming language. It is a package that implements relation algebras using the matrix algebra approach and can be imported by other programs and/or languages such as Java and Haskell when programming or manipulating arbitrary relations. We implemented RelMDD using the matrix algebra over a suitable basis approach hence, it is capable of manipulating both the classes of standard and the non-standard models of relation algebra.

The implementation requires a more advanced form of OBDD's (data structure) [13, 15], since OBDD's are limited to Boolean values. We need a data structure similar to OBDD's that is able to store arbitrary coefficient of a matrix instead of just Boolean values. We shall assume that the reader is familiar with OBBDs, otherwise, the reader should refer to the previous chapter or see [5, 13, 15]. Before we give a full account of the system, we will first of all look at the data structures used for the implementation of RelMDD. The main data structure used is multiple-value decision diagrams (MDDs) [17, 32, 33, 34]. However, we will get to know later on in this chapter that this data structure is best or usually implemented by encoding them into another form of decision diagram called algebraic decision diagrams (ADDs) [3]. We will refer to [13, 15] for definitions and examples of MDDs.

4.1 Data Structure and Implementation

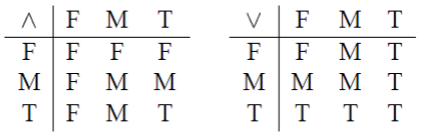
A multiple-valued decision diagram (MDD) is a natural extension of the reduced ordered binary decision diagram (ROBDD) to the multiple-valued case [17, 32]. MDDs are considered to be more efficient and require smaller memory size than the ROBDDs. In this section, we will examine various properties and examples of MDDs.

4.1.1 Multiple-Valued Decision Diagram (MDD)

Let V be a set of finite size r . An r-valued function f is a function mapping V^n for some n to V. We will identify the n input values of f using a set of variables $X = \{x_0, x_1, ..., x_n\}$. Each x_i as well as $f(X)$ is r-valued, i.e., it represents an element from V. The function f can be represented by a multi-valued decision diagram. Such a decision diagram is a directed acyclic graph (DAG) with up to r terminal nodes each labeled by a distinct value from V. Every non-terminal node is labeled by an input variable x_i and has r outgoing edges [17, 32].

An MDD is ordered (OMDD) if there is an order on the set of variables X so that for every path from the root to a leaf node all variables appear in that order. Furthermore, an MDD is called reduced if the graph does not contain isomorphic subgraphs and no nodes for which all r children are isomorphic (i.e. there is no redundant node in which two edges leaving a node point to the same next node within the graph). An MDD that is ordered and reduced is called a reduced ordered multi-valued decision diagram (ROMDD). Both ROBDDs and ROMDDs have widely been studied. Most of the techniques used when implementing a package for the creation and manipulation of an ROMDD are those already known from the binary case. These techniques include edge negation, adjacent level interchange, operator nodes and logical operation [17, 32, 34].

Example 14. *Let f be a multiple valued logic defined over the truth values T, M, F such that $T = True$, $M = Maybe$ and $F = False$. The truth table below defines the conjunction and disjunction for this function. For instance, from the table $F \wedge T = F$, $T \vee M = T$. Using this table we can produce an MDD for the function $f = y \wedge x$ as shown in Figure 4.1. Figure 4.2 shows the reduced multiple valued decision diagram for f after applying reduction rules to the MDD on Figure 4.1a [17].*



∧	F	M	T
F	F	F	F
M	F	M	M
T	F	M	T

∨	F	M	T
F	F	M	T
M	M	M	T
T	T	T	T

Figure 4.1:

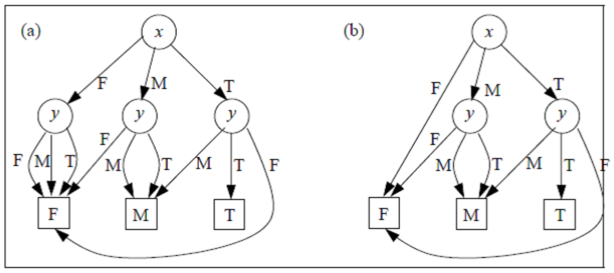


Figure 4.2: $f = y \wedge x$, (a)MDD; (b)ROMDD [17]

4.1.2 Heterogeneous Multiple-Valued Decision Diagram

The following definitions and examples gives a more general version of MDDs called a heterogeneous MDDs.

Definition 15. *Let V(X) be two-valued logic function such that $X = \{x_1, x_2, x_3, ..., x_n\}$ and $x_i (i = 1, 2, ..., n)$ are binary variables. If $X = X_1 \cup X_2 \cup ... \cup X_u$ and $X_i \cap X_j = \varnothing (i \neq j)$, then $(X_1, X_2, ..., X_u)$ is a partition of X. An ordered set of variable X_i is called a super variables. If $|X_i| = k_i (i = 1, 2, ..., u)$ and $k_1 + k_2 + ... + k_u = n$, then $V(X)$ can be represented by $V(X_1, X_2, ..., X_u) : P_1 \times P_2 \times P_3 \times ... \times P_u \to \mathbb{B} = \{0, 1\}$ where $P_i = \{0, 1, 2, ...2^{k_i} - 1\}$. This is called heterogeneous MDD [34].*

We shall illustrate this with the following example:

Example 15. *Suppose (X_1, X_2) is a partition of X, where $X = (x_1, x_2, x_3, x_4, x_5)$ and each x_i is a binary variable. When $X_1 = (x_1, x_2)$ and $X_2 = (x_3, x_4, x_5)$, $k_1 = 2, k_2 = 3, P_1 = \{0, 1, 2, 3\}$, and $P_2 = \{0, 1, ..., 7\}$, we can represent a 5-variable logic function V(X) by the multi-valued function $V(X_1, X_2) : P_1 \times P_2 \to \mathbb{B}$ such that X_1 takes 4 values and X_2 takes 8 values.*

Definition 16. *If $X = (x_1, x_2, ..., x_n)$ is partitioned into $(X_1, X_2, ..., X_u)$, a ROMDD representing a logic function $f(X)$ is called a heterogeneous MDD. In the case where $k_1 = k_2 = ... = k_u$, a ROMDD representing logic function $f(X)$ is called homogeneous MDD. A homogeneous MDD is also denoted by $MDD(k)$, where $k = k_1 = k_2 = ... = k_u$.*

Example 16. *Let $f = x_1x_2x_3 \vee x_2x_3x_4 \vee x_3x_4x_1 \vee x_4x_1x_2$. Figure 4.3 and Figure 4.4 show various decision diagrams for f. In Figure 4.3(a) the solid lines represents the 1-edges and the 0-edges are represented by the dotted lines. In Figure 4.3(b) the input variables of $X = (x_1, x_2, x_3, x_4, x_5)$ are partitioned into (X_1, X_2) such that $X_1 = (x_1, x_2)$ and $X_2 = (x_3, x_4)$, i.e., it shows an MDD(2) for f. Finally, Figure 4.4* *shows two additional heterogeneous MDDs for f with the partitions $X_1 = (x_1, x_2, x_3)$ and $X_2 = (x_4)$ respectively $X_1 = (x_1)$ and $X_2 = (x_2, x_3, x_4)$.*

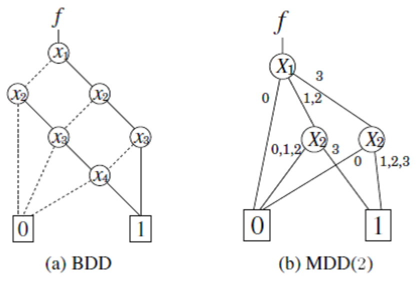


Figure 4.3: MDD

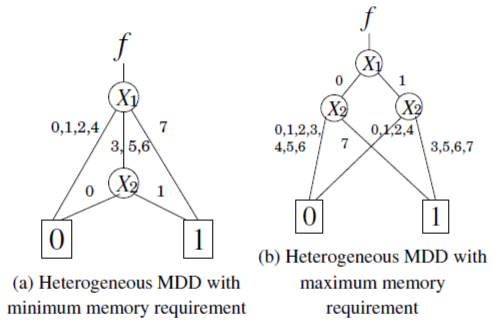

f
X1
0,1,2,4
7
3, 5,6
X2
0
1
0
1
(a) Heterogeneous MDD with minimum memory requirement
f
X1
0
1
X2
X2
0,1,2,3, 4,5,6
7
0,1,2,4
3,5,6,7
0
1
(b) Heterogeneous MDD with maximum memory requirement

Figure 4.4: MDD

MDDs are considered to be more efficient and can represent logic functions with smaller memory size and shorter path length than ROBDDs [17, 32, 34].

MDDs are usually traversed in one of the following three ways:

1. A depth-first traversal starting at the top node and moving along the edges from each node to the descendants or child nodes. This technique is a very well-known conventional graph traversal.

2. ROMDDs can be traversed horizontally by moving from one node to another of all nodes labeled by the same variable. This corresponds to a specific breath-first traversal.

3. ROMDDs can also be traversed by applying both techniques described above simultaneously [32].

There are several techniques that can be used to reduce the size of MDDs. This is very important if one wants to minimize the amount of storage capacity needed to store an

MDD as well as the time and speed required to manipulate MDDs. These techniques include edge negation, variable reordering, operator nodes and logical operation, and adjacent level interchange. For detailed descriptions on these techniques refer to [32].

4.1.3 Implementation of Relations using MDDs

In this section, we want to show how MDDs can be used to implement relations. We shall assume that relations are given or stored as matrices. The elements of the matrices become the leaf or terminal nodes of an MDD after encoding the domain and the range of a relation.

We will illustrate the implementation by examples using fuzzy relations. Figure 4.5 shows the Hasse-diagram of a lattice that we will use as membership values of fuzzy relations. Notice that, this lattice is actually a Heyting algebra. See [2, 51] for details on Heyting algebras.

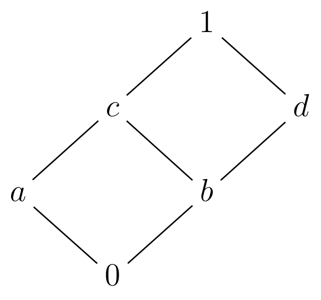


Figure 4.5: Heyting algebra $\mathcal{L}$

Hence, from the figure above, we can derive operation tables for the meet and the join operations of the lattice $\mathcal{L}$ or the structure described above. These tables are depicted in Table 4.1.

$\sqcap$	0	a	b	c	d	1
0	0	0	0	0	0	0
a	0	a	0	a	0	a
b	0	0	b	b	b	b
c	0	a	b	b	b	c
d	0	0	b	b	d	d
1	0	a	b	c	d	1

$\sqcup$	0	a	b	c	d	1
0	0	a	b	c	d	1
a	a	a	c	c	1	1
b	b	c	b	c	d	1
c	c	c	c	c	1	1
d	d	1	d	1	d	1
1	1	1	1	1	1	1

Table 4.1: Meet and join operations in the Heyting algebra $\mathcal{L}$

Now suppose that R and S are two fuzzy relations of the structure above represented by the following matrices.

$$R = \begin{pmatrix} a & b & a \\ 0 & 1 & c \end{pmatrix} \quad S = \begin{pmatrix} a & a & 1 \\ 0 & b & d \end{pmatrix}$$

Using the meet operation of the underlying structure described above, we can compute the intersection (or meet) of R and S as:

$$T := R \sqcap S = \begin{pmatrix} a & 0 & a \\ 0 & b & b \end{pmatrix}$$

In order to implement this relation using MDDs, we must first convert R and S into MDDs by encoding their corresponding matrices first as functions and then as graphs. We will adopt an approach similar to the method used in the RelView system. We will start by encoding R. Let $\mathbb{L} = \{0, a, b, c, d, 1\}$ be a set (universe) containing all the elements under consideration, and $X = (x_1, x_2)$ and $Y = (y_1, y_2, y_3)$ be the elements labeling the domain and range of R respectively. Since $\mathbb{L}$ has 6 elements we need only one variable for the rows and the columns of R and S respectively. However, in order to obtain a totally defined function, we have to enlarge the matrices so that its size is a power of the number of elements, i.e., the corresponding function is defined for every possible input for each variable. We will fill the new entries with 0's. The figure below shows R enlarged to a proper size with its rows and columns labeled by values, i.e., by elements of the lattice $\mathbb{L}$, for the row variable u and the column variable v [2].

$$\begin{array}{c|cccccc} & 0 & a & b & c & d & 1 \\ \hline 0 & a & b & a & 0 & 0 & 0 \\ a & 0 & 1 & c & 0 & 0 & 0 \\ b & 0 & 0 & 0 & 0 & 0 & 0 \\ c & 0 & 0 & 0 & 0 & 0 & 0 \\ d & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Let $f_R : \mathbb{L}^2 \to \mathbb{L}$ be a multivalued function which takes elements from $\mathbb{L}^2$ to $\mathbb{L}$ with variables encoding the domain defined as $C_X(x_1) = 0$, $C_X(x_2) = a$ and that of the range are $C_Y(y_1) = 0$, $C_Y(y_2) = a$, and $C_Y(y_2) = b$ such that $f_R(u,v) = w \Leftrightarrow C^{-1}(u)$ and $C^{-1}(u)$ are related through w, where u and v are variables such that $u < v$. For example $f_R(x_1, y_2) = b$ since x_1 and y_2 are related through b. The table below shows the corresponding function f_R encoding R where _ stands for an arbitrary parameter not listed before.

u	v	$f_R(u,v)$
0	0	a
0	a	b
0	b	a
a	0	0
a	a	1
a	b	c
_	_	0

Table 4.2: Encoding of f_R

Now based on the variable ordering $u < v$, we can produce an MDD representing f_R as shown on the left in the figure below. In addition, we present the corresponding reduced graph on the right-hand side of the figure. Notice that, we only display the essential part of R in both graphs for brevity. The additional 0's are skipped:

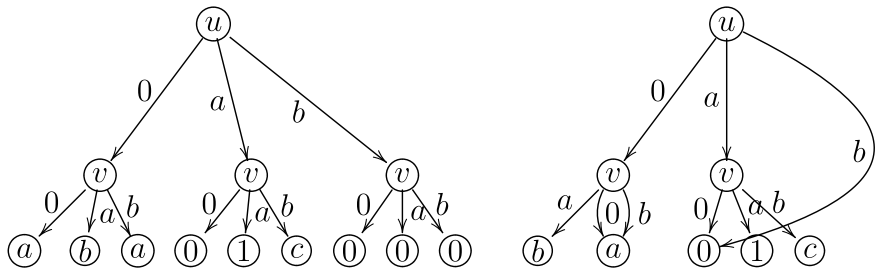


Figure 4.6: MDD and ROMDD for f_R

Now in the same fashion as above, we will construct an MDD for S. So let $\mathbb{L} = \{0, a, b, c, d, 1\}$ be a set (universe) containing all the elements under consideration, and $X = (x_1, x_2)$ and $Y = (y_1, y_2, y_3)$ be the elements labeling the domain and range of R respectively. This is depicted in the figure below:

$$\begin{array}{c c} & \begin{array}{cccccc} 0 & a & b & c & d & 1 \end{array} \\ \begin{array}{c} 0 \\ a \\ b \\ c \\ d \\ 1 \end{array} & \begin{pmatrix} a & a & 1 & 0 & 0 & 0 \\ 0 & b & d & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Let $f_S : \mathbb{L}^2 \to \mathbb{L}$ be a multivalued function which takes elements form $\mathbb{L}^2$ to $\mathbb{L}$ with variables encoding the domain defined as $C_X(x_1) = 0$, $C_X(x_1) = a$ and that of the range are $C_Y(y_1) = 0$, $C_Y(y_2) = a$, and $C_Y(y_2) = b$ such that $f_S(u, v) = w \Leftrightarrow C^{-1}(u)$ and $C^{-1}(u)$ are related through w, where u and v are variables such that $u < v$. The table below shows the complete encoding of f_S.

u	v	$f_S(u, v)$
0	0	a
0	a	a
0	b	1
a	0	0
a	a	b
a	b	d
_	_	0

Table 4.3: Encoding of f_S

Using this table, we get the MDD and its reduced form as:

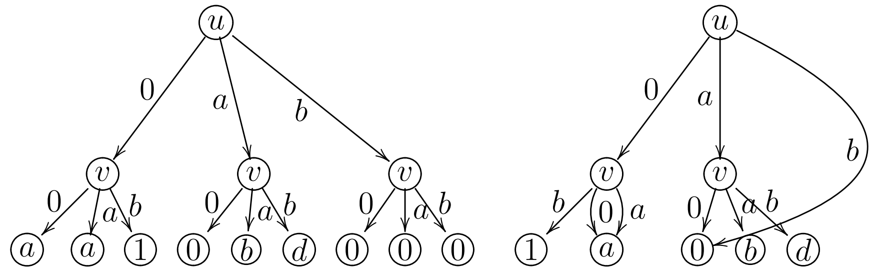


Figure 4.7: MDD and ROMDD for f_S

Now, define an `apply` operation on the MDDs of R and S as an operation which applies a function to a leaf node of R and the corresponding leaf node of S. Corresponding leaf nodes are determined by the same path from the root to the leaf in both graphs, i.e., the same spot in each matrix. In order to find T, the intersection of R and S, we shall derive T from the MDDs of R and S by using the `apply` operation defined above. For example, the leaf node of the edges labeled 00 from Figure 4.6 is a, while the leaf node of the edges labeled 00 of Figure 4.7 is also a. So upon applying the `apply` operation we get an MDD in which the path 00 leads to the node labeled a because $a \sqcup a = a$. We can also deduce the union of R and S in the same procedure as above, the only difference here is the operation (union) that is passed into `apply`, otherwise all derivations are the same.

Now suppose we want to find the converse (transpose) of S which is given by Figure 4.8 and using the same encoding and values of S given above; we can find the transpose of S ($S^{\smile}$) from the MDD of S by just flipping or swapping adjacent edges of the same variables at the same level of ordering of the MDD of S producing the MDD in Figure 4.8. However, in general, we also have to apply the underlying converse operation to the leaves. In this particular example we are using fuzzy relations where the converse of a membership value is the value itself. Another version of the `apply` operation applies a unary function to each leaf node of a single ROMDD. This version can be used to implement relation algebraic operations such as complement as well as converse.

$$S^{\smile} = \begin{pmatrix} a & 0 \\ a & b \\ 1 & d \end{pmatrix}$$

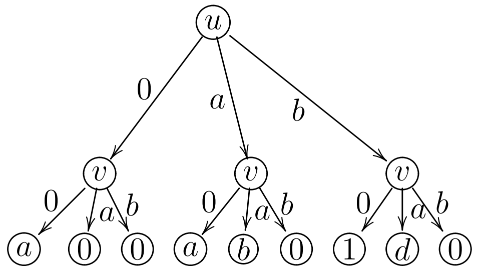


Figure 4.8: MDD for the transpose of S

In our last example, we want to see how the composition of two relations can be computed. Suppose we want to find the product of R and S. i.e.

$$R; S^{\smile} = \begin{pmatrix} a & b & a \\ 0 & 1 & c \end{pmatrix} \; ; \; \begin{pmatrix} a & 0 \\ a & b \\ 1 & d \end{pmatrix}$$

$$= \begin{pmatrix} (a \cap a) \cup (b \cap a) \cup (a \cap 1) & (a \cap 0) \cup (b \cap b) \cup (a \cap d) \\ (0 \cap a) \cup (1 \cap a) \cup (c \cap 1) & (0 \cap 0) \cup (1 \cap b) \cup (c \cap d) \end{pmatrix}$$

$$M = \begin{pmatrix} a & b \\ 1 & c \end{pmatrix}$$

Now suppose that, an `abstraction` operation is an operation on ROMDDs which applies a function to all elements of an entire row or column of a matrix. The product of R and S can be computed by a combination of the `apply` and `abstraction` operations on the MDDs of these matrices. This is similar to the well known matrix multiplication. See below for full description of `apply` and `abstraction` operations.

Generally, let us suppose that T is an m by n matrix representing a relation R, and let $X = \{x_1, x_2, ..., x_n\}$ and $Y = \{y_1, y_2 ... y_m\}$ be the domain and range of R respectively such that $\mathbb{L} = \{k_1, k_2, ..., k_n\}$ is a set containing all the elements in R, and $C_X(x_1, x_2, ..., x_n) = l_1 \in \mathbb{L}$ be the variables encoding the domain and $C_Y(y_1, y_2, ..., y_m) = l_2 \in \mathbb{L}$ be the variables encoding the range of R, then we can define a multiple value function $f : \mathbb{L} \times \mathbb{L} \to \mathbb{L}$ such that $f(x_i, y_i) = l_i$ if and only if $C_X^{-1}(x_i)$ and $C_Y^{-1}(y_i)$ are related through w where u and v are variables such that $u < v$. It is worthwhile to know that for every variable order and relation R there is exactly one reduced ordered MDD.

4.2 Algebraic Decision Diagrams(ADD) or Multiple Terminal Decision Diagram(MTDD)

One of the best methods to implement MDDs package is to make use of ADDs. This also makes it easier to use existing packages of ADDs. In this section, we shall give a brief description of ADDs and how they can be used as an efficient means to implement MDDs.

Multi-Terminal Binary Decision Diagrams (MTBDD) is a kind of BDD that provides an efficient means for arithmetic symbolic computation [3]. One significant feature is that they have multiple terminal nodes. Because they are applicable to different algebras, and because of their foundation in large Boolean algebras, such BDDs with multiple terminal nodes are usually termed Algebraic Decision Diagrams (ADDs). ADDs have been applied to solve various algorithmic problems such as computational problems involving the use of a special class of matrices called Walsh transforms, and the representation of matrices, shortest-path computation using ADDs. An ADD is basically a BDD whose leaves take on values belonging to a set of constants different from 0 or 1 [3]. In this section we will refer to [3] for definitions and examples of ADDs.

Example 17. *Suppose that T is a set of constants defined as one of the following $T = \{0,1,2\}$, $T = \{apple, orange, banana\}$ or $T = \{a,b,c\}$. We can represent ADDs as Boolean functions: Let $T \subseteq Q$ such that $|Q| = 2^m$, for some m, then we can represent the ADD with the Boolean function $f : \{0,1\}^n \to Q$. Hence we can extend all the theorems from Boolean Algebra to ADDs. An example of such theorem is the Boole's expansion theorem which is defined as $f(x,y,z,...) = x.f(1,y,z,..) + x'.f(0,y,z,...)$ [12]. The above example was taken from [3]*

4.3 Representations of Boolean Functions using ADD

Suppose that an algebraic structure is composed of a finite carrier, a set of operations and a set of distinguished elements. An ADD is a directed acyclic graph $(V, \sqcup\varphi\sqcup \mathrm{T}, \mathrm{E})$, representing a set of functions $f_i : \{0,1\}^n \to S$ such that:

1. S represents the finite carrier of the algebraic structure over which the ADD is defined.

2. V is the set of internal nodes such that the out-degree of $u \in V$ is 2. The two out-going arcs for a node $v \in V$ are labeled *then* and *else*, respectively. Each

node $v \in V$ is labeled $l(v) \in \{0, ..., n-1\}$. The label specifies a variable on which the $f_i's$ depend.

3. φ denotes the set of the function nodes such that the out-degree and in-degree of $\psi \in \varphi$ is 0 and 1 respectively.

4. T represents the set of terminal nodes such that each node $t \in T$, is labeled with an element of the carrier S denoted by $s(t)$.

5. E is the set of edges connecting the nodes of the graph. The edge connecting the node v_i to v_j is denoted by (v_i, v_j). if v_j is a descendant of v_i. then the variables of the ADD are ordered. $(i.e., (v_i, v_j) \in E$, then $l(v_i) < l(v_j))$ [3].

The following definition specifies how an ADD can be used to represent a Boolean function.

Definition 17. *An ADD representing a set of Boolean functions, one for each function node is defined as follows:*

1. *Let t be a terminal node, then a function of t is the constant function s(t), such that s(t) is an element of a Boolean algebra larger than or equal to the size of S.*

2. *If $v \in V$ and let "." and "+" be the Boolean conjunction and disjunction respectively, then we can define the function of a node as $l(v).f_{then} + l(v)'.f_{else}$, where f_{then} and f_{else} are the functions of the* then *and* else *children.*

3. *The only child of the function is $\phi \in \varphi$.*

[3].

Suppose that $S = \{0, ..., r-1\}$, such that r is a power of 2 and define S as a carrier of Boolean algebra such that its zero and one are the r-bit binary codes 0...0 and 1...1 respectively and the r atoms are given by the "1-hot "codes 0...01, 0..10,...,10...0. We can represent an ADD with a Boolean function $f(x)$ of n variables as

$$f(x) : \{0,1\}^n \to S$$

This functions obeys the Boole's expansion theorem [3].

Theorem 3. *if $f : \{0,1\}^n \rightarrow \{0,1\}$ is a Boolean function, then $f(x_1, x_2, ...x_n) = x_1'.f(0, x_2, ..., x_n) + x_1.f(1, x_2, ..., x_n)$, for all $(x_1, ...x_n) \in \{0,1\}^n$. Now recursively application of Boole's expansion to f leads to the following minterm in the canonical form [14]: $f(x_1, ..., x_{x-1,x_n}) = f(0, .., 0, 0)x'...x_{n-1}'x_n + ... + f(1, ..., 1, 1)x_1, ..., x_{x-1}, x_n$. such that $f(0, ..0, 0), ..., f(1, ..., 1, 1) \in S$ and are termed as* discriminants *of the function f, and $x'...x_{n-1}'x_n, x'...x_{n-1}'x_n, .., x'...x_{n-1}'x_n$ are called the minterms [3].*

A complete enumeration of minterm expansion results in an ADD. Minterms with the same discriminants are grouped together during reordering. Hence, an ADD consists of a Boolean function, and the set of all such functions for a given carrier S. This forms a Boolean function algebra. Moreover, the values of the leaf nodes are members of the algebraic system S. Hence, we can have a set of operation on an ADD. These are Boolean operations and operations on matrices built on S. The operation on matrices is further compose of arithmetic and abstraction operations.

4.4 ADD Representation of Matrices

In this section, we want to recall how ADD can be used as a means to represent matrix. ADDs are very efficient data structures for the representation of graphs and matrices. We will illustrate this briefly with the following example: Let G be a simple weighted graph shown in Figure 4.9(a) whose adjacency matrix is shown in Figure 4.9(b). In this matrix the element 0 indicates that there is no connection between the two nodes. Figure 4.9(c) shows the corresponding ADD of G.

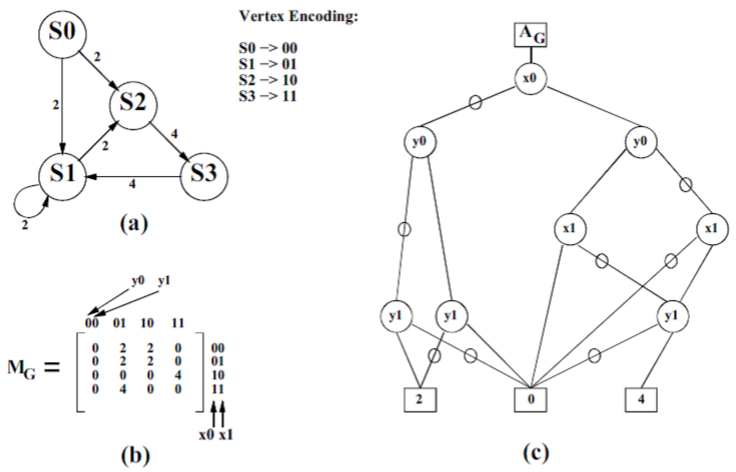


Figure 4.9: ADD Representation of Matrices [3]

In Figure 4.9(c) edges with an open circle on them denote the *else child* while those without open circles denote *then child.* S is given by the set $\{0, 2, 4\}$ and each path that corresponds to an edge that is not an element of G is mapped to the zero terminal node (0). Suppose that N is the number of vertices in the graph G, then there will be $2n$ encoding variables, such that $n = |x| = |y| = \lceil \log N \rceil$, where x and y are the row variable and column variable respectively. From the above example we have a 4×4 square matrix. This matrix has size $n = 4$ i.e. n^2. Therefore, in the case where n is not a power of 2, then, we must "pad" the rows and columns with dummy values (usually zero) so that the argumented matrix satisfies the power of 2 rule. In other words, we must ensure that the number of rows and columns is exactly a power of 2 otherwise, we cannot use ADD to represent the matrix [3].

Let x and y be the top variable and the next variable of the ADD structure respectively, then, we can partition the matrix above with respect to x into 2 rectangular sub-matrices. The upper part represents the *else-child* and the lower part represents the *then-child.* Further cofactoring in terms of x and y divides the 2 sub-matrices into 4 square sub-matrices. These 4 square matrices represents 4 *grand-children.* Repeated and continuous application of this process until all row and column variables have been cofactored will result in a 1×1 sub-matrices. This 1×1 sub-matrix is actually one of the constant terminal elements of the given ADD. Suppose that, the given ADD denotes an $N \times N$ matrix then, we have $n = \log(N)$ row variables and $n = \log(N)$ column variables [3]. This notion can be visualized in the following diagram:

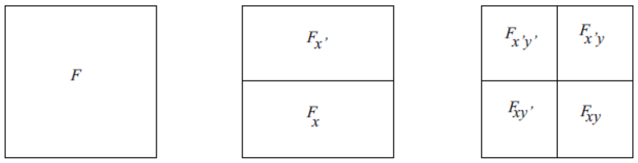


Figure 4.10: ADD Representation of Matrices [3]

4.4.1 Operations on ADDs

ADD can be manipulated using three sets of operations namely, Boolean, arithmetic and abstraction operations. These operations have made the implementation of symbolic algorithms for some problems possible. We shall give a brief description of arithmetic and abstraction operations since it is relevant to this thesis. For further readings on Boolean operations please refer to [3].

4.4.2 Arithmetic operations

As mentioned above, an `apply` operation in ADDs can similarly be used to accomplish several arithmetic operations. In most ADDs implementations, there is a generic operation which inherits the properties of the `apply` operations, this may be used to accomplish a large number of matrix operations. The following is the definition of an apply operation.

Definition 18. *Let f and g be generic ADDs and op be an operator such as +,-,*,/, min, max, etc. then an* `apply` *operation can be defined as* `Apply(f,g,op)= f` *such that f op g is defined as a function whose set of discriminants is obtained by applying op to corresponding pairs of discriminants of f and g. i.e.* $D=\{d_1^f \text{ op } d_1^g,\ldots,d_q^f \text{ op } d_q^g\}$ *where*

$$D^f = \{d_1^f, \ldots, d_q^f\} = \{f(0,\ldots,0,0), f(0,\ldots,0,1), \ldots, f(1,\ldots,1,1)\}$$

$$D^g = \{d_1^f, \ldots, d_q^g\} = \{g(0,\ldots,0,0), g(0,\ldots,0,1), \ldots, g(1,\ldots,1,1)\}$$

.

Example 18. *Suppose that we want to compute* `apply(f,g,+)` *for the following matrices. where f,g,+ are the parameters accepted by this* `apply` *operation. Then*

$$f = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \quad \textit{and} \quad g = \begin{pmatrix} 5 & 5 & 5 & 5 \\ 5 & 5 & 5 & 5 \\ 2 & 2 & 2 & 2 \\ 2 & 2 & 2 & 2 \end{pmatrix}$$

$\Rightarrow$

$$\texttt{apply(f,g,+)} = \begin{pmatrix} 6 & 6 & 5 & 5 \\ 6 & 6 & 5 & 5 \\ 2 & 2 & 3 & 3 \\ 2 & 2 & 3 & 3 \end{pmatrix}$$

4.4.3 Abstraction Operations

We mentioned abstraction operation for MDDs, now we want to see how it is implemented in ADDs. Abstraction operation provides generic operations that can be used to reduce the dimensionality of its argument function by using variable abstractions, for instance, if T is an $N \times M$ matrix such that the abstraction variable is the entire row of encoding variables, then the result produced after abstraction is a $1 \times M$ matrix. There is still loss of dimensionality even if the abstraction variable are not set to the entire row or column encoding variable set. Some abstraction operations include taking the sum, product, minimum, join of the rows of a matrix [3]. The following is a formal definition for an abstraction operation taken from [3].

Definition 19. *Let $f(u)$ be a function of u, and let x and y be the two subsets of u such that $x \sqcup y = u$. The generic abstraction of variables x from function f(u), denoted by $f_x^{op}(y) = \pi_x^{op} f(u)$ is defined as follows. For each minterm y of $f_x(y)$, we define the discriminant set D^y as the set of discriminants of all the minterms (x, y) of $f(x, y)$. Then, the discriminant of minterm y of $f_x(y)$ is defined as the result of applying the operation op uniformly, in right to left order, over the members of set D^y. i.e.,*

$$f_x^{op}(y) = \pi_x^{op} f(u) = (d_1 \ op \ (d_2 \ op ...(d_q - 1 \ op \ d_q))...).$$

It must be noted that for some operations the above definition depends on the order in which the operator *op* is applied. However, given an algebraic system which obeys the usual associativity, identity and closure laws, then the result of abstraction is independent of the order in which the operator *op* is applied. We want to illustrate this with the following example.

Example 19. *Give that*

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 0 & 0 & 1 & 2 \\ 2 & 2 & 2 & 2 \end{pmatrix}$$

We can compute the sum over the rows of A by using an abstraction operation. This is given by

$$\texttt{abstract(A,+)} = \begin{pmatrix} 8 & 10 & 13 & 16 \end{pmatrix}$$

Similarly, taking the minimum over the rows of A results in

$$\begin{pmatrix} 1 \\ 5 \\ 0 \\ 2 \end{pmatrix}$$

Please refer to [3] for algorithms used to implement the above operations.

4.5 RelMDD - Design Specification

This section will give a detailed documentation of the structure, implementation and technical detail of the RelMDD system. We will give the technical details of the system as well as platform and system requirements. We shall also look at the other packages needed in order to use RelMDD. We shall give an account of the system architecture, the software components, the user interface, and syntax of files containing a basis.

4.5.1 Overview of RelMDD

As mentioned above, RelMDD is a C-Library that implements arbitrary heterogeneous relation algebras using the matrix algebra approach represented by ROMDDs. RelMDD is a library written in the programming language C. It is a package that can be imported by other programs and/or languages such as Java and Haskell when programming or manipulating arbitrary relations.

The implementation is currently restricted to the basic operations of relation algebras, i.e., union, intersection, composition, converse, and complement as well as three special relations known as the identity relation, empty relation and universal relation. By design, the package is capable of manipulating relations from both classes of models, standard models and non-standard models of relation algebras. The main purpose of this project is to:

1. Implement heterogeneous relation algebras using matrix algebra.
2. Understand how relations represented as matrices could be implemented using MDDs.
3. Make available a library to aid researchers to manipulate relations in their own programs.

In RelMDD system, there are two sets of functions, namely, internal and external functions. The internal functions consist of functions that manipulate decision diagrams whiles the external functions does the manipulation of relations. Users of RelMDD will normally and frequently use the external functions to manipulate relations.

There are three major generic internal functions that are used to manipulate relations represented as decision diagrams. These functions are imported from CUDD [46]. They are `Cudd_addApply()`, `Cudd_addMonadicApply()`, `Cudd_addMatrixMult-iply()`. `Cudd_addApply()` is used for manipulating operations including meet and join, `Cudd_addMonadicApply()` is used to manipulate converse and complement where as `Cudd_addMatrixMultiply()` is used to compute the products of relations. Relation in RelMDD is a C structure consisting of a matrix (relation), list of source and target objects, the size of the matrix and the decision diagram representing this relation. RelMDD is basically a set of C files, each containing several functions packaged into a static library. Basically, the system reads relations from a file represented as matrices and list of objects, and represents them as decisions diagrams internally. It then does manipulations using decision diagrams and then output the result of the relations as a list. So the user does not interact with the decision diagrams since they represent relations internally. At input the matrices are stored in Harwell-Boeing Exchange Format.

4.5.2 Language and Operating System Platforms

We chose to implement RelMDD in C programing language (ANSI standard C) because we needed a programming language which is very fast in terms of execution of programs and C is one of the fastest so far, hence C. Moreover, most of the packages such as CUDD and libxml used in RelMDD are all implemented in C, hence it will be easier if we also implement RelMDD in C. RelMDD is currently targeted to Linux operating system platforms which made it easier to utilize other packages in our system.

4.5.3 RelMDD naming convention

In RelMDD, to name the package, files, variables, methods, and structures we have followed ANSI standard C. By convention, identifiers are mostly in lowercase letters.

4.6 Components and Technical Details of RelMDD

In this section we shall discuss the various components and the technical details of RelMDD. These components include structure of basis File, XML parsing, the Harwell-Boeing exchange format of a matrix, RelMDD operations, internal and external functions.

4.6.1 Structure of Basis File

As in ReAlM [54], basis for relation algebras in RelMDD is stored in XML format for flexibility and easy accessibility. Without a basis file, RelMDD cannot be used, since the result of operations depends on values provided by the basis file. If the program starts, the content of basis file are read into memory and then used for computations. The contents are stored in C structures using dynamic memory allocations. A basis file is associated with XML schema definitions (.xsd). The XML schema file is used to validate the contents of a basis file. This ensures that, data stored in the XML files are stored in a specific format to be recognized by the system. One should call the XML schema validation functions to validate the XML file before parsing it. RelMDD includes a schema file, which serves as the default schema file within the system.

The content of a basis file consist of several XML tags. The root element "RelationBasis" which is the starting point of the basis file, specifies the type of relations in its attribute name called "TypeOfRelation". `Example: <RelationBasis TypeOfRelation ="FiniteRelAlg">`. The next tag is the relation tag `<relation>... </relation>`. This tag contains all other tags within a basis and differentiates one basis from the other. So it is possible to have more than one basis in one file. The next tag is `<comment> ...</comment>`. This tag can be used to provide a brief description of the basis. Example: `<comment> this is a basis file for a sample relation algebra between two objects </comment>`. The tag label by "object" lists the set of objects under consideration. `Example: <objects>A , B </objects>.` 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="4"/>`. This tag indicates that there are four relations between the objects A and A. The identity tag is used to store the identity of an object. `Example: <identity object="A" relation="1"/>` indicates that, the identity relation of an object A is represented by value 1. The bottom tag is used to store the bottom relations between objects. An example is `<bottom source="A" target="A" relation="4"/>` indicates that bottom relation of an object A is represented by value 4. The top tag is used to store the top relations between objects. `Example: <top source="A" target="A" relation="0"/>` indicates that the top relation of an object A is represented by value 0.

The later half of the basis file defines operations between pair of objects for all possible combinations. Basis file only defines the standard operations. These operations are defined in union, intersection, composition_Union, composition_Intersection, transposition and complement tags. For example, union can be used in the following way to define a union operation between two objects A and B.

```
<union source="A" 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>
```

The union tag uses ";" as a delimiter among different entries in the operation table. The above union tag specifies sixteen different union operations. The first operation "0, 0, 0" implies that the union operation of "0" and "0" with "A" and "B" as the source and target objects respectively is "0". For a detailed structure of the basis and XML schema file we refer to the appendix.

4.6.2 XML Parsing

To manipulate relation using RelMDD, a basis file must be loaded into the system by calling the function `RelMDDParseXmlFile(const char *xmlFilename)` which will initialize all XML functions. However, `RelMDDValidateXmlFile(const char *xmlPath)` should be used to validate the document before parsing. RelMDD uses a hash map technique to map the relations to the objects. The system has several functions for processing XML content.

These functions are divided into two parts. The first one is used by the system internally and are located in a C file named `RelMDDXmlParser.c`. They are used to retrieve and store the content of the XML files. We adopted the usual technique in which the content of the XML file is traversed from the root node to the child node until we fetch the data we want. The parsing and processing of XML data was made easier by using a well known library called libxml2. See below for details on libxml2

[61]. The other set of functions are defined in the file `RelXmlRead.c` and are available for the users to use to process basis file content when using RelMDD. Table 4.4, Table 4.5 and Table 4.7 show various functions and structures in RelMDD.

RelMDDXmlParser.c:-Internal Functions
contains all the internal functions that are use to process XML file internally

xmlChar *relXmlGetComments(xmlDocPtr doc, xmlNodePtr cur);
void relXmlDisplayComments(xmlDocPtr doc, xmlNodePtr root);
xmlChar *relXmlGetListObject(xmlDocPtr doc, xmlNodePtr cur);
relationPtr relxmlReadrelations(xmlNode *root, xmlDocPtr doc);
char **relXmlGetSourceRelation(xmlChar *pt, int i);
char **relXmlGetTargetRelation(xmlChar *pt, int i);
char **relXmlGetNumberRelation(xmlChar *pt, int i);
identityPtr relxmlReadIdentity(xmlNode *root, xmlDocPtr doc);
char **relXmlGetObjectIdentity(xmlChar *pt, int i);
double *relXmlGetRelationIdentity (xmlChar *pt, int i);
topPtr relxmlReadTop(xmlNode *root, xmlDocPtr doc);
bottomPtr relxmlReadBottom(xmlNode *root, xmlDocPtr doc);
xmlChar *relXmlGetTopSource(xmlChar *pt, int i);
xmlChar *relXmlGetTopTarget(xmlChar *pt, int i);
xmlChar *relXmlGetTopRelation(xmlChar *pt, int i);
unionsPtr relxmlReadUnion(xmlNode *root, xmlDocPtr doc);
char **relXmlGetSource(xmlChar *pt, int i);
char **relXmlGetTarget(xmlChar *pt, int i);
intersectionPtr relxmlReadIntersection(
xmlNode *root, xmlDocPtr doc);
compoUnionPtr relxmlReadCompostion_Union(xmlNode *root, xmlDocPtr doc);
compoInterPtr relxmlReadCompostion_Inter(xmlNode *root, xmlDocPtr doc);
conversePtr relxmlReadTransposition(xmlNode *root, xmlDocPtr doc);
complementPtr relxmlReadComplement(xmlNode *root, xmlDocPtr doc);
complementPtr complIni();
intersectionPtr interIni();
unionsPtr unionIni();
xmlDocPtr getdoc();
compoUnionPtr compoUionIni();
conversePtr coverselIni();
compoInterPtr compoInterIni();
double *readString(unsigned char *s);

Table 4.4: RelMDDXmlParser.c

RelMDDXmlParser.c:-Structure Types
Structures used to store the content of the xml files
typedef struct relation{ int *number; char **source; char ** target; };
typedef struct identity{ char **object; double *rel ; int len; };
typedef struct bottom{ char **source; char ** target; double *rel; int len; };
typedef struct top{ char **source; char ** target; double *rel; int len; };
typedef struct unions{ double *content; char **source; char ** target; int len; int *len_Content; };
typedef struct intersection{ double *content; char **source; char ** target; int len; int *len_Content; };
typedef struct compoUnion{ double *content; char **source; char **target; int len; int *len_Content; };
typedef struct compoInter{ double *content; char **source; char **target; int len; int *len_Content; };
typedef struct converse{ double *content; char **source; char **target; int len; int *len_Content; };
typedef struct complement{ double *content; char **source; char **target; int len; int *len_Content; };

Table 4.5: RelMDDXmlParser.c

RelMDDXmlReader.c:-External Functions
Contains all the external functions that the user can use to process XML files

static xmlDocPtr xmlDocument;
extern int RelMDDValidateXmlFile(const char *xmlPath);
extern int RelMDDParseXmlFile(const char *xmlFilename);
extern unionsPtr RelXmlGetUnion();
extern intersectionPtr RelXmlGetIntersection();
extern complementPtr RelXmlGetComplement();
extern compoUnionPtr RelXmlGetComposition_Union();
extern compoInterPtr RelXmlGetComposition_Intersection();
extern conversePtr RelXmlGetConverse();
extern topPtr RelXmlGetTop();
extern bottomPtr RelXmlGetBottom();
extern identityPtr RelXmlGetIdentity();
extern relationPtr RelXmlGetRelation();
extern void relXmlDisplayComments();
extern char ** relXmlGetObjects();

Table 4.6: RelMDDXmlReader.c

4.6.3 lixml2

In parsing XML files, we used a C parser and toolkit called Libxml2 developed for the Gnome project. It can also be used outside the Gnome platform. This software is free and available under the MIT License. Libxml2 includes complete XPath, XPointer and XInclude implementations. We used several functions from this library in our work which made it very easy for us to process XML files. For detailed descriptions of this package please refer to [61].

4.6.4 CUDD

CUDD (Colorado University Decision Diagram) is a package that provides functions to manipulate Binary Decision Diagrams (BDDs), Algebraic Decision Diagrams (ADDs), and Zero-suppressed Binary Decision Diagrams (ZDDs), CUDD is written in C by Fabio Somenzi at the Department of Electrical and Computer Engineering, University of Colorado at Boulder. RelMDD is built on CUDD version 2.4.2. It provides a large set of operations on BDDs, ADDs, and ZDDs, and a large assortment of variable reordering methods. In fact it is one of the best and well known packages that implements decision diagrams more efficiently by incorporating several optimization techniques. For detailed overview of this system please refer to [46, 13].

4.6.5 RelMDD Operations and Special Relations

Internally, RelMDD uses three generic functions imported from the CUDD package for manipulation of decision diagrams. `Cudd_addApply()` accepts a binary operator such as meet or join with two ADDs representing two matrices. Hence, we used this function to tackle the problem of meet and join. `Cudd_addMonadicApply()` accepts a unitary operator and one ADD representing a matrix. We used this function to implement operations such as the converse, and the complement of relations. However, in the case of converse we had to swap variables of the ADD before using `addMonadicApply()`. Lastly, we used `Cudd_addMatrixMultiply()` to accomplish composition by modifying it slightly to suite our purpose. In the following paragraphs, we will describe various functions that can be used in RelMDD to compute relations. Table 4.7 list the functions in RelMDD for manipulating relations. For detailed description of how to use the package please refer to the user manual accompanying the RelMDD package.

Union: `RelMDDUnionPtr RelMDDUnionOperation(...);` The function above is used to for calculating the union of two relations. It accepts six parameters as input. The first two parameters are the size of rows and columns of the matrices respectively. The next two parameters are the list of source and target objects of both matrices respectively. It should be noted that, to compute the union of two relation, the source and target objects of the both matrices are the same. This function when called returns a structure type containing the list of source and target objects, and the relations.

Intersection: `RelMDDIntstersectionPtr RelMDDIntersectionOperation(...);` The function above is used to find the meet of two relations. It behaves similarly to the join operation described above.

Compostion: To find the composition of two matrices, we use the function `RelMDDC-ompostionPtr RelMDDCompositionOperation(...);`. This function takes the size of the rows and columns of the first matrix as the first two parameters and then the size of the columns of the second matrix, followed by the list of source and target objects of the first matrix and then the target objects of the second matrix. The last two parameters are the two matrices that we want to find their product. The function returns details of the result of computations in a C structure. Note that, the source objects of the first matrix is always the same as the target objects of the second matrix.

Converse: `RelMDDConversePtr RelMDDConverse_Operation(...);` is use to accomplish the converse operation. It accepts parameters similarly to the union and intersection operations described above except that it takes only one matrix.

Complement: `RelMDDComplementPtr RelMDDComplement_Operation(...);` is used for the computation of the complement of a relation. It behaves similarly to the converse relation describe above.

Identity: To find the identity element of a given list of objects we use the function `RelMDDIdentityPtr RelMDDIdentityRelation(...);` The first two parameters are the of size of the rows and columns of the matrix we want to find its identity. This is followed by the list of objects. Here the source and target objects are the same. This function returns a structure made up of the identity relation and the list of objects.

Zero Element: `RelMDDBottomPrt RelMDDBottomRelation(...);` is used for the derivation of the zero element of a given set of objects. As usual the first two parameters are the size of the rows and columns of the list of source and target objects respectively. It returns a structure that consist of the bottom element and the list of objects.

Universal Relation: `RelMDDTopPtr RelMDDTopRelation(...);` It behaves similarly to the zero element function described above.

RelMDD.c: Internal and external functions
Contains functions for manipulating relations

extern DdNode *Transpose(DdManager * dd, DdNode * f);
extern CUDD_VALUE_TYPE transposeRelation(CUDD_VALUE_TYPE F);
extern DdNode *RelMDD_SwapVariables(DdManager * dd , int roww, int collh);
extern DdNode *Complement(DdManager * dd, DdNode * f);
extern CUDD_VALUE_TYPE complementRelation(CUDD_VALUE_TYPE F);
extern void RelMDDComplement_Operation(int tra_rn, int tra_col, char **source_objects, char **target_objects, double *matrimx);
extern void RelMDDConverse_Operation(int tra_rn, int tra_col, char **source_objects, char **target_objects, double *matrimx) ;
extern fileInforPtr RelMdd_ReadFile(char *filename);
extern double setSequence(double value);
extern DdNode *RelMDD_RenameRelation(DdManager * dd, int roww ,int collh);
extern CUDD_VALUE_TYPE complementRelation(CUDD_VALUE_TYPE F);
extern int *RelMDD_ListSequence(DdNode *E, int roww, int coll);
static FILE *open_file (char *filename, const char *mode);
extern DdNode *Cudd_addUnion_Meet(DdManager * dd, DdNode ** f,DdNode ** g);
extern DdNode *Cudd_addIntersection(DdManager * dd,DdNode ** f,DdNode ** g);
extern RelMDDIntersectionPtr RelMDDIntersectionOperation(int no_rows, int no_cols,char **source_objects , char **target_objects, double *matrix1, double *matrix2);
extern RelMDDUnionPtr RelMDDUnionOperation(int no_rows, int no_cols, char **source_objects, char **target_objects, double *matrix1, double *matrix2);
extern DdNode * node_Complement(DdNode * mat);
extern CUDD_VALUE_TYPE identityRelation(CUDD_VALUE_TYPE F);
extern RellMDDIdentityPtr RelMDDIdentityRelation(int row, int col, char **objects);
extern RellMDDBottomPtr RelMDDBottomRelation(int row, int col, char **source_objects , char **target_objects);
extern CUDD_VALUE_TYPE BottomRelation(CUDD_VALUE_TYPE F);
extern RellMDDTopPtr RelMDDTopRelation(int row, int col, char **source_objects , char **target_objects);
extern CUDD_VALUE_TYPE topRelation(CUDD_VALUE_TYPE F);
extern int RelMdd_CreateRelation(...);
extern RelMDDCompostionPtr RelMDDCompositionOperation(int no_rows, int no_cols, char **sourceA_objects,char **targetA_objects,char **targetB_objects, double *matrix1, double *matrix2);

Table 4.7: RelMDD.c

4.6.6 Loading a Matrix

Since relations are represented as matrices, RelMDD has a special format to read in matrices from files. This format adhere to the format of Harwell-Boeing benchmark suite. This format specifies how matrices should be stored and read from a file. In our case the first item on the file specifies the number of rows and column of the matrix. The second and third items on the file specify the list of source and target objects respectively. This is followed by the matrix in which the row and column indices of each element of the matrix is also specified. Refer to the appendix for sample file.

Chapter 5

Conclusion and Future Work

In this thesis, we reviewed various properties of heterogeneous relation algebras. We went ahead to introduce a system called RelView which only works in the standard model of relations algebra. This is because the underlying structure of this system is based on Boolean matrix which restricts its ability to be used outside the class of standard models.

Now, it has been proved in [49] that for 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 $\mathcal{R}$. Based on the above theorem, we developed a system called RelMDD which works within both the standard and the non-standard models of relation algebras using the matrix approach. In order to do this, we used an advanced and more efficient data structure called multiple valued decision diagrams (MDDs) which is similar to the data structure used by RelView system in the Boolean matrix case.

This implementation combines two major advantages over a regular array implementation of matrices. Both RelView and RelMDD systems have proven that an implementation of relations using decision diagrams is of great benefit. The RelMDD system is a library written in C which can be imported by other languages such as Java or Haskell.

The package implements all standard operations on relations. A future project will add further operations such as sums and splittings. The latter will then also allow to compute relational powers and so-called weak relational products [52]. Another project will be a suitable module for the programming language Haskell that makes the RelMDD package available in this language. An advanced project will be to extend the current system into a programming language which will allow programming using relations. One can also work on integrating the package into the RelView system.

Appendix A

Appendix

A.1 XML schema file

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault =
"qualified">

<!-- definition of simple elements -->
<xs:element name="comment" type="xs:string"/>
<xs:element name="objects"/>


<!-- definition of attributes -->
<xs:attribute name="TypeOfRelation" type="xs:string"/>
<xs:attribute name="source"  type="xs:NCName"/>
<xs:attribute name="target"  type="xs:NCName"/>
<xs:attribute name="number"  type ="xs:integer"/>
<xs:attribute name="object"  type="xs:NCName"/>
<xs:attribute name="relation"  type="xs:integer"/>

<!-- definition of complex elements -->
<xs:element name="relations">
  <xs:complexType mixed="true">
  <xs:attribute ref="source"  use="required"/>
  <xs:attribute ref="target"  use="required"/>
  <xs:attribute ref="number"  use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="identity">
  <xs:complexType mixed="true">
  <xs:attribute ref="object"  use="required"/>
  <xs:attribute ref="relation"  use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="bottom">
  <xs:complexType mixed="true">
  <xs:attribute ref="source"  use="required"/>
  <xs:attribute ref="target"  use="required"/>
  <xs:attribute ref="relation"  use="required"/>
  </xs:complexType>
</xs:element>

 <xs:element name="top">
  <xs:complexType mixed="true">
  <xs:attribute ref="source"  use="required"/>
  <xs:attribute ref="target"  use="required"/>
  <xs:attribute ref="relation"  use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="union">
  <xs:complexType mixed="true">
  <xs:attribute ref="source"  use="required"/>
  <xs:attribute ref="target"  use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="intersection">
  <xs:complexType mixed="true">
     <xs:attribute ref="source"  use="required"/>
     <xs:attribute ref="target"  use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="complement">
  <xs:complexType mixed="true">
     <xs:attribute ref="source"  use="required"/>
     <xs:attribute ref="target"  use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="composition_Union">
  <xs:complexType mixed="true">
  <xs:attribute ref="source"  use="required"/>
  <xs:attribute ref="target"  use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="composition_Intersection">
  <xs:complexType mixed="true">
  <xs:attribute ref="source"  use="required"/>
  <xs:attribute ref="target"  use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="transpose">
  <xs:complexType mixed="true">
     <xs:attribute ref="source"  use="required"/>
     <xs:attribute ref="target"  use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="relation">
  <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_Union" maxOccurs="unbounded"/>
        <xs:element ref="composition_Intersection" maxOccurs="unbounded"/>
      <xs:element ref="transpose" maxOccurs="unbounded"/>
       <xs:element ref="complement" maxOccurs="unbounded"/>
    </xs:sequence>

  </xs:complexType>
</xs:element>

<xs:element name="RelationBasis" >
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="relation" maxOccurs="unbounded"/>
      </xs:sequence>
   <xs:attribute ref="TypeOfRelation" use="required"/>
     </xs:complexType>
</xs:element>


</xs:schema>
```

A.2 Sample Basis File

```
<?xml version="1.0" encoding="utf-8"?>
<FiniteRelAlg name ="two_object_basis">

<comment>
Our very first example.
</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_Union source="A"  target="B">
        0,0,0;
        0,1,0;
        1,0,0;
        1,1,1
</composition_Union >

<composition_Union  source="B" target="A">
        0,0,0;
        0,1,0;
        1,0,0;
        1,1,1
</composition_Union >


<composition_Union  source="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_Union >

<composition_Union  source="B" target="B">
        0,0,0;
        0,1,0;
        1,0,0;
        1,1,1
</composition_Union >


<composition_Intersection source="A"  target="B">
        0,0,0;
        0,1,0;
        1,0,0;
        1,1,1
</composition_Intersection>

<composition_Intersection source="B" target="A">
        0,0,0;
        0,1,0;
        1,0,0;
        1,1,1
</composition_Intersection>


<composition_Intersection source="A" " target="A">
        0,0,0;
        0,1,0;
        1,0,0;
        1,1,1
</composition_Intersection>

<composition_Intersection source="B" target="B">
       0,0,0;
       0,1,0;
       0,2,0;
       0,3,0;

       1,0,0;
       1,1,1;
       1,2,2;
       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_Intersection>






<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>
```

A.3 Sample Matrix in a specified file format

```
4 4
[A,B,B,B]
[A,B,A,B]
0 0 1
0 1 0
0 2 1
0 3 1
1 0 2
1 1 0
1 2 0
1 3 2
2 0 0
2 1 0
2 2 3
2 3 3
3 0 0
3 1 1
3 2 0
3 3 1
```

Bibliography

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

[2] Atampore F., Winter M.: Relation Algebras, Matrices, and Multi-Valued Decision Diagrams. Brock University, Dept. of Computer Science Report CS-12-03 (2012). `http://www.cosc.brocku.ca/sites/all/files/downloads/research/cs1203.pdf`

[3] Bahar I. R., Frohm E. A., Gaona C. M., Hachtel G. D., Macii E., Pardo A., Somenzi F.: Algebraic Decision Diagrams and their Applications. In Proceedings of the International Conference on Computer-Aided Design, 188-191, Santa Clara, CA, (1993).

[4] Behnke R., Berghammer R., Schneider P.: Machine Support of relational Computations: The Kiel Relview System. Christian-Albrechts-Universität Kiel, Germany

[5] 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).

[6] Berghammer R., Hoffmann T.: Modeling Sequences within the RelView System. Christian-Albrechts-Universitat Kiel, Germany, Journal of Universal Computer Science, vol. 7, no. 2, 107-123 (2001).

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

[8] Berghammer R., Neumann F.: RELVIEW- An OBDD-Based Computer Algebra System for Relations. CASC 2005, LNCS 3718, pp.40-50, Springer-Verlag Berlin Heidelberg (2005).

[9] Berghammer R., Schmidt G., Winter M.: RelView and RATH - Two System for Dealing with Relations. Institut für Informatik und parktische Mathematik Universät zi Kiel, Fakultät für Informatik. München, Germany.

[10] Berghammer R., Fronk A.: Considering Design Tasks in OO-Software Engineering using Relations and Relation-based Tools. Journal on Relational Methods in Computer Science, Vol. 1, 73 - 92 (2004).

[11] Berghammer R.: Computation of Cut Completions and Concept Lattices Using Relational Algebra and RelView. Journal on Relational Methods in Computer Science, Vol. 1, 50 - 72 (2004).

[12] Boole G.: The Mathematical analysis of Logic, Macmillan, 1847, Reprinted by B. Blackwell, Oxford, Uk, (1951)

[13] Brace K. S., Rudell L. R., Bryant E. R.: Efficient Implementation of a BDD Package 27 ACM/IEEE Design Automation Conference (1990).

[14] Brown F. M.: Boolean Reasoning: The Logic of Boolean equations. Kluwer Academic Publishers, (1990).

[15] Bryant R. E.: Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers, C-35(8),677-691, August 1986.

[16] Desharnais J.: Basic Relation Algebra. Universite Laval, Dept. d'informatique et de genie logiciel, Quebec, Canada.

[17] Devereux B., Chechik M.: Edge-Shifted Decision Diagrams for Multiple-Valued Logic. Department of Computer Science, University of Toronto, (2001).

[18] Dwyer B.: LIBRA: A Lazy Interpreter of Binary Relational Algebra. Computer Science Technical Report 95-10, University of Adelaide.

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

[20] Furusawa H.: Algebraic Formalisations of Fuzzy Relations and their Representation Theorems. PhD-Thesis, Department of Informatics, Kyushu University, Japan (1998).

[21] Glanfield J.: Relaps: A Proof Assistant for Relational Categories. "`http://www.joelglanfield.com/relaps`."

[22] Gonzalia C.: Heterogeneous relation algebra as a kind of allegory. Chalmers University of Technology, Göteborg, Sweden (Dec 1998).

[23] Graham M.H.: Between Functions and Relations in Calculating Programs. Ph.D thesis, University of Glasgow (1992).

[24] Hattensperger C.: Rechnergestütztes Beweisen in heterogenen Relationenalgebren. PhD thesis, Fakultät für Informatik, Universität der Bundeswehr MÍunchen, 1997. Dissertationsverlag NG Kopierladen, München, ISBN 3-928536-99-0.

[25] Hattensperger C., Berghammer R., Schmidt G.: Ralf - A relation-algebraic formula manipulation system and proof checker (Notes to a system demonstration). In Nivat et al. [21], pages 405-406. Proc. 3rd Int'l Conf. Algebraic Methodology and Software Technology (AMAST '93), University of Twente, Enschede, The Netherlands, Jun 21-25, 1993

[26] Jipsen P.: Foundations of relations and Kleene algebra. Chapman University. 1-84, September 4, 2006

[27] Kahl W., Schmidt G.: Exploring (Finite) Relation Algebras Using Tools Written in Haskell, Technical Report Nr. 2000-02, Fakultät für Informatik Universität der Bundeswehr München.

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

[29] Maddux R. D.: Studies In Logic And The Foundations Of Mathematics. Volume 150, Department of Mathematics Iowa State University, USA, (2006).

[30] McKenzie R.: Representations of integral relation algebras. Michigan Mathematical Journal 17, 279-287 (1970).

[31] Meertens L.: Category theory for program construction by calculation. Utrecht University, CWI, Amsterdam and Dept. of Computer Science 1-122, (September 5, 1995).

[32] Miller D. M., Drechsler R.: Implementing a Multiple-Valued Decision Diagram Package. Proceedings of the 28th IEEE International Symposium on Multiple-Valued Logic(ISMVL-98) (1998).

[33] Miller D. M., Radomir S. S.: A Heterogeneous Decision Diagram Package. Dept. of Computer Science, Faculty of Engineering University of Victoria Victoria, BC, Canada V8W 3P6, Dept. of Computer Science, Faculty of Electronics University of Ni's, 18000 N's, Serbia

[34] Nagayama S., Sasao T.: Compact Representations of Logic Functions using Heterogeneous MDDs. Proceedings of the 33rd International Symposium on Multiple-Valued Logic(ISML'03,) (2003 IEEE).

[35] Oheimb D. V., Gritzner T. F.: RALL: Machine supported proofs for relation algebra. Fakultät für informatic, Technische München Germany and University der Bundeswehr, Munchen & Neubiberg Germany (1997).

[36] Olivier J. P., Serrato D.: Categories de Dedekind. Morphismes dans les Categories de Schröder. C.R. Acad. Sci. Paris 290, 939-941 (1980).

[37] Olivier J. P., Serrato D.: Squares and Rectangles in Relational Categories - Three Cases: Semilattice, Distributive lattice and Boolean Non-unitary. Fuzzy sets and systems 72, 167-178 (1995).

[38] Rudell R.: Dynamic variable ordering for ordered binary decision diagrams, Proc. IEEE/ACM ICCAD, 43-47, 1993.

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

[40] 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).

[41] Schmidt G.: Decomposing Relations - Data Analysis Techniques for Boolean Matrices. Technical Report 2002-09, Fakultät für Informatik, Universität der Bundeswehr München, 2002. `http://ist.unibw-muenchen.de/People/schmidt/DecompoHomePage.html,79pages.`

[42] Schmidt G.: A Proposal for a Multilevel Relational Reference Language. Journal on Relational Methods in Computer Science, Vol. 1, 2004, pp. 314 - 338 `http://www.cosc.brocku.ca/Faculty/Winter/JoRMiCS/Vol1/PDF/v1n13.pdf`

[43] Schmidt G.: Relational Language-Revised Version 2 Institute for Software Technology Department of Computing Science Federal Armed Forces University Munich, Neubiberg, Germany (2004) `http://mucob.dyndns.org:30531/~gs/Papers/RelLangTex20041025.pdf`

[44] Siddavaatam P., Winter M.: Refining relational algebras for qualitative spatial reasoning, Ph.D. Student Programme of 12th International Conference on Relational and Algebraic Methods in Computer Science (RAMiCS 12), (2011).

[45] Siddavaatam P.: Generating Relation Algebras for Qualitative Spatial Reasoning. Master's thesis, Brock University (2011).

[46] Somenzi F.: CUDD: CU decision diagram package; Release 2.5.0. "`http://vlsi.colorado.edu/~fabio/CUDD/node1.html`" (2012).

[47] __,Splitting atoms in relational algebras, Relational and Algebraic Methods in Computer Science (Rotterdam) (de Swart, H. , ed.), LNCS 6663, 331-346, (2011).

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

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

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

[51] Winter M.: Goguen Categories - A Categorical Approach to L-fuzzy Relations. Trends in Logic 25 (2007).

[52] Winter M.: Weak relational products. Relations and Kleene Algebra in Computer Science (RelMiCS'06/AKA'06), LNCS 4136, 417-431 (2006).

[53] Winter M.: Goguen Categories Department of Computer Science, Brock University, St. Catharines, Ontario, Canada. Journal on Relational Methods in Computer Science, Vol. 1,339 - 357 (2004).

[54] Zafor A.: Realm - A system to manipulate relations. Master's thesis, Brock University (2009).

[55] Zhang S.: Generating Finite Integral Relation Algebras. Masters's thesis, Brock University (2010).

[56] “`www.cs.umbc.edu/~artola/fall02/Relations.ppt-UnitedStates`”

[57] “`http://www2.cs.unibw.de/Proj/relmics/html/`”

[58] “`http://www.informatik.uni-kiel.de/~progsys/relview/screenshots.html`”

[59] “`http://cs.adelaide.edu.au/~dwyer/TR95-10_TOC.html`”

[60] “`http://relmics.mcmaster.ca/html/tools.html`”

[61] “`http://www.xmlsoft.org/`”

[62] “`http://www.ift.ulaval.ca/~desharnais/Recherche/Tutoriels/TutorielRelMiCS10.pdf`”