Generating Finite Integral Relation Algebras

by

© Si Zhang

Submitted in partial fulfilment

of the requirements for the degree of

Master of Science

Department of Computer Science

Brock University

July 2010

St. Catharines, Ontario

Abstract

Relation algebras and categories of relations in particular have proven to be extremely useful as a fundamental tool in mathematics and computer science. Since relation algebras are Boolean algebras with some well-behaved operations, every such algebra provides an atom structure, i.e., a relational structure on its set of atoms. In the case of complete and atomic structure (e.g. finite algebras), the original algebra can be recovered from its atom structure by using the complex algebra construction. This gives a representation of relation algebras as the complex algebra of a certain relational structure. This property is of particular interest because storing the atom structure requires less space than the entire algebra.

In this thesis I want to introduce and implement three structures representing atom structures of integral heterogeneous relation algebras, i.e., categorical versions of relation algebras. The first structure will simply embed a homogeneous atom structure of a relation algebra into the heterogeneous context. The second structure is obtained by splitting all symmetric idempotent relations. This new algebra is in almost all cases an heterogeneous structure having more objects than the original one. Finally, I will define two different union operations to combine two algebras into a single one.

Acknowledgements

I treasure the memory of the many discussion sessions I have had with Professor Michael Winter who has not only directed me to this area of research but also provided me edifying experience. I would not have come this far without his unfailing encouragement and guidance. He always makes time and efforts when I need his help. I thank him for his patience, his constant support and assistance for the duration of my thesis.

Besides my supervisor, I would like to thank the rest of my supervisory committee: Dr. S. Houghten and Dr. B. Ross, who reviewed my work, asked me good questions, and gave insightful comments. I thank Brock University for the financial support during the course of this work.

Let me also say thank you to my dear friends: Yifeng Li, Jing Sun, Liang He, Fan Zhang, Xiang Yin, Wei Wei, Huan Zhang, Chen Han and Yusi Cheng. Without their help, I could not finish my studies.

Last, but certainly not least, I thank my parents, Zhigang Zhang, and Huanzhen Dong, for giving me life in the first place, and for their enduring support. I would like to thank my wife, Yutong Zhou for all the love and support she has given me during my years of graduate studies at Brock University.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Relation algebras have proven to be extremely useful as a fundamental tool in mathematics and computer science, e.g., program semantics, specification, derivation, set theory, and logic (see for example [2, 10, 11, 13]).

The major advantage of relation algebras is that any property formulated in first-order logic with at most 3 variables can be translated into an equation in the language of relation algebras. Requiring some extra structure such as relational products this translation can be extended to formulas with arbitrarily many variables. Since the theory of relation algebras is equational, proofs of arbitrary properties become algebraic manipulations of equations similar to the kind of manipulation known from high school. For this reason, relation algebras are considered to be part of algebraic logic. However, since proving theorems can be difficult anyone facing such a task appreciates any tool assisting him/her. In particular, having a vast amount of examples,

e.g., a collection of finite relation algebras, on which to test properties and to find counterexamples is extremely useful.

Since relation algebras are Boolean algebras with some well-behaved operations, every such algebra provides an **atom structure**. If the given relation algebra is complete and atomic, then the original algebra can be recovered from its atom structure by using the **complex algebra** construction. This gives a representation of relation algebras as the complex algebra of a certain relational structure (on its set of atoms [9]). This property is of particular interest because storing the atom structure requires exponentially less space than the entire algebra. An example of this reduction is given in Table 1.1. The first column represents the number of atoms of the relation algebra. The second column represents the size of the composition table, which contains the largest portion of data needed to store relation algebra. Finally, the last column represents the size of the composition table reduced to atoms.

Table 1.1: Space Compare: Algebra vs. Atom

Atoms in RA	The composition table size	Reduced to atoms size
1	4	1
2	16	4
3	64	9
4	256	16
5	1024	25
$\vdots$	$\vdots$	$\vdots$
n	$2^n \times 2^n$	$n \times n$

In addition to reduction above it can be shown that considering so-called integral algebras is sufficient. In fact, every relation algebra is equivalent to a matrix algebra over a basis built from integral algebras [14, 15].

1.2   Goals

In this thesis I want to introduce and implement three structures representing atom structures of integral heterogeneous relation algebras, i.e., categorical versions of relation algebras. As already mentioned above those algebras are sufficient to generate all finite heterogeneous relation algebras. The first structure will simply embed a homogeneous atom structure of a relation algebra into the heterogeneous context. The second structure is obtained by splitting all symmetric idempotent relations. This new algebra is in almost all cases an heterogeneous structure having more objects than the original one. Finally, I will define two different union operations to combine two algebras into a single one.

1.3   Outline

Now, we outline the contents of the subsequent chapters.

Chapter 2: presents the necessary background required for the thesis, including an introduction of heterogeneous relation algebras, and an overview of homogeneous relation algebras.

Chapter 3: describes three methods of generating finite integral heterogeneous relation algebras.

Chapter 4: presents a system which has implemented those three methods designed in the previous chapter.

Chapter 5: offers a detailed user manual for the system.

Chapter 6: contains some concluding remarks, and some ideas about future work.

Parts of the thesis have been published in a conference [17]. In this paper the three structures for generating finite integral relation algebras as described in Chapter 3 are introduced. In addition the implementation which is presented in Chapter 4.3 was outlined.

Chapter 2

Background

Relation algebras have been studied since the latter half of the nineteenth century. De Morgan (1864) and Peirce (1870) first founded the calculus of binary relations. Schröder (1890 - 1905) carried on with their work. More than forty years later, Tarski (1941) gave a first formal introduction to relation algebras. For more information, see [9].

Originally, the theory of relations is homogeneous, i.e., every relation is considered to be defined on a single and fixed universe U. In recent years a variant of this theory based on category theory was developed in order to include relations with different source and target. These approaches can be called *heterogenous*. Unlike the homogeneous relation algebras, relations act between different sets in heterogeneous relation algebras, e.g., the relation 'is_married_to' which is a relation between the two different sets of men and women. A classical (homogeneous) relation algebra can be seen as a heterogeneous relation algebra with just one object (or single universe).

We have attempted to keep our notation and conventions in agreement with those

of the closely related subject of relation algebras, especially as presented in Maddux's *Relation Algebras* [9]. We assume that the reader is familiar with the basic notions from set theory and the theory of lattices.

In this chapter, first we will introduce the basic concepts of Boolean algebras and category theory. Secondly, we will introduce heterogeneous relation algebras. Last, we will review some fundamentals on (homogeneous) relation algebras.

2.1 Boolean Algebras

Since a relation algebra is based on a Boolean algebra, it is useful to review some basic properties of Boolean algebras as well. For further reference see [3, 7, 9].

Definition 1. *An algebraic structure of the form* $\mathfrak{B} = \langle B, \sqcup, \bar{\ } \rangle$ *where* $\sqcup$ *is a binary and* $\bar{\ }$ *is a unary operation is called a Boolean algebra if and only if it satisfies the following three axioms (for all $x, y, z \in B$):*

1. $x \sqcup y = y \sqcup x$.

2. $x \sqcup (y \sqcup z) = (x \sqcup y) \sqcup z$.

3. $\overline{\overline{x} \sqcup \overline{y}} \sqcup \overline{\overline{x} \sqcup y} = x$.

The operation $\sqcup$ *is called union, and the operation* $\bar{\ }$ *is called complementation.*

In addition to the union ($\sqcup$) and complementation ($\bar{\ }$) operation of a Boolean algebra one may define intersection ($\sqcap$) by $x \sqcap y = \overline{\overline{x} \sqcup \overline{y}}$. Furthermore, there are two constants, the zero or empty element 0 and the one or universal element 1, which are defined by $0 = x \sqcap \overline{x}$ and $1 = x \sqcup \overline{x}$ for an arbitrary $x \in B$, respectively. A

Boolean algebra is also equipped with a partial order defined by: $x \sqsubseteq y$ if and only if $x \sqcup y = y$. Notice that the property $x \sqcup y = y$ is equivalent to $x \sqcap y = x$. The element $x \sqcup y$ is the greatest lower bound of x and y with respect to the order $\sqsubseteq$, i.e., $x \sqcup y \sqsubseteq x$ and $x \sqcup y \sqsubseteq y$ and whenever $z \sqsubseteq x$ and $z \sqsubseteq y$, then $x \sqcup y \sqsubseteq z$. Analogously, $x \sqcap y$ is the least upper bound of x and y. The concept of a greatest lower and a least upper bound extends to any finite subset of B by iterating the operations $\sqcup$ and $\sqcap$, respectively. A Boolean algebra $\mathfrak{B}$ is called complete iff[1] every subset of B has a greatest lower and a least upper bound. Not every Boolean algebra is complete. For an example we refer to [3, 7].

Any Boolean algebra is a distributive lattice, i.e., the equations $x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$ and $x \sqcup (y \sqcap z) = (x \sqcup y) \sqcap (x \sqcup z)$ hold for all x, y and z.

The powerset $\mathcal{P}(A)$ of a set A is an example of a Boolean algebra. The operation $(\sqcup)$ and the operation $(^-)$ are given by union and complementation (relative to the set A) of sets.

Definition 2. *An element x of $\mathfrak{B}$ is called an atom of $\mathfrak{B}$ iff $x \neq 0$ and there is no nonzero element of $\mathfrak{B}$ is below x. We denote by $At\mathfrak{B}$ the set of atoms of $\mathfrak{B}$, i.e.,*
$$At(\mathfrak{B}) = \{x \mid x \neq 0, x \in \mathfrak{B} \text{ and } \forall y (y \in \mathfrak{B}, y \sqsubseteq x \Rightarrow y = x \text{ or } y = 0)\}.$$

For example, when B is the Boolean algebra with two elements 0 and 1 with $0 \sqsubseteq 1$, then 1 is the only atom in the algebra. The singleton sets are the atoms in the power set. There are also Boolean algebras that contain no atoms but we will not consider these algebras throughout this thesis.

Definition 3. *A Boolean algebra is atomic if for every element $x \neq 0$ there is an*

---

[1]Throughout this thesis we will use 'iff' as a shorthand for 'if and only if'.

atom a with a $\sqsubseteq x$.

In particular every finite Boolean algebra is atomic. As already mentioned above there are also non-atomic Boolean algebras but since we are mainly interested in finite algebras we will only consider atomic ones.

In order to compare two different Boolean algebras the notion of a homomorphism, i.e., a function preserving the structure of Boolean algebras, is of particular interest.

Definition 4. *A homomorphism f between two Boolean algebras $\mathfrak{B}_1$ and $\mathfrak{B}_2$ is a function which preserves the Boolean operations, i.e., for all $x, y \in \mathfrak{B}_1$ we have*

1. $f(x \sqcup y) = f(x) \sqcup f(y)$,

2. $f(\overline{x}) = \overline{f(x)}$.

If it is possible to obtain a Boolean algebra from a given one by simply renaming the elements, we consider the two algebras as being essentially the same, i.e., they are isomorphic.

Definition 5. *A homomorphism $f : \mathfrak{B}_1 \to \mathfrak{B}_2$ is called an isomorphism iff it is a bijective homomorphism. $\mathfrak{B}_1$ and $\mathfrak{B}_2$ are isomorphic iff there is an isomorphism from $\mathfrak{B}_1$ to $\mathfrak{B}_2$.*

For example, if $A_1 = \{x_1, \ldots, x_n\}$, $A_2 = \{y_1, \ldots, y_n\}$ are two sets, then there is a bijective function $j : A_1 \to A_2$ defined by $j(x_i) = y_i$ for all $1 \le i \le n$. This function j can be extended to an isomorphism $k : \mathcal{P}(A_1) \to \mathcal{P}(A_2)$ between the Boolean algebras $\mathcal{P}(A_1)$ and $\mathcal{P}(A_2)$ by $k(B) = \{j(x) \mid x \in B\}$.

2.2 Category Theory

Later in Chapter 2.3, we will introduce a variant form of relation algebras whose definition is based on category theory. In this variant of the theory relations are heterogeneous, i.e., they may have different source and target such as the relation 'is_married_to' already mentioned at the beginning of this chapter.

Definition 6. *A category $\mathcal{C}$ consists of the following two collections:*

1. *A class of objects $\mathrm{Obj}_{\mathcal{C}}$;*

2. *A class of morphisms $\mathrm{Mor}_{\mathcal{C}}$. Each morphism f has a source object A and a target object B, which is usually indicated by the notation $f : A \rightarrow B$. The collection of all morphisms with source A and target B is denoted by $\mathcal{C}[A, B]$. $\mathcal{C}$ is called locally small if $\mathcal{C}[A, B]$ is a set for every pair of objects A and B.*

In addition, a category has a binary operation which is called composition (;). Composition is defined for each pair of morphisms $f : A \rightarrow B$ and $g : B \rightarrow C$, and yields a morphism, $f; g : A \rightarrow C$. The following axioms are valid:

Identity: *For every object A, there is a morphism $\mathbb{I}_A$, called the identity morphism for A, such that for every morphism $f : A \rightarrow B$, we have $f; \mathbb{I}_A = \mathbb{I}_B; f = f$.*

Associativity: *For all $f : A \rightarrow B$, $g : B \rightarrow C$, and $h : C \rightarrow D$, we have $f; (g; h) = (f; g); h$.*

In the definition above composition of morphisms has to be read from left to right, i.e., $f; g$ means first f, then g.

Below we have listed some typical examples of concrete categories:

9

Set: The category with sets as objects and functions as morphisms.

Vec: The category with vector spaces as objects and linear maps as morphisms.

Top: The category with topological spaces as objects and continuous functions as morphisms.

There are also examples of categories that are not based on functions between some (structured) sets. For instance, any ordered set is a category, where the objects are elements of the set and there is exactly one morphism with source x and target y iff $x \leq y$. The axioms of a category are satisfied because the reflexivity of the order induces the identity morphisms and its transitivity implies the associativity of composition.

A homomorphism between categories is called a *functor*.

Definition 7. *Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be categories. A functor $F : \mathcal{C}_1 \to \mathcal{C}_2$ is a pair of operations $F_{\mathrm{Obj}} : \mathrm{Obj}_{\mathcal{C}_1} \to \mathrm{Obj}_{\mathcal{C}_2}, F_{\mathrm{Mor}} : \mathrm{Mor}_{\mathcal{C}_1} \to \mathrm{Mor}_{\mathcal{C}_2}$ such that for each $f : A \to B, g : B \to C$ in $\mathcal{C}_1$,*

- $F_{\mathrm{Mor}}(f) : F_{\mathrm{Obj}}(A) \to F_{\mathrm{Obj}}(B)$

- $F_{\mathrm{Mor}}(f; g) = F_{\mathrm{Mor}}(f); F_{\mathrm{Mor}}(g)$

- $F_{\mathrm{Mor}}(\mathbb{I}_A) = \mathbb{I}_{F_{\mathrm{Obj}}(A)}$

*A functor F is called **full** if for all $A, B \in \mathrm{Obj}_{\mathcal{C}_1}$ and every $i \in \mathcal{C}_2[F(A), F(B)]$ there is $f \in \mathcal{C}_1[A, B]$ such that $i = F(f)$.*

*F is called **faithful** if for all $A, B \in \mathrm{Obj}_{\mathcal{C}_1}$ and for all $f, h \in \mathcal{C}_1[A, B]$, $F(f) = F(h)$ implies $f = h$.*

In category theory the weaker notion of equivalent categories is usually used instead of the stronger notion of isomorphism. However, since we are mainly interested in finite structures we will use the stronger concept of isomorphism.

Definition 8. *A functor is called an isomorphism iff it is full, faithful and bijective on objects. Two categories $\mathcal{C}_1$ and $\mathcal{C}_2$ are isomorphic if there exists an isomorphism $F : \mathcal{C}_1 \to \mathcal{C}_2$.*

Two isomorphic categories share all properties that are defined in terms of category theory.

2.3   Heterogeneous Relation Algebras

In this section we recall some fundamentals on heterogeneous relation algebras. For further details we refer to [5, 11, 10].

Definition 9. *A heterogeneous relation algebra is a locally small category $\mathcal{R}$. Its morphisms are usually called relations. In addition, there is a unary operation $\breve{}_{AB} : \mathcal{R}[A, B] \longrightarrow \mathcal{R}[B, A]$ between the sets of morphisms, called conversion. The operations satisfy the following rules:*

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

2. *The Schröder equivalences*

$$Q; R \sqsubseteq_{AC} S \iff Q^{\smallsmile}; \overline{S} \sqsubseteq_{BC} \overline{R} \iff \overline{S}; R^{\smallsmile} \sqsubseteq_{AB} \overline{Q}$$

11

hold for relations $Q : A \rightarrow B$, $R : B \rightarrow C$, and $S : A \rightarrow C$.

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.

Let Rel be the relation algebra which has sets as objects and concrete binary relations as morphisms. A concrete binary relation R between A and B is a set of pairs from A and B, i.e., $R \subseteq A \times B$ where $A \times B$ is the cartesian product of the sets A and B. In the remainder of this thesis we will also write xQy for $\langle x, y \rangle \in Q$. Suppose $Q, S \subseteq A \times B$ and $R \subseteq B \times C$. Then the operations are defined as follows:

Converse: $Q^{\smallsmile} = \{\langle y, x \rangle \in B \times A \mid xQy\}$,

Complement: $\overline{Q} = \{\langle x, y \rangle \in A \times B \mid \langle x, y \rangle \notin Q\}$,

Union: $Q \sqcup S = \{\langle x, y \rangle \in A \times B \mid xQy \text{ or } xSy\}$,

Intersection: $Q \sqcap S = \{\langle x, y \rangle \in A \times B \mid xQy \text{ and } xSy\}$,

Composition: $Q; R = \{\langle x, z \rangle \in A \times C \mid (\exists y \in B)\ xQy \text{ and } yRz\}$.

The following is an example of a composition in the heterogenous relation algebra Rel.

Suppose $A = \{1, 2, 3\}, B = \{a, b, c\}, C = \{True, False\}$ are objects in Rel, and $Q = \{(1, True), (2, True), (3, False)\}, S = \{(False, a), (False, b), (True, c)\}, T = \{(1, c), (2, c), (3, a), (3, b)\}$ are relations with $Q : A \rightarrow C$, $S : C \rightarrow B$ and $T : A \rightarrow B$.

Relations in Rel can alternatively be represented by Boolean matrices. For example, the first matrix below represents Q in the following sense. The three rows

represent the three elements in A, and the two columns the two elements in C. Therefore, the 1 in the upper left corner indicates that the first element of A, the element 1, is in relation Q with the first element of C, the element True. Analogously, the 0 in the lower left corner indicates that $(3, True) \notin Q$.

Then

$$Q; S = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} ; \begin{pmatrix} 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} = T$$

Lemma 1. *Given relations* $R : A \to B$, $Q : A \to B$, $S : B \to C$, $T : B \to C$, $P : A \to C$, *then we have:*

(a) $R; (S \sqcap T) \sqsubseteq R; S \sqcap R; T$

(b) $R; S \sqcap P \sqsubseteq R; (S \sqcap R^{\smile}; P)$ *and* $R; S \sqcap P \sqsubseteq (R \sqcap P; S^{\smile}); S$

(c) $R; (S \sqcup T) = R; S \sqcup R; T$

(d) $(R; S)^{\smile} = S^{\smile}; R^{\smile}$

The proof of this lemma can be found in either of [9, 10, 11].

Since a heterogeneous relation algebra is a locally small category, where every hom-set is a Boolean algebra, we define homomorphisms between relation algebras based on functors and homomorphisms of Boolean algebras.

Definition 10. *A homomorphism between two relation algebras* $\mathcal{R}_1$ *and* $\mathcal{R}_2$ *is a functor* $F : \mathcal{R}_1 \to \mathcal{R}_2$ *so that the function induced by* F *between* $\mathcal{R}_1[A, B]$ *and*

$\mathcal{R}_2[F(A), F(B)]$ *is a homomorphism between Boolean algebras for every pair of objects A and B.*

Definition 11. *A homomorphism $F : \mathcal{R}_1 \to \mathcal{R}_2$ is called an isomorphism iff it is full, faithful and bijective on objects. Two relation algebras $\mathcal{R}_1$ and $\mathcal{R}_2$ are isomorphic ($\mathcal{R}_1 \cong \mathcal{R}_2$), iff there is an isomorphism from $\mathcal{R}_1$ to $\mathcal{R}_2$.*

Notice that if $F : \mathcal{R}_1 \to \mathcal{R}_2$ is an isomorphism between relation algebras, then the induced function between $\mathcal{R}_1[A, B]$ to $\mathcal{R}_2[F(A), F(B)]$ is an isomorphism between Boolean algebras for every pair of objects A and B.

2.3.1 Integral Object

Integral objects are of particular interest since it can be shown that certain heterogenous relation algebras are equivalent to matrix algebras with coefficients given by relations between integral objects. For further detail see [14].

Following the notion used in algebra, we call an object A integral if there are no zero divisors within the subalgebra $\mathcal{R}[A, A]$. Later on, the class of integral objects will define the basis of $\mathcal{R}$.

Definition 12. *An object A of a relation algebra is called integral iff $\bot\!\!\!\bot_{AA} \neq \top\!\!\!\top_{AA}$ and for all $Q, R \in \mathcal{R}[A, A]$ the equation $Q; R = \bot\!\!\!\bot_{AA}$ implies either $Q = \bot\!\!\!\bot_{AA}$ or $R = \bot\!\!\!\bot_{AA}$.*

The following example shows that not all objects are integral.

Let $A = \{1, 2\}$ be an object of Rel, and $Q = \{(1, 1)\}, S = \{(2, 1)\}$ be relations in $\mathcal{R}[A, A]$.

14

Then

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

However, neither $Q = \mathbb{L}$ nor $S = \mathbb{L}$, i.e., the object A is not integral. Actually, the integral objects in Rel are the singleton sets.

It can be shown [14, 15] that the property of being integral is equivalent to the properties '$\mathbb{I}_A$ is an atom' or 'every relation not equal $\mathbb{L}_{AA}$ is total'.

The *basis* $\mathcal{B}_\mathcal{R}$ of $\mathcal{R}$ is defined as the full subcategory given by the class of all integral objects.

2.3.2 Symmetric Idempotent Relation

The notion of a splitting combines subobjects (or subsets) and the set of equivalence classes of an equivalence relation into one concept [5]. In order to define this concept we first need the notion of a partial equivalence or a symmetric idempotent relation.

Definition 13. *A relation $X : A \to A$ is called a symmetric idempotent relation, iff $X^\smile = X$ and $X; X = X$.*

Suppose $A = \{\text{Peter, Jon, Mary, Paul, Carly, Anna, Debbie, Dave}\}$ is a set of persons and an object in Rel. Let $X : A \to A$ be the relation on this set of persons that relates two persons if they went to the same country for vacation this year. We have $X = \{\langle\text{Peter, Peter}\rangle, \langle\text{Peter, Mary}\rangle, \langle\text{Peter, Paul}\rangle, \langle\text{Jon, Jon}\rangle, \langle\text{Mary, Peter}\rangle, \langle\text{Mary, Mary}\rangle, \langle\text{Mary, Paul}\rangle, \langle\text{Paul, Peter}\rangle, \langle\text{Paul, Mary}\rangle, \langle\text{Paul, Paul}\rangle, \langle\text{Anna, Debbie}\rangle, \langle\text{Anna, Anna}\rangle,$

⟨Debbie, Debbie⟩, ⟨Debbie, Anna⟩} or as a Boolean matrix:

	Peter	Jon	Mary	Paul	Carly	Anna	Debbie	Dave
Peter	1	0	1	1	0	0	0	0
Jon	0	1	0	0	0	0	0	0
Mary	1	0	1	1	0	0	0	0
Paul	1	0	1	1	0	0	0	0
Carly	0	0	0	0	0	0	0	0
Anna	0	0	0	0	0	1	1	0
Debbie	0	0	0	0	0	1	1	0
Dave	0	0	0	0	0	0	0	0

Notice that Carly and Dave did not go on vacation this year so that they are not in relation to themselves. X is a partial equivalence relation that has three equivalence classes – the sets {Peter, Mary, Paul}, {Jon} and {Anna, Debbie} – and two elements that do not belong to any equivalence class – the elements Carly and Dave.

Some basic properties of symmetric idempotent relations needed throughout the thesis are listed in the following lemma.

Lemma 2. *If $X : A \to A$ and $Y : B \to B$ are symmetric idempotent relations, then*

(a) $X;(Q \sqcap R);Y = X;Q;Y \sqcap X;R;Y$ *for all* $Q, R : A \to B$ *with* $X;Q;Y \sqsubseteq Q$,

(b) $X;\overline{X;\overline{Q;Y}};Y \sqsubseteq Q$ *for all* $Q : A \to B$,

(c) $Q \sqsubseteq \overline{X;\overline{Q};Y}$ *or all* $Q : A \to B$ *with* $X;Q;Y \sqsubseteq Q$,

(d) $X;\overline{X;\overline{Q;Y}};Y = Q$ *for all* $Q : A \to B$ *with* $X;Q;Y = Q$.

16

Proof. (a) We have $X; (Q \sqcap R); Y \sqsubseteq X; Q; Y \sqcap X; R; Y$ Lemma 1 (a)

For the converse inclusion we have

$$X; Q; Y \sqcap X; R; Y \sqsubseteq X; (X^\smile; X; Q; Y \sqcap R; Y) \quad \text{Lemma 1 (b)}$$

$$= X; (X; Q; Y \sqcap R; Y) \quad\quad X^\smile = X \text{ and } X; X = X$$

$$\sqsubseteq X; (X; Q; Y; Y^\smile \sqcap R); Y \quad \text{Lemma 1 (b)}$$

$$= X; (X; Q; Y \sqcap R); Y \quad\quad Y^\smile = Y \text{ and } Y; Y = Y$$

$$\sqsubseteq X; (Q \sqcap R); Y \quad\quad\quad X; Q; Y \sqsubseteq Q$$

(b) The assumption follows from

$$X; \overline{Q}; Y \sqsubseteq X; \overline{Q}; Y$$

$$\Leftrightarrow \overline{X; \overline{Q}; Y}; Y^\smile \sqsubseteq \overline{X; \overline{Q}} \quad\quad \text{Schröder equivalences}$$

$$\Leftrightarrow \overline{X; \overline{Q}; Y}; Y \sqsubseteq \overline{X; \overline{Q}} \quad\quad Y^\smile = Y$$

$$\Leftrightarrow X; \overline{Q} \sqsubseteq \overline{\overline{X; \overline{Q}; Y}; Y}$$

$$\Leftrightarrow X^\smile; \overline{X; \overline{Q}; Y}; Y \sqsubseteq Q \quad\quad \text{Schröder equivalences}$$

$$\Leftrightarrow X; \overline{X; \overline{Q}; Y}; Y \sqsubseteq Q \quad\quad X^\smile = X$$

17

(c) We have

$$X;Q;Y \sqsubseteq Q \qquad\qquad \text{Assumption}$$

$$\Leftrightarrow \overline{Q};Y^{\smile} \sqsubseteq \overline{X;Q} \qquad\qquad \text{Schröder equivalences}$$

$$\Leftrightarrow \overline{Q};Y \sqsubseteq \overline{X;Q} \qquad\qquad Y^{\smile} = Y$$

$$\Leftrightarrow X;Q \sqsubseteq \overline{\overline{Q};Y}$$

$$\Leftrightarrow X^{\smile};\overline{Q};Y \sqsubseteq \overline{Q} \qquad\qquad \text{Schröder equivalences}$$

$$\Leftrightarrow X;\overline{Q};Y \sqsubseteq \overline{Q} \qquad\qquad X^{\smile} = X$$

$$\Leftrightarrow Q \sqsubseteq \overline{X;\overline{Q};Y}$$

(d) From (b), we obtain the inclusion $\sqsubseteq$. The converse inclusion follows from $Q = X;Q;Y \sqsubseteq X;\overline{X;\overline{Q};Y};Y$ using (c). $\qquad\qquad \square$

Now, we are ready to define the concept of a splitting.

Definition 14. *Let $\mathcal{R}$ be a relation algebra, and $X : A \to A$ be a symmetric idempotent relation. Then an object B together with a relation $R : B \to A$ is called a splitting of X iff $R;R^{\smile} = \mathbb{I}_B$ and $R^{\smile};R = X$.*

Consider again the set A of persons and the symmetric and idempotent relation $X : A \to A$ from above. The object B of the splitting of X is the set of the equivalence classes, i.e., it is the set containing the three sets $\{\text{Peter}, \text{Mary}, \text{Paul}\}$, $\{\text{Jon}\}$ and $\{\text{Anna}, \text{Debbie}\}$. The splitting $R : B \to A$ is the relation that relates

18

every equivalence class to its elements, i.e., it is given by $R =$

$$
\begin{array}{c}
 \\
\{\text{Peter}, \text{Mary}, \text{Paul}\} \\
\{\text{Jon}\} \\
\{\text{Anna}, \text{Debbie}\}
\end{array}
\begin{array}{cccccccc}
\text{Peter} & \text{Jon} & \text{Mary} & \text{Paul} & \text{Carly} & \text{Anna} & \text{Debbie} & \text{Dave} \\
\left(\begin{array}{cccccccc}
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0
\end{array}\right)
\end{array}
$$

Theorem 1. *Every relation algebra can be faithfully embedded into an algebra that provides all splittings.*

Proof. Let $\mathcal{R}$ be a relation algebra, and define a relation algebra $\mathcal{R}^+$ as follows. The class of object $\mathrm{Obj}_{\mathcal{R}^+}$ is the class of all SID (symmetric idempotent relations) from $\mathcal{R}$. Given two SID $X : A \to A$ and $Y : B \to B$, the set of relations between X and Y in $\mathcal{R}^+$ is defined by $\mathcal{R}^+[X,Y] = \{R : A \to B \mid X; R; Y = R\}$. All operations except complement are inherited from $\mathcal{R}$. The complement of $S \in \mathcal{R}^+[X,Y]$ is given by $X; \overline{S}; Y$. Therefore, given $T, S \in \mathcal{R}^+[X,Y]$, $Q \in \mathcal{R}^+[Y,Z]$ and $R \in \mathcal{R}^+[X,Z]$ we need to show:

1. $X; T; Q; Z = T; Q,$

2. $X; (T \sqcap S); Y = T \sqcap S,$

3. $X; (T \sqcup S); Y = T \sqcup S,$

4. The operation $\sqcup$ together with the complement $X; \overline{S}; Y$ for $S \in \mathcal{R}^+[X,Y]$ forms a Boolean algebra, i.e., we have $X; \overline{X; \overline{Q}; Y \sqcup X; \overline{R}; Y}; Y \sqcup X; \overline{X; \overline{Q}; Y \sqcup R}; Y = Q$ for all $Q, R \in \mathcal{R}^+[X,Y]$.

19

5. $Y; S^\smile; X = S^\smile$,

6. $\mathbb{I}_x = X$,

7. The Schröder equivalences are valid, i.e.,

$$S; Q \sqsubseteq R \iff S^\smile; X; \overline{R}; Z \sqsubseteq Y; \overline{Q}; Z \iff X; \overline{R}; Z; Q^\smile \sqsubseteq X; \overline{S}; Y.$$

We now show each of Equations 1-7 above:

1. Suppose $T \in \mathcal{R}^+[X, Y]$ and $Q \in \mathcal{R}^+[Y, Z]$. Then we have

$$
\begin{aligned}
X; T; Q; Z &= X; X; T; Y; Y; Q; Z; Z && T = X; T; Y \text{ and } Q = Y; Q; Z \\
&= X; T; Y; Y; Q; Z && X = X; X \text{ and } Z = Z; Z \\
&= T; Q && X; T; Y = T \text{ and } Y; Q; Z = Q
\end{aligned}
$$

2. Suppose $T, S \in \mathcal{R}^+[X, Y]$. Then we have

$$
\begin{aligned}
X; (T \sqcap S); Y &= X; T; Y \sqcap X; S; Y && \text{Lemma 2 (a)} \\
&= T \sqcap S && T = X; T; Y \text{ and } S = X; S; Y \\
T \sqcap S &= X; T; Y \sqcap S && T = X; T; Y \\
&\sqsubseteq X; (T; Y \sqcap X^\smile; S) && \text{Lemma 1 (b)} \\
&\sqsubseteq X; (T \sqcap X^\smile; S; Y^\smile); Y && \text{Lemma 1 (b)} \\
&= X; (T \sqcap X; S; Y); Y && X = X^\smile \text{ and } Y = Y^\smile \\
&= X; (T \sqcap S); Y && S = X; S; Y
\end{aligned}
$$

20

3. Suppose $T, S \in \mathcal{R}^+[X, Y]$. Then we have

$$X;(T \sqcup S);Y = (X;T \sqcup X;S);Y \qquad \text{Lemma 1 (c)}$$

$$= X;T;Y \sqcup X;S;Y \qquad \text{Lemma 1 (c)}$$

$$= T \sqcup S \qquad T = X;T;Y \text{ and } S = X;S;Y$$

4. $X;\overline{X;\overline{Q};Y \sqcup X;\overline{R};Y};Y \sqcup X;\overline{X;\overline{Q};Y \sqcup R};Y$

$$= X;(\overline{X;\overline{Q};Y} \sqcap \overline{X;\overline{R};Y});Y \sqcup X;(\overline{X;\overline{Q};Y} \sqcap \overline{R});Y \qquad \text{Definition of } \sqcap$$

$$= (X;\overline{X;\overline{Q};Y};Y \sqcap X;\overline{X;\overline{R};Y};Y) \sqcup (X;\overline{X;\overline{Q};Y};Y \sqcap X;\overline{R};Y) \qquad \text{Lemma 2 (a)}$$

$$= (Q \sqcap R) \sqcup (Q \sqcap X;\overline{R};Y) \qquad \text{Lemma 2 (d)}$$

$$= Q \sqcap (R \sqcup X;\overline{R};Y) \qquad \text{Definition of } \sqcup$$

$$= Q \sqcap (X;R;Y \sqcup X;\overline{R};Y)$$

$$= Q \sqcap X;(R \sqcup \overline{R});Y \qquad \text{Lemma 1 (c)}$$

$$= X;Q;Y \sqcap X;(R \sqcup \overline{R});Y$$

$$= X;(Q \sqcap (R \sqcup \overline{R}));Y \qquad \text{Lemma 2 (a)}$$

$$= X;Q;Y$$

$$= Q$$

5. Suppose $S \in \mathcal{R}^+[X, Y]$. Then we have

$$Y;S^\smile;X = Y^\smile;S^\smile;X^\smile \qquad X = X^\smile \text{ and } Y = Y^\smile$$

$$= X;S;Y^\smile \qquad \text{Lemma 1 (d)}$$

$$= S^\smile$$

21

6. Suppose $S \in \mathcal{R}^+[X, Y]$. We have to show that $X; S = S$ for all $S \in \mathcal{R}^+[X, Y]$ and $T; X = T$ for all $T \in \mathcal{R}^+[Z, X]$.

$$X; S = X; X; S; Y \qquad\qquad\qquad S = X; S; Y$$

$$= X; S; Y \qquad\qquad\qquad\qquad X = X; X$$

$$= S \qquad\qquad\qquad\qquad\qquad X; S; Y = S$$

7. $S; Q \sqsubseteq R$

$$\Leftrightarrow S^\smile; \overline{R} \sqsubseteq \overline{Q} \qquad\qquad\qquad\qquad \text{Schröder equivalences}$$

$$\Rightarrow Y; S^\smile; \overline{R}; Z \sqsubseteq Y; \overline{Q}; Z$$

$$\Leftrightarrow Y; Y; S^\smile; X; \overline{R}; Z \sqsubseteq Y; \overline{Q}; Z \qquad (5)$$

$$\Leftrightarrow Y; S^\smile; X; X; \overline{R}; Z \sqsubseteq Y; \overline{Q}; Z \qquad Y; Y = Y \text{ and } X; X = X$$

$$\Leftrightarrow S^\smile; X; \overline{R}; Z \sqsubseteq Y; \overline{Q}; Z \qquad (5)$$

The converse implication is shown by

$$S^\smile; X; \overline{R}; Z; \sqsubseteq Y; \overline{Q}; Z$$

$$\Leftrightarrow S; \overline{Y; \overline{Q}; Z} \sqsubseteq \overline{X; \overline{R}; Z} \qquad\qquad \text{Schröder equivalences}$$

$$\Rightarrow X; S; \overline{Y; \overline{Q}; Z}; Z \sqsubseteq X; \overline{X; \overline{R}; Z}; Z$$

$$\Leftrightarrow X; X; S; Y; \overline{Y; \overline{Q}; Z}; Z \sqsubseteq X; \overline{X; \overline{R}; Z}; Z \qquad X; S; Y = S$$

$$\Leftrightarrow X; S; Y; Y; \overline{Y; \overline{Q}; Z}; Z \sqsubseteq X; \overline{X; \overline{R}; Z}; Z \qquad X; X = X \text{ and } Y; Y = Y$$

$$\Leftrightarrow X; S; Y; Q \sqsubseteq R \qquad\qquad\qquad \text{Lemma 2 (d)}$$

$$\Leftrightarrow S; Q \sqsubseteq R \qquad\qquad\qquad\qquad X; S; Y = S$$

The second equivalence is shown analogously.

We have just shown that $\mathcal{R}^+$ is a relation algebra, which inherits all operations from $\mathcal{R}$. The next thing we have to prove is that $\mathcal{R}$ can be faithfully embedded into $\mathcal{R}^+$.

Define a functor $F : \mathcal{R} \to \mathcal{R}^+$ by $F(A) = \mathbb{I}_A$ for every object A in $\mathcal{R}$, and $F(T) = T$ for every relation. This functor is obviously full and faithful.

The last thing we have to show is that $\mathcal{R}^+$ provides all splittings. Let U be **SID** in $\mathcal{R}^+$, where $U : X \to X$. By definition we know that $U^\smile = U$, $U;U = U$ and $X;U;X = U$ Therefore U is also an object in $\mathcal{R}^+$. Furthermore, we have

$$U;U;X = U;X \qquad\qquad U;U = U$$

$$= X;U;X;X \qquad\qquad U = X;U;X$$

$$= X;U;X \qquad\qquad X;X = X$$

$$= U \qquad\qquad X;U;X = U$$

i.e., $U \in \mathcal{R}^+[U, X]$. From

$$U;U^\smile = U = \mathbb{I}_U \text{ and } U^\smile;U = U$$

We conclude that the object U and the relation $U \in \mathcal{R}^+[U, X]$ is a splitting of $U \in \mathcal{R}^+[X, X]$. $\qquad\qquad\square$

Later, in Chapter 3, we will use the concept of symmetric idempotent relations to define one structure that represents atom structures of heterogenous relation algebras.

Definition 15. *Given two relation algebras $\mathcal{R}_1$ and $\mathcal{R}_2$, define their* **direct sum** *$\mathcal{R}_1 +_0 \mathcal{R}_2$ as follows:*

- *The class of object $\mathrm{Obj}_{(\mathcal{R}_1 +_0 \mathcal{R}_2)}$ is the union of all objects from $\mathcal{R}_1$ and $\mathcal{R}_2$.*

- *The set of relations between objects A and B is defined by:*

$$(\mathcal{R}_1 +_0 \mathcal{R}_2)[A, B] = \begin{cases} \mathcal{R}_1[A, B], & A, B \in \mathrm{Obj}_{\mathcal{R}_1} \\ \mathcal{R}_2[A, B], & A, B \in \mathrm{Obj}_{\mathcal{R}_2} \\ \{\bot_{AB}\}, & \text{otherwise.} \end{cases}$$

- *For $Q : A \to B$, $S : B \to C$, the composition $(;)$ is defined by:*

$$Q; S = \begin{cases} Q; S, & A, B, C \in \mathrm{Obj}_{\mathcal{R}_1} \ \text{or} \ A, B, C \in \mathrm{Obj}_{\mathcal{R}_2} \\ \bot_{AC}, & \text{otherwise.} \end{cases}$$

Theorem 2. *The directed sum of two relation algebras is also a relation algebra, and the given relation algebras can be embedded into it.*

Proof. Since $\mathcal{R}_1$ and $\mathcal{R}_2$ are relation algebras, and $\bot$ is the only element between objects of different inner relation algebras, each set $(\mathcal{R}_1 +_0 \mathcal{R}_2)[A, B]$ is a Boolean algebra.

For each triple of morphisms $T \in (\mathcal{R}_1 +_0 \mathcal{R}_2)[A, B]$, $S \in (\mathcal{R}_1 +_0 \mathcal{R}_2)[B, C]$ and $Q \in (\mathcal{R}_1 +_0 \mathcal{R}_2)[C, D]$; the associativity $(T; S); Q = T; (S; Q)$ holds if A, B, C, D are all either from $\mathcal{R}_1$ or $\mathcal{R}_2$. Otherwise one of the relations T, S or Q equals $\bot$. Assume $T = \bot_{AB}$, then we have

$$(T; S); Q = (\bot_{AB}; S); Q$$

$$= \bot_{AC}; Q$$

$$= \bot_{AD}$$

$$T; (S; Q) = \bot_{AB}; (S; Q)$$

$$= \bot_{AD}$$

The other cases are similar.

Therefore $(T;S);Q = T;(S;Q)$, and the associativity is valid.

If all relations are in one of the relation algebras $\mathcal{R}_1$ or $\mathcal{R}_2$, the Schröder equivalences are vaild. Otherwise assume $T \in (\mathcal{R}_1 +_0 \mathcal{R}_2)[X,Y]$, $S \in (\mathcal{R}_1 +_0 \mathcal{R}_2)[X,Z]$ and $Q \in (\mathcal{R}_1 +_0 \mathcal{R}_2)[Y,Z]$, where $X,Y \in \mathrm{Obj}_{\mathcal{R}_1}$ and $Z \in \mathrm{Obj}_{\mathcal{R}_2}$.

$$
\begin{aligned}
T;Q = T;\bot\!\!\!\bot_{YZ} && Q \in (\mathcal{R}_1 +_0 \mathcal{R}_2)[Y,Z]\\
= \bot\!\!\!\bot_{XZ} && \text{the definition of composition}\\
\sqsubseteq_{XZ} S && S \in (\mathcal{R}_1 +_0 \mathcal{R}_2)[X,Z]\\
T^\smile;\overline{S} = T^\smile;\bot\!\!\!\bot_{XZ} && S \in (\mathcal{R}_1 +_0 \mathcal{R}_2)[X,Z]\\
= \bot\!\!\!\bot_{YZ} && \text{the definition of composition}\\
\sqsubseteq_{YZ} \overline{Q} && Q \in (\mathcal{R}_1 +_0 \mathcal{R}_2)[Y,Z]
\end{aligned}
$$

Therefore $T;Q \sqsubseteq_{XZ} S \Leftrightarrow T^\smile;\overline{S} \sqsubseteq_{YZ} \overline{Q}$. The other equivalences are shown analogously.

$\mathcal{R}_1$ and $\mathcal{R}_2$ are embedded into $(\mathcal{R}_1 +_0 \mathcal{R}_2)$ because the class of object $\mathrm{Obj}_{(\mathcal{R}_1 +_0 \mathcal{R}_2)}$ is the union of all objects from $\mathcal{R}_1$ and $\mathcal{R}_2$. $\qquad\square$

In the case of integral relation algebras the directed sum can be enlarged by using a two-element Boolean algebra instead of the trivial one-element Boolean algebra as the intermediate morphism sets.

Definition 16. *Given two **integral** relation algebras $\mathcal{R}_1$ and $\mathcal{R}_2$, define their **enlarged sum** $\mathcal{R}_1 +_1 \mathcal{R}_2$ as follows:*

- *The class of object $\mathrm{Obj}_{(\mathcal{R}_1 +_1 \mathcal{R}_2)}$ is the union of all objects from $\mathcal{R}_1$ and $\mathcal{R}_2$.*

25

- *The set of relations between objects A and B is defined by:*

$$(\mathcal{R}_1 +_1 \mathcal{R}_2)[A, B] = \begin{cases} \mathcal{R}_1[A, B], & A, B \in \mathrm{Obj}_{\mathcal{R}_1} \\ \mathcal{R}_2[A, B], & A, B \in \mathrm{Obj}_{\mathcal{R}_2} \\ \{\bot_{AB}, \top_{AB}\}, & \text{otherwise.} \end{cases}$$

- *For $Q : A \to B$, $S : B \to C$, the composition $(;)$ is defined by:*

$$Q; S = \begin{cases} Q; S, & A, B, C \in \mathrm{Obj}_{\mathcal{R}_1} \text{ or } A, B, C \in \mathrm{Obj}_{\mathcal{R}_2} \\ \bot_{AC}, & Q = \bot_{AB} \text{ or } S = \bot_{BC} \\ \top_{AC}, & \text{otherwise} \end{cases}$$

Theorem 3. *The enlarged sum of two integral relation algebras is also a relation algebra, and the given relation algebras can be embedded into it.*

Proof. The proof of this theorem is very similar to the proof of Theorem 2. Therefore, we just show the Schröder equivalences as an example.

Suppose $T \in (\mathcal{R}_1 +_1 \mathcal{R}_2)[X, Y]$, $S \in (\mathcal{R}_1 +_1 \mathcal{R}_2)[X, Z]$ and $Q \in (\mathcal{R}_1 +_1 \mathcal{R}_2)[Y, Z]$, where $X, Y \in \mathrm{Obj}_{\mathcal{R}_1}$ and $Z \in \mathrm{Obj}_{\mathcal{R}_2}$.

If $Q = \bot_{YZ}$, then we obtain

$$T; Q = T; \bot_{YZ}$$

$$= \bot_{XZ} \qquad\qquad\qquad \text{the definition of composition}$$

$$\sqsubseteq_{XZ} S \qquad\qquad\qquad S \in (\mathcal{R}_1 +_1 \mathcal{R}_2)[X, Z]$$

$$T^\smile; \overline{S} \sqcap Q \sqsubseteq T^\smile; (\overline{S} \sqcap T; Q) \qquad\qquad \text{Lemma 1 (b)}$$

$$= T^\smile; \bot_{XZ} \qquad\qquad\qquad T; Q \sqsubseteq_{XZ} S$$

$$= \bot_{YZ} \qquad\qquad\qquad \text{the definition of composition}$$

If $T = \bot_{XY}$, then

$$T;Q = \bot_{XY};Q$$

$$= \bot_{XZ} \qquad\qquad \text{the definition of composition}$$

$$\sqsubseteq_{XZ} S \qquad\qquad S \in (\mathcal{R}_1 +_1 \mathcal{R}_2)[X, Z]$$

$$T^{\smile};\overline{S} = \bot_{YX};\overline{S}$$

$$= \bot_{YZ} \qquad\qquad \text{the definition of composition}$$

Otherwise we get

$$T;Q = T;\mathbb{T}_{YZ} \qquad\qquad Q \in (\mathcal{R}_1 +_1 \mathcal{R}_2)[Y, Z]$$

$$= \mathbb{T}_{XZ} \qquad\qquad \text{the definition of composition}$$

$$\sqsubseteq_{XZ} S \qquad\qquad S \in (\mathcal{R}_1 +_1 \mathcal{R}_2)[X, Z]$$

$$T^{\smile};\overline{S} = T^{\smile};\mathbb{T}_{XZ} \qquad\qquad S \in (\mathcal{R}_1 +_1 \mathcal{R}_2)[X, Z]$$

$$= \mathbb{T}_{YZ} \qquad\qquad \text{the definition of composition}$$

$$\sqsubseteq_{YZ} \overline{Q} \qquad\qquad Q \in (\mathcal{R}_1 +_1 \mathcal{R}_2)[Y, Z]$$

Therefore $T;Q \sqsubseteq_{XZ} S \Leftrightarrow T^{\smile};\overline{S} \sqsubseteq_{YZ} \overline{Q}$. The other equivalences are shown analogously. $\square$

Another way of constructing a relation algebra from given ones is to consider pairs of relations.

Definition 17. *Given two relation algebras $\mathcal{R}_1$ and $\mathcal{R}_2$, their product $\mathcal{R}_1 \times \mathcal{R}_2$ is defined by:*

- *The class of object $\mathrm{Obj}_{(\mathcal{R}_1 \times \mathcal{R}_2)} = \{(A, B) | A \in \mathrm{Obj}_{\mathcal{R}_1}, B \in \mathrm{Obj}_{\mathcal{R}_2}\}$*

27

- *The set of relations in $\mathcal{R}_1 \times \mathcal{R}_2$ is a set of pairs, the first element is a relation from $\mathcal{R}_1$ and the second element is a relation from $\mathcal{R}_2$.*

- *The composition is inherited from $\mathcal{R}_1$ and $\mathcal{R}_2$ and applied componentwise.*

Theorem 4. *The product of two relation algebras is also a relation algebra.*

Proof. Given two relation algebras $\mathcal{R}_1$ and $\mathcal{R}_2$, and we define the product of two relation algebras $\mathcal{R}_1 \times \mathcal{R}_2$ by following its definition. The set of relations in $\mathcal{R}_1 \times \mathcal{R}_2$ carries the structure of a complete atomic Boolean algebra.

For each set of morphisms in $\mathcal{R}_1 \times \mathcal{R}_2$, we need to show the associativity. Let $Q_1 : X_1 \rightarrow Y_1, S_1 : Y_1 \rightarrow Z_1$, and $T_1 : Z_1 \rightarrow X_1$, where $X_1, Y_1, Z_1 \in \mathrm{Obj}_{R_1}$; and $Q_2 : X_2 \rightarrow Y_2, S_2 : Y_2 \rightarrow Z_2$, and $T_2 : Z_2 \rightarrow X_2$, where $X_2, Y_2, Z_2 \in \mathrm{Obj}_{R_2}$.

$$(Q_1, Q_2); ((S_1, S_2); (T_1, T_2)) = (Q_1, Q_2); (S_1; T_1, S_2; T_2)$$

$$= (Q_1; S_1; T_1, Q_2; S_2; T_2)$$

$$((Q_1, Q_2); (S_1, S_2)); (T_1, T_2) = (Q_1; S_1, Q_2; S_2); (T_1, T_2)$$

$$= (Q_1; S_1; T_1, Q_2; S_2; T_2)$$

Therefore $(Q_1, Q_2); ((S_1, S_2); (T_1, T_2)) = ((Q_1, Q_2); (S_1, S_2)); (T_1, T_2)$

Now we need to show the Schröder equivalences.

$$(Q_1, Q_2); (S_1, S_2) \sqsubseteq (T_1, T_2) \Leftrightarrow (Q_1; S_1, Q_2; S_2) \sqsubseteq (T_1, T_2)$$

$$\Leftrightarrow Q_1; S_1 \sqsubseteq T_1 \text{ and } Q_2; S_2 \sqsubseteq T_2$$

$$\Leftrightarrow Q_1^{\smile}; \overline{S_1} \sqsubseteq \overline{T_1} \text{ and } Q_2^{\smile}; \overline{S_2} \sqsubseteq \overline{T_2}$$

The other equivalences follow analogously. $\qquad\qquad\square$

28

2.4 Homogeneous Relation Algebras

In this section we review the basic concepts of homogeneous relation algebras. For further details we refer to [9, 4]. A homogeneous relation algebra is a heterogenous relation algebra with one object.

2.4.1 Atom Structure

Every relation algebra provides an atom structure, i.e., a relational structure on its set of atoms. For further details not mentioned in this chapter we refer to [8].

The operations of the underlying Boolean algebra correspond to the set-theoretic operations of union, intersection and complement on sets of atoms. Some additional structure on atoms is needed to recover composition and converse of the relations. Notice that the converse of an atom is an atom again. For a proof see [8].

Definition 18. *Let $\mathcal{R}$ be a finite homogenous relation algebra. The atom structure of $\mathcal{R}$ is a structure of form $\mathfrak{At}(\mathcal{R}) = \langle At(\mathcal{R}), C, \smile, I \rangle$, where $At(\mathcal{R})$ is the set of atoms of $\mathcal{R}$, $C = \{\langle x, y, z \rangle : z \sqsubseteq x; y\}$ is a ternary relation on $At(\mathcal{R})$, i.e., $C \subseteq At(\mathcal{R}) \times At(\mathcal{R}) \times At(\mathcal{R})$ and $I = \{x : x \in At(\mathcal{R}), x \sqsubseteq \mathbb{I}\}$.*

We need to prove the following theorem in order to define the concept of a cycle. This concept can be used to represent the ternary relation C in a more compact manner.

Theorem 5. *Let $\mathfrak{At}(\mathcal{R})$ be the atom structure of a finite homogeneous relation algebra. Then $\langle x, y, z \rangle \in C$ iff $\langle \breve{x}, z, y \rangle \in C$ iff $\langle y, \breve{z}, \breve{x} \rangle \in C$ iff $\langle \breve{y}, \breve{x}, \breve{z} \rangle \in C$ iff $\langle \breve{z}, x, \breve{y} \rangle \in C$ iff $\langle z, \breve{y}, x \rangle \in C$.*

29

Proof. $\langle x, y, z \rangle \in C$

$$\Leftrightarrow z \sqsubseteq x; y$$

$$\Leftrightarrow z \sqcap x; y \neq 0 \qquad\qquad z \text{ is an atom.}$$

$$\Leftrightarrow x; y \not\sqsubseteq \overline{z} \qquad\qquad \text{Boolean algebra property.}$$

$$\Leftrightarrow x^{\smile}; z \not\sqsubseteq \overline{y} \qquad\qquad \text{Schröder equivalences.}$$

$$\Leftrightarrow x^{\smile}; z \sqcap y \neq 0 \qquad\qquad \text{Boolean algebra property.}$$

$$\Leftrightarrow y \sqsubseteq x^{\smile}; z \qquad\qquad y \text{ is an atom.}$$

$$\langle x^{\smile}, z, y \rangle \in C$$

All other implications are shown analogously. □

Due to the theorem, we define a cycle [x,y,z] by

$$[x, y, z] = \{ \langle x, y, z \rangle, \langle y, \breve{z}, \breve{x} \rangle, \langle z, \breve{y}, x \rangle \langle \breve{x}, z, y \rangle, \langle \breve{y}, \breve{x}, \breve{z} \rangle, \langle \breve{z}, x, \breve{y} \rangle \}$$

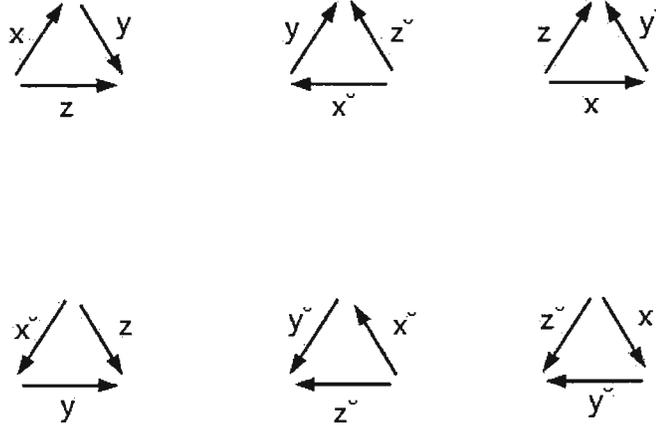A cycle can nicely be visualized by the following diagrams.

Figure 2.1: Cycle law for binary relations.

If one of the directed triangles in Figure 2.1 is in the relation C of an atom structure of a relation algebra, then so are the others. In other words, the relation C is the union of certain cycles. Furthermore, it is sufficient to store only one element as the representative of a cycle.

For an example consider the set $A = \{1, 2\}$ and the relation algebra $\mathcal{R}$ with object A and all binary relations on A. Represented as Boolean matrices, the atoms of $\mathcal{R}$ are the following four relations:

$$a = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, b = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}, c = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}, d = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

The atom structure is, therefore, given by

$$At(\mathfrak{B}) = \{a, b, c, d\}, \text{ and } I(\mathfrak{B}) = \{a, d\}.$$
$$\breve{a} = a, \; \breve{b} = c, \; \breve{c} = b, \; \breve{d} = d.$$

By the definition of cycle structures, we obtain

$$C(\mathfrak{B}) = \{[a, a, a], [b, c, a], [c, b, d], [d, d, d]\}.$$

The atom structure of $\mathfrak{B}$ is $\mathfrak{At}(\mathfrak{B}) = \langle At(\mathfrak{B}), C(\mathfrak{B}), \breve{}, I(\mathfrak{B}) \rangle$.

2.4.2 Complex Algebra

We are now interested in the converse process, i.e., defining a relation algebra if an atom structure is given.

Definition 19. *Let be* $\mathfrak{S} = \langle U, C, f, I \rangle$ *a relational structure such that* $C \subseteq U \times U \times$ $U, f : U \to U$ *and* $I \subseteq U$. *The complex algebra of* $\mathfrak{S}$ *is* $\mathfrak{Cm}(\mathfrak{S}) = \langle \mathcal{P}(U), \cup, \bar{\ }, ; , \ \breve{\ }, I \rangle$, *where* $X ; Y = \{z : (\exists x \in X)(\exists y \in Y)(\langle x, y, z \rangle \in C)\}$, *and* $X^{\breve{\ }} = \{fx : x \in X\}$, *for any subset* $X, Y \subseteq U$.

We continue with the example which was given in the previous section. In this case, the given relational structure is $\mathfrak{S} = \mathfrak{At}(\mathfrak{B}) = \langle At(\mathfrak{B}), C(\mathfrak{B}), \ \breve{\ }, I(\mathfrak{B}) \rangle$.

A finite relation algebra is completely determined by the action of composition on its atoms. The cycles completely determine the composition operation.

Let $X = \{a, b\}$, $Y = \{c, d\}$, we have $\{a, b\}; \{c, d\} = \{a, b\}$ by the definition of the relation $C(B)$. This corresponds to the composition of the following two matrices in the original algebra.

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

By the definition of the relation $C(B)$, and $\{a, b\}^{\breve{\ }} = \{fa, fb\} = \{a, c\}$. This corresponds to the union of the following two matrices in the original algebra.

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \cup \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}$$

The connections between atom structures and complex algebras have been presented in [8] as following: If $\mathfrak{A}$ is a complete and atomic relation algebra, then $\mathfrak{A} \cong \mathfrak{Cm}(\mathfrak{At}(\mathfrak{A}))$. Furthermore, any two complete and atomic relation algebras $\mathfrak{A}$ and $\mathfrak{B}$, $\mathfrak{A} \cong \mathfrak{B}$ iff $\mathfrak{At}(\mathfrak{A}) \cong \mathfrak{At}(\mathfrak{B})$.

32

It is well-known that every atomic relation algebra induces a particular structure on its atoms: its atom structure. In particular, composition is represented by a set of triples of atoms, i.e., by a ternary relation. Conversely, given an atom structure one may form the complex algebra thereof. The structure obtained is, in general, neither integral nor a relation algebra.

Following properties in [9] characterize the atom structure that lead to a relation algebra.

Theorem 6. *Given* $\mathfrak{A} = \langle U, C, \smile, I \rangle$, $\mathfrak{Cm}(\mathfrak{A})$ *is a relation algebra iff*

1. if $\langle x, y, z \rangle \in C$, then $[x, y, z] \subseteq C$.

2. $x = y$ iff there is some $i \in I$ such that $\langle x, i, y \rangle \in C$,

3. if $\langle v, w, x \rangle, \langle x, y, z \rangle \in C$ then there exists $u \in U$ such that $\langle w, y, u \rangle, \langle v, u, z \rangle \in C$ as shown in Figure 2.2.
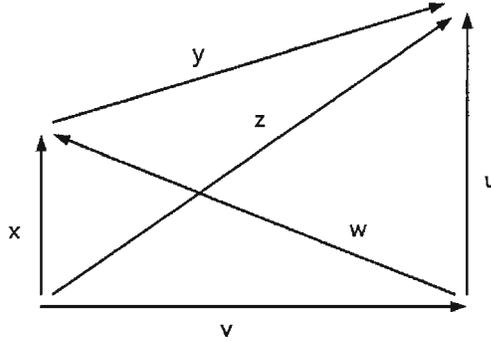

Figure 2.2: Associativity for composition.

When $1'$ is an atom, I is a singleton, i.e., $I = \{e\}$.

33

Theorem 7. *Given* $\mathfrak{A} = \langle U, C, \smile, \{e\} \rangle$,

- $\mathfrak{Cm}(\mathfrak{A})$ *is integral iff* $\mathfrak{A}$ *satisfies condition (a).*

- $\mathfrak{Cm}(\mathfrak{A})$ *is a relation algebra, iff* $\mathfrak{A}$ *satisfies conditions (a) and (b).*

(a) *if* $x \neq y$ *then there is some* $w \in U \sim \{e\}$ *such that* $\langle x, w, y \rangle \in C$.

(b) *if* $\langle v, w, x \rangle, \langle x, y, z \rangle \in C$ *and* $v \neq z$, $\breve{w} \neq y$ *then there exists* $u \in U \sim \{e\}$ *such that* $\langle w, y, u \rangle, \langle v, u, z \rangle \in C$.

Later, in Chapter 4, we will use these properties to check whether the complex algebra of a given relational structure on a set is a relation algebra .

Chapter 3

Three Structures representing Relation Algebras

In this chapter we want to introduce three structures representing atom structures of integral heterogenous relation algebras. The first structure simply embeds a homogeneous atom structure into the heterogenous context. The second structure is obtained by splitting all symmetric idempotent relations. Finally, we define a structure representing the direct and the enlarged sum of two algebras.

3.1 The Structure HomNA

In this section we define the structure HomNA. Recall that this structure resembles the atom structure of a relation algebra. The abbreviation NA stands for nonassociative. If condition **(b)** of **Theorem 7** is not satisfied, then the composition in the complex algebra is not associative, and hence, the use of this name.

The following structures and operations were implemented in the programming

language Haskell. Therefore, we present the structures in a Haskell-pseudocode, i.e., using constructors and tuple notation to represent structures.

Definition 20. *The structure* `HNA(name,n,s,i,cyc,isRA)` *is called a HomNA where*

- `n` *is the number of atoms,*

- `s` *is the number of symmetric atoms, i.e.,* $1 \leq s \leq n$,

- `i` *is the index of the structure within all possible structures with* `n` *atoms and* `s` *symmetric atoms,*

- `cyc` *is the ternary relation for composition given by cycles (cf. [9]),*

- `isRA` *is true iff the structure is an integral relation algebra.*

As a representation of the atoms we use natural numbers $1, \ldots, n$ with the following conventions. The identity, which is an atom in an integral structure, will always be atom 1. The atoms $1, \ldots, s$ are symmetric, and the remaining atoms $s + 1, \ldots, n$ are non-symmetric with $m^{\smile} = m + 1$ if $m - s$ is odd and $m^{\smile} = m - 1$ if $m - s$ is even. Consequently, $n - s$ must be even.

The Boolean part of the complex algebra of an HomNA is defined as usual, and the converse operation on sets is defined componentwise. In the following we want to briefly explain how to compute composition in the complex algebra. First of all, the ternary relation `cyc` is given by cycles [9]. Recall that a cycle is a triple of atoms $[x, y, z]$ representing up to six triples, i.e.,

$$[x, y, z] = \{(x, y, z), (x^{\smile}, z, y), (z, y^{\smile}, x), (y^{\smile}, x^{\smile}, z^{\smile}), (y, z^{\smile}, x^{\smile}), (z^{\smile}, x, y^{\smile})\}.$$

36

Furthermore, we omit the cycles involving the identity, i.e., the cycles $[x, 1, x]$ for every atom x. cyc can now be computed as the union of the identity cycles and the cycles of the structure.

We want to illustrate the whole concept by an example. Consider the HomNA HNA(4,2,98,{[2,2,2],[2,2,3],[3,3,3]},True). This structure has 4 atoms with the following converse operation

$$1^\smile = 1, \ 2^\smile = 2, \ 3^\smile = 4, 4^\smile = 3.$$

Expanding the set of cycles we get

$$\begin{aligned}
\texttt{cyc} = \{&(1,1,1), (2,1,2), (2,2,1), (1,2,2), (3,1,3), (4,3,1), (1,4,4), (4,1,4),\\
&(3,4,1), (1,3,3), (2,2,2), (2,2,3), (2,3,2), (3,2,2), (2,2,4), (2,4,2),\\
&(4,2,2), (3,3,3), (4,3,3), (3,4,3), (4,4,4), (3,4,4), (4,3,4)\}.
\end{aligned}$$

Composition in the complex algebra is now defined by

$$X;Y := \{z | x \in X, y \in Y, (x, y, z) \in \texttt{cyc}\}.$$

Now, consider the sets of atoms $A = \{2, 3\}$ and $B = \{2\}$. Then we have

$$A^\smile = \{2^\smile, 3^\smile\} = \{2, 4\}, \ A;B = \{2;2\} \cup \{3;2\} = \{1, 2, 3, 4\}.$$

3.2   The Structure SplitNA

In this section we want to introduce the structure SplitNA. A SplitNA is generated by splitting all symmetric idempotent relations of a HomNA. For each of the additional objects of this category according to the embedding sketched in Section 3.1 a HomNA

37

is computed and stored in the new structure. In addition, for each heterogeneous hom-sets, i.e., the sets of relation with different source and target, the number of atoms is computed. For each such number its extension, i.e., the original relation, is stored.

Definition 21. *The structure* SNA(root,objs,atoms) *is called a SplitNA where*

- root *is a HomNA (the root of this structure),*

- objs *is a set of HomNAs representing the objects,*

- atoms *is a function mapping hom-set to a list of relations of the root object.*

Notice that in the actual implementation the elements of objs are in fact just triples consisting of n, s and i. This information together with the enumeration algorithm implemented is sufficient to identify the corresponding HomNA.

We illustrate this structure by an example. The whole SplitNA including the function atoms is visualized in the Diagram 3.1:
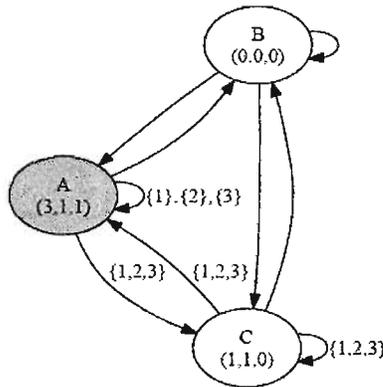

Figure 3.1: The Structure SplitNA.

We start with the HomNA with n = 3, s = 1 and i = 1 as the root object. This algebra has exactly three symmetric and idempotent relations: the identity, the empty and the universal relation. The identity generates the original algebra as an object of SplitNA. The other two relations generate HomNAs with n = 0, s = 0 and i = 0 and n = 1, s = 1 and i = 0, respectively. Therefore *objs* = [(0, 0, 0), (3, 1, 1), (1, 1, 0)] and we indicate each object by its index in *objs*. In this case, *atoms* = [((0,0),[]),((0,1),[]),((0,2),[]), ((1,0),[]),((1,1),[[1],[2],[3]]),((1,2),[[1,2,3]]), ((2,0),[]),((2,1),[[1,2,3]]),((2,2),[[1,2,3]])].

For example, there are no atoms between the objects (0,0,0) (index 0) and (3,1,1) (index 1). On the other hand, there is exactly one atom between (1,1,0) and itself. This atom is the the union of the atom 1,2 and 3 of the original relation algebra, i.e, the object (3,1,1).

3.3   CombineNA Structure

In this section we introduce the structure CombineNA. This structure implements the directed as well as the enlarged sum of two algebras as defined in Section 2. Therefore, it consists of a list of inner NAs, i.e., each element is a HomNA, SplitNA or CombineNA. Furthermore, it indicates whether the hom-sets between objects of different inner NAs has 0 or 1 atom. The structure is visualized in Figure 3.2. In this example there are two NAs. NA#0 is a HomNA, and NA#1 is a SplitNA. The CombineNA was generated by specifying that there should be 0 atoms in the hom-sets between objects of different inner NAs (dashed arrows). i.e., that we use the direct sum construction.
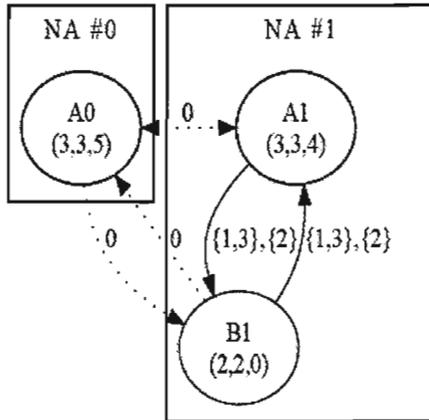
Figure 3.2: CombineNA Structure.

Composition of relations within one of the inner structures of a CombineNA is computed within that inner structure. If one of the parameters of a composition is a relation between inner structures, the result is uniquely determined by the axioms of integral nonassociative relation algebras.

Chapter 4

Implementation

4.1 Overview

In this chapter, we present the implementation of our work. In general, the system is capable of doing the following three tasks.

1. Compute all relation algebras with n atoms and s symmetric atoms. This works using the tables on the disk. We first divide the table entries into two parts: non-isomorphic NAs and isomorphic NAs. For isomorphic NAs entries, we just need to check isRA property for the first element in the isomorphic list, and the rest is filled automatically. Conversely, non-isomorphic NAs entries have to check the isRA property all the time.

2. Provide a user interface in which a user can design a basis using the three structures from Chapter 3. During this process the structure, i.e., how the basis is built, is kept.

3. Export a basis as a heterogeneous algebra suitable to be used by another program. The algebras are stored in XML. This format can be used by the ReAlM system [1]. The additional information of how the basis was constructed is not stored.

The system has been developed using the functional programming language Haskell. A Haskell program consists of a collection of modules. A module defines a collection of values, datatypes, type synonyms, classes, etc. We will discuss how each module of the system relates to our work. Figure 4.1 shows the dependencies between the modules of the system. The user interface has been developed by using GTK and
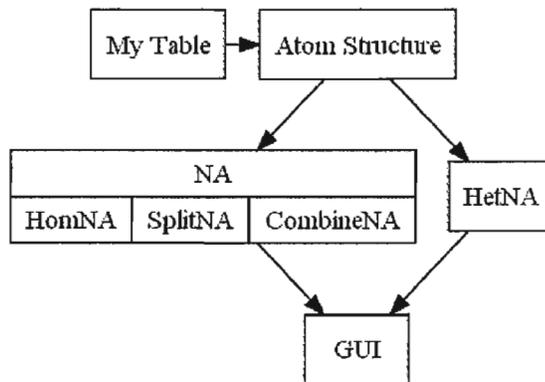

Figure 4.1: The Relationship of Modules.

the Haskell library Gtk2Hs. The graphic has been generated by Graphviz which is a graph visualization software. Throughout this chapter we assume that the reader is familiar with the language Haskell and its standard libraries.

4.2 Atom Structure Module

The `Atom Structure Module` groups a set of related functionalities into a single package and manages them. It exports some of these resources and makes them available for other modules, e.g., `HomNA`, `HetNA`.

In Chapter 2, we introduced the atom structure, which is a relational structure on a set of atoms of a finite relation algebra. In our module, atoms are integers from 1 to n, where 1 is the identity. The atoms $1, \ldots, s$ are symmetric, and the remaining atoms $s+1, \ldots, n$ are non-symmetric with $m^\smile = m+1$ if $m-s$ is odd and $m^\smile = m-1$ if $m-s$ is even. Consequently, $n-s$ must be even, in order to obtain a well-defined converse operation.

We also define three data types:

```
type Ternary = (Int,Int,Int)
type Cycle = [Ternary]
type IsoMap = IORef (M.Map (Int,Int) [[Int]])
```

As a representation of the atoms we use the integer numbers; therefore `Ternary` is a triple of `Int`. Recall that a cycle is a triple of atoms, which are related by the property of Theorem 5. We use a list of `Ternary` to represent a `Cycle`. `IsoMap` is an `IO Reference` that helps to access the data from files. It makes the data on disk available to other programs that might be reading it concurrently. Given n and s, the system will load the list of [Int] from a file, each [Int] represents an isomorphism.

With the converse operation and the type `Cycle`, we can define the cycle structure by its definition.

```
cycleSet :: Eq a => (a->a) -> (a,a,a) -> [(a,a,a)]
```

43

```
cycleSet f (x,y,z) = nub [(x,y,z),(f x,z,y),(y,f z,f x),
                          (f y,f x,f z),(f z,x,f y),(z,f y,x)]
```

In order to determine whether a given atom structure leads to a relation algebra, we need to implement the two properties mentioned at the end of Chapter 2.

```
isRA :: (Num a, Enum a) => (a->a) -> a -> [(a,a,a)] -> Bool
isRA f n na = (isIntegral f n na) && (and
              [all (existU f n na) (match f t1 t2)| t1<-na, t2<-na])
```

The isIntegral function implements property (a) from Theorem 7.

```
isIntegral :: (Num a, Enum a) => (a->a) -> a -> [(a,a,a)] -> Bool
isIntegral f n c = and [any (\w -> cycElem f (x,w,y) c) [2..n]
                   | x<-[2..n], y<-[2..n], x/=y]
```

The existU function implements property (b) from Theorem 7.

```
existU :: (Num a,Enum a) =>
          (a->a) -> a -> [(a,a,a)] -> (a,a,a,a,a) -> Bool
existU f n c (v,w,x,y,z) = any (\u -> (isIn (w,y,u) c) &&
                                       (isIn (v,u,z) c)) [2..n]
          where isIn = cycElem f
```

Recall that a homogeneous relation algebra can be seen as a heterogeneous relation algebra with just one object. In HomNA module, atoms are simply integers; however each atom has a source object and a target object in HetNA module. Since atom structures are a commonly used module in our system, all functions are implemented polymorphically, i.e., they are universally quantified in some way over all types.

For instance, function `cycleSet` takes a polymorphic function f as a parameter. This function represents the converse operation of the algebra. In the case of a HomNA atoms are simply integers and one might pass function `converse` applied to the numbers n and s. On the other hand, if we work with any of the heterogeneous structures, the atoms are triples consisting of source and target object and the number of the atom so that we have to use a different converse operation.

4.3 NA Structure

The implementation of the modules `HomNAs`, `SplitNAs`, and `CombineNAs` follow the same pattern. There is only a difference in the way they are organized. For example, we are able to represent atoms by integers in HomNA structure, i.e., a relation is an element of the datatype (`Set Int`). We have to distinguish each object and represent atoms by adding a source object and a target object in the SplitNA structure, i.e., an atom is of the form (`Int, Int, Int`) where the first two integers indicate the objects and the third integer the actual atom. Consequently, a relation is of type (`Int, Int, Set Int`). In CombineNA structure, the source object and the target object are represented by a pair of integers instead, i.e., ((`Int, Int`), (`Int, Int`), `Set Int`). If the source object and target object are from the same category, then the first integers in the pairs are the same. The second integer in the pair is the index of objects in the category. So that we declare a new type called `NA`, and use case expressions to handle the differences.

There are three ways to create an NA. We can create an NA by being given a HomNA, a SplitNA, or a CombineNA.

```
data NA = NaH HomNA

        | NaS SplitNA

        | NaC CombineNA

    deriving (Ord,Eq,Show,Read)
```

The function `checkRA` is used to test whether a NA is a RA or not.

```
checkRA :: NA -> Bool

checkRA (NaH hna) = beRA hna

checkRA (NaS sna) = beRA (rootObject sna)

checkRA (NaC cna) = and [checkRA x|x<-naList cna]
```

4.3.1   HomNA Module

By Definition 20 in Chapter 3, we defined a new data type `HomNA` using the **data** keyword.

```
data HomNA = HNA {

    name     :: String,

    atomN    :: Int,

    atomS    :: Int,

    index    :: Integer,

    beRA     :: Bool,

    cycList  :: Cycle

} deriving (Ord,Eq,Show,Read)
```

Recall that a HomNA resembles the atom structure of a nonassociative relation algebra. By importing the **Atom Structure** module, the user only needs to provide three parameters in order to construct such a complicated structure.

46

```
createHNA :: Int -> Int -> Integer -> HomNA
```

The user can produce a new HomNA out of two given HomNAs by calling the function prodHNA.

```
prodHNA :: HomNA -> HomNA -> HomNA
```

Given a HomNA, the following operations are defined.

Table 4.1: Functions of HomNA

Nullary Operations	Unary Operations	Binary Operations
idHNA	converseHNA	unionHNA
zeroHNA	complementHNA	intersectHNA
topHNA		compositHNA

Following Definition 14 in Chapter 2, we defined a function which returns all symmetric idempotent relations for a given HomNA. We first check symmetry and idempotent then find the relations that satisfy the predicate among all relations of the given HomNA.

```
allSidHNA :: HomNA -> Set (Set Int)
allSidHNA na = filter (\r -> (converseHNA na r)==r
                        && (compositHNA na r r)==r
                   ) (allRelationsHNA na)
```

The method homToDot is a simple mechanism for translating a Haskell date type into a Dot-Format text string. The string will be used by Graphviz to create a graph representation for this Haskell date type.

47

```
homToDot :: HomNA -> String
```

4.3.2 SplitNA Module

By Definition 21 in Chapter 3, we defined a new data type `SplitNA` using the **data** keyword.

```
data SplitNA = SNA {
        rootObject    :: HomNA,
        splitObjects  :: [Object],
        atomCorresp   :: [((Int,Int),[S.Set Int])]
} deriving (Ord,Eq,Show,Read)
```

The object type is just a triple consisting of n, s and i. It is sufficient to identify the corresponding HomNA.

```
type Object = (Int,Int,Integer)
```

Recall that a SplitNA is generated by splitting all symmetric idempotent relations of a HomNA. By importing the `HomNA` module, the user can generate a SplitNA structure easily.

```
createSNA :: HomNA -> SplitNA
```

The user can produce a new SplitNA out of two given SplitNAs by calling the function `prodSNA`.

```
prodSNA :: SplitNA -> SplitNA -> SplitNA
```

Given a SplitNA, the following operations are defined.

48

Table 4.2: Functions of SplitNA

Nullary Operations	Unary Operations	Binary Operations
idSNA	converseSNA	unionSNA
zeroSNA	complementSNA	intersectSNA
topSNA		compositSNA

Since two different SplitNAs may contain the same HomNAs as objects, the user should give a name in order to distinguish one from another. Compared with the homToDot function, there is one more string parameter.

```
splitToDot :: SplitNA -> String -> String
```

4.3.3 CombineNA Module

In Chapter 3, we introduced the direct sum and the enlarged sum as unions of relation algebras. The structure CombineNA is their representation in Haskell.

```
data CombineNA = CNA {
        naList       :: [NA],
        interAtoms   :: Int
} deriving (Ord,Eq,Show,Read)
```

For each of the two versions of a sum of relation algebras, there is a corresponding operation to create an instance of a CombineNA: createComb0, createComb1.

```
createComb0 :: [NA] -> CombineNA

createComb1 :: [NA] -> CombineNA
```

Table 4.3: Functions of CombineNA

Nullary Operations	Unary Operations	Binary Operations
idCNA	converseCNA	unionCNA
zeroCNA	complementCNA	intersectCNA
topCNA		compositCNA

Given a CombineNA, the operations above are defined.

As usual, there is a function for translating a Haskell date type into a Dot-Format text string. There is an integer list as a parameter, because it will be used for displaying each member of the CombineNA in the same cluster when Graphviz creates the graph.

```
combToDot :: CombineNA -> [Int] -> String -> String
```

4.4   HetNA Module

Following Definition 9 in Chapter 2, we defined the data type HetNA as follows:

```
data HetNA = HtNA {
        nList    :: [(Int,Int,Int)],
        sList    :: [(Int,Int)],
        objNum   :: Int,
        cList    :: [Triple],
        raVal    :: Bool
} deriving (Ord,Eq,Show)
```

50

`nList` is the list of atoms where the first and the second integer denote the source and target object. `sList` is the list of symmetric atoms. Since symmetric atoms must have same source and target there is just one integer for those objects. The number of objects is represented by `objNum`. `cList` is the ternary relation for composition given by cycles. If an element of type `HetNA` is a relation algebra then `raVal` equals to true and to false otherwise.

Given an NA, we can construct a HetNA by calling `createHtNA`.

```
createHtNA :: NA -> HetNA
```

Those HomNA's operations can also be applied to HetNAs by adding type rules.

Table 4.4: Operation Functions of HetNA

Nullary Operations	Unary Operations	Binary Operations
idHtNA	converseHtNA	unionHtNA
zeroHtNA	complementHtNA	intersectHtNA
topHtNA		compositHtNA

The following function can be used to store an HetNA in an XML file:

```
saveHetNA :: HetNA -> String -> String -> IO ()
```

The first string parameter is the file name, and the second string parameter is the name of the structure.

51

4.5 MyTable Module

The system is capable of computing all integral homogeneous relation algebras with n atoms and s symmetric atoms. In order to do so, the system checks the properties of Theorem 7 for all possible sets of cycles. The result is saved in a list on the hard disc.

During the computing process, the system could be stopped at any point. If we did not save our data before the system was halted, we would have to compute the same data again. For optimization reasons, the system is designed to handle unexpected halt.

We imagine that there is a table in our system like Figure 4.5. Given n and s, there are $2^{Q(n,s)}$ possible sets of cycles where $Q(n,s) = \frac{1}{6}(n-1)[(n-1)^2 + 3s - 1]$ are described in [8]. The first column stores the indices of $2^{Q(n,s)}$ atom structures. The second column stores the Boolean values which indicates whether the atom structure leads to a relation algebra. The third column stores the list indices of atom structures that are isomorphic to the current one.

Table 4.5: MyTable

Index	isRA	Isomorphic List
1	T	[2]
2	T	[1]
3	F	[18,19,20]
⋮	⋮	⋮
$2^{Q(n,s)}$	F	[]

52

Recall that any two complete and atomic relation algebras are isomorphic to each other if and only if their atom structures are isomorphic. In our example in Figure 4.5, we already have computed the first row and it is isomorphic to the second row, so that it is not necessary to compute the Boolean value `isRA` for this atom structure again.

Our implementation does not use the same kind of table as in Figure 4.5, since such a table would become extremely large for larger numbers of atoms. We only use one byte to represent one row in the table above and we write them in sequential order.

So far we have just computed this table for $n = 6$ and $s = 2$. For further comments on the limitation of the system with respect to the value of n and s see Chapter 6.

Since there are $2^{Q(n,s)}$ possible sets of cycles, it is easy to represent the index of atom structures in $Q(n, s)$ binary form.

For example, given $n = 4$ and $s = 2$, then the set of all possible cycles is:

$$[(2,2,2), (2,2,3), (2,3,3), (2,3,4), (2,4,4), (3,3,3), (3,3,4)]$$

where each ternary in the set represents its own cycle set.

The atom structure $[(2,4,4), (3,3,3), (3,3,4)]$ has index 7, which is 0000111 in binary form.

Given an index, the `indexTo` function finds the corresponding ternary list.

```
indexTo :: Int -> Int -> Integer -> Cycle
indexTo n s i  | i<(2^len)  = [t|(b,t) <- list,b == 1]
    where list = zip (bList len (binary i)) (allCycles (converse n s) n)
          len = qNS n s
```

Given a ternary list, the `fromIndex` function finds the corresponding index number.

```
fromIndex :: Int -> Int -> Cycle -> Integer
fromIndex n s ts | ts == []  = 0
                 | ts /= []  = decimal [b|(b,t) <-(update n s ts list)]
    where list = zip (bList len (binary 0)) (allCycles (converse n s) n)
          len = qNS n s
```

We implemented the isomorphic list in the following steps: First, find the atom structures at the given index and convert the given index to a $Q(n, s)$ binary form. Secondly, find all map functions for atoms, then apply every map function on each cycle of the atom structures componentwise; after that we will receive a list of new atom structures. Finally, find the indices for each new atom structures and sort the list, so that the first element is the minimum isomorphic index.

Note that we obtain all map functions by calling `mapFunctions`. It simply combines the direct product of the symmetric atoms' permutation list and the asymmetric atoms' permutation list.

```
mapFunctions :: Int -> Int -> [[Int]]
mapFunctions n s = sList 'combine' asList [[x,x+1] | x <- [s+1,s+3..n]]
    where sList = permutations [2..s]
          asList l = concatMap (\x->foldl combine [[]] (map permutations x))
                                    (permutations l)
```

In the `MyTable` module, we define a new data type `Table` by given a handle and a start position of the file.

```
newtype Table = Tab (Handle,Int)

type RATable = MVar Table
```

We will create a new `Table` without filling the `isRA` field if no such a file exists. Otherwise we will load a `Table` from an existing file.

Haskell I/O Handle allows us to perform specific operations at any locations in the file.

The function `getRAValue` will return isRA value in the given Table at the index

```
getRAValue :: RATable -> Integer -> IO (Maybe Bool)
```

Given a Table, Handle Position, isRA Value, the function `updateRATable` updates the Table value at the file position

```
updateRATable :: RATable -> Integer -> Bool -> IO ()
```

Haskell maintains internal buffers for files. Our data may not be flushed out to the operating system until we call `closeRATable`.

```
closeRATable :: RATable -> IO ()
```

4.6 GUI

The framework of the graphic user interface is developed using Glade which is an interface designer that allows the user to graphically lay out an application window and its dialogs. Below in Figure 4.2 there is a screenshot of Glade during the design of the graphical user interface of our system. Glade saves the interface in XML files, that will be loaded by the application at runtime.

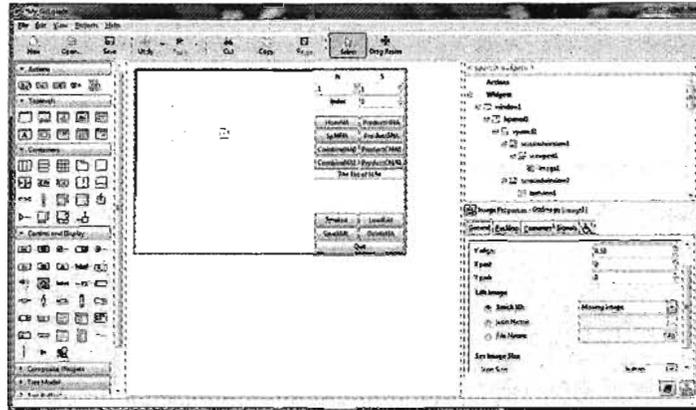The library `Gtk2Hs` is the Haskell binding for GTK and Glade.

Figure 4.2: The Framework of GUI

```
import qualified Graphics.UI.Gtk.Glade as Glade

Just xml <- Glade.xmlNew "My_GUI.glade"

window <- Glade.xmlGetWidget xml Gtk.castToWindow "window1"
```

The Haskell code above shows how to import and use the user interface designed in Glade. First, we have to import the module Graphics.UI.Gtk.Glade. The XML file generated by Glade is loaded in the next line, and finally, the main window is created using its name used within Glade. For further details of developing and using graphical user interfaces with Glade and Haskell we refer to [12].

The system interface is made up of four main parts: the graph display area, the system message box, the internal list of algebras created so far, and the user input area.

The graph display area shows result pictures which are corresponding to the user inputs. By selecting an item in the NA list store,the corresponding picture will be displayed in the graph display area. Each NA structure has a toString method that

56

can convert the data type into the Dot language. Figure 4.3 shows the running process. The Dot language is used by Graphviz (a graph visualization software) to generate pictures. For further details we refer to [6].
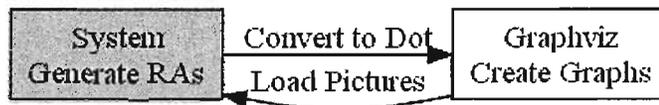

Figure 4.3: Running Process

The system message box shows system messages which helps the user to understand each step of the system operations. It displays error messages, file location messages, and user Input/Output messages.

The internal list of algebras store presents the internal process results. It allow the user to choose items in the list. Those chosen items can be used for creating a CombineNA structure, deleting, or saving as a xml file.

Chapter 5

User Manual

In this chapter, we present how to use our system. We start by giving a brief overview of the system. After that we describe how we organize the system documents. At the end, we give a thorough example to demonstrate our system.

5.1   Overview

The system was intended for generating finite integral relation algebras using Windows operating system. It provides graph representations to visualize different structures of finite integral relation algebras. The system has interrupt capability to handle unexpected halting. It allows the users to save their final results as files in xml format. The intermediate results and structure graphs are available for reloading. The system has an easy to use graphical user interface. Figure 5.1 is a screenshot of the system's graphical user interface.
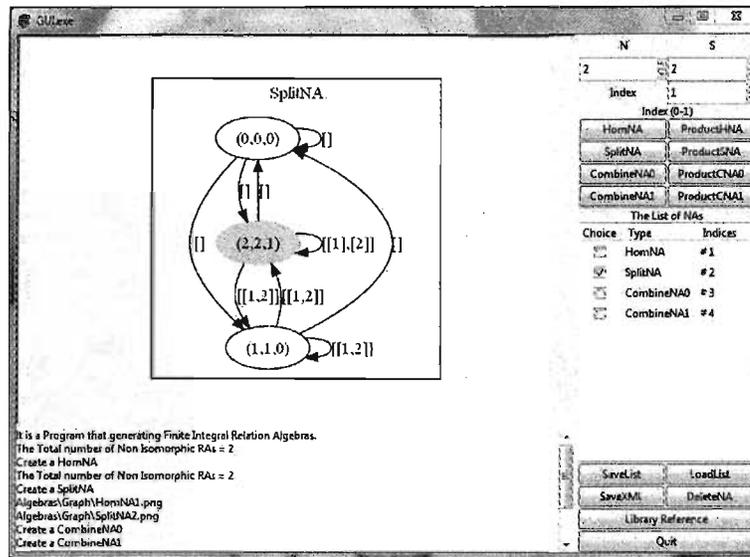
Figure 5.1: Graphic User Interface.

5.2   User Instruction

In order to run our system properly, the user has to install a GUI library which is called **Gtk2Hs**. The user should consult the gtk2hs downloads site at http://www.haskell.org/gtk2hs/download/.

In order to create a homogeneous relation algebra, the user should specify the number of atoms (n), the number of symmetric atoms (s), and the index of structure within all possible structures with n and s. There are three spin buttons for user input in the top right corner which can be seen in Figure 5.2. Two numbers n and n are positive integers which $n - s$ is even and n is greater or equal to s. The index is start from 0 to $2^{Q(n,s)}$. The system will automatically check whether the input index is in the range or not. If user inputs invalid data, the system will report an error message:
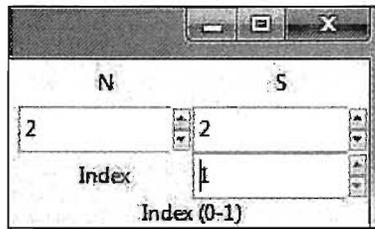
59

Figure 5.2: User Input Area.

```
"Invalid Input Data.(1<=S<=N && even (N-S) && 0<=i<2^Q(N,S))"
```

The messages will be displayed by the system message box at the bottom left corner of the GUI. Furthermore, when the user applies any kind of operations, the system message box will also display the correspondent messages. Below the user input area, there are eight operation buttons.

HomNA: create a HomNA and save in the NA list below.

SplitNA: create a SplitNA and save in the NA list below.

CombineNA0: create a CombineNA which has 0 inter atom in the hom-sets between objects of different inner NAs.

CombineNA1: create a CombineNA which has 1 inter atom in the hom-sets between objects of different inner NAs.

ProductHNA: given two HomNAs, produce a new one.

ProductSNA: given two SplitNAs, produce a new one.

ProductCNA0: given two CombineNAs which both have 0 inter atom, produce a new one.

60

ProductCNA1: given two CombineNAs which both have 1 inter atom, produce a new one.

There is a view on the right side which shows all the NAs currently stored in the system, like in Figure 5.3. In our case, there are two NAs.
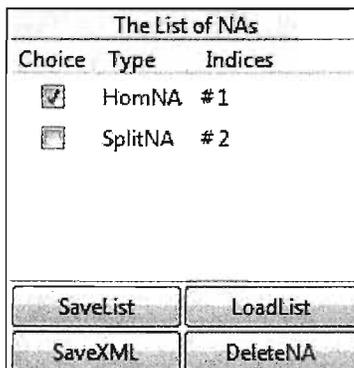
Figure 5.3: List Store Area.

The **Choice** indicates whether the NA is chosen. The **Type** means the type of NA. The **Indices** is the current NA's index amount all NAs be stored in the system. By selecting an item in the view, the corresponding picture will be displayed. The user has the freedom to choose a list of NAs which would be used for creating a CombineNA structure or would be deleted from the system.

The user also is free to choose an NA to save as an XML file, and the XML file will be used as inputs of the other system which is called RelAIM. When the user saves a NA to a file, the system will ask for the file name and the name of the output structure (the default name is **hetRel**). Not only is the user able to save the current list to a file, but also to load a list from a file. At the end, the user can close the

61

program by clicking on the **Quit** button.

5.3 File Organization

There are two root directories. The first one is **System**, the second one is **Algebras**. The **System** contains all the intermediate files are created by the system during the generation of the finite relation algebras. These files are only used by the system.

Docs: contains the Haskell source code documentation of the system.

IsoMap: contains all Isomorphisms of the given RA.

RAs: contains the RA tables.

Algebras contains all the intermediate files created by the system during the representation of the finite relation algebras. These files may be used by an external program.

DotFile: contains dot files used by an external program.

Graph: contains graph representations for NA Structures.

NaList: contains the lists of NA which are loaded by the system.

XML: contains the information about heterogeneous relation algebras.

5.4 Example

In this section we provide a thorough example for the use of the system. In this example we apply all kinds of operations and save the result in different formats.

1. **Create a HomNA:**

    First, we are going to create a homogeneous relation algebra. After selecting n=4 and s=2 in the input area on the top right corner of the graphical user interface, the proper index range, i.e., 0-127, is displayed. We select 6 as the index and press the button labeled 'HomNA'. The system creates the corresponding structure, adds it to the list of structures in the lower left corner, and displays the result as shown in Figure 5.4.
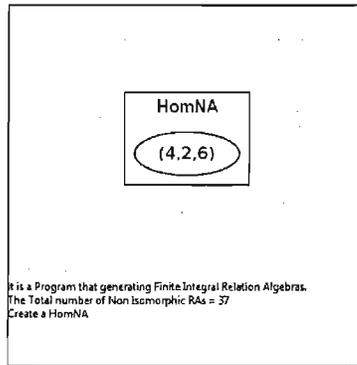

Figure 5.4: HomNA 6@(4,2)

If this is the first time ever that a homogeneous relation algebra with 4 atoms and 2 symmetric atoms is generated, the system will generate the files 'IsoMap_4_2' in the folder 'IsoMap' and the file 'RAs_4_2' in the folder 'RAs' as described in Chapter 4. Notice that this may take some time depending on N and S. In addition, the system will produce two file related to the image shown in Figure 5.4. These files are the file 'HomNA1.dot' in the folder 'DotFile', and the file 'HomNA1.png' in the folder 'Graph'.

2. Create a SplitNA:

Secondly, we create an instance of the SplitNA structure. Similar to creation of the first structure we enter N=3 and S=1 on the top right corner. Again, the proper index range (0-3) is displayed, and we select 3. This time we press the button labeled 'SplitNA'. As before the system generates the structure and displays it as a graph as seen in Figure 5.5. The system will generate similar files
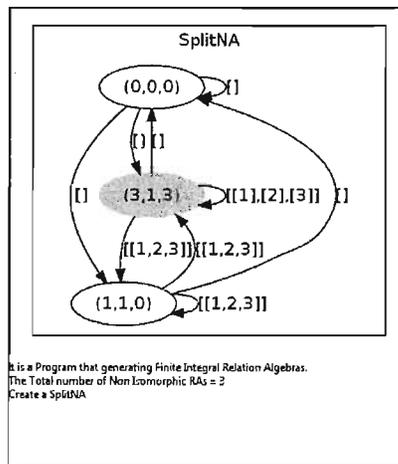

Figure 5.5: SplitNA 3@(3,1)

as in our first example on the disc. This time the files are called 'IsoMap_3_1', 'RAs_3_1', 'SplitNA2.dot' and 'SplitNA2.png', of course.

3. Create a CombineNA:

So far, we have created two algebras which are stored in the internal list of algebras. Now we use those algebras in order to build a more complex structure. First, we select the previous two structures in the list of algebras in the lower right corner of the application. Then we press the button labeled 'Com-

BineNA0', and the system will create a `CombinNA0` structure. A new algebra called 'ComBineNA0 #3' is created and added to the list. We select this new algebra and press the button labeled 'CombineNA1'. The result is shown in Figure 5.6. This time the system only generates the dot and png files for the
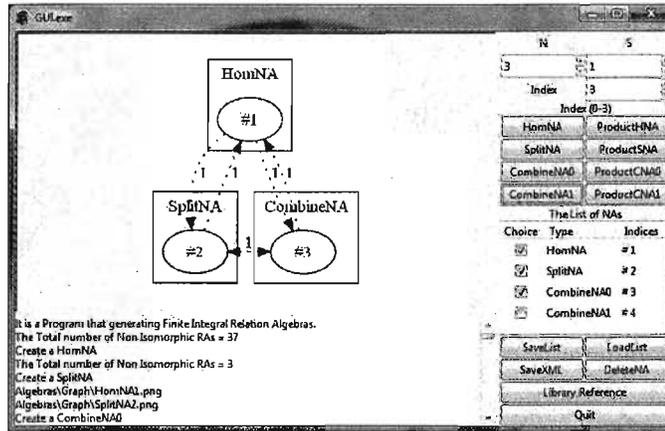

Figure 5.6: CombineNA

two new algebras in the corresponding folders.

4. **Product of two NAs:**

In order to demonstrate the product functions, the user should have already created at least two structures of the same type. For example in Figure 5.7 the algebras HomNA 1@(2,2) and HomNA 3@(3,1) have already been created, and the algebra HomNA 20503@(6,2) was obtained by using the ProductHNA function. As before, dot and png files were created during this process.

5. **Save and Load NA List:**

The system has created seven NAs in the list since its inception. To save them,
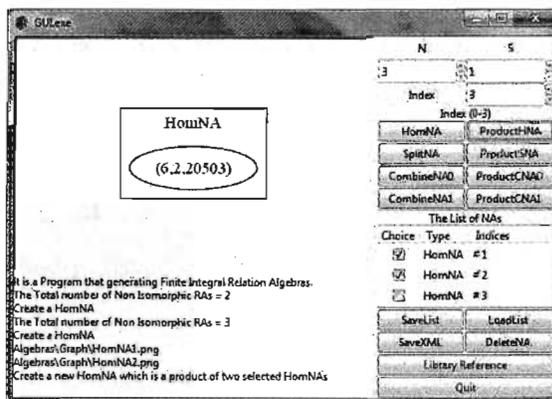
65

Figure 5.7: ProductHNA

press **SaveList** button and name your file in the upcoming dialog. In the subfolder **NaList** of the folder **Algebras** the file `NAseven` will be generated. When we need to use the list again, we should press the **LoadList** button and select the file in the upcoming dialog.

6. **Output the chosen NA as an XML File:**

We are able to save `HomNAs, SplitNAs, and CombineNAs` in an XML format. The only difference depending on the kind of structure that is stored is in the way the atoms are represented. In the heterogeneous structures, the atoms are triples consisting of source and target object and the number of the atom. (Recall Section 4.3 of the previous chapter).

The following shows the content of the file **hetH.xml** in the folder **XML** which was generated by saving the algebra HomNA #1 in XML format.

```
<hetRel name="hetRel#1" isRA="False">

<objects number="1" />
```

66

```
   <atoms source="0" target="0" number="4" symmetric="2" />
  <cycles>
   ((0,0,1),(0,0,1),(0,0,1)) ((0,0,1),(0,0,2),(0,0,2))
   ((0,0,1),(0,0,3),(0,0,3)) ((0,0,1),(0,0,4),(0,0,4))
   ((0,0,2),(0,0,4),(0,0,4)) ((0,0,3),(0,0,3),(0,0,3))
  </cycles>
 </hetRel>
```

The next example shows the content of the file 'hetS.xml' which was generated by saving the algebra SplitNA #2 in XML format.

```
<hetRel name="hetRel#2" isRA="True">
 <objects number="3" />
   <atoms source="0" target="0" number="0" symmetric="0" />
   <atoms source="0" target="1" number="0" />
   <atoms source="0" target="2" number="0" />
   <atoms source="1" target="0" number="0" />
   <atoms source="1" target="1" number="3" symmetric="1" />
   <atoms source="1" target="2" number="1" />
   <atoms source="2" target="0" number="0" />
   <atoms source="2" target="1" number="1" />
   <atoms source="2" target="2" number="1" symmetric="1" />
 <cycles>
   ((1,1,1),(1,1,1),(1,1,1)) ((1,1,1),(1,1,2),(1,1,2))
   ((1,1,1),(1,1,3),(1,1,3)) ((1,1,1),(1,2,1),(1,2,1))
   ((1,1,2),(1,1,1),(1,1,2)) ((1,1,2),(1,1,2),(1,1,2))
```

```
((1,1,2),(1,1,2),(1,1,3)) ((1,1,2),(1,1,3),(1,1,1))

((1,1,2),(1,1,3),(1,1,2)) ((1,1,2),(1,1,3),(1,1,3))

((1,1,2),(1,2,1),(1,2,1)) ((1,1,3),(1,1,1),(1,1,3))

((1,1,3),(1,1,2),(1,1,1)) ((1,1,3),(1,1,2),(1,1,2))

((1,1,3),(1,1,2),(1,1,3)) ((1,1,3),(1,1,3),(1,1,2))

((1,1,3),(1,1,3),(1,1,3)) ((1,1,3),(1,2,1),(1,2,1))

((1,2,1),(2,1,1),(1,1,1)) ((1,2,1),(2,1,1),(1,1,2))

((1,2,1),(2,1,1),(1,1,3)) ((1,2,1),(2,2,1),(1,2,1))

((2,1,1),(1,1,1),(2,1,1)) ((2,1,1),(1,1,2),(2,1,1))

((2,1,1),(1,1,3),(2,1,1)) ((2,1,1),(1,2,1),(2,2,1))

((2,2,1),(2,1,1),(2,1,1)) ((2,2,1),(2,2,1),(2,2,1))

  </cycles>

 </hetRel>
```

Our final example shows the content of the file 'hetC.xml' which was generated by saving the algebra CombineNA0 #3 in XML format.

```
<hetRel name="hetRel#3" isRA="False">

 <objects number="4" />

  <atoms source="0" target="0" number="4" symmetric="2" />

  <atoms source="1" target="1" number="0" symmetric="0" />

  <atoms source="1" target="2" number="0" />

  <atoms source="1" target="3" number="0" />

  <atoms source="2" target="1" number="0" />

  <atoms source="2" target="2" number="3" symmetric="1" />

  <atoms source="2" target="3" number="1" />
```

68

```
    <atoms source="3" target="1" number="0" />

    <atoms source="3" target="2" number="1" />

    <atoms source="3" target="3" number="1" symmetric="1" />

  <cycles>

   ((0,0,1),(0,0,1),(0,0,1))  ((0,0,1),(0,0,2),(0,0,2))

   ((0,0,1),(0,0,3),(0,0,3))  ((0,0,1),(0,0,4),(0,0,4))

   ((0,0,2),(0,0,4),(0,0,4))  ((0,0,3),(0,0,3),(0,0,3))

   ((2,2,1),(2,2,1),(2,2,1))  ((2,2,1),(2,2,2),(2,2,2))

   ((2,2,1),(2,2,3),(2,2,3))  ((2,2,1),(2,3,1),(2,3,1))

   ((2,2,2),(2,2,1),(2,2,2))  ((2,2,2),(2,2,2),(2,2,2))

   ((2,2,2),(2,2,2),(2,2,3))  ((2,2,2),(2,2,3),(2,2,1))

   ((2,2,2),(2,2,3),(2,2,2))  ((2,2,2),(2,2,3),(2,2,3))

   ((2,2,2),(2,3,1),(2,3,1))  ((2,2,3),(2,2,1),(2,2,3))

   ((2,2,3),(2,2,2),(2,2,1))  ((2,2,3),(2,2,2),(2,2,2))

   ((2,2,3),(2,2,2),(2,2,3))  ((2,2,3),(2,2,3),(2,2,2))

   ((2,2,3),(2,2,3),(2,2,3))  ((2,2,3),(2,3,1),(2,3,1))

   ((2,3,1),(3,2,1),(2,2,1))  ((2,3,1),(3,2,1),(2,2,2))

   ((2,3,1),(3,2,1),(2,2,3))  ((2,3,1),(3,3,1),(2,3,1))

   ((3,2,1),(2,2,1),(3,2,1))  ((3,2,1),(2,2,2),(3,2,1))

   ((3,2,1),(2,2,3),(3,2,1))  ((3,2,1),(2,3,1),(3,3,1))

   ((3,3,1),(3,2,1),(3,2,1))  ((3,3,1),(3,3,1),(3,3,1))

  </cycles>

 </hetRel>
```

Chapter 6

Conclusions

This chapter begins with a summary of the work in this thesis and the conclusions reached. This is followed by a discussion of the areas that require further investigation and the work planned in the future.

6.1 Summary

In this thesis, we focused on generating atom structures of finite integral heterogeneous relation algebras. We were able to define three new data types and operations representing such structures. Following the theoretical work, we have implemented a Haskell library which can be used to compute atom structures of finite integral relation algebras. In addition, we have developed a system with a graphical user interface that allows users to create, visualize, and store such atom structures for later use. Except for a simple program enumerating the non-isomorphic integral homogenous relation algebras up to six atoms [9], the system developed in this thesis is the first of its kind. In particular, there is no system that generates any heterogeneous

70

structures. However, the results of our system, i.e., the algebras constructed, can be used by the system developed by Z. Ahmed in [1]. His program allows users to execute computations in matrix algebras where the coefficients are relations from a heterogeneous integral relation algebra. Recall that every algebra is equivalent to a matrix algebra.

6.2 Future Work

The current system also has its limitations. The huge amount of homogenous integral relation algebras for $n > 5$ makes it almost impossible to use any algebras of that size. In fact, the number of non-isomorphic relation algebras with $n = 6$ and $s = 6$ is already 3,849,820. For any $n > 6$ the number of algebras is still unknown [9].

Although we have accomplished a lot in this thesis, there is room for additional work, which can be built on the current implementation. Some of these ideas are presented below.

Concurrency: The computation of all atom structures with n atoms could be done using multiple threads or even multiple computers in order to gain a significant speed up. The current state of concurrency in Haskell did not allow us to implement this feature so far.

External implementation: This point is related to the first one. Instead of using the concurrency of Haskell, one could implement the search of all atom structures with n atoms in a different language. Since efficiency is crucial in that task a suitable language could be C or C++. The functionality of those programs

could be utilized within Haskell by using the foreign function interface FFI. In addition, this C or C++ program could be a parallel program.

More kinds of unions: We have implemented two kinds of unions of heterogenous relation algebras, the direct and the enlarged sum. These two structures introduce 0 and 1 atom in between the input algebras, respectively. There are examples of a basis that has larger intermediate structures. For example, the number of relations between two objects A and B in a SplitNA is usually larger than 2. These objects are HomNAs themselves so that it should be possible to create the algebra consisting of A and B and all relations between using a sum. How to create such algebras in general is unknown so far, and subject of further research.

Bibliography

[1] Z. Ahmed. Realm - a system to manipulate relations. Master's thesis, Brock University, 2009.

[2] R. Bird and O. de Moor. *Algebra of Programming*. Prentice-Hall, 1997.

[3] G. Brikhoff. *Lattice Theory*. AMS, third edition, 1995.

[4] I. Düntsch. Relation algebras and their application in temporal and spatial reasoning. *Artificial Inteligence Review*, 23:315–357, 2005.

[5] P. Freyd and A. Scedrov. *Categories, Allegories*. North-Holland, 1990.

[6] E. Gansner, E. Koutsofios, and S. North. *Drawing graphs with dot*, January 2006.

[7] S. Koppelberg. *General Theory of Boolean Algebras*. North Holland, 1989.

[8] R. D. Maddux. Finite integral relation algebras. In *Lecture Notes in Mathematics 1149*, pages 175–197. Springer-Verlag, 1985.

[9] R. D. Maddux. *Relation Algebras*, volume 150 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science, 2006.

[10] G. Schmidt, C. Hattensperger, and M. Winter. Heterogeneous relation algebras. In C.Brink, W.Kahl, and G.Schmidt, editors, *Relational Methods in Computer Science*, Advances In Computing Science, chapter 2, pages 39–53. Springer-Verlag, 1997.

[11] G. Schmidt and T. Ströhlein. *Relations and Graphs. Discrete Mathematics for Computer Scientists*. EATCS Monographs in Theoretical Computer Science. Springer-Verlag, 1993.

[12] B. O. Sullivan, J. Goerzen, and D. Stewart. *Real World Haskell*. Oreilly, first edition, November 2008.

[13] A. Tarski and S. Givant. *A Formalization of Set Theory without Variables*. Handbook of Boolean Algebras (Vol. 1). AMS, 1988.

[14] M. Winter. Relation algebras are matrix algebras over a suitable basis. Technical Report 1998-05, Universität der Bundeswehr München, November 1998.

[15] M. Winter. A pseudo representation theorem for various categories of relations. *TAC Theory and Applications of Categories*, 7(2):23–37, 2000.

[16] M. Winter. Products in categories of relations. *The Journal of Logic and Algebraic Programming*, pages 145–159, 2008.

[17] S. Zhang and M. Winter. Three structures for generating finite integral relation algebras. *PhD Program of RelMiCS/AKA*, pages 28–32, 2009.