An Abstract Algebraic Theory of $\mathcal{L}$-Fuzzy Relations for Relational Databases

Abdul Wazed Chowdhury
Department of Computer Science

Supervisor:
Dr. Michael Winter

Submitted in partial fulfilment of the requirements for the degree of Master of Science

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

BROCK UNIVERSITY

Abstract

Faculty of Mathematics and Science
Computer Science

Master of Science

An Abstract Algebraic Theory of $\mathcal{L}$-Fuzzy Relations for Relational Databases

by Abdul Wazed CHOWDHURY

Classical relational databases lack proper ways to manage certain real-world situations including imprecise or uncertain data. Fuzzy databases overcome this limitation by allowing each entry in the table to be a fuzzy set where each element of the corresponding domain is assigned a membership degree from the real interval $[0 \ldots 1]$. But this fuzzy mechanism becomes inappropriate in modelling scenarios where data might be incomparable. Therefore, we become interested in further generalization of fuzzy database into $\mathcal{L}$-fuzzy database. In such a database, the characteristic function for a fuzzy set maps to an arbitrary complete Brouwerian lattice $\mathcal{L}$. From the query language perspectives, the language of fuzzy database, FSQL extends the regular Structured Query Language (SQL) by adding fuzzy specific constructions. In addition to that, $\mathcal{L}$-fuzzy query language $\mathcal{L}$FSQL introduces appropriate linguistic operations to define and manipulate inexact data in an $\mathcal{L}$-fuzzy database. This research mainly focuses on defining the semantics of $\mathcal{L}$FSQL. However, it requires an abstract algebraic theory which can be used to prove all the properties of, and operations on, $\mathcal{L}$-fuzzy relations. In our study, we show that the theory of arrow categories forms a suitable framework for that. Therefore, we define the semantics of $\mathcal{L}$FSQL in the abstract notion of an arrow category. In addition, we implement the operations of $\mathcal{L}$-fuzzy relations in Haskell and develop a parser that translates algebraic expressions into our implementation.

Acknowledgements

I owe my deepest gratitude to my supervisor Dr. Michael Winter for everything, everything. Besides him, I would like to thank everyone in the advisory committee and all my friends in the department.

And love to Durdana Rahman, my wife, for her continuous support throughout the study.

Contents

List of Figures

To my father
Golam Mostafa Chowdhury

Chapter 1

Introduction

1.1 Introduction

In mathematics the theory of sets and relations forms the basis of many other mathematical concepts. While set theory studies collections of entities (commonly known as elements) and various operations on those collections, the theory of relations deals with the association between individual elements. The importance of organizing a huge collection of data on the basis of their relationship has been proven to be both mathematically sound and practically useful. The term *database* better describes such an organization of data. Our research concentrates on defining the semantics of a language for $\mathcal{L}$-fuzzy databases. This particular generalization of classical databases overcomes its shortcomings in handling real-world problems like imprecision in data.

1.2 Databases

Over the last decade databases have become an indispensable part of all kinds of software applications. The contact list on a cell phone probably is the most common example of a database that we deal with in our daily life. Databases might be as big as the collection of client information of a bank or a global email service provider like Gmail or Yahoo to the central governmental database of a country.

In computing, a *database* is an organized collection of information facilitating its easy storage, management, and retrieval. Databases are usually characterized by the organizational approach they follow. There are relational approaches, hierarchical approaches, object-oriented approaches, and so on. Among them the relational model is the one most widely used nowadays.

1.2.1 Relational Database

In a relational database data are stored and presented in tables with rows and columns. Each column of a table refers to an attribute of an object (also called an entity) whereas a row is considered to be the object having a number of attributes. A table can be thought of a relation in the sense that it is a collection of objects of the same type. For example the contact records on someone's phone might be organized in a table as follows. Here we have records (or tuples) of four persons containing their names and phone numbers.

Name	**Phone no.**
John	+12222222222
Kevin	+13333333333
Linda	+14444444444
Richy	+15555555555

In a relational database a Data Definition Language (DDL) is used to build and modify the structure of the data and a Data Manipulation Language (DML) is used to populate that structure and fetch useful information. The most typical example of a database language is Structured Query Language (SQL). SQL comes with a DML as well as a DDL component. SQL-DDL contains statements for defining database structure. Examples of such statements include CREATE to create a table, ALTER to modify the structure of table, etc. DML statements, on the other hand, are used for managing records. For instance, a SELECT statement retrieves information from a database, an INSERT statement adds a record to an existing table, and so on.

1.2.2 Missing or Imprecise Data in Relational Databases

Well-structured relational databases can manage almost all kinds of information. In particular, missing information are usually denoted by *null* in such a database. For example, presently almost all cellular phones allow users to enter additional information like someone's birth-date, address, company, etc., while creating a new contact. But many of such contacts contain only name and phone numbers. Therefore, *null* can be used to represent information which are absent in such a contact list. Here, as shown below, we have two derived attributes: *Distance* obtained from Address, and *Age* from Date of Birth.

Name	**Phone no.**	**Address**	***Distance***	**Date of birth**	***Age***
John	+12222222222	19 York street	3	*null*	*null*
Kevin	+13333333333	*null*	*null*	*null*	*null*
Linda	+14444444444	*null*	*null*	March 15, 1970	45
Richy	+15555555555	125 Perfect road	6	August 29, 1998	17

Also, there are a number of real-world scenarios where data might be imprecise or uncertain. Unfortunately classical relational databases lack a proper way to handle that. For example, someone's phone might have contact records where dates of birth and addresses of his friends are missing but he can very well tell if Kevin is *young* or *old*, whether soccer or hockey is *more popular* among his friends, who lives *closest* to him, and so on. Using this kind of language expressions is a common phenomenon of our daily life. In regard to a relational database, we have only two alternatives to represent such an expression: either use a *null* or guess a value appropriate for a label. For example, if we only know that Kevin is *young* but not his precise age, then using a *null* does not provide any information about Kevin's age, nor does it preserve the information already known that Kevin is *young*. In the second approach we pick a random value that we consider *young* which actually does provide some information but is most likely to be wrong. In order to handle such imprecision in data the concepts of fuzzy relational databases and its query language, Fuzzy Structured Query Language (FLSQL), have emerged.

1.2.3 Fuzzy Database

In a fuzzy relational database (or simply fuzzy database) we can store a fuzzy set for each field whereas a classical relational database allows only a single value per field. Fuzzy sets can be distinguished from their classical counterparts by their characteristic function. For a classical set, a characteristic function also called a membership function, has the form $\chi_B : A \to \mathbb{B}$ where $\mathbb{B}$ is the set of Boolean values, i.e., $\{True, False\}$. However, as mentioned before such a Boolean function is very inappropriate in managing vagueness in data which requires us to have more alternatives than simply *yes* and *no*. In order to handle such situations Lotfi A. Zadeh (1965) [36] introduced the notion of fuzzy sets. He changed the standard characteristic function to map to the unit interval of real numbers: $\chi_B : A \to [0 \ldots 1]$ where 1 indicates full membership, 0 not a member at all with all other values in between representing how strongly an element belongs to a set. For instance let's say we have a group of students {Russel, Peter, Andy, James} and their marks {62,95,46,75} in some course. Now if we are interested to make a list of good students, then that list can be expressed as a fuzzy set where each of the students would be there with different degrees of membership. A membership function for such a set could be:

$$\chi_{good}(sname) = \begin{cases} 0 & \text{iff } marks_{sname} < 50, \\ \dfrac{marks_{sname} - 50}{50} & \text{iff } 50 \leqslant marks_{sname} \leqslant 100 \end{cases}$$

Therefore, we get this: {0.24/Russel, 0.9/Peter, 0/Andy, 0.5/James}. It is evident that Russel, Peter, and James have passed the course and Peter can be considered better than Russel and James for this course. In contrast, a classical representation of this list, which would be {1/Russel,

1/Peter, 0/Andy, 1/James}, provides no information other than just pass and fail. Representing data of different ranges by the unit interval thus provides a better way for working with inexactness in data.

Note that fuzzy sets can wisely be used to provide mathematical interpretation of language expression (also called linguistic labels) like *young*, *popular*, etc. in a fuzzy database. As an example, *young* can be represented by a fuzzy set with the following characteristic function:

$$\chi_{young}(x) = \begin{cases} 0 & \text{iff } x > 25 \text{ years}, \\ \frac{25 - x}{10} & \text{iff } 15 \text{ years} \leqslant x \leqslant 25 \text{ years}, \\ 1 & \text{iff } 15 \text{ years} < x \end{cases}$$

1.2.4 Querying a Database

An SQL query on a relational database produces a list of tuples that satisfies the condition. Such a query has three major parts: SELECT, FROM and WHERE. SELECT clause is used to choose one or more attributes of our interest. In the FROM clause we specify the tables that we want to retrieve data from. Finally, the WHERE clause which is the only optional part, tells about the condition that all resultant tuples should satisfy. We continue using the contact list example above for further demonstrations. At this point we are interested to know about only those people who have an age smaller than or equal to 20. The result shows that only the record for Richy fulfils the condition.

SELECT Name, Phone no., Age
FROM Contacts
WHERE Age $\leqslant$ 20

Name	**Phone no.**	***Age***
Richy	+15555555555	17

In contrast to SQL, a query in FSQL results in a list of tuples each with a degree value up to which it satisfies the condition. In order to make it possible working on imprecise data in a database, the language of FSQL adds to the regular SQL statements operations which are specific to fuzzy sets. For example, in FSQL each of the comparison operations $=$, $\leqslant$, and $<$ of regular SQL is available in two forms: a possibility operation $F=$, $F\leqslant$, $F<$, and a necessity operation $NF=$, $NF\leqslant$, $NF<$. In our contact-list example, a possibility comparison of *age F= $Young* computes the degree that someone possibly has an age which is considered to be young. A necessity comparison of *age NF= Young*, on the other hand, computes a degree up to which every potential age a person could have is considered to be young.

Moreover, FSQL provides additional constructs to specify a threshold (THOLD) for the minimum degree up to which a resultant tuple should satisfy the condition. It also allows the use

t-norms and t-conorms in place of min and max while computing logical connective *and* and *or*. In fuzzy theory Triangular Norm (t-norm) and Triangular Conorm (t-conorm) are binary operations that are used to compute intersection and union of two fuzzy sets respectively. It is to be noted that minimum function is the largest t-norm whereas maximum function is the smallest t-conorm.

A linguistic label in FSQL is always preceded by a $ sign. A fuzzy database uses a meta-database to store fuzzy-specific constructs such as characteristic functions for linguistic labels, t-norms and t-conorms, etc. Here is a fuzzy version of the contact list database where we keep only the relevant fields because of our interest on *Age*. We use the same query before except that $\leqslant$ is changed to $F\leqslant$ and a threshold of 0.4 is enforced. This query when executed returns only those tuples that satisfy the condition and obtain a membership degree greater than or equal to 0.4.

Name	**Phone no.**	***Age***
John	+12222222222	$Old
Kevin	+13333333333	$Young
Linda	+14444444444	45
Richy	+15555555555	17

SELECT Name, Phone no., Age

FROM Contacts

WHERE Age F$\leqslant$ 20 THOLD 0.4

Name	**Phone no.**	***Age***
Kevin	+13333333333	$Young
Richy	+15555555555	17

1.3 $\mathcal{L}$-Fuzzy Database

It is evident that the unit interval of $[0\ldots1]$ is linearly ordered meaning that for any $x, y \in [0\ldots1]$, we have either $x \leqslant y$ or $y \leqslant x$. Elements of an ordered set for which the ordering relation ($\leqslant$ here) holds commutatively are called "comparable". This property implies that we are always able to tell for any two elements a and b which is more in a given fuzzy set B by comparing $\chi_B(a)$ and $\chi_B(b)$. But in a number of real world applications this might not be the case. As an example we assume that Kevin wants to buy a cell phone and his primary concern is the size of a phone's internal memory as there are tradeoffs between memory sizes and their associated costs. We consider to model the memory size of different cell phones as a fuzzy set. The degree of membership of a given memory size in the set of *good* sizes indicates how well that particular size serves our requirement. An internal memory of 64GB might be *good* because of greater storage capacity but not so good because of the extra cost incurred. 16GB of memory, on the other hand, can be considered *good* because of the standard phone price, but not good enough as approximately half of the memory is occupied by the phone operating system which results in limited storage capacity available for the user. For these reasons both memory

sizes should be in the fuzzy set of *good* memory sizes up to a certain degree. However, it seems hard, or even impossible or sometimes unwanted, to decide which memory size is better, i.e., we do not want that $\chi_{good}(16GB) \leqslant \chi_{good}(64GB)$ or vice versa. So we become interested in a set where there might be incomparable elements alongside comparable ones. Such a set is called a partially ordered set or poset. An useful example of poset might be the set of all divisors of 24 which is $\{1, 2, 3, 4, 6, 8, 12, 24\}$ with divisibility ($|$) as the induced relation. Note that neither 3 divides 4, nor does 4 divide 3 which means that 3 and 4 are incomparable. The two most common operations on a poset is the greatest lower bound or meet ($\wedge$) and the least upper bound or join ($\vee$). A lower bound of two elements x and y of a poset $(P, \leqslant)$ is an element $z \in P$ so that $z \leqslant x$ and $z \leqslant y$. In the previous example of divisors, 4 and 6 has two lower bounds 1 and 2. Therefore, their meet would be the greatest element of all their lower bounds which is 2 here. Dually an upper bound for $x, y \in P$ is an element z so that $x \leqslant z$ and $y \leqslant z$. It is clear that 4 and 6 in the example above have two upper bounds $\{12, 24\}$. As a result, the least upper bound or join of 4 and 6 is 12. A poset in which a meet and a join exist for any two elements is called a lattice. Clearly the poset $(\{1, 2, 3, 4, 6, 8, 12, 24\}, |)$ is a lattice. However, $(\{2, 3, 4, 10, 12, 20, 25\}, |)$ cannot be a lattice because $\{12, 20\}$ doesn't have a join as well as $\{2, 5\}$ doesn't have a meet. A lattice $\mathcal{L}$ might contain an element 1 such that for all $x \in L$, $x \wedge 1 = x$ and $x \vee 1 = 1$. Such an element is called the top element or upper bound of L. The dual of top element is called the bottom of $\mathcal{L}$, expressed as 0, such that for all $x \in \mathcal{L}$, $x \wedge 0 = 0$ and $x \vee 0 = x$. Note that top and bottom elements of a lattice are unique; however, if they exist.

In our study we are interested in Goguen (1967)'s generalization of fuzzy sets to $\mathcal{L}$-fuzzy sets where $\mathcal{L}$ stands for an arbitrary complete Brouwerian lattice. The characteristic function for $\mathcal{L}$-fuzzy sets has the form $\chi_B : A \to \mathcal{L}$, i.e., elements have membership degrees chosen from a lattice $\mathcal{L}$ with a meet ($\wedge$) and a join ($\vee$) operation and a least element 0 and a greatest element 1. As because understanding of poset, lattice, $\mathcal{L}$-fuzzy relations and their operations requires explanations, we defer their details to Chapter 2.

Therefore, an $\mathcal{L}$-fuzzy database can be said to generalize a fuzzy database by allowing us to store an $\mathcal{L}$-fuzzy set for each column of a table. Such a database thus can appropriately handle imprecise as well as incomparable data.

1.4 Motivation

In this research, we are motivated to have an abstract algebraic theory for $\mathcal{L}$-fuzzy relations where we can prove all the formal properties and operations of $\mathcal{L}$-fuzzy relations as well as of classical relations along with the associated axioms. Category theory is a good choice to start with as a large collection of concepts and theorems have already been available based on categorical axioms only. A Dedekind category essentially forms a suitable framework to deal with

binary relations. Unfortunately it is very weak in expressing crispness. The concept of crispness, although quantitative opposite to fuzziness, is very important in the world of fuzziness. A crisp relation is an $\mathcal{L}$-fuzzy relation in which each membership value is either 0 (the least element of L) or 1 (the greatest element of L). These relations provide a natural way to embed regular relations in the fuzzy world and so should be expressed properly by any algebraic theory of $\mathcal{L}$-fuzzy relations. The theory of arrow categories extends Dedekind categories by introducing two arrow operations that provides an appropriate way of dealing with crispness.

The main purpose of this thesis is to define the semantics of $\mathcal{L}$-fuzzy query language $\mathcal{L}$FSQL using the theories of arrow categories. In doing so, we have implemented the operations of $\mathcal{L}$-fuzzy relations in Haskell and developed a parser that would translate algebraic expressions into our implementation. Apart from this, our work uses the relation algebraic system RelView for demonstration purposes.

Prior to discussing our implementation, we recall all the relevant mathematical preliminaries with examples in Chapter 2. It starts with the basic definition of classical relations, proceeds toward the $\mathcal{L}$-version of it, discusses the different categorical concepts available for relations, and ends up with the theory of Arrow categories. Chapter 3 solely discusses the semantics of $\mathcal{L}$FSQL followed by our concrete implementations in Chapter 4. We give our concluding remarks and some useful directions for future works in Chapter 5.

1.5 Main Contribution of the Thesis

The purpose of this section is to clearly identify the unique contribution of this research in contrast to E. Adjei's work [1]. The following points summarize this.

- In Chapter 3 we present an abstract semantics for the $\mathcal{L}$-fuzzy query language, $\mathcal{L}$FSQL in arrow categories and this can be considered to be the main contribution of this thesis.
- As a part of the realization of the semantics provided here, we have implemented the concrete arrow category of $\mathcal{L}$-fuzzy relations between finite sets in Haskell.
- Last but not the least, we have developed a parser that can be used to translate relation algebraic terms into our concrete implementation and a structure to execute those terms using relational instances.

Chapter 2

Mathematical Preliminaries

This chapter reviews the different mathematical concepts that are crucial to our study. As relations and their algebra have been used throughout our research, this chapter begins with the basic definition of classical relation, describes its different properties and operations before we switch to the fuzzy version of it. As mentioned earlier we use $\mathcal{L}$-fuzzy relations to interpret $\mathcal{L}$-fuzzy databases while defining semantics for its query language $\mathcal{L}$FSQL. Therefore, $\mathcal{L}$-fuzzy relations and its calculus are accounted elaborately in this chapter. It was also mentioned that the theory of arrow categories forms a suitable algebraic framework for $\mathcal{L}$-fuzzy relations. So, we would include an in-depth demonstration of this theory with all the required proofs as well as appropriate examples whenever and wherever applicable. The ideas and concepts included here would suffice to start with the semantics of $\mathcal{L}$FSQL in the very next chapter.

2.1 Classical Relations

In mathematics and computer science, relating entities of different types entails great results. The best example might be a relational database where we organize information according to their relationship. In the contact list example from Chapter 1, a contact record is nothing but relating a *Name* entity with a *Phone no*. entity. Below there are some of the mathematical statements demonstrating how we relate entities usually.

$$x \neq y,\ 10 < 20,\ 2 \in \mathbb{N},\ 5 \geqslant 5,\ \mathbb{N} \subseteq \mathbb{Z},\ 10\,kg = 4.5\,lb,\ \sqrt{2} \approx 1.414$$

In each of these statements a single symbol, used infix, expresses the relationship between the quantities on either sides. These are examples of binary relations as they associate a pair of objects.

In mathematics the operation of *Cartesian product* forms ordered pairs from two sets of objects. By the term "ordered" we mean that the first element of a pair comes from the first participating set while the second from the second set. If A and B are two sets then their *Cartesian product* is defined as

$$A \times B = \{(a, b) \mid a \in A \text{ and } b \in B\}$$

In the contact list example below, although the *Cartesian product* of *Name* and *Phone no.* produces four pairs, only two of them state the relationship accurately. These are:
i. (John, +12222222222), and ii. (Kevin, +13333333333).

Name	**Phone no.**
John	+12222222222
Kevin	+13333333333

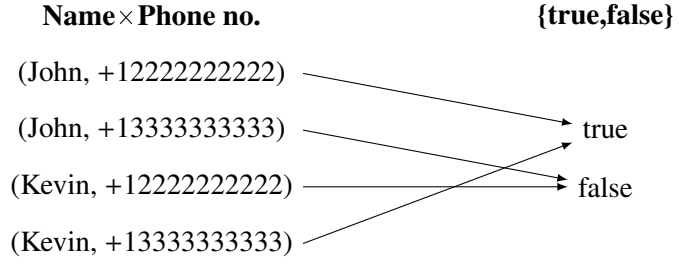
Name = {John, Kevin}
Phone no. = {+12222222222,+13333333333}

Name × Phone no. = { (John, +12222222222), (John, +13333333333), (Kevin, +12222222222), (Kevin, +13333333333)}

Therefore, a binary relation R between two sets A and B can be defined in either of the following two ways.

1. **As a subset of Cartesian product**: $R \subseteq A \times B$.
   It means that R contains only those ordered pairs that have the association induced by R. If we think of R as contact records in our example, i.e., $R \subseteq Name \times PhoneNo.$, then we get the following.
   R = {(John, +12222222222), (Kevin, +13333333333)}
   It is well-known that *power set* of a set consists of all the possible subsets. Therefore, there might be as many as $2^{|A|*|B|}$ different binary relations from A to B.

2. **As a Boolean function**: $R : A \times B \rightarrow \mathbb{B}$ where $\mathbb{B} = \{true, false\}$.
   This notation maps an ordered pair (a, b) to *true* if it is related by R, otherwise it is mapped to $false$. Such a function is called the *characteristic function* of the set. Therefore, a pair (a, b) has a membership degree of *true* if $(a, b) \in R$, $false$ otherwise.



We call A the source of R and B, the target. So, we write $R : A \to B$. To express the fact that an $a \in A$ is related to some $b \in B$ by R, we simple write aRb. To visualize relations we would prefer to use matrices as in [27]. In such a matrix source elements are set to the labels of rows and those from the target set are used as column-labels. An entry at row a and column b is 1 if aRb, otherwise it is a 0.

$$\begin{array}{c} \\ \textit{John} \\ \textit{Kevin} \end{array} \begin{array}{c} \begin{array}{cc} +12222222222 & +13333333333 \end{array} \\ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array}$$

The *domain* of a relation $R : A \to B$ is the set $\{a \in A \mid \exists b \in B : R(a,b)\}$ [31]. Dually, the *codomain* or *range* or *image* of R is the set $\{b \in B \mid \exists a \in A : R(a,b)\}$.

Note that the *Cartesian product* is not associative, neither commutative. It means that for any sets A, B, and C

- *Non-associativity*: $(A \times B) \times C \neq A \times (B \times C)$, but these two products are related by a bijective function which means that they are isomorphic. We will define bijection and isomorphism later in this chapter.
- *Non-commutativity*: $A \times B \neq B \times A$.

All the relations we have used so far associate elements from two different sets, thus called *heterogeneous* binary relations. An special version of this relates elements within a single set, i.e., *homogeneous*. As an example consider the set of divisors of 24 which is $\{1,2,3,4,6,8,12,24\}$. The divisibility relation $(|)$ on this set could be represented as follows.

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 6 \\ 8 \\ 12 \\ 24 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 6 & 8 & 12 & 24 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

FIGURE 2.1: Divisibility relation on the divisors of 24

From the above matrix it is evident that the divisibility relation $(|)$ is included in the smaller than or equal $(\leqslant)$ relation, i.e., $| \sqsubseteq \leqslant$. Note that we use $\sqsubseteq$ for inclusions on relations whereas $\subseteq$ denotes inclusions on sets. For definition of $\sqsubseteq$, please refer to the next section.

2.1.1 Set Theoretic Operations on Relations

Interpreting a relation $R : A \to B$ as a subset of $A \times B$ implies that certain set-theoretic operations be immediately applicable on relations as well. For demonstrating these operations we use sets of prime and perfect numbers. Recall that a prime is a natural number greater than 1 that is divisible by only 1 and itself. A perfect number, on the other hand, is characterized by the fact that it is a positive integer for which the proper divisors add up to itself. It is also interesting to remember the *Euclid–Euler theorem* that relates these two: if $2^p - 1$ is prime (also known as *Mersenne prime*) for a prime p, then $2^{p-1}(2^p - 1)$ is an even perfect number. To keep it simple for now we limit our first set A to the first, second, and fourth prime number and the second set B to the first two perfect numbers. For relations, we assume $R, S : A \to B$ to be the "divides" ($|$) or "is a divisor of", and the "smaller than or equal to" ($\leqslant$) relations, respectively.

$$R = \begin{array}{c} \\ 2 \\ 3 \\ 7 \end{array} \begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array} \text{ and, } S = \begin{array}{c} \\ 2 \\ 3 \\ 7 \end{array} \begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 1 \end{pmatrix} \end{array}$$

- *Union*: $R \sqcup S = \{(a, b) \in A \times B \,|\, (a, b) \in R \text{ or } (a, b) \in S\}$.

$$\begin{array}{c} \\ 2 \\ 3 \\ 7 \end{array} \begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array} \sqcup \begin{array}{c} \\ 2 \\ 3 \\ 7 \end{array} \begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 1 \end{pmatrix} \end{array} = \begin{array}{c} \\ 2 \\ 3 \\ 7 \end{array} \begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 1 \end{pmatrix} \end{array}$$

- *Intersection*: $R \sqcap S = \{(a, b) \in A \times B \,|\, (a, b) \in R \text{ and } (a, b) \in S\}$.

$$\begin{array}{c} \\ 2 \\ 3 \\ 7 \end{array} \begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array} \sqcap \begin{array}{c} \\ 2 \\ 3 \\ 7 \end{array} \begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 1 \end{pmatrix} \end{array} = \begin{array}{c} \\ 2 \\ 3 \\ 5 \end{array} \begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array}$$

- *Complement*: $\overline{S} = \{(a, b) \in A \times B \,|\, (a, b) \notin S\}$.
  It is clear from the above definition of S that $\overline{S}$ represents the "greater than" ($>$) relation. Note that the complement operation is involutory. In mathematics an involution is such a

function if applied twice, produces the original information. This means that $\overline{\overline{S}} = S$.

$$\overline{S} = \begin{array}{c} \\ 2 \\ 3 \\ 7 \end{array} \begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix} \end{array}$$

- *Inclusion*: $R \sqsubseteq S \iff \forall a \in A : \forall b \in B : \big[(a,b) \in R \implies (a,b) \in S\big]$. Notice that this property (R is included in S) is equivalent to $R \sqcap \overline{S} = \varnothing$.

$$\begin{array}{c} \\ 2 \\ 3 \\ 7 \end{array} \begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array} \sqcap \begin{array}{c} \\ 2 \\ 3 \\ 7 \end{array} \begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix} \end{array} = \varnothing$$

  Therefore, R is included into S. It entails from the fact that a divisor is always less than or equal to a dividend.

- *Empty or null relation* ($\perp\!\!\!\perp_{AB}$): It is called the *bottom* relation from A to B as it doesn't associate an element of A with any element of B. Clearly the equality relation ($=$) between the sets A and B as defined above produces the empty relation here as they do not have any element in common.

- *Universal relation* ($\top_{AB}$): This is the largest relation from A to B and relates every element of A with all the elements of B. This is why it is also know as the *top* relation from A to B. In this particular example S as well as $R \sqcup S$ equals to $\top_{AB}$. Therefore, it is evident that for any relation $X : A \to B$, $X \sqcup \top_{AB} = \top_{AB}$.

2.1.2 Relational Operations

This section defines the characteristic operations of relations. These operations play vital roles in almost any study on relations. For the demonstrations, we continue using the above example and add the greatest common divisor (gcd) and least common multiple (lcm) of 6 and 28 as the set C. We also define $T : B \to C$ to be another instance of "divides" ($|$) or "is a divisor of" relation.

$$T = \begin{array}{c} \\ 6 \\ 28 \end{array} \begin{array}{c} \begin{array}{cc} 2 & 84 \end{array} \\ \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \end{array}$$

- *Converse or transpose*: It converts $R : A \to B$ into a relation from B to A such that $R^{\smile} = \{(b, a) \in B \times A \mid (a, b) \in R\}$. On an $n \times n$ matrix this operation simply exchanges row- and column-labels and mirros the matrix-entries along the diagonal from upper left to lower right. In our case $R^{\smile}$ means the 'divides by'" relation.
  It is intuitive that the *converse or transpose* operation is involutory too, i.e., $R^{\smile\smile} = R$. In addition, the order of application of the two involutions (*complement* and *converse*) doesn't matter.

$$R^{\smile} = \begin{matrix} \\ 6 \\ 28 \end{matrix}\begin{matrix} \begin{matrix} 2 & 3 & 7 \end{matrix} \\ \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \end{matrix}, \quad \overline{R^{\smile}} = \begin{matrix} \\ 6 \\ 28 \end{matrix}\begin{matrix} \begin{matrix} 2 & 3 & 7 \end{matrix} \\ \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \end{matrix}, \text{ and } \overline{R}^{\smile} = \begin{matrix} \\ 6 \\ 28 \end{matrix}\begin{matrix} \begin{matrix} 2 & 3 & 7 \end{matrix} \\ \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \end{matrix}$$

- *Composition or multiplication*: The composition of $R : A \to B$ and $T : B \to C$ is another relation defined by

$$R;T = \{(a, c) \in A \times C \mid \exists b \in B : (a, b) \in R \text{ and } (b, c) \in T\}$$

$$\begin{matrix} \\ 2 \\ 3 \\ 7 \end{matrix}\begin{matrix} \begin{matrix} 6 & 28 \end{matrix} \\ \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \end{matrix}; \begin{matrix} \\ 6 \\ 28 \end{matrix}\begin{matrix} \begin{matrix} 2 & 84 \end{matrix} \\ \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \end{matrix} = \begin{matrix} \\ 2 \\ 3 \\ 7 \end{matrix}\begin{matrix} \begin{matrix} 2 & 84 \end{matrix} \\ \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \end{matrix}$$

  Note that for two relations to be composable, the target of the first relation should be the same as the source of the second relation. In this example, one can easily reason about the *composition* as, "a divides b" and "b divides c" eventually mean that "a divides c". Throughout our study we write $R;T$ to indicate that we first apply R and then T. There are left and right unit relations that act as the unit elements for *composition*. For $R : A \to B$, the unit relations are $\mathbb{I}_A$ and $\mathbb{I}_B$ such that $\mathbb{I}_A;R = R;\mathbb{I}_B = R$. These are also known as *identity relations* and they relate every element to itself, i.e., for example, $\mathbb{I}_A = \{(a, a) \mid a \in A\}$ [31].

$$\mathbb{I}_A = \begin{matrix} \\ 2 \\ 3 \\ 7 \end{matrix}\begin{matrix} \begin{matrix} 2 & 3 & 7 \end{matrix} \\ \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{matrix} \text{and } \mathbb{I}_B = \begin{matrix} \\ 6 \\ 28 \end{matrix}\begin{matrix} \begin{matrix} 6 & 28 \end{matrix} \\ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{matrix}$$

  Note that *identity relations* are always of the form $n \times n$ while visualized as matrices with all diagonal entries being 1 along top-left to bottom-right. In case of the source of R above, which is $\{2, 3, 7\}$, $\mathbb{I}_A$ is 3×3 diagonal matrix.

  Composition has certain properties which are important in the calculus of relations. For $X : D \to E$, $Y : E \to F$, $W : D \to F$, and $Z : F \to G$,

- *Composition* is associative, i.e., $(X;Y);Z = X;(Y;Z)$, but not commutative, i.e., $X;Y \neq Y;X$.
- The *converse* of $X;Y$ is given by $(X;Y)^{\smile} = Y^{\smile};X^{\smile}$.
- *Taraski rule*: For all $X \neq \perp\!\!\!\perp_{DE}$, $\pi_{DD};X;\pi_{EE} = \pi_{DE}$.
- *Schröder equivalences*: It depicts the interaction between composition, converse, and complement, with respect to containment [28].
  $X;Y \sqsubseteq W \iff X^{\smile};\overline{W} \sqsubseteq \overline{Y} \iff \overline{W};Y^{\smile} \sqsubseteq \overline{X}$.
- *Dedekind rule*: This is a variant of Schröder equivalences and does not involve complements.
  $X;Y \sqcap W \sqsubseteq (X \sqcap W;Y^{\smile});(Y \sqcap X^{\smile};W)$.

The proofs are trivial and could be found in any book on relation algebra including [27] and [28]. From now on we omit the indices of π, $\perp\!\!\!\perp$, and $\mathbb{I}$ if those are clear from the context.

2.1.3 Composite Operations on Relations

These operations are derived from the above operations and yield useful results in our study. As composition is also known as multiplication, it is worth asking if there exists a quotient of one relation with respect to another like in the case of usual multiplication, for instance, $2 * x = 6$ entails $x = 3$. This is why, the operations of residuals are defined. Since composition is not commutative, we have two residuals, *left residual* and *right residual*. For the demonstrations, we use the following sets of prime numbers (A and C) and perfect numbers (B and D).

$$A = \{2,3,5\},\ B = \{7,11\},\ C = \{6,28\},\ D = \{496,8128\}$$

We define three relations $R : B \to C$, $S : A \to C$, and $T : A \to D$, all representing the "divide" relation between different sets of numbers.

$$R = \begin{array}{c} \\ 7 \\ 11 \end{array}\!\!\begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \end{array}, S = \begin{array}{c} \\ 2 \\ 3 \\ 5 \end{array}\!\!\begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 0 \end{pmatrix} \end{array}, \text{ and } T = \begin{array}{c} \\ 2 \\ 3 \\ 5 \end{array}\!\!\begin{array}{c} \begin{array}{cc} 496 & 8128 \end{array} \\ \begin{pmatrix} 1 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \end{array}$$

- *Left residual*: The *left residual* of two relations $R : B \to C$ and $S : A \to C$ (S over R), both having the same range, is defined as $S/R = \overline{\overline{S};R^{\smile}}$ [31]. Component wise,

$$S/R = \{(a,b) \mid \forall c : R(b,c) \implies S(a,c)\}$$

.

$$R^{\smile} = \begin{array}{c} \\ 6 \\ 28 \end{array} \begin{array}{c} \begin{array}{cc} 7 & 11 \end{array} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \end{array}, \overline{S} = \begin{array}{c} \\ 2 \\ 3 \\ 5 \end{array} \begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \end{array}, \overline{S};R^{\smile} = \begin{array}{c} \\ 2 \\ 3 \\ 5 \end{array} \begin{array}{c} \begin{array}{cc} 7 & 11 \end{array} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \end{array}$$

$$\text{Therefore, } S/R = \overline{\overline{S};R^{\smile}} = \begin{array}{c} \\ 2 \\ 3 \\ 5 \end{array} \begin{array}{c} \begin{array}{cc} 7 & 11 \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix} \end{array}$$

Note that S/R is the largest of all relations $X : A \to B$ such that $X;R \sqsubseteq S$. In our example, the *left residual* $S/R : A \to B$ relates two numbers from $\{2,3,5\}$ and $\{7,11\}$ if every perfect number from $\{6,28\}$ that can be divided by the first number can also be divided by the second.

- *Right residual*: The *right residual* of $S : A \to C$ over $T : A \to D$, having identical source, is another relation defined as $T\backslash S = \overline{T^{\smile};\overline{S}}$. We define $T\backslash S$ componentwisely as follows

$$T\backslash S = \{(d,c) \mid \forall a : T(a,d) \implies S(a,c)\}$$

$$T^{\smile} = \begin{array}{c} \\ 496 \\ 8128 \end{array} \begin{array}{c} \begin{array}{ccc} 2 & 3 & 5 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix} \end{array}, \overline{S} = \begin{array}{c} \\ 2 \\ 3 \\ 5 \end{array} \begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix} \end{array}, T^{\smile};\overline{S} = \begin{array}{c} \\ 496 \\ 8128 \end{array} \begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \end{array}$$

$$\text{Therefore, } T\backslash S = \begin{array}{c} \\ 496 \\ 8128 \end{array} \begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \end{array}$$

One can easily verify that the residual $T\backslash S : D \to C$ in our example, relates a perfect number from $\{496, 8128\}$ to another from $\{6,28\}$ if every prime number of $\{2,3,5\}$ that divides the first also divides the second.

It is to be noted that, $T\backslash S = S^{\smile}/T^{\smile\smile}$.

2.1.4 Properties of Relations

Relations can be distinguished by the properties they may satisfy. For example, the relation of "smaller than or equal to" ($\leqslant$) holds for any two consecutive numbers $x, y \in \mathbb{Z}$, while "smaller than" ($<$) does not. Again, a number can be associated to more than one number in terms of divisibility while a person can be attached to exactly one other person on paternal relationship.

Among the different properties of relations, important are those that connects to ordering and to functions [27]. Our research requires some of these properties to be explored in details. Note that these include properties like *reflexivity, symmetry, and transitivity*, that apply to homogeneous relations only. We start with the following "divides" relations of arbitrary integers as examples. For clarity, we define their source and target as $P : A \to B$, $Q : C \to D$, and $R : A \to E$.

$$P = \begin{array}{c} \\ 2 \\ 3 \\ 5 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} 25 & 26 & 27 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \end{array}, \quad Q = \begin{array}{c} \\ 7 \\ 11 \end{array}\!\!\begin{array}{c} \begin{array}{cc} 28 & 29 \end{array} \\ \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \end{array}, \quad R = \begin{array}{c} \\ 2 \\ 3 \\ 5 \end{array}\!\!\begin{array}{c} \begin{array}{cccc} 30 & 31 & 32 & 33 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

- *Univalent* relation: A relation $S : G \to H$ is univalent if it associates an element of the source to at most one element of the target. Mathematically, S is *univalent* iff

$$\forall g \in G : \forall h_1, h_2 \in H : (g S h_1 \text{ and } g S h_2) \implies h_1 = h_2$$

  In the contact list example, a phone number is held by exactly one person, thus forming a *univalent* relation. Clearly P is *univalent* here. The *univalency* property of S is equivalent to $S^\smile; S \sqsubseteq \mathbb{I}_H$. For the above relation P, we have

$$P^\smile = \begin{array}{c} \\ 25 \\ 26 \\ 27 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} 2 & 3 & 5 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \end{array} \text{ and } P^\smile; P = \begin{array}{c} \\ 25 \\ 26 \\ 27 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} 25 & 26 & 27 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{array} \sqsubseteq \mathbb{I}_B.$$

  Similarly, Q is also univalent while R is not.

$$Q^\smile; Q = \begin{array}{c} \\ 28 \\ 29 \end{array}\!\!\begin{array}{c} \begin{array}{cc} 28 & 29 \end{array} \\ \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \end{array} \sqsubseteq \mathbb{I}_D, \text{ and } R^\smile; R = \begin{array}{c} \\ 30 \\ 31 \\ 32 \\ 33 \end{array}\!\!\begin{array}{c} \begin{array}{cccc} 30 & 31 & 32 & 33 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \end{array} \not\sqsubseteq \mathbb{I}_E$$

  Univalent relations are also known as *partially defined functions* as there might be source elements which are related to none of the target elements. Note that some of these results, $R^\smile; R$ for instance, might seem to make no sense in regard to the original "divides" relation at the first sight. But, all of these actually have an intuitive interpretation. For example, a 1 at $(x, y) \in (R^\smile; R)$ indicates whether x and y share a divisor or not. It is clear that 30 has a common divisor with each of $\{30, 32, 33\}$. However, we will skip any future interpretation of this kind as they are not important to the contexts.

- *Injective* relation: A relation is said to be *injective* iff the converse is *univalent*. Here, P and Q are *injectives* as both $P^{\smile}$ and $Q^{\smile}$ are *univalent*. However, R is not *injective* as 30 has more than one image in $R^{\smile}$.

$$R^{\smile} = \begin{array}{c} \\ 30 \\ 31 \\ 32 \\ 33 \end{array} \begin{array}{c} \begin{array}{ccc} 2 & 3 & 5 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \end{array}, \text{ and } Q^{\smile} = \begin{array}{c} \\ 28 \\ 29 \end{array} \begin{array}{c} \begin{array}{cc} 7 & 11 \end{array} \\ \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \end{array}$$

- *Total* relation: A relation $S : G \rightarrow H$ is *total* if each of its source-element is associated with at least one element from the target. From [27],

$$S \text{ total} : \Longleftrightarrow \forall g \in G, \exists h \in H : gS\,h$$

This property is equivalent to $\mathbb{I}_{GG} \sqsubseteq S\,;S^{\smile}$ [31]. Clearly, in our example, P and R are total while Q is not.

$$R;R^{\smile} = \begin{array}{c} \\ 2 \\ 3 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccc} 2 & 3 & 5 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \end{array} \sqsupseteq \mathbb{I} \text{ and } Q;Q^{\smile} = \begin{array}{c} \\ 28 \\ 29 \end{array} \begin{array}{c} \begin{array}{cc} 28 & 29 \end{array} \\ \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \end{array} \not\sqsupseteq \mathbb{I}$$

- *Surjective* relation: A relation is *surjective* iff its converse is *total*. Here P is *surjective*. However, neither Q, nor R is surjective as 31 in $R^{\smile}$ and 29 in $Q^{\smile}$ are not related to any of their target elements, respectively.

- *Bijective* relation: A relation is called a bijective if it is both injective and surjective. In our example, only P is *bijective*.

- *Mapping*: A relation which is total and univalent is called a *mapping*. One may easily find that P defined above is a mapping.

  Note that relations have the flexibility of being able to express more things than functions or mappings do. This is because a relation may assign zero, one, or more values to a member of the source. As a result, it appears to be a better mathematical tool to apply to numerous applications.

- *Vector* and *point*: These relations provide a way to correspond to an element or a subset of elements [27]. Thus we denote them by lower case letters. In our study these are particularly useful in selecting tuples from a database table that satisfy certain condition(s).

A relation $v : G \to H = \{(g,h) \mid h \in H\} \sqsubseteq G \times H$ is a *vector* if it is *column-constant*, i.e., $v = \pi_{GG}; v$.

As in the definition, a *vector* makes only the first element of the pairs conditional and thus provides a way to characterize a subset of the source set. Since the source of a vector is not important, one often chooses a singleton set $1 = \{*\}$ and uses that in the right example. For instance, if we are interested in the entries for 32 and 33 in R above, then the corresponding *vector* would be

$$v : A \to E = \begin{array}{c} \\ 2 \\ 3 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccc} 30 & 31 & 32 & 33 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix} \end{array} \text{ or simply } \begin{array}{c} \\ \{*\} \end{array} \begin{array}{c} \begin{array}{cccc} 30 & 31 & 32 & 33 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

Notice that, for any relation $S : G \to H$, $(S; \pi_{HH})^\smile$ and $\pi_{GG}; S$ are the two vectors that define the domain and the range of S respectively. At the end of this section we will see an example for that.

A single element within a set, on the other hand, can be interpreted by a point relation.

Mathematically, a relation $p : G \to H = \{(g,h) \mid h \in H\} \sqsubseteq G \times H$ is a *point* if it is

- *column-constant*, i.e., $p = \pi_{GG}; p$.
- *univalent*, i.e., $p^\smile; p \sqsubseteq \mathbb{I}_{HH}$.
- *total*, i.e., $\mathbb{I}_{GG} \sqsubseteq p; p^\smile$.

A *point* relation corresponding to 32 in R is given by

$$p : A \to E = \begin{array}{c} \\ 2 \\ 3 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccc} 30 & 31 & 32 & 33 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \text{ which is equivalent to } \begin{array}{c} \\ \{*\} \end{array} \begin{array}{c} \begin{array}{cccc} 30 & 31 & 32 & 33 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

As a short representation of the usage of *vector* and *point* relations in our work, we borrow the example of student-marks from Chapter 1. Our interest here is to know the name of the students who scored 70 and above. Figure 2.2c shows the corresponding *vector* for this.

Now we apply this vector to S by evaluating $v; S^\smile$. The result below shows that both Peter

Name	**Marks**
Russel	62
Peter	95
Andy	46
James	75

(A) A tabular relation S

$$\begin{array}{c} \\ \textit{Russel} \\ \textit{Peter} \\ \textit{Andy} \\ \textit{James} \end{array} \begin{array}{c} \begin{array}{cccc} 62 & 95 & 46 & 75 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

(B) Matrix representation of S

$$\begin{array}{c} \\ \{*\} \end{array} \begin{array}{c} \begin{array}{cccc} 62 & 95 & 46 & 75 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

(C) A vector v

FIGURE 2.2: Example of a vector relation

and James obtained 70 or more in that course.

$$\begin{array}{c} \begin{array}{cccc} 62 & 95 & 46 & 75 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 \end{pmatrix} \end{array} ; \begin{array}{c} \\ 62 \\ 95 \\ 46 \\ 75 \end{array} \begin{array}{c} \begin{array}{cccc} \textit{Russel} & \textit{Peter} & \textit{Andy} & \textit{James} \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array} = \begin{array}{c} \begin{array}{cccc} \textit{R.} & \textit{P.} & \textit{A.} & \textit{J.} \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 \end{pmatrix} \end{array}$$

- *Reflexivity*: A homogeneous relation $T : A \to A$ is said to be *reflexive* if each element of A can be associated to itself by T, i.e., aTa for all $a \in A$. Thus, T being reflexive indicates that $\mathbb{I} \sqsubseteq T$. The "divides" relation is reflexive on $\mathbb{Z}$ as any integer divides itself. We define *irreflexive* relations to be relations in which no element is related to itself. Therefore, irreflexive relations are, by definition, not reflexive, but not all non-reflexive relations are irreflexive. The "smaller than" ($<$) relation on $\mathbb{Z}$ is irreflexive as an integer x cannot be smaller than itself. In our example above, P and Q are not reflexive, R, however, is neither reflexive nor irreflexive. If a relation T is irreflexive, then $T \sqsubseteq \bar{\mathbb{I}}$. To summarize these, we include three relations in the following figure which are reflexive, irreflexive, and none of them, from left to right.

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

- *Symmetry*: A homogeneous relation $T : A \to A$ is *symmetric* if $a_1 T a_2$ implies $a_2 T a_1$ for all $a_1, a_2 \in A$. It can easily be shown that both $=$ and $\neq$ are *symmetric*. For a *symmetric* relation T, $T^{\smile} \sqsubseteq T$. Relations which are not *symmetric* may satisfy one of the following properties [27]. $T : A \to A$ is
  - *asymmetric* iff $\forall a_1, a_2 : (a_1, a_2) \in T \implies (a_2, a_1) \notin T$. If T is *asymmetric*, then $T \sqcap T^{\smile} \sqsubseteq \perp\!\!\!\perp$.
  - *antisymmetric* means that $\forall a_1, a_2 : a_1 \neq a_2 \implies \{(a_1, a_2) \notin T \text{ or } (a_2, a_1) \notin T\}$. *Antisymmetry* in T entails that $T \sqcap T^{\smile} \sqsubseteq \mathbb{I}$.

In the following figure, the left relation is *asymmetric* while the one on the right is *anti-symmetric*.

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

- *Transitivity*: This property is useful in defining ordering and equivalence relations. A homogeneous relation $T : A \to A$ is transitive, for any $a_1, a_2, a_3 \in A$, if a_1 is related to a_2 and a_2 is related to a_3, then a_1 is also related to a_3. Mathematically, T is *transitive* if and only if

$$\forall a_1, a_2, a_3 \in A : \{(a_1, a_2) \in T \text{ and } (a_2, a_3) \in T\} \implies (a_1, a_3) \in T$$

The "divides" relation on $\mathbb{Z}$ is *transitive* as because whenever x divides y and y divides z, then x also divides z. For any *transitive* relation T, $T; T \sqsubseteq T$.

Having defined the above operations and properties of relations, one could easily verify that the domain and the codomain of a relation $R : A \to B$ are given by the vectors [27]

$$\mathsf{dom}(R) = (R; \pi_B)^\smile \text{ and } \mathsf{cod}(R) = \pi_A; R$$

$$R = \begin{array}{c} \\ 2 \\ 3 \\ 5 \end{array} \begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 0 \end{pmatrix} \end{array}, \; R; \pi_B = \begin{array}{c} \\ 2 \\ 3 \\ 5 \end{array} \begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 0 \end{pmatrix} \end{array}; \begin{array}{c} \\ 6 \\ 28 \end{array} \begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} \end{array} = \begin{array}{c} \\ 2 \\ 3 \\ 5 \end{array} \begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix} \end{array}$$

$$\mathsf{dom}(R) = (R; \pi_B)^\smile = \begin{array}{c} \\ 6 \\ 28 \end{array} \begin{array}{c} \begin{array}{ccc} 2 & 3 & 5 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix} \end{array} \text{ which is equivalent to } \begin{array}{c} \\ \{*\} \end{array} \begin{array}{c} \begin{array}{ccc} 2 & 3 & 5 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 \end{pmatrix} \end{array}$$

$$\mathsf{cod}(R) = \pi_A; R = \begin{array}{c} \\ 2 \\ 3 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccc} 2 & 3 & 5 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \end{array}; \begin{array}{c} \\ 2 \\ 3 \\ 5 \end{array} \begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix} \end{array} = \begin{array}{c} \\ 2 \\ 3 \\ 5 \end{array} \begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix} \end{array}$$

$$\text{or equivalently, } \mathsf{cod}(R) = \begin{array}{c} \\ \{*\} \end{array} \begin{array}{c} \begin{array}{cc} 6 & 28 \end{array} \\ \begin{pmatrix} 1 & 1 \end{pmatrix} \end{array}$$

Considering the differences between relations and functions or mappings, one usually intends to know which of the source elements are assigned which and how many of the values on the target side. This can be done by determining the univalent and the multivalent part of a relation.

For any arbitrary relation R, the *univalent part* ($\mathtt{unp}$) and the *multivalent part* ($\mathtt{mup}$) is given by [28]

$$\mathtt{unp}(R) = R \sqcap \overline{R;\bar{\mathbb{I}}} = R \sqcap R^{\smile}\backslash\mathbb{I} \text{ and } \mathtt{mup}(R) = R \sqcap R;\bar{\mathbb{I}}$$

$$\begin{array}{c|cc} & 6 & 28 \\ 2 & 1 & 1 \\ 3 & 1 & 0 \\ 5 & 0 & 0 \end{array} \qquad \begin{array}{c|cc} & 6 & 28 \\ 6 & 0 & 1 \\ 28 & 1 & 0 \end{array} \qquad \begin{array}{c|cc} & 6 & 28 \\ 2 & 1 & 1 \\ 3 & 0 & 1 \\ 5 & 0 & 0 \end{array} \qquad \begin{array}{c|cc} & 6 & 28 \\ 2 & 0 & 0 \\ 3 & 1 & 0 \\ 5 & 0 & 0 \end{array} \qquad \begin{array}{c|cc} & 6 & 28 \\ 2 & 1 & 1 \\ 3 & 0 & 0 \\ 5 & 0 & 0 \end{array}$$

(A) R (B) $\bar{\mathbb{I}}$ (C) $R;\bar{\mathbb{I}}$ (D) $\mathtt{unp}(R)$ (E) $\mathtt{mup}(R)$

FIGURE 2.3: Univalent and multivalent part of a relation

Therefore, every relation splits into its univalent and multivalent parts which results in a disjunction [28]

$$R = \mathtt{unp}(R) \sqcup \mathtt{mup}(R), \quad \mathtt{unp}(R) \sqcap \mathtt{mup}(R) = \perp\!\!\!\perp$$

2.2 Orders and Lattices

Ordering appears commonly in various contexts of real life. For example, comparing things is very usual while buying something so that someone gets the best-match. Chapter 1 included such an example that explained the concepts behind contrasting goods from an order theoretic view point. In mathematics, order theory provides an algebraic way to look into orders using binary relations. In our research, order relations and lattices form the basis for working with database relations and interpreting their various operations. In this section, we investigate partial orderings and lattices in greater depth. However, we would like to start with the definition of equivalence relations and splittings.

2.2.1 Equivalence Relation, Quotient Set, and Splitting a Relation

In a number of situations we have objects that exhibit similar behaviours under certain relations. In the matrix representation of such a relation, objects own identical rows. These objects can be considered equivalent although they are different individuals. Such a group of object is called a *equivalent class*. One can choose an arbitrary object from a equivalent class as the representative and apply the operations of partially ordered relations. Now we formalize equivalent relation and equivalent classes mathematically.

A homogeneous relation $\Xi : A \to A$ is said to be an *equivalence* if Ξ is reflexive ($\mathbb{I} \sqsubseteq \Xi$), transitive ($\Xi;\Xi \sqsubseteq \Xi$), and symmetric ($\Xi^{\smile} \sqsubseteq \Xi$).

If we make this definition flexible by removing the constraint for Ξ being reflexive, then Ξ becomes a *partial equivalent* relation.

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

(A) A equivalence relation

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

(B) A partial equivalence relation as $(d,d) \notin \Xi$

FIGURE 2.4: Equivalence relations

An equivalence relation Ξ on a set A yields a partitioning of A into equivalence classes [28]. We recall the definition of a *partition* on A to be a collection of non-empty disjoint subsets of A, i.e., $A_i \cap A_j = \varnothing$ for $i \neq j$, and $\bigcup_{1 \leqslant i \leqslant n} A_i = A$.

Now, the *equivalence class* of any element $a \in A$ with respect to a equivalence relation Ξ is the set of $b \in A$ such that $(a,b) \in \Xi$. The set of all equivalence classes on A is called the *quotient set* and is denoted by A/Ξ. For example, the quotient sets for the above two matrices are $\{\{a,b\},\{c,d\}\}$ and $\{\{a,b\},\{c\}\}$ respectively.

Given an equivalence relation $\Xi : A \to A$, a set B together with a relation $R : B \to A$ is called a *splitting* of Ξ if and only if $R;R^{\smile} = \mathbb{I}_B$ and $R^{\smile};R = \Xi$. As an example, consider Ξ to be the second relation in Figure 2.4, $B = \{\{a,b\},\{c\}\}$ which is the quotient set on A, and R be the relation that relates an equivalence class to all elements it contains.

$$R;R^{\smile} = \begin{array}{c|cccc} & a & b & c & d \\ \hline \{a,b\} & 1 & 1 & 0 & 0 \\ \{c\} & 0 & 0 & 1 & 0 \end{array}; \begin{array}{c|cc} & \{a,b\} & \{c\} \\ \hline a & 1 & 0 \\ b & 1 & 0 \\ c & 0 & 1 \\ d & 0 & 0 \end{array} = \begin{array}{c|cc} & \{a,b\} & \{c\} \\ \hline \{a,b\} & 1 & 0 \\ \{c\} & 0 & 1 \end{array} = \mathbb{I}_B$$

$$R^{\smile};R = \begin{array}{c|cc} & \{a,b\} & \{c\} \\ \hline a & 1 & 0 \\ b & 1 & 0 \\ c & 0 & 1 \\ d & 0 & 0 \end{array}; \begin{array}{c|cccc} & a & b & c & d \\ \hline \{a,b\} & 1 & 1 & 0 & 0 \\ \{c\} & 0 & 0 & 1 & 0 \end{array} = \begin{array}{c|cccc} & a & b & c & d \\ \hline a & 1 & 1 & 0 & 0 \\ b & 1 & 1 & 0 & 0 \\ c & 0 & 0 & 1 & 0 \\ d & 0 & 0 & 0 & 0 \end{array} = \Xi$$

The operations above clearly state that R splits Ξ into equivalence classes by removing the duplicated and the zero rows.

2.2.2 Partial Order and Total Order

Consider the set of divisors of 8 which is $\{1, 2, 4, 8\}$. The elements of this set can be compared on the basis of divisibility ($|$). By comparing two numbers x and y in our set $\{1, 2, 4, 8\}$ we recognize that either $x|y$ or $y|x$. Therefore, $(\{1, 2, 4, 8\}, |)$ is said to be a *total* or *linear* order as it contains *comparable* elements only. On the other hand, the set of divisors of 6, i.e., $\{1, 2, 3, 6\}$ includes *incomparable* elements. This is because neither $2|3$, nor does $3|2$. Thus, we cannot arrange them to form a *total* order on $|$, therefore, $(\{1, 2, 3, 6\}, |)$ is a *partial* order. Such a set is called a *partially ordered set* or *poset*. We use the symbol $\leqslant$ to denote both partial and total orders. At this point, we give the formal definition of these orders.

As defined in [31], a pair $(P, \leqslant)$ consisting of a set and a binary relation on it is a *partially ordered set* or *poset* if

- $\leqslant$ is *reflexive*, i.e., $x \leqslant x$ for all $x \in P$,
- $\leqslant$ is *transitive*, i.e., if $x \leqslant y$ and $y \leqslant z$, then $x \leqslant z$ for all $x, y, z \in P$, and
- $\leqslant$ is *antisymmetric*, i.e., if $x \leqslant y$ and $y \leqslant x$, then $x = y$ for all $x, y \in P$.

Given a set P, an order on it, expressed as $<$, is said to be *strict* if

- $<$ is *transitive*, i.e., if $x < y$ and $y < z$, then $x < z$ for all $x, y, z \in P$, and
- $<$ is *asymmetric*, i.e., $\forall x, y \in P : x < y \implies y \not< x$

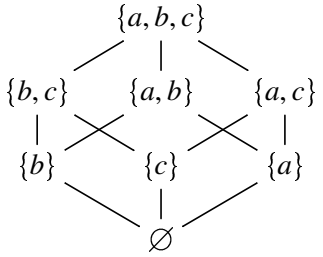
Now, an order E is said to be *total* if $E \sqcup E^{\smile} = \pi$ whereas a strict order C is *total* if $C \sqcup C^{\smile} \sqcup \mathbb{I} = \pi$. Notice that every partial order induces a strict order and vice versa.

The power set of any set together with the order of inclusion is a good example of *poset*. This means that, for the set $\{a, b\}$, $(\{\varnothing, \{a\}, \{b\}, \{a, b\}\}, \subseteq)$ is a *poset* as $\{a\} \nsubseteq \{b\}$ and $\{b\} \nsubseteq \{a\}$, i.e., $\{a\}$ and $\{b\}$ are incomparable. As an example of *totally* ordered set, on the other hand, we can think of the unit interval of real numbers which is used as the range of the membership function in *fuzzy sets*. That is, $[0, 1] = \{x \in \mathbb{R} \,|\, 0 \leqslant x \leqslant 1\}$ [31] together with the relation of "less than or equal to" ($\leqslant$) is a *total* order.

2.2.3 Hasse Diagram

In most situations having a diagrammatic representation of finite ordered sets aids in investigating their properties. *Hasse diagram* is such a handy tool named after the German mathematician *Helmut Hasse*.

In a *Hasse diagram*, each element of the ordered set is represented by a node and its immediate successors are placed above the node and connected to it by line segments or curves. This is why *Hasse diagrams* are also known as *upward drawings*. A *Hasse diagram* does not include any transitive relationship between the nodes, i.e., if $x \leqslant z$, then there is no node y such that $x \leqslant y \leqslant z$. Figure 2.5 shows the Hasse diagrams of a partial order and a total order.



(A) A poset $(\mathcal{P}(\{a,b,c\}), \subseteq)$

(B) A strict order $(\{1,2,4,8\}, |)$

FIGURE 2.5: Two Hasse diagrams

2.2.4 Lower and Upper Bounds: Meet and Join

While dealing with an ordering, we usually become interested in knowing the greatest and the smallest element in the corresponding set. It becomes even more important while dealing with partial orderings as they might contain incomparable elements. However, things become different for orderings which are not finite. Here we focus on the lower and upper bounds of partial orders.

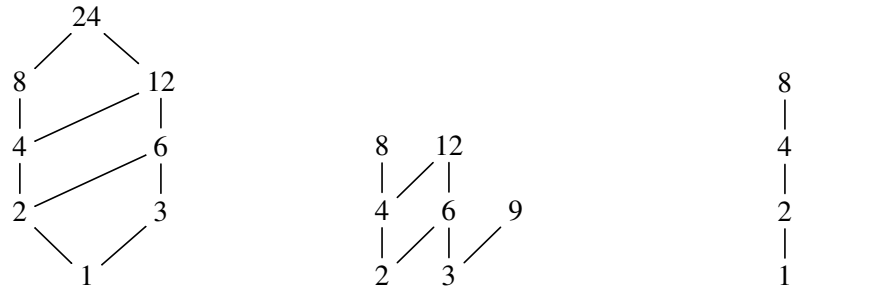
Let $(P, \leqslant)$ be a partial order and an arbitrary subset $Q \subseteq P$.

An element $m \in Q$ is a *maximal element* of Q if no other element in Q is strictly greater than m, i.e., $\forall n \in Q - \{m\} : m \not< n$.

An element $m \in Q$ is a *minimal element* of Q if no other element in Q is strictly smaller than m, i.e., $\forall n \in Q - \{m\} : n \not< m$.

These definitions apply to strict orders too as because all strict orders are essentially posets. In the following figure, (a) is the poset $(\{1,2,3,4,6,8,12,24\}, |)$ in which $\{1,2,3\}$ has a minimal element 1 but two maximal elements, 2 and 3. However, the subset $\{2,4,8\}$ has a minimal element 2 and a maximal element 8. The poset in Figure 2.6b contains two minimal elements, 2 and 3, and three maximal elements, 8, 12, and 9. The strict order in Figure 2.6c, however, has a minimal and maximal element for any subset.

This is clear that a subset of a partial order can have 0, 1, or more maximal as well as minimal elements. However, in our study we are interested in posets which have only one maximal or minimal elements. This maximal element is then called the *maximum or greatest element* as it is



(A) divisors of 24 (B) $(\{2,3,4,6,9,8,12\},|)$ (C) $(\{1,2,4,8\},|)$

FIGURE 2.6: Maximal and minimal elements of posets

larger than any other element in the poset. Dually, a poset P with exactly one minimal element m is said to have a *minimum or least element* as because for all $n \in P$, $m \leqslant n$.

Therefore, the maximum and the minimum elements of a poset are unique, if they exist. In our study, we respectively use 0 and 1 to express the least and the greatest elements of partial orders. It is evident that, $\{1,2,3\}$ in Figure 2.6a has the minimum element 1, but no maximum element as none of the maximal elements 2 and 3 greater than the other. However, for the whole poset in the same figure, 24 and 1 are the greatest and the least element, respectively. The poset in Figure 2.6b has neither a maximum, nor a minimum element. It is easy to recognize that all strict orders like the one in Figure 2.6c include greatest and least elements.

It can also be seen from Figure 2.6 that, for a subset $Q \subseteq P$, the superset P might contain an element m which is strictly greater or smaller than all elements of Q. For example, although the subset $\{1,2,3\}$ in (a) doesn't include a maximal element that belong to itself, each number of $\{6,12,24\}$ can be thought of as a maximal for it. Thus, we generalize the concept of maximal and minimal elements to upper and lower bounds by letting a maximal element be outside of a subset.

An element $u \in P$ is an *upper bound* of the set $Q \subseteq P$ if $\forall q \in Q : q \leqslant u$.

An element $l \in P$ is an *lower bound* of the set $Q \subseteq P$ if $\forall q \in Q : l \leqslant q$.

Therefore, $\{2,4,8\}$ in Figure 2.6a has two upper bounds, 8 and 24, and two lower bounds, 2 and 1. However, $\{4,6\}$ in Figure 2.6b has only one upper bound and one lower bound which are 12 and 2, respectively.

Apart from this, we can compute the upper bounds and lower bounds for each row of a relation by using residuals. That is, for a given relation $R : A \to B$ with ordering E, the set of upper bounds (ubd) and lower bounds (lbd) are given by

$$ubd_E(R) = R^{\smile} \backslash E \text{ and } lbd_E(R) = R^{\smile} \backslash E^{\smile}$$

In order to demonstrate the calculation of upper bounds, we use the divisibility relation in Figure 2.1 (Hasse diagram is given in Figure 2.6a) as the order E and define R to be the relation in Figure 2.7. Notice that each of the four rows of R indicates the numbers that we are interested to compute the upper bounds of. For example, in the second row we compute the upper bounds for 2, 3, and 4. As given in the resultant matrix, one could easily verify with Figure 2.6a that the upper bounds of 2, 3, and 4 are 12 and 24.

$$R = \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 6 & 8 & 12 & 24 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

$$ubd_E(R) = R^{\smile}\backslash E = \overline{R^{\smile\smile}; \overline{E}} = \overline{R; \overline{E}} = \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 6 & 8 & 12 & 24 \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

FIGURE 2.7: Calculating upper bounds for each row of a relation

It is evident from the above examples that some set might have more than one upper bounds or lower bounds. At this point we therefore define the least upper bound and the greatest lower bound for sets.

An upper bound a of Q is the *least upper bound* or *supremum* or *join* of Q if only if, for any upper bound b of Q, we have $a \leqslant b$, i.e., the minimum element in the set of upper bounds. We write this as $a = \bigvee Q$.

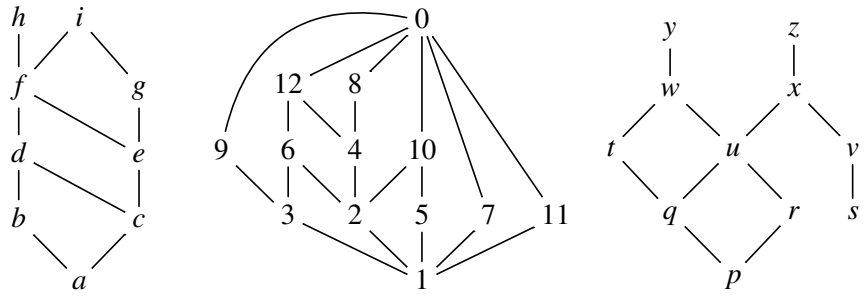
Dually, a lower bound a of Q is the *greatest lower bound* or *infimum* or *meet* of Q if only if, for any upper bound b of Q, $b \leqslant a$. It is written as $a = \bigwedge Q$.

However, for a subset $\{x, y\} \sqsubseteq Q$, we write $x \vee y$ and $x \wedge y$ to express respectively, the *join* and *meet* of x and y.

In the first poset of the following figure, the upper bounds of $\{a, b, c\}$ is the set $\{d, f, h, i\}$ and its only lower bound is a. It is more than evident that d is strictly smaller than any other lower bound in the set, so it is the least upper bound or join of $\{a, b, c\}$. At the same time, its only lower bound a is the greatest lower bound or meet. Looking at the partial order in 2.8b, we find that $8 \wedge 10 = 2$, $9 \vee 4 = 0$, $\bigwedge\{6, 10, 11\} = 1$, etc. On the right poset of Figure 2.8, $\{u, v\}$ has two upper bounds, x, and z. Since $x \leqslant z$, therefore, the join for $\{u, v\}$ is x. However, it doesn't have a meet.

Note that, the left poset of Figure 2.8 has a least element a, two maximal elements h and i, but neither $h \leqslant i$, nor $i \leqslant h$, therefore, none of them is the maximum. But, the ordering on the

middle has a greatest element 0 as well as a least element 1. Finally, the partial order on the right has neither a greatest element, nor a least element.



(A) a poset (B) divisibility on $[0 \dots 12]$ (C) another poset

FIGURE 2.8: Meet and join on posets

2.2.5 Lattices

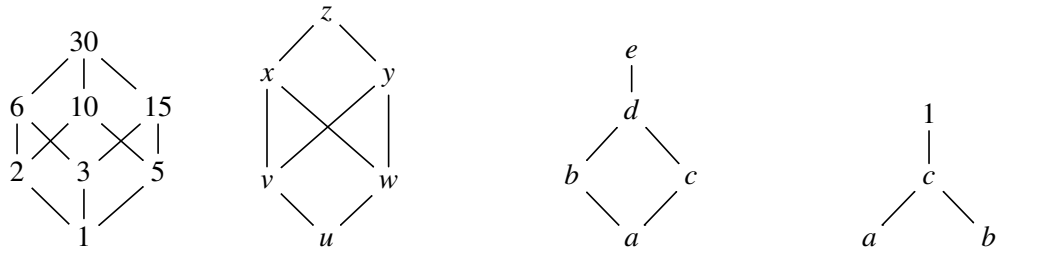
Lattices constitute a particular class of partially ordered sets that has been found very useful in different branches of mathematics including logic, topology, algebra, and so on. In fact, the study of lattices was brought to life by Richard Dedekind more than a hundred of years ago when he published two fundamental papers on this theory [20, 25]. However, it was given a huge boost by a series of papers from Garrett Birkhoff who then wrote a textbook in 1940 [3].

A poset $(\mathcal{L}, \leqslant)$ is a *lattice* if for every two-element subset there exists a greatest lower as well as a least upper bound, i.e., $\forall a, b \in \mathcal{L}$, $a \vee b$ and $a \wedge b$ exist [6, 27].

A lattice in which any subset has a least upper and a greatest lower bound is called a *complete lattice*. Evidently every complete lattice has a least element 0 and a greatest element 1 with $0 = \bigwedge \mathcal{L} = \bigvee \varnothing$ and $1 = \bigvee \mathcal{L} = \bigwedge \varnothing$ [31].

It is sometimes useful to define a lattice in terms of semilattices. A *lower semilattice* $\mathcal{L}$ is a poset for which every pair of elements x and y has a greatest lower bound or meet, $x \wedge y$. It is called *complete* iff every subset $M \neq \varnothing$ of $\mathcal{L}$ has a meet denoted by $\bigwedge M$. Dully, we define $\mathcal{L}$ to be a *complete upper semilattice* iff every nonempty subset $M \subseteq \mathcal{L}$ has a least upper bound or join denoted by $\bigvee M$. It is evident that a complete lower semilattice has a least element $0 = \bigwedge \mathcal{L}$ while a complete upper semilattice has a greatest element $1 = \bigvee \mathcal{L}$. Finally, a poset is a *lattice* if it is both a lower semilattice and upper semilattice with respect to the same partial order.

There are four posets in the following figure out of which only the second one is not a lattice. This is because, in the Figure 2.9b, although $\{x, y\}$ has three lower bounds, u, v, and w, none of them is greater than the other two. Therefore, $\{x, y\}$ does not have the greatest lower bound or meet. Similarly, $\{v, w\}$ does not have a join.



(A) divisors of 30 (B) a poset (C) a lattice (D) an upper semilattice

FIGURE 2.9: Lattice examples

Notice that for a lattice $\mathcal{L}$, $\vee$ and $\wedge$ are the two characteristic binary operations that map $\mathcal{L}^2$ ($\mathcal{L} \times \mathcal{L}$) to $\mathcal{L}$. They satisfy the following axiomatic identities [6]

- *Idempotency*: $a \vee a = a$,
  $a \wedge a = a$
- *Commutativity*: $a \vee b = b \vee a$,
  $a \wedge b = b \wedge a$
- *Associativity*: $(a \vee b) \vee c = a \vee (b \vee c)$,
  $(a \wedge b) \wedge c = a \wedge (b \wedge c)$
- *Absorption*: $a \vee (b \wedge c) = a$,
  $a \wedge (b \vee c) = a$

Therefore, a lattice $(\mathcal{L}, \leqslant)$ is equivalent to the algebra $(\mathcal{L}; \wedge, \vee)$ iff $\mathcal{L}$ is nonempty with $a \leqslant b \iff a \wedge b = a$ [6]. This entails that the axioms for idempotency are redundant above as $a \leqslant a$ implies $a \vee a = a \wedge a = a$.

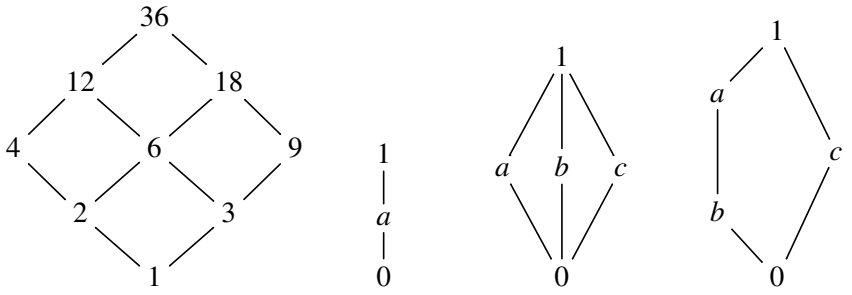
In our study, as we would see, we need the lattices to be bounded and distributive. However, these two properties are generally used to distinguish between lattices of different kinds.

2.2.5.1 Distributive lattice

In many lattices, the meet and join operations behave analogous to the arithmetic multiplication (*) and addition (+) operations where the first distributes over the later. So, we become interested in the distributivity of one of these lattice operations on the other.

A lattice $(\mathcal{L}, \wedge, \vee)$ is said to be distributive if, for any $a, b, c \in \mathcal{L}$, it satisfies the following [6]

$$a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$$
$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$$



(A) Divisors of 36 (B) $[0 \dots 1]$ (C) M_3 (D) N_5

FIGURE 2.10: Two distributive and two non-distributive lattices

A common example of distributive lattice is $(\mathbb{Z}+, |)$, i.e., the set of positive integers with the greatest common divisor as the meet and the least common multiple as the join. It can be proved intuitively that any chain is also distributive with the two functions of max and min being the join and the meet, respectively. However, the lattices in Figure 2.10c and 2.10d are not distributive. In fact, M_3, the *diamond lattice*, is the simplest non-distributive lattice. In the *pentagon lattice* N_5, $a \wedge (b \vee c) = a \wedge 1 = a$ and $(a \wedge b) \vee (a \wedge c) = b \vee 0 = b$, which proves its non-distributivity.

M_3 and N_5 are important in the study of lattices as they can be used to identify if an arbitrary lattice is distributive: a lattice is *distributive* if none of its sublattices is isomorphic to M_3 or N_5. We define a *sublattice* to be a nonempty subset $M \subseteq \mathcal{L}$ which itself is a lattice with the same meet and join operations as $\mathcal{L}$. This means that, for any $a, b \in M$, we have $a \wedge b$ and $a \vee b$ in M. Two lattices $\mathcal{L}_1 = (L_1, \leqslant)$ and $\mathcal{L}_2 = (L_2, \leqslant)$ are said to be *isomorphic* [6], and the map $\varphi : L_1 \to L_2$ is an *isomporphism* iff φ is a bijection and

$$a \leqslant b \text{ in } \mathcal{L}_1 \text{ iff } \varphi(a) \leqslant \varphi(b) \text{ in } \mathcal{L}_2.$$

2.2.5.2 Bounded lattice

A lattice $(\mathcal{L}, \wedge, \vee, 0, 1)$ is called a *bounded* lattice with the greatest and least element 1 and 0 if for any $a \in \mathcal{L}$, $a \vee 0 = a$ and $a \wedge 1 = a$.

All the lattices in Figure 2.10 are bounded. However, as explained before, M_3 and N_5 are not distributive.

Having defined bounded lattices, one might easily coincide this definition with that of a complete lattice, but there are differences. Complete lattices require that for any subset $M \subseteq \mathcal{L}$, we have $\bigvee M$ and $\bigwedge M$. In the case of emptyset, every lattice element can be treated as a lower bound as well as an upper bound. Therefore, for $M = \varnothing$, $\bigvee M = 0$ and $\bigwedge M = 1$. As both complete and bounded lattices include binary meet and join, complete lattices can thus be thought of a special class of bounded lattices. Surely, there are bounded lattices which are not complete.

At this point of our study, we become interested in defining complements for lattice elements which yields useful results in lattice theory. We start with the definition of complements.

It is easy to recall that *complement* of a set, so of a relation, consists of those elements that it does not include. If the *universe of discourse* is defined, then the computed complement is called the *absolute complement*. However, the type of complement we are going to work on is known as the *relative complement* as we calculate complement of an element relative to another.

As in [6], in a bounded lattice $\mathcal{L}$, b is a *complement* of a iff

$$a \wedge b = 0, \text{ and } a \vee b = 1.$$

Generally, a lattice element might have zero or more complements. If every element of a bounded lattice $\mathcal{L}$ has a complement, then it is called a *complemented lattice*. But if $\mathcal{L}$ is distributive, also called a *bounded distributive lattice*, then any element $a \in \mathcal{L}$ can have at most one complement [6]. For example, in Figure 2.10d, c has two complements, a and b. So, although N_5 is bounded, it is not distributive. In the bounded distributive lattice of Figure 2.9c, on the other hand, none of b, c and d has a complement. Therefore, it is not complemented. However, divisors of 30 in Figure 2.9a constitute a bounded distributive lattice which is complemented too.

It is to be noted that in a bounded distributive lattice $\mathcal{L}$, if b is a complement of a, then b is the largest element x of $\mathcal{L}$ such that $a \wedge x = 0$ [6]. More generally, let $\mathcal{L}$ be a lattice with 0; an element $\neg a$ is a *pseudocomplement* of a ($\in \mathcal{L}$) iff $a \wedge \neg a = 0$, and $a \wedge x = 0$ implies that $x \leqslant \neg a$. There may be at most one pseudocomplement for an element. A lattice in which every element has a *pseudocomplement* is called a *pseudocomplemented lattice*.

For $x, y \in \mathcal{L}$, the *relative pseudocomplement* of x in y is an element $x \to y$ so that for all $z \in \mathcal{L}$

$$z \leqslant x \to y \iff x \wedge z \leqslant y \text{ [31]}$$

A lattice in which the relative pseudocomplement exists for every pair of elements is called a *Brouwerian lattice* or a *Heyting algebra* [31]. However, it can be proved that every finite distributive lattice is a Brouwerian lattice. Heyting algebras, introduced by Arend Heyting (1930) [13], are important in our study and would be used in Dedekind categories, an algebraic framework for binary relations.

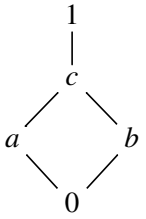
A bounded lattice $\mathcal{L}$ together with greatest and least element 1 and 0, and a binary implication operation $\to$ forms a *Heyting algebra* iff for all $x, y, z \in \mathcal{L}$ it satisfies the following axioms:

1. $x \to x = 1$
2. $x \wedge (x \to y) = x \wedge y$

3. $y \wedge (x \to y) = y$

4. $x \to (y \wedge z) = (x \to y) \wedge (x \to z)$ (Distributivity of $\to$)

As an example, we consider the lattice in 2.11a and compute the relative pseudocomplement for each of its elements. However, for two relations, we calculate relative pseudocomplements componentwise.



(A) A Heyting algebra $\mathcal{L}$

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

(B) $x \to y$ for all $x, y \in \mathcal{L}$

FIGURE 2.11: Relative pseudocomplement

2.3 Fuzzy Sets and Relations

The word "fuzziness" means the ambiguity that one can find in the definition of something. For example, "low pressure", "small fish", "tall building", etc. are some common phrases that have uncertainty included in their definitions. As mentioned in Chapter 1, we use the term "linguistic label" to refer to these phrases. Note that this type of uncertainties are different than probabilities as the former do not depend on the occurrence of phenomena or some particular tests [30].

Feeling the essence of a new mathematical theory for dealing with such vagueness in information, Lotfi A. Zadeh in 1965 [36] introduced the concepts of fuzzy sets and relations. This theory defines a mathematical grade for each element from the unit interval $[0 \ldots 1]$ to which an element is included in a set. The function that produces the degrees is called the *characteristic* or *membership function*. Contrast to classical sets, fuzzy sets thus provide a convenient way of dealing with situations where the we require to define memberships more general than a simple true or false. Therefore, as mentioned in Chapter 1, a *fuzzy set* thus generalizes the concept of classical sets by replacing the bivalent target of its membership function $\chi_B : A \to \mathbb{B}$ by the real interval $[0 \ldots 1]$. Mathematically, a *fuzzy set* A over a universe of discourse X (finite or infinite) is a set of pairs [8]

$$A = \{\chi_A(x)/x : x \in X, \chi_A(x) \in [0 \ldots 1] \subseteq \mathbb{R}\}$$

If $\chi_A(x) = 1$ for some x, then it is called a full member of X. A degree of 0, on other hand, indicates that x does not belong to X at all. As an example, we continue using from Chapter 1, the problem of modelling memory sizes of cell phones by a fuzzy set in order to find a

"good" match. However, in order to make the example more realistic, we would like add a few more performance factors that affect customers' choices of cell phones. These include processor speed, screen size, battery life, and camera resolution. We write the set of factors as $Z = \{processor, memory, screen, battery, camera\}$. One could easily notice that each of these factors has tradeoffs between customer's satisfaction and the associated cost. In addition, each factor has its own benefits and disadvantages from a customer's view point. For example, usually the faster the processor of a phone, the more power it consumes; although larger screens produce better view but drain the battery quickly and also make it difficult to handle the phone; the capacity of a battery more or less affects its size, and so on. We say that a phone is good for a customer if he/she rate it 80 or more for each of the these factors. Therefore, it is reasonable to view a cell phone as a vector where the components are ratings for the factors. Thus, selecting a "good" cell phone is a optimization problem where someone needs to select a phone that best meets his/her requirements. A characteristic function to evaluate such a set of cell phones against a specific factor might be

$$\chi_{good}(phone) = \begin{cases} 1 & \text{iff } x_{factor} >= 80, \\ \frac{x_{factor}}{80} & \text{otherwise} \end{cases}$$

It is often useful to visualize a fuzzy set by trapezoidal diagram. Figure 2.12 depicts such a diagram for the above function. It is clear that the "goodness" of a cell phone increases as its score approaches 50 at which point it gains a membership degree of 1. Note that each of these factors could actually be represented by a distinct fuzzy set in a real world application.

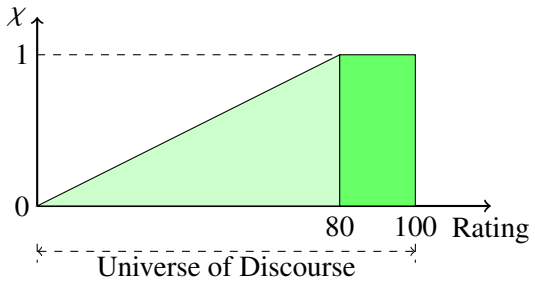


FIGURE 2.12: Trapezoidal fuzzy set defining "good" cell phones

If $X = \{Brand1, Brand2, Brand3, Brand4\}$ be the set of cell phones and $Y = \{66, 94, 49, 81\}$ be their respective ratings by a customer on "internal memory", then the following fuzzy set $A : X \rightarrow [0 \ldots 1]$ represents how well the customer is satisfied with these phones for their internal memories.

$$A = \{0.825/Brand1, 1/Brand2, 0.6125/Brand3, 1/Brand4\}$$

Although we are using simple numeric examples here, one could easily understand that different types and ranges of data can actually be represented by the unique interval $[0 \ldots 1]$. This tremendous feature of fuzzy sets has made researchers able to express mathematically ambiguity in human thinking as well as in real world in a much better way.

Like crisp sets, we have the different set theoretic operations on fuzzy sets. However, we define them in terms of the membership functions [30]. For the demonstrations here, we define another fuzzy set B which indicates the level of customer satisfaction regarding battery life of the four cell phones. However, we use the same membership function χ_{good} as before.

$$B = \{0.75/Brand1, 0.68/Brand2, 1/Brand3, 0.92/Brand4\}$$

- The *union* of two fuzzy sets A and B, $A \cup B$, is the fuzzy set defined by the following membership function:

  $$\chi_{A\cup B}(x) = \chi_A(x) \vee \chi_B(x)$$

  where $\vee$ denotes the maximum of the two values and therefore the join of two real values over the unit interval $[0 \ldots 1]$. In our example,

  $$A \cup B = \{0.825/Brand1, 1/Brand2, 1/Brand3, 1/Brand4\}.$$

- The *intersection* of two fuzzy sets A and B is another fuzzy set with the following membership function:

  $$\chi_{A\cap B}(x) = \chi_A(x) \wedge \chi_B(x)$$

  where $\wedge$ represents the minimum (meet) of the two values. In our example,

  $$A \cap B = \{0.75/Brand1, 0.68/Brand2, 0.6125/Brand3, 0.92/Brand4\}.$$

- The complement of a fuzzy set A has the following membership function:

  $$\chi_{\overline{A}}(x) = 1 - \chi_A(x).$$

  In our example, $\overline{B} = \{0.25/Brand1, 0.32/Brand2, 0/Brand3, 0.08/Brand4\}$.

- A fuzzy set A is said to be included in another fuzzy set B if and only if $\chi_A(x) \leqslant \chi_B(x)$ for all $x \in A$.

In order to process imprecise information in relational structures, the concept of fuzzy relations was introduced in [36]. Like fuzzy sets, fuzzy relations use characteristic functions to assign values to members indicating their degrees of membership.

If A and B are two universes of discourse and $\chi_{A\times B} : A \times B \to [0 \ldots 1]$, then a *fuzzy relation* is defined as [8]

$$R = \{\chi_{A\times B}(a, b)/(a, b) : (a, b) \in A \times B, \chi_{A\times B}(a, b) \in [0 \ldots 1] \in \mathbb{R}\}.$$

For $(a, b) \in A \times B$, $R(a, b)$ indicates the degree how far a and b are associated under R. It is easy to remember from Chapter 1 that, likewise fuzzy sets generalizes classical sets, fuzzy relations are a generalization of classical relations. As an example of fuzzy relations, we focus on the individual factors of each phone-brand in the previous example. Rather than defining a

sophisticated membership function for that, we just divide the ratings by 100 to get the corresponding values between 0 and 1. Therefore, an entry in the matrix representation of such a relation $R : A \times B \to [0 \dots 1]$ would indicate the customer rating of a phone in the relevant factor.

$$R = \begin{array}{c} \\ \textit{Brand}1 \\ \textit{Brand}2 \\ \textit{Brand}3 \\ \textit{Brand}4 \end{array} \begin{array}{c} \begin{array}{ccccc} \textit{processor} & \textit{memory} & \textit{screen} & \textit{battery} & \textit{camera} \end{array} \\ \begin{pmatrix} 0.72 & 0.66 & 0.8 & 0.34 & 0 \\ 1 & 0.94 & 0.88 & 0.95 & 0.74 \\ 0.37 & 0.41 & 0.44 & 0.25 & 0.1 \\ 0.89 & 0.81 & 1 & 0.78 & 0.60 \end{pmatrix} \end{array}$$

Before introducing $\mathcal{L}$-fuzzy relations, we recall that the composition of fuzzy relations is also known as *sup-min composition*. For a complete Browerian lattice $\mathcal{L}$ and two $\mathcal{L}$-fuzzy relations $Q : A \to B$, $S : B \to C$, their composition is another $\mathcal{L}$-fuzzy relation defined by

$$(Q;S)(a,c) := \bigvee_{b \in B} (Q(a,b) \wedge S(b,c)).$$

Also, *Semi-scalar multiplication* of a fuzzy relation R by a scalar k produces a fuzzy relation kR such that $kR(x,y) = kR(x,y)$.

From the discussions on posets and lattices in Section 2.2 we find that the closed unit interval $[0 \dots 1]$ is completely ordered i.e., it is a chain. But there are numerous situations where this linear set of fuzzy membership values is not sufficient to express fuzzy data. In Chapter 1, we included an example that modelled the difficulties in selecting a "good" size of cell phone memory. In order to handle such situations where multiple factors contribute to memberships, we move to the generalization of fuzzy sets and relations by J. A. Goguen (1967) [7].

2.4 $\mathcal{L}$-fuzzy Sets and Relations

For an arbitrary lattice $\mathcal{L}$, an $\mathcal{L}$-fuzzy set A on a set B is a function $A : B \to \mathcal{L}$ [7]. Note that the set of all $\mathcal{L}$-fuzzy sets on B is $\mathcal{L}^B$. In order to capture the notion of fuzziness properly, Goguen in his paper [7] imposed $\mathcal{L}$ to be a complete Brouwerian lattice (see Section 2.2.5.2 for definitions).

Similarly, for a complete Brouwerian lattice $\mathcal{L}$, an $\mathcal{L}$-fuzzy relation R between two sets X and Y is a function $R : A \times B \to \mathcal{L}$. In other words, R is an element of $\mathcal{L}^{A \times B}$ [7]. We will see later that the set of all $\mathcal{L}$-fuzzy relations between A and B forms a complete Brouwerian lattice with a least element 0 and a greatest element 1. We can summarize the three versions of relations in terms of their characteristic functions as follows.

- Classical relation: $R : A \times B \to \mathbb{B}$.

- Fuzzy relation: $R : A \times B \to [0 \dots 1]$.
- $\mathcal{L}$-fuzzy relation: $R : A \times B \to \mathcal{L}$.

It is more than evident that an $\mathcal{L}$-fuzzy relation could be specialized to a normal fuzzy relation by replacing the partial order $\mathcal{L}$ by the chain $[0 \dots 1]$, which then could be further specialized to a classical relation by replacing the chain $\{0, 1\}$, a bivalent chain representing *true* and *false* in $\mathbb{B}$.

In our example, let us replace $[0 \dots 1]$ by the following lattice for degrees of membership. We interpret the lattice elements as: for a particular factor, 0 denotes a "bad" rating, g_c and g_e respectively denote "good" ratings based on price (cost effective) and customer experience (customer satisfaction), b indicates "better" ratings, and finally, 1 denotes the "best" rating.

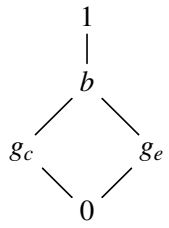


FIGURE 2.13: A complete Brouwerian lattice $\mathcal{L}$

Using the above lattice, we can define an $\mathcal{L}$-fuzzy relation $R : X \to Z$ as follows.

$$R = \begin{array}{c} \\ \textit{Brand}1 \\ \textit{Brand}2 \\ \textit{Brand}3 \\ \textit{Brand}4 \end{array} \begin{array}{c} \begin{array}{ccccc} \textit{processor} & \textit{memory} & \textit{screen} & \textit{battery} & \textit{camera} \end{array} \\ \begin{pmatrix} b & g_e & b & g_c & b \\ 1 & 1 & b & 1 & g_e \\ 0 & g_c & g_c & 0 & g_c \\ 1 & b & g_e & b & g_c \end{pmatrix} \end{array}$$

2.4.1 Operations on $\mathcal{L}$-Fuzzy Relations

At this point, we define certain operations on $\mathcal{L}$-fuzzy relations which are important in our study. Notice that for $\mathcal{L} = \mathbb{B}$, these operations coincide with those of classical relations defined in Section 2.1. We thus omit examples for their demonstrations.
For $\mathcal{L}$-fuzzy relations $Q, R : A \to B$, $S : B \to C$, and $T : D \to B$, [31]

- *Union*: $(Q \sqcup R)(a, b) := Q(a, b) \vee R(a, b)$
- *Intersection*: $(Q \sqcap R)(a, b) := Q(a, b) \wedge R(a, b)$
- *Converse*: $Q^{\smile}(a, b) := Q(b, a)$

- *Composition*: $(Q;S)(a,c) := \bigsqcup_{b\in B}(Q(a,b) \wedge S(b,c))$
- *Inclusion*: $Q \sqsubseteq R \iff \forall a \in A, b \in B : Q(a,b) \leqslant R(a,b)$
- *Empty and Universal relation*: $\perp\!\!\!\perp_{AB} := 0$, $\top_{AB} := 1$ where 0 and 1 are the least and greatest element of $\mathcal{L}$, respectively.
- *Identity relation*: $\mathbb{I}_A(a_1,a_2) := \begin{cases} 1 & \text{iff } a_1 = a_2, \\ 0 & \text{otherwise} \end{cases}$
- *Left residual*: $(Q/T)(a,d) := \bigsqcap_b T(d,b) \to Q(a,b)$ where $\to$ denotes the relative pseudocomplement.

In order to define the semantic constructions for $\mathcal{L}$FSQL appropriately, we look at the following properties of these operations. The corresponding proofs can be found in [31].

For a complete Brouwerian lattice $\mathcal{L}$ and $\mathcal{L}$-fuzzy relations $Q, Q_2, Q_i : A \to B$, $R, R_i : B \to C$, $S : C \to D$ for $i \in I$ and $T : A \to C$, we have

1. $Q;\mathbb{I}_B = Q$ and $\mathbb{I}_B;R = R$
2. $(Q;R);S = Q;(R;S)$
3. $(Q \sqcap Q_2)^\smile = Q^\smile \sqcap Q_2{}^\smile$
4. $(Q;R)^\smile = R^\smile;Q^\smile$
5. $(Q^\smile)^\smile = Q$
6. $Q;(\bigsqcap_{i\in I} R_i) \sqsubseteq \bigsqcap_{i\in I}(Q;R_i)$ and $(\bigsqcap_{i\in I} Q_i);R \sqsubseteq \bigsqcap_{i\in I}(Q_i;R)$
7. $Q;R \sqcap T = Q;(R \sqcap Q^\smile;T)$
8. $Q;\perp\!\!\!\perp_{BC} = \perp\!\!\!\perp_{AC}$
9. $Q;(\bigsqcup_{i\in I} R_i) = \bigsqcup_{i\in I}(Q;R_i)$ and $(\bigsqcup_{i\in I} Q_i);R = \bigsqcup_{i\in I}(Q_i;R)$

2.4.2 Crispness in $\mathcal{L}$-Fuzzy Relations

As $\mathcal{L}$-fuzzy relations generalizes classical relations, it is intuitive to say that the later is contained in the former. In fuzzy world, these relations that represents the presence or absence of association between elements is called *crisp relations*. Crispness is fundamental to the study of fuzziness. Goguen wrote in his paper [7], "Things unfuzzified or only trivially fuzzified are *crisp*; crispness is the qualitative opposite of fuzziness, although technically it is a special case."

An $\mathcal{L}$-fuzzy relation Q is called *0-1 crisp*, iff for all $(x, y) \in Q$, $Q(x, y) = 0$ or $Q(x, y) = 1$.

Therefore, when the least and the greatest element of $\mathcal{L}$ represents *true* and *false* of $\mathbb{B}$, $0 - 1$ crisp relations may be regarded as regular classical relations. Note that, these relations are closed under the operations of $\mathcal{L}$-fuzzy relations defined above.

2.4.3 Scalar Relations

Given a set of $\mathcal{L}$-fuzzy relations, one usually becomes interested to know about the structure $\mathcal{L}$. Scalar relations provide a means to abstractly identify the underlying lattice $\mathcal{L}$. The notion of scalar relations was first introduced in [11] and [17].

A relation $\alpha : A \to A$ is called a scalar on A iff $\alpha \sqsubseteq \mathbb{I}_A$ and $\pi_{AA}; \alpha = \alpha; \pi_{AA}$.

Notice that scalars are *partial identities*. Therefore, for some $l \in \mathcal{L}$ and $x, y \in A$, a scalar $\alpha : A \to A$ could alternatively be defined as

$$\alpha^l_A(x, y) = \begin{cases} l & \text{iff } x = y, \\ 0 & \text{otherwise} \end{cases}$$

If A is a three-element set and $l \in \mathcal{L}$, then one could easily verify that the followings two relations are scalars on A.

$$\begin{pmatrix} l & 0 & 0 \\ 0 & l & 0 \\ 0 & 0 & l \end{pmatrix} \qquad \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

(A) Scalar relation $\mathbb{I}_A$ (B) Scalar relation $\perp\!\!\!\perp_{AA}$

FIGURE 2.14: Two scalar relations

Jónsson and Tarski in [16] introduced the notion of *ideals* which are equivalent to scalar relations and thus provide an alternative way of generating a special classe of $\mathcal{L}$-fuzzy relations isomorphic to the underlying $\mathcal{L}$.

A relation $R : A \to B$ is said to be an *ideal relation* iff $\pi_{AA}; R; \pi_{BB} = R$.

As an example, consider the $\mathcal{L}$-fuzzy relations on the sets $A = \{2, 3, 5\}$, $B = \{6, 28\}$ and the lattice in Figure 2.13. It is clear that there might be as many as $|\mathcal{L}|^{|A|*|A|} = 5^9 = 1953125$ different $\mathcal{L}$-fuzzy relations on A, but only 5 of them have the form of Figure 2.14a which are scalars. Similarly, although there might be $|\mathcal{L}|^{|A|*|B|} = 5^6 = 15625$ different $\mathcal{L}$-fuzzy relations from A to B, only 5 of them look like the following relation for some $l \in \mathcal{L}$ which are essentially ideals.

$$\begin{pmatrix} l & l \\ l & l \\ l & l \end{pmatrix}$$

FIGURE 2.15: An ideal relation

The set of scalars on A, similarly, the set of ideals between A and B are isomorphic to the underlying lattice shown in Figure 2.13. This implies that abstractly they behave the same way the original lattice elements behave when applied to the different lattice operations.

2.4.4 α-Cuts and Arrow Operations

It is of common interest in the fuzzy world to generate crisp relations. For example, although a fuzzy database stores fuzzy information, the result of an user query usually results in something crisp. In order to select a specific element from a fuzzy set of alternatives in fuzzy decision theory, several cut operations were introduced in [5, 19]. Here we are interested in α-cuts which can generate crisp relations.

For an $\mathcal{L}$-fuzzy relation R, an *α-cut* for some $\alpha \in \mathcal{L}$ is defined as the following 0-1 crisp relation [31]

$$R_\alpha = \begin{cases} 1 & \text{iff } R(x,y) \geqslant \alpha \\ 0 & \text{otherwise} \end{cases}$$

It is clear that this cut operation produces crisp relations that associate only those pairs of elements for which the membership degree is at least α. For pairs with smaller degrees, it sets them to the least element 0. Thus, in an $\mathcal{L}$-fuzzy database, α-cuts can be used to model queries with thresholds, i.e., to select only those tuples that satisfy the condition with a degree greater or equal to the threshold.

We now define two special α-cuts that are called up-arrow ($\uparrow$) and down-arrow ($\downarrow$) operations.

For an $\mathcal{L}$-fuzzy relation R, the arrow operations are defined as

$$R^\uparrow = \begin{cases} 1 & \text{iff } R(x,y) \neq 0 \\ 0 & \text{otherwise} \end{cases} \qquad R^\downarrow = \begin{cases} 1 & \text{iff } R(x,y) = 1 \\ 0 & \text{otherwise} \end{cases}$$

The $\uparrow$ operation, when applied to a relation, raise the the membership degrees, which are not zero, to 1. Therefore, it produces the least 0-1 crisp relation containing R. The $\downarrow$ operation, on the other hand, sets the membership degrees, which are smaller than 1, to zero. As a result, it produces the greatest 0-1 crisp relation that R contains. The relations $R^\uparrow$ and $R^\downarrow$ are known as the *support* and the *kernel* of R in the fuzzy world.

In our example of modelling factors affecting customer choice of cell phones from Section 2.4, the $\uparrow$ and $\downarrow$ operations obtain the following relations.

$$R^{\uparrow} = \begin{array}{c} \\ B1 \\ B2 \\ B3 \\ B4 \end{array} \begin{array}{c} \begin{array}{ccccc} p & m & s & b & c \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \end{array}, \quad R^{\downarrow} = \begin{array}{c} \\ B1 \\ B2 \\ B3 \\ B4 \end{array} \begin{array}{c} \begin{array}{ccccc} p & m & s & b & c \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

As we will see later, the two arrow operations would be used in defining the algebraic theory of Arrow categories. So, we proceed to explore certain useful properties of these operations.

Let $Q, R : A \to B$ and $S : B \to C$ are three $\mathcal{L}$-fuzzy relations on the complete Brouwerian lattice $\mathcal{L}$. Then we have

1. Q is 0-1 crisp iff $Q^{\uparrow} = Q$, or equivalently iff $Q^{\downarrow} = Q$
2. $(R^{\smile}; S^{\downarrow})^{\uparrow} = R^{\uparrow\smile}; S^{\downarrow}$
3. $(Q \sqcap R^{\downarrow})^{\uparrow} = Q^{\uparrow} \sqcap R^{\downarrow}$
4. if $l(\in \mathcal{L}) \neq 0$, then ${\alpha_A^l}^{\uparrow} = \mathbb{I}_A$

Once again, the proofs can be found in [31].

It is important to note that an $\mathcal{L}$-fuzzy relation R can be represented by the set of all its α-cuts, i.e., by the set of all crisp relations R_α such that $(x, y) \in R_\alpha$ iff $\alpha \leqslant R(x, y)$. This is known as the *α-cut Theorem* in fuzzy theory. It states that, for an $\mathcal{L}$-fuzzy relation $R : A \to B$ on the complete Brouwerian lattice $\mathcal{L}$, we have

$$R = \bigsqcup_{l \in \mathcal{L}} (\alpha_A^l ; R_\alpha)$$

As an example, we consider the following arbitrary relation using the lattice in Figure 2.13

$$R = \begin{pmatrix} 1 & g_c & 0 \\ b & b & 0 \\ 0 & g_e & 1 \end{pmatrix}$$

We define the following α-cuts on R.

In a matrix R_l above, we set a 1 for (x, y) iff $l \leqslant R(x, y)$. For example, as there are only four entries in R, namely b's and 1's which are greater than or equal to b, they become 1's in R_b. We

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

(A) R_0 (B) R_{g_c} (C) R_{g_e} (D) R_b (E) R_1

FIGURE 2.16: α-cuts on R

now take the union of the above relations according to the α-cut Theorem.

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \sqcup \begin{pmatrix} g_c & g_c & 0 \\ g_c & g_c & 0 \\ 0 & 0 & g_c \end{pmatrix} \sqcup \begin{pmatrix} g_e & 0 & 0 \\ g_e & g_e & 0 \\ 0 & g_e & g_e \end{pmatrix} \sqcup \begin{pmatrix} b & 0 & 0 \\ b & b & 0 \\ 0 & 0 & b \end{pmatrix} \sqcup \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & g_c & 0 \\ b & b & 0 \\ 0 & g_e & 1 \end{pmatrix} = R$$

We conclude the section by introducing another important construction.
For a relation $R : A \to A$ and a scalar $\alpha^l : A \to A$ with some $l \in \mathcal{L}$, the α-cut of R, R_α can be computed by evaluating the relational expression $(\alpha^l \backslash R)^{\downarrow}$. For the demonstration, we continue using the above relation R with the scalar α^b.

$$(\alpha \backslash R)^{\downarrow} = \left(\begin{pmatrix} b & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & b \end{pmatrix} \backslash \begin{pmatrix} 1 & g_c & 0 \\ b & b & 0 \\ 0 & g_e & 1 \end{pmatrix} \right)^{\downarrow} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & g_e & 1 \end{pmatrix}^{\downarrow} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

2.5 Algebra of Relations

While analysing some mathematical structure intensively, it eventually becomes important to have a systematic formalization of its behaviour. The term algebra is widely used by mathematicians to mean such type of formalizations.

As defined in [26], an *algebra* is a domain or set of elements together with some functions defined on this domain and taking values in it.

2.5.1 Algebra of Classical Relations

All modern development on relations is affected by the works of A. De Morgan, C. S. Peirce, and W. Schröder in the late nineteenth century. George Boole, in his book *The Mathematical Analysis of Logic* (1847), first introduced an algebraic formalization of relations. However, his works focused only on unary relations, i.e., relations with one parameter, $R(x)$. Recall that a unary relation is simply a subset of a given set.

De Morgan, in 1860, studied the properties of and operations on binary relations, $R(x, y)$. However, in 1870, C. S. Peirce extended Boole's works and eventually produced a good general algebra of mathematical logic [24]. In the next few decades, Morgan and Peirce individually investigated several operations on relations many of which later on were found to coincide between their works. During this period, Schröder studied Boole's and De Morgan's work on logic and added several important constructions. The contributions of De Morgan, Peirce, and Schröder on early versions of relation algebra were summarized in the 1911 edition of the *Dictionary of Philosophy and Psychology* by J. M. Martin. However, A. Tarski (1941) was the first person who attempted to axiomatise the algebra of binary relations in terms of equational postulates.

Note that all these people worked on homogeneous binary relations, i.e., relations from and to the same set. Their works use Boolean algebra and therefore, include the concept of complements. However, the authors in [29] talk about heterogeneous relations.

2.5.2 Algebra of Fuzzy Relations

Algebras for fuzzy relations, and therefore, for $\mathcal{L}$-fuzzy relations, require special attention because of their generalized nature. It follows that fuzzy relation algebras are not Boolean relation algebras because they deal with more that two values. The authors in [18] investigated algebraic formalization of fuzzy relations. As because fuzzy relations do not have complements, the authors in [18], intend to replace Schröder rule with the complement-free version of it, the Dedekind rule.

In addition, multiplication by a real number from the interval $[0 \dots 1]$ is common in the study of fuzzy relations. This leads to semi-scalar multiplication and requires appropriate axioms for it. Recall that the semi-scalar kR of a fuzzy relation $R : X \to X$ by a scalar $k \in [0 \dots 1]$ is a fuzzy relation such that $(kR)(x, y) = kR(x, y)$ for all $x, y \in X$. However, such a semi-scalar multiplication might not exist for an arbitrary complete Brouwerian lattice.

Moreover, in our study, heterogeneous $\mathcal{L}$-fuzzy relations are of no less importance than the homogeneous ones.

As a result, none of these algebraic approaches formalizes $\mathcal{L}$-fuzzy relations appropriate for our research. We therefore, switch to the categorical version of it.

2.6 Categories of Relations

In this section, we will discuss about the various categorical approaches that have been introduced in literature to formalize relations. However, our objective is to deduce an abstract algebraic framework for $\mathcal{L}$-fuzzy relations which we would use to define the semantics for the query language of $\mathcal{L}$-fuzzy database.

2.6.1 Categories

Category theory is a relatively young branch of mathematics which is originated from algebraic topology. It provides a bag of concepts which, through abstraction, describes many different structures and formalisms from the various branches of mathematics in a uniform way. A systematic study of categories thus allows us to axiomatically capture the common characteristics of these structures and relate between them by functions. Therefore, categories are essentially collections of objects (abstract version of a mathematical structure) and morphisms (abstract version of structure preserving functions) between them. The two basic properties of a category are the ability to compose morphisms associatively and that each object has a identity morphism that maps to itself. We now present a formal definition of categories.

As defined in [31], A *category* C consists of

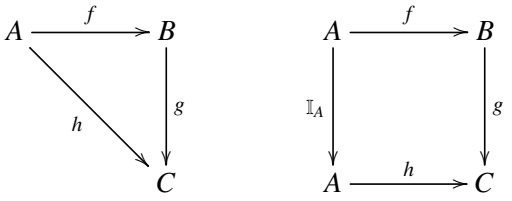
1. a class of objects Obj_C,
2. for every pair of objects A and B a class of morphisms $C[A, B]$,
3. an associative binary (partial) operation ; that maps each pair of morphisms f in $C[A, B]$ and g in $C[B, C]$ to a morphism $f; g$ in $C[A, C]$,
4. for every object A a morphism $\mathbb{I}_A$ such that for all f in $C[A, B]$ and g in $C[C, A]$ we have $\mathbb{I}_A; f = f$ and $g; \mathbb{I}_A = g$.

We will write $f : A \to B$ to express a morphism in $C[A, B]$ which is denotationally consistent with a relation $R : A \to B$ as we would describe relations as morphisms in categories.

The theory of categories has inherent interaction with set theory. While in set theory we deal with membership and equality of those abstract collections called sets, in category theory, we speak about composition and equality of those abstract functions, called morphisms. An object

in set theory is determined by its content while in category theory we study objects in terms of its relationship with other objects of the same category (using morphisms) and of related categories (using functors).

Commonly, categories are visualized by diagrams in which we represent objects by nodes and a morphism f in $C[A, B]$ as an arrow from A to B. This is why, the term "arrow" is interchangeably used for "morphism". We say that such a digram commutes iff $\forall a \in A : (f;g)(a) = h(a)$ and we write $f;g = h$ (Figure 2.17a). However, we can add identity morphisms without affecting commutation as shown in Figure 2.17b.



(A) A category (B) Addition of $\mathbb{I}_A$

FIGURE 2.17: Categorical diagram

A category can be as simple as consisting of a single object 0 and an arrow $\mathbb{I}_0$. Figure 2.17a depicts a category of three objects A, B, and C, together with six morphisms: f, g, h, and three identity morphisms not shown here. Probably the most common example of a category is the category of sets, **Set**, in which sets are objects and functions between sets are treated as morphisms. If we consider the category of all posets, written as **PO**, the posets are objects and monotonic functions are morphisms. Recall that in order theory, a *monotonic function* f between two orders $(P_1, \leqslant_1)$ and $(P_2, \leqslant_2)$ is one that preserves the given order, i.e., $\forall x, y \in P_1 : x \leqslant_1 y \implies f(x) \leqslant_2 f(y)$. The dual of a monotone is an *antitone or order-reversing* which is characterized by the property that for all x and y in its domain, $x \leqslant y \implies f(x) \geqslant f(y)$.

In **Rel**, the category of relations, sets are objects and relations are considered morphisms. Correspondingly, $\mathcal{L}$**-Rel**, the category of $\mathcal{L}$-fuzzy relations, has nonempty sets as objects and $\mathcal{L}$-fuzzy relations for morphisms.

2.6.2 Categorical Terminologies

Before we start the details of categories for $\mathcal{L}$-fuzzy relations, we would like to define some of the categorical terminologies that play important role in our study. These include initial, terminal, and null objects, categorical product, and finally, categorical sum or coproduct.

2.6.2.1 Initial, Terminal, and Null Objects

An object I of a category C is an *initial object* if for every object X in C there exists exactly one morphism from I to X. Dually, an object T is a *terminal object* in C if for every object X in C there exists a unique morphism $X \to T$.

An object, which is both an initial and terminal object, is called a *zero object* or *null object* in C.

This is important to note that initial and terminal objects need not exist in a category. However, if they do exist, they are unique up to isomorphism. It means that if I_1 and I_2 are two initial objects, then there exists a unique isomorphism between them. Moreover, any object which is isomorphic to an initial object, is also considered an initial object. The same concept applies to terminal objects.

As the first example for initial and terminal objects, we consider the category of sets. Here the empty set is the only initial object and every singleton (one-element) set is a terminal object. However, there is no zero object. If we consider the category of non-empty sets, then there are no initial objects. Although every set admits a function from a singleton set in this category, this function is in general not unique. Therefore, the singleton sets are not initials here. If we interpret a poset $(P, \leqslant)$ as a category, then the elements of P are the objects and for $x, y \in P$ there is a single morphism from x to y iff $x \leqslant y$. Such a category has an initial object and a terminal object iff P has a least and a greatest element, respectively.

In the category of relations **Rel**, the empty set is the only zero object. One can easily justify this by the fact that the smallest relation on the empty set is equal to the greatest relation on it, i.e., $\amalg_{\varnothing\varnothing} = \pi_{\varnothing\varnothing}$.

2.6.2.2 Categorical Product

In category theory, the product of two objects forms an abstraction of cartesian product.

Let A and B be two objects of a category C. The *product* of A and B (if it exists) consists of an object $A \times B$ of C and two arrows $\pi : A \times B \to A$ and $\rho : A \times B \to B$ of C such that for every object C of C and every pair of arrows $f : C \to A$ and $g : C \to B$ of C there exists a unique arrow $\langle f, g \rangle : C \to A \times B$ such that $f = \langle f, g \rangle ; \pi$ and $g = \langle f, g \rangle ; \rho$.
Here, the unique morphism $\langle f, g \rangle$ is called the product of morphisms f and g, and π and ρ are called the first and second projections, respectively.

As an example, if we consider the category of sets **Set**, then categorical product is simply the cartesian product. In **Rel**, the categorical product is given by disjoint union of sets. However, if a poset is treated as a category, then products correspond to greatest lower bounds or meets.

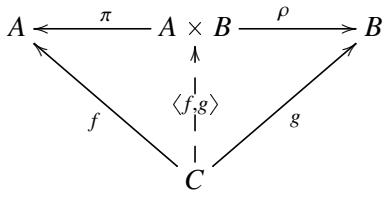


FIGURE 2.18: Categorical product

It is interesting to know that in category theory, every categorical property, structure, or theorem has a dual which is known as the *duality principle*. For example, a morphism $f : A \to B$ in a category C has a dual morphism $f^{op} : B \to A$ in the opposite category C^{op}. The dual of an initial object is a terminal object. Similarly, the dual of a product is a coproduct.

2.6.2.3 Categorical Sum or Coproduct

Coproduct is the category-theoretic dual notion of product, which means that it has the same definition as categorical product with all arrows reversed.

The *coproduct* of two objects $A, B \in Obj_C$ is an object $A + B$ together with two morphisms $\iota : A \to A + B$ and $\kappa : B \to A + B$ such that, for any object C and morphisms $f : A \to C$ and $g : B \to C$, there exists exactly one morphism $[f, g] : A + B \to C$ such that the following diagram commutes.

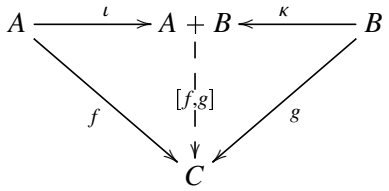


FIGURE 2.19: Categorical sum or coproduct

The morphisms i_1 and i_2 are called injections. As an exmple, in **Set**, the coproduct is the disjoint union of sets. In the category of a poset, the coproduct is the least upper bound, however, if it exists.

2.6.3 Categories of $\mathcal{L}$-Fuzzy Relations

At the beginning of our study for a suitable categorical framework of $\mathcal{L}$-fuzzy relations, we present $\mathcal{L}$-**Rel**, the category of $\mathcal{L}$-fuzzy relations.

Let $\mathcal{L}$ be a complete Brouwerian lattice. Then the structure $\mathcal{L}$-**Rel** is defined as follows:

1. The objects are nonempty sets,
2. A relation $R : A \to B$ between two sets A and B is a function $A \times B \to \mathcal{L}$,
3. For $R : A \to B$ and $S : B \to C$, composition is defined by
   $(R;S)(a,c) := \bigsqcup_{b \in B} R(a,b) \sqcap S(b,c)$,
4. For $R : A \to B$, the converse is defined by $R^{\smile}(a,b) := R(b,a)$,
5. Meet and join for $R, R_2 : A \to B$ are defined by
   $(R \sqcap R_2)(a,b) := R(a,b) \sqcap R_1(a,b)$
   $(R \sqcup R_2)(a,b) := R(a,b) \sqcup R_1(a,b)$
6. The universal, zero, and the identity elements are defined by

$$\top_{AB}(x,y) := 1, \quad \bot_{AB}(x,y) := 0$$

$$\mathbb{I}_A(a_1,a_2) = \begin{cases} 0 & \text{if } a_1 \neq a_2 \\ 1 & \text{if } a_1 = a_2 \end{cases}$$

While $\mathcal{L}$-**Rel** defines the basic operations on $\mathcal{L}$-fuzzy relations, we need a stronger theory with suitable axioms to define other algebraic operations and rules. In [4], Freyd and Scedrov introduced and extended "allegories" as a categorical relational calculus.

2.6.3.1 Allegories

As defined in [31], an *allegory* $\mathcal{R}$ is a category that satisfies the following:

1. For all objects A and B the class of morphisms $\mathcal{R}[A,B]$ is a lower semilattice. Meet and the induced ordering are denoted by $\sqcap, \sqsubseteq$, respectively. The elements in $\mathcal{R}[A,B]$ are called relations.
2. There is a monotone operation $^{\smile}$ (called the converse operation) such that for all relations $Q, R : A \to B$ and $S : B \to C$ the following holds:

$$Q;S^{\smile} = S^{\smile};Q^{\smile} \quad \text{and} \quad (Q^{\smile})^{\smile} = Q$$

3. For all relations $Q : A \to B, R, S : B \to C$ we have

$$Q;(R \sqcap S) \sqsubseteq Q;R \sqcap Q;S$$

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

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

Recalling the operations in 2.4.1 and their properties, it can be proved that the category of $\mathcal{L}$-fuzzy relations $\mathcal{L}$-**Rel** with the set theoretic intersection and conversion as respectively the meet and converse forms an allegory. However, this structure only satisfies properties 1-7 as in 2.4.1. For the remaining, we need to add the union operation and its associated axioms.

The collection of binary relations on a fixed set, on the other hand, constitutes a distributive lattice with a least element [31]. Therefore, we intend to use distributive lattices as the order in allegories replacing the lower semilattices.

A *distributive allegory* $\mathcal{R}$ is an allegory satisfying the following:

1. The classes $\mathcal{R}[A, B]$ are distributive lattices with a least element. Union and the least element are denoted by $\sqcup, \perp\!\!\!\perp_{AB}$, respectively.
2. For all relations $Q : A \to B$ we have $Q; \perp\!\!\!\perp_{BC} = \perp\!\!\!\perp_{AC}$.
3. For all relations $Q : A \to B, R, S : B \to C$ we have

$$Q; (R \sqcup S) = Q; R \sqcup Q; S$$

Having defined this distributive structure, we can show that the allegory $\mathcal{L}$-**Rel** of $\mathcal{L}$-fuzzy relations with set theoretic union is a distributive allegory [31].

A distributive allegory $\mathcal{R}$ is said to be *locally complete* if $\mathcal{R}[A, B]$ is a complete lattice for all objects A and B and if composition and finite intersection distribute over arbitrary unions: that is, given $Q : A \to B, R_i : B \to C$ for $i \in I$, we have $Q; (\bigsqcup_{i \in I} R_i) = \bigsqcup_{i \in I} (Q; R_i)$ [4].

By adding the abstraction of the division (residual) operation of relation algebra to a distributive allegory, we get a division allegory.

A *division allegory* $\mathcal{R}$ is a distributive allegory such that ; has an upper left adjoint, i.e., for all relations $R : B \to C$ and $S : A \to C$ there is a relation $S/R : A \to B$ (called the left residual of S and R) such that for all $Q : A \to B$ the following holds:

$$Q; R \sqsubseteq S \iff Q \sqsubseteq S/R.$$

Similarly, ; has an upper right adjoint, denoted by $Q \backslash S$ and called the right residual of S and Q.

Another important categorical approach to formalize binary relations has been proposed by Olivier and Sarrato in [22, 23]. They used the name "Dedekind categories" for which they

required the underlying order structure to be complete. As allegories, Dedekind categories are equivalent to locally complete distributive allegories [4]. In our study, we use Dedekind categories as the fundamental theory to reason about $\mathcal{L}$-fuzzy relations.

2.6.3.2 Dedekind Categories

A *Dedekind category* $\mathcal{R}$ is a category satisfying the following:

1. For all objects A and B the collection $\mathcal{R}[A, B]$ of morphisms of A into B is a complete Heyting algebra. Meet, join, the induced ordering, the least element, and the greatest element are denoted by $\sqcap, \sqcup, \sqsubseteq, \top_{AB}$, **and** $\bot_{AB}$, respectively.

2. There is a monotone operation $^\smile$ (called converse) mapping a relation $Q : A \to B$ to $Q^\smile : B \to A$ such that for all relations $Q : A \to B$ and $R : B \to C$ the following holds:

$$Q;R^\smile = R^\smile;Q^\smile \quad \text{and} \quad (Q^\smile)^\smile = Q.$$

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

$$(Q;R) \sqcap S \sqsubseteq Q;(R \sqcap (Q^\smile;S))$$

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

$$X;R \sqsubseteq S \iff X \sqsubseteq S/R.$$

Recall that the left residual defined for the above structure implies the existence of a right residual characterized by

$$Q;Y \sqsubseteq S \implies Y \sqsubseteq Q\backslash S.$$

In fact, $Q\backslash S = (S^\smile/Q^\smile)^\smile$. Also, note that both residuals are monotone in one argument and antitone in the other. This means that, if $S \sqsubseteq S', R' \sqsubseteq R$ and $Q' \sqsubseteq Q$, then $S/R \sqsubseteq S'/R'$ and $Q\backslash S \sqsubseteq Q'\backslash S'$.

It can be shown that the class of $\mathcal{L}$-fuzzy relations form a Dedekind category. Unfortunately, this category is too weak to express some standard properties of $\mathcal{L}$-fuzzy relations, particularly, the 0-1 crispness. By the use of scalar elements with certain assumptions on the underlying lattice, the authors in [12, 17] attempted to define several notion of crispness for an arbitrary Dedekind category. Moreover, the author in [32] proved that there is no formula in the theory of Dedekind categories expressing the fact that a given $\mathcal{L}$-fuzzy relation is 0-1 crisp. Therefore, we

need an extension of this theory to define a suitable algebraic framework for $\mathcal{L}$-fuzzy relations. This was done in [35] by means of two arrow operations, the up-arrow ($\uparrow$) and the down-arrow ($\downarrow$). These new structures are then called Arrow categories and serve as a complete algebraic theory of $\mathcal{L}$-fuzzy relations in our study. Recall from Section 2.4.4 that the $\uparrow$ and $\downarrow$ operations, when applied to a relation R, respectively produce the least 0-1 crisp relation containing R and the greatest 0-1 crisp relation that R contains.

Before switching to the definition of arrow categories, we would like to define some important constructions within Dedekind categories.

The category **Rel** of binary relations between sets with the usual definition of the operations form a Dedekind category. In **Rel**, the empty set and singleton sets play an important role. We recall that the empty set is a zero object in **Rel** (Section 2.6.2.1). Similarly, we call an object 0 of a Dedekind category an zero object iff $\perp\!\!\!\perp_{00} = \top_{00}$. Singleton sets in **Rel** are terminal objects in the subcategory of maps. In **Rel** itself they can be characterized as so-called units. A *unit* 1 is an object of a Dedekind category so that $\mathbb{I}_1 = \top_{11}$ and $\top_{A1}$ is total for all objects A.

The relational product of two objects A and B in a Dedekind category is the object $A \times B$ together with two relations $\pi : A \times B \to A$ and $\rho : A \times B \to B$ so that the following equations hold

$$\pi^\smile;\pi \sqsubseteq \mathbb{I}_A, \quad \rho^\smile;\rho \sqsubseteq \mathbb{I}_B, \quad \pi^\smile;\rho = \top_{AB}, \quad \pi;\pi^\smile \sqcap \rho\rho^\smile = \mathbb{I}_{A\times B}.$$

Note that the relational product is equivalent to the categorical product (Section 2.6.2.2) in the subcategory of maps. Similarly, the notion of relational sum (categorical coproduct in maps) of two objects A and B is defined to be another object $A + B$ together with two relations $\iota : A \to A + B$ and $\kappa : B \to A + B$ with the following axioms being satisfied.

$$\iota;\iota^\smile = \mathbb{I}_A, \quad \kappa;\kappa^\smile = \mathbb{I}_B, \quad \iota;\kappa^\smile = \perp\!\!\!\perp_{AB}, \quad \iota^\smile;\iota \sqcup \kappa^\smile;\kappa = \mathbb{I}_{A+B}$$

2.6.3.3 Arrow Categories

An arrow category $\mathcal{A}$ is a Dedekind category with $\top_{AB} \neq \perp\!\!\!\perp_{AB}$ for all A, B and two operations $\uparrow$ and $\downarrow$ satisfying the following axioms for all $Q, R : A \to B$, $S : B \to A$, and $T : B \to C$.

1. $R^\uparrow, R^\downarrow : A \to B$.
2. $(^\uparrow, ^\downarrow)$ forms a Galois correspondence, i.e., $Q^\uparrow \sqsubseteq R$ iff $Q \sqsubseteq R^\downarrow$.
3. $(S^\smile;T^\downarrow)^\uparrow = S^{\uparrow\smile};T^\downarrow$.
4. $(Q \sqcap R^\downarrow)^\uparrow = Q^\uparrow \sqcap R^\downarrow$.
5. If $\alpha_A \neq \perp\!\!\!\perp_{AA}$ is a non-zero scalar then $\alpha_A^\uparrow = \mathbb{I}_A$.

A relation that satisfies $R^{\uparrow} = R$, or equivalently $R^{\downarrow} = R$, is called crisp. Notice that the complete Heyting algebra of scalar relations on each object are isomorphic. In addition, we recall from Section 2.4.4 that $(\alpha \backslash R)^{\downarrow}$ is called the α-cut of R while α is a scalar on R.

In fuzzy theory t-norms and t-conorms are essential for defining new operations for fuzzy sets or relations. For the details of t-norms and t-conorms, we refer to [8]. A generalization of these operations for arbitrary complete lattices was introduced in [7], called complete lattice-ordered semigroups.

A *semigroup* in mathematics is a set S together with a binary operation $* : S \times S \to S$ such that for any $a, b, c \in S$, the equation $(a * b) * c = a * (b * c)$ holds.

A *complete lattice-order semigroup*, as defined in [7], is a complete lattice $\mathcal{L}$ which is also a semigroup with identity under $*$ and which satisfies the following distributive laws,

$$a * \bigvee_i b_i = \bigvee_i (a * b_i) \quad \text{and} \quad (\bigvee_i a_i) * b = \bigvee_i (a_i * b)$$

Given such an operation $* : \mathcal{L} \times \mathcal{L} \to \mathcal{L}$ we may define a new meet or composition based operation on $\mathcal{L}$-fuzzy relations $Q, R : A \to B$ and $S : B \to C$ by

$$(Q \sqcap_* R)(x, y) = Q(x, y) * R(x, y) \quad \text{and} \quad (Q ;_* S)(x, z) = \bigsqcup_{y \in B} Q(x, y) * S(y, z).$$

In an abstract arrow category we require $*$ to be defined of the complete Heyting algebra of scalar elements. As shown in [31, 34] the corresponding operations on relations are defined as follows.

Let Q, R be relations, $\otimes \in \{\sqcap, ;\}$ such that $Q \otimes R$ is defined, and $*$ be the operation of a complete lattice-ordered semigroup on the set of scalar relations. Then we define

$$Q \otimes_* R := \bigsqcup_{\alpha,\beta \text{ scalars}} (\alpha * \beta); ((\alpha \backslash Q)^{\downarrow} \otimes (\beta \backslash R)^{\downarrow}).$$

We distinguish two kinds of commutative complete lattice-ordered semigroup operations corresponding to either t-norms or t-conorms. If the neutral element of the semigroup is equal to 1 (greatest element of the lattice) we call the operation a t-norm like operation. t-norm like operations will be used together with $\sqcap$ and ; to form new operations on relations. If the neutral element of the semigroup is equal to 0 (smallest element of the lattice) we call the operation a t-conorm like operation. These operations will only be used together with $\sqcap$.

Notice that we also have residuals based on semigroup operations. They are defined as the left (resp. right) adjoint of $;_*$. For more details on these constructions we refer to [31].

Chapter 3

$\mathcal{L}$-Fuzzy Structured Query Language

In this chapter we present a semantics for the query language of lattice-based fuzzy databases, commonly known as $\mathcal{L}$-fuzzy databases. Arrow categories, as discussed in the previous chapter, comprises a complete algebraic theory to abstractly work with $\mathcal{L}$-fuzzy databases. $\mathcal{L}$-fuzzy databases extend or, generalize to be precise, classical relational databases by introducing additional constructs in order to be able to handle inexactness in data. In a similar fashion, the SQL extension for $\mathcal{L}$-fuzzy databases, $\mathcal{L}$FSQL, captures appropriate linguistic operations to define and manipulate imprecise data in an $\mathcal{L}$-fuzzy database. Apart from the syntactic additions to $\mathcal{L}$FSQL, our main concern here is to present a formal semantics of this language in the theory of arrow categories. In doing so, we will be using the different mathematical theories and primitive structures that have been discussed in Chapter 2.

3.1 $\mathcal{L}$-Fuzzy Databases

Similar to a regular database a table in an $\mathcal{L}$-fuzzy database contains a collection of objects represented by a set of attributes or columns. A row of attribute-values is also called a tuple. Each attribute has a set of possible values from where an object or tuple takes values for that attribute. This value-set is commonly known as the domain of the attribute. Before going further, we would like to recall the contact-list example from Chapter 1. Let us assume that the following table contains some of the contact records on Peter's cell phone.

Name	**Phone no.**	**Address**	***Distance***	**Date of birth**	***Age***
John	+12222222222	19 York street	3	*null*	*null*
Kevin	+13333333333	*null*	*null*	*null*	*null*
Linda	+14444444444	*null*	*null*	March 15, 1970	45
Richy	+15555555555	125 Perfect road	6	August 29, 1998	17
Tijo	+16666666666	*null*	*null*	April 4, 1996	19

To make this example simpler and more useful, we shrink the above table to the following by discarding the columns "Address" and "Date of Birth". At the same time we introduce fuzzy constructions like linguistic labels and set of values as table-entries.

Name	**Phone no.**	**Distance**	**Age**
John	+12222222222	3	$Old
Kevin	+13333333333	$Close	$\{20, 21, 22\}$
Linda	+14444444444	*null*	45
Richy	+15555555555	6	17
Tijo	+16666666666	$\{4, 5, 6\}$	19

FIGURE 3.1: $\mathcal{L}$-fuzzy contact list

Algebraically, each construct of an $\mathcal{L}$-fuzzy database is represented by a relation. Eventually, each table is modelled as a big relation of all the attribute-value pairs whereas a database is an even bigger relation formed off all of its tables.

Unlike classical databases, every single entry for an attribute at a tuple in an $\mathcal{L}$-fuzzy database is an $\mathcal{L}$-fuzzy subset of the corresponding attribute-domain. However, a single (crisp) value x is just an abbreviation for a fuzzy set that has degree 1 for x and 0 otherwise. For example, the *Name* attribute in our table has the domain {John,Kevin,Linda,Richy,Tijo}. However, we define the domains for *Age* and *Distance* to be the sets {1,2,..,120} and {1,2,..,100}, respectively. This means, for example, that someone could have an age between 1 to 120 and so on. As mentioned before, a concrete value, "Kevin" for example, would be denoted by the $\mathcal{L}$-fuzzy subset $\{0/\text{John}, 1/\text{Kevin}, 0/\text{Linda}, 0/\text{Richy}, 0/\text{Tijo}\}$ or simply $\{1/\text{Kevin}\}$. A *null* which means absence of data, would be represented by the empty set $\{\}$. In a similar fashion, the set $\{4, 5, 6\}$ indicates that only 4, 5, and 6 of the domain of "Distance" have degrees greater than 0 in it. We will see some examples later in this chapter.

Each of these domains comes with some comparison operations associated with them. *Unordered domains*, where elements can not be put into some order, have at least the equality operation defined on it. As an example of such domain we can consider the Boolean attribute "Active" in some "Users" table which indicates whether an user account is active or not. This is clear that in this domain we can compare values for equality only. In our first table above,

the attribute "Address" has an unordered domain. Although, the "Name" field seems to be unordered at first sight, but it can actually be sorted lexicographically. In addition to equality, *ordered domains* provide $\leqslant$ and the respective strict order $<$ such that for any two values v_1 and v_2, $v_1 \leqslant v_2$ iff $v_1 < v_2$ or $v_1 = v_2$. All the attributes in Table 3.1 have ordered domains as the values within a domain are comparable in terms of equality, less than, and less than or equal. Notice that one could easily obtain the corresponding reverse order, greater than ($>$) for example, by taking complement of the smaller than or equal ($\leqslant$) relation. Later in this chapter we will see examples on how to use these operations in queries.

There are some domains that come with even binary approximate equalities ($\equiv$). Such an operation returns an $\mathcal{L}$-value as the membership degree which indicates the level up to which two given elements are considered to be equal. To illustrate this, let us say we are interested in sameness of age. We might say that two people have the same age if their actual ages differ by not more than one year. In that case, we use a higher lattice value to emphasize on closeness of the values. As the difference between the two ages grows, we start to assign lower degrees respectively. Note that an approximate equality is reflexive in nature. This is because any value is approximately equal to itself with the highest degree of membership i.e., $v \equiv v = 1$. Moreover, such an approximate equality is needed to be symmetric, i.e., $v_1 \equiv v_2 = v_2 \equiv v_1$ [31]. However, it cannot be transitive. This means that, for example, if an age value v_1 approximately equals to another age v_2 and v_2 in turn approximately equals to a third age v_3, then v_1 might not be approximately equal to v_3 because of the cumulative differences.

3.1.1 Metadatabase

Likewise regular databases, $\mathcal{L}$-fuzzy databases use metadatabase to store database configuration settings supporting metadata management. Typically an $\mathcal{L}$-metadatabase contains the lattice $\mathcal{L}$, any t-norm and t-conorm like operation on $\mathcal{L}$, and the $\mathcal{L}$-fuzzy sets that represents the linguistic labels. Throughout this chapter, we choose to use the following lattice called D_6 as the target of all membership functions.

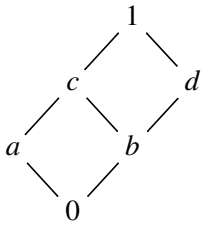


FIGURE 3.2: A complete distributive lattice D_6

It can be shown that D_6 is isomorphic to the product $\{0,1\} \times \{0,m,1\} = \{(0,0),(0,m),(0,1),$-$(1,0),(1,m),(1,1)\}$ of two linear orderings $0 \leqslant 1$ and $0 \leqslant m \leqslant 1$. This makes D_6 an ideal

structure to model two aspects of membership similar to the cell phone example in Section 1.3 of Chapter 1. The first aspect is a "yes-no" and the second aspect a "yes-maybe-no" relationship. For example, $a = (0, m)$ represents the degree of not being in the set with respect to the first aspect and "maybe" with respect to the second aspect.

As discussed before, *t-norm* resp. *t-conorm (or s-norm)* generalize the union and intersection on $\mathcal{L}$-fuzzy sets. Such an operation must have these properties: commutativity, associativity, monotonicity, and border conditions [8]. Some of these operations are listed below. Note that the minimum function is the largest t-norm while drastic product comprises the smallest. Drastic sum, on the other hand, is the largest t-conorm with the maximum function being the smallest t-conorm. Also note that all of these operations are for the unit interval $[0, \ldots, 1]$. However, some of them work in a more general setting where the *min* becomes *meet*, *max* becomes *join* and the drastic product and sum work unmodified.

t-norms	**Expression** ($f(x,y)$)	**t-conorms**	**Expression** ($f(x,y)$)
Minimum	$min(x,y)$	Maximum	$max(x,y)$
Algebraic product	$x \times y$	Algebraic sum	$x + y - xy$
Drastic product	$\begin{cases} xy, \text{ if } y = 1 \\ y, \text{ if } x = 1 \\ 0, \text{otherwise} \end{cases}$	Drastic sum	$\begin{cases} x, \text{ if } y = 0 \\ y, \text{ if } x = 0 \\ 1, \text{otherwise} \end{cases}$
Einstein product	$\frac{xy}{1+(1-x)+(1-y)}$	Einstein sum	$\frac{x+y}{1+xy}$

Some $\mathcal{L}$-metadatabases also contain pre-implemented characteristic functions, i.e., functions that map from certain domains into the lattice $\mathcal{L}$. These functions facilitate the creation of new $\mathcal{L}$-fuzzy subsets explicitly. Like ordinary functions, they can be parametric. In addition, they might be restricted to the lattice they are defined with. Figure 3.3 shows such a function in Java that accepts two names as parameters and returns a D_6 value as the degree of lexicographic closeness between the two strings. Note the dependency of this function on D_6 as it doesn't work with a different lattice. If $\mathcal{L}$ is the unit interval, then a preimplemented function would generate triangular or trapezoidal fuzzy subsets of some linear ordered set.

3.1.2 Linguistic Labels and $\mathcal{L}$-Fuzzy Sets

As we know every individual entry in a table of an $\mathcal{L}$-fuzzy database could be modelled as an $\mathcal{L}$-fuzzy set ($\mathcal{L}$-fuzzy subset of the domain, to be precise), irrespective of whether it is a single-valued entry or a multivalued set. In our $\mathcal{L}$-database, the entry for Kevin's age which is $\{20, 21, 22\}$ might be represented by the $\mathcal{L}$-fuzzy set $\{d/20, c/21, a/22\}$. For the two linguistic labels \$Old and \$Close which are stored in the metadatabse, we assume that any age greater

```
public static LatticeValue nameMatching(String name1, String name2){
    //convert names into uppercase
    name1 = name1.toUpperCase();
    name2 = name2.toUpperCase();
    //find out the minimum length of the two
    int minLength;
    if(name1.length() < name2.length())
        minLength = name1.length();
    else
        minLength = name2.length();
    float sum = 0;
    int diff;
    int matches = 0;    // count the characters that match
    for(int i = 0; i < minLength; i++){ //loop through the strings for the minimum length
        diff = Math.abs(name1.charAt(i) - name2.charAt(i)); // lexicographic distance
        if(diff == 0) //if same character
            matches++;
        sum = sum + diff;
    }
    //if no character matched, divide by 1, otherwise divide by the number of matches
    float result = sum/(matches == 0 ? 1 : matches);
    // return lattice value accordingly
    if(result <= 1)
        return '1';
    else if(result > 1 && result <= 5)
        return 'd';
    else if(result > 5 && result <= 15)
        return 'b';
    else
        return '0';
}
```

FIGURE 3.3: A Java method as preimplemented function

than 50 would be considered "old" and a distance less than or equal to 5 km would be regarded as "close". Furthermore, we define the label \$Young for the ages less than 21.

$$\$\text{Old} = \{\ldots, 0/39, 0/40, \ldots, b/47, c/48, d/49, 1/50, 1/51, 1/52, \ldots\}$$
$$\$\text{Young} = \{1/1, 1/2, 1/3, \ldots, 1/19, 1/20, 0/21, 0/22, \ldots\}$$
$$\$\text{Close} = \{1/1, 1/2, 1/3, 1/4, 1/5, c/6, a/7, 0/8, 0/9, \ldots\}$$

In addition to the $\mathcal{L}$-fuzzy subsets in the metadatabase, we can define new $\mathcal{L}$-fuzzy sets either explicitly or by modifying already existing ones. These newly created sets can then be stored back to the metadatabase with new names. Also, they can be directly used in query statements.

For an arbitrary domain $D = \{d_1, d_2, \ldots, d_n\}$, an $\mathcal{L}$-fuzzy subset of D is expressed as $C = \{l_1/d_1, \ldots, l_n/d_n\}$ where $l_i \in \mathcal{L}$ for $1 \leqslant i \leqslant n$. Such a set has the following membership function.

$$\chi_C(x) = \begin{cases} l_1 & \text{iff } x = d_1, \\ \vdots & \quad\vdots \\ l_n & \text{iff } x = d_n. \\ 0 & \text{otherwise} \end{cases}$$

As mentioned before, preimplemented functions which are stored in the metadatabase can be used to define $\mathcal{L}$-fzzy sets. For such a function f, we define the corresponding $\mathcal{L}$-fuzzy set to be

$\#f$ which use f as its membership function.

Obtaining new $\mathcal{L}$-fuzzy sets from previously defined ones could be done in either of the following two ways.

1. **By taking lower or upper bounds of an $\mathcal{L}$-fuzzy set**: In this approach we take the upper bounds or lower bounds of $\mathcal{L}$-fuzzy sets to produce new ones. It requires the domain to have the order defined, so it can not be applied to unordered domains. In Chapter 2 we have seen how to compute the lower bounds and upper bounds of some relation R over the order E ($ubd_E(R)$ and $lbd_E(R)$). For an $\mathcal{L}$-fuzzy set m on domain D which has an order E, we can get the set of upper bounds by computing the meet of the relative pseudocomplements $\bigwedge_{y\in m} : d_y \rightarrow d_{yEm}$ for every $x \in D$ where d_y is the degree up to which y is in m and d_{yEx} is the entry (y, x) in the relation E. As an example, lets consider a domain of 1 to 4km for the attribute "Distance" and an arbitrary $\mathcal{L}$-fuzzy set $m = \{c/2, b/3\}$. While computing the upper bounds for m we get, for $x = 1, c \rightarrow 0 \wedge b \rightarrow 0 = 0 \wedge a = 0$, for $x = 2$ we get $c \rightarrow 1 \wedge b \rightarrow 0 = 1 \wedge a = a$, and so on. Eventually, it produces the new $\mathcal{L}$-fiuzzy set $\{0/1, a/2, 1/3, 1/4\}$ or simply $\{a/2, 1/3, 1/4\}$. Note that the computation of lower bounds and upper bounds can also be based on a t-norm like operation ($ubd(*, m)$) stored in the metadatabase in which case $*$ replaces the meet in a composition and the *lbd* and *ubd* are based on the residual corresponding to that composition. At the end of Chapter 2, we have seen how to define $*$-based composition ($;_*$) where the $*$ represents an operation of a complete lattice-ordered semigroup. As compositions and residuals are called adjoints to each other, whenever we have ; defined on a complete lattice structure, the residuals do exist automatically.

2. **By intensifying or weakening approximate equality**: This approach is applicable to a domain that provides approximate equality $\equiv$ defined on it. This approximate relation can be used to modify the notion given by an $\mathcal{L}$-fuzzy set m. In order to intensify it, we define extremely$(\equiv, m)$ and very$(\equiv, m)$. On the other hand, more_or_less$(\equiv, m)$ and roughly$(\equiv, m)$ will be used in order for weakening the approximation. Notice that these sets satisfy the following chain of inclusions

$$\text{extremely}(\equiv, m) \sqsubseteq \text{very}(\equiv, m) \sqsubseteq m \sqsubseteq \text{more\_or\_less}(\equiv, m) \sqsubseteq \text{roughly}(\equiv, m).$$

While defining the semantics later in this chapter, we would see an example on how to compute these approximate equalities componentwise between $\mathcal{L}$-fuzzy sets on the same domain. For now, if we think of a domain of 1 to 4km for the attribute "Distance" and a difference of 1km to be considered "roughly equal", then the following relation might be

one representation the approximation.

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 1 & d & 0 & 0 \\ d & 1 & c & 0 \\ 0 & c & 1 & d \\ 0 & 0 & d & 1 \end{pmatrix} \end{array}$$

3.2 $\mathcal{L}$-Fuzzy Structured Query Language ($\mathcal{L}$FSQL)

As we already know, the language of $\mathcal{L}$FSQL extends the query language for fuzzy databases (FSQL) in order to deal with lattice-based membership values. Our version of $\mathcal{L}$FSQL is inspired by the work on FSQL in [8], [9] and [10]. This section begins with a through investigation of the syntactic definitions of different DDL and DML operations namely, CREATE, INSERT, DELETE, and SELECT. In the next section we present a semantics of these statements in the abstract theory of arrow categories. However, we would like to start with the different binary comparison operations that are available for $\mathcal{L}$-fuzzy sets.

3.2.1 $\mathcal{L}$-Fuzzy Comparators

A binary comparison operator or comparator on a set A can be thought of a relation $C : A \to A$ which compares two elements of A and produce a Boolean value in the result. Typical examples of such comparators include $=$, $\leqslant$ and $<$. These comparators are often used in queries to combine multiple conditions as well as in building complex ones. As we know every single entry of a table in an $\mathcal{L}$-fuzzy database is modelled as an $\mathcal{L}$-fuzzy set irrespective of whether it is a single value or an explicit set. As a result, for such a comparator to be available for $\mathcal{L}$-fuzzy database, it needs to be lifted from comparing elements to comparing sets. This can be done in multiple ways. In our study we adapt the notation used in [8] and define the followings based on a typical binary comparator C. We would like to use the regular set $\{4,5,6\}$ from our database which represents the probable distance from Peter's place to his friend Tijo's place.

1. *Possibility fuzzy comparison*, denoted by FC: As the name suggests, this comparator binds a possibility factor to the condition. As an example, let us say we want to evaluate the condition $\{4,5,6\}$ $F=$ \$*Close* where \$Close is a linguistic label stored in the meta-database that considers all distances smaller or equal to 5km as *Close*. That is, we would like to know if Tijo's place is possibly *close* to Peter's place. In other words, is it possible that there is a distance value in $\{4,5,6\}$ which is regarded as *Close*? The answer is yes, because both 4km and 5km are considered *Close*.

2. *Necessity fuzzy comparison (NFC)*: This type of comparison produces a "true" if the set of values on the left hand side of the comparator is essentially a subset of the set on the right. However, there are exceptions as we would see shortly. In our example, the comparison $\{4, 5, 6\}$ $NF =$ \$*Close* would require all the values of 4, 5, and 6 to be in the set *Close* for the output to be a "true". As because 6 is not considered *Close*, the answer is a "false" this time.

   This is intuitive that for a binary comparator C which is symmetric, a necessity fuzzy comparator based on that doesn't required to be so. As an example, the set $\{2, 3\}$ of distances is necessarily fuzzy equal to \$*Close* but not the other way around as there are distances, namely, 1, 4, and 5, which are also considered *Close*.

From the example above one could easily infer that possibility comparator generalizes necessity comparator, or in other words, fuzzy-necessity is included in fuzzy-possibility. Eventually this lead to the fact that a query with a possibility comparator returns more tuples than its necessity counterpart [9]. But this is true only when the fuzzy sets are total, i.e., sets for which the join of all degrees equals 1 (the top element) [2]. In FSQL (as in [9]) all fuzzy sets are trapezoidal and therefore, are normalized. Recall that a normalized fuzzy set is one that has at least one element with degree 1. From the two definitions it is evident that every normalized fuzzy set is itself total, but not vice versa. However, if we consider the lattice $\mathcal{L}$ for membership degrees to be the unit interval $[0 \dots 1]$, then both becomes equivalent. Relation algebraically an $\mathcal{L}$-fuzzy set expressed as a vector $v : 1 \to A$ is total iff $\mathbb{I}_A \sqsubseteq vv^{\smile}$ and normalized iff $v^{\downarrow} \neq 0$.

In our study not all fuzzy sets are total, so a possibility comparator does not always produce more tuples. As an example, let us say we want to check if the empty set is necessarily equal to \$Close (i.e., $\{\}$ $NF =$ \$*Close*). The answer is intuitively "true". But when the condition uses possibility comparator instead, i.e., $\{\}$ $F =$ \$*Close*, then it returns "false" because at least one of the elements of $\{\}$ should have to be in *Close* which is not satisfied.

3.2.2 The CREATE Statement

The CREATE statement does the first step in building a database. It creates a new table with the attributes specified by the user. Such a table is initially empty, i.e, it has no rows in it. Along with the attributes, the user has to provide the corresponding domains. Here is the general form of the CREATE statement.

$$\text{CREATE TABLE } R(A_1 : D_1, \dots, A_n : D_n);,$$

where R is the new table to be crated with attributes $A_1, \dots, A_n$ and $D_1, \dots, D_n$ are the corresponding domains. For a CREATE statement to be successful it is required that R is a new name

and the domains are defined. We write $R.A_i$ to indicate the column A_i of R. However, if the attribute A_i is unique in regard to the context, we remove the prefix R from $R.A_i$.

3.2.3 The INSERT Statement

Once a table is created we add tuples by using the DML statement INSERT. It has the following general form.

$$\text{INSERT INTO } R \text{ VALUES } (m_1, \ldots, m_n);,$$

where R is an existing table with attributes $A_1{:}D_1, \ldots, A_n{:}D_n$ and $m_1, \ldots, m_n$ are $\mathcal{L}$-fuzzy subsets defined on the domains $D_1, \ldots, D_n$, respectively. As mentioned before one can define these fuzzy sets within the INSERT statement or refer to a linguistic label in the metadatabase.

3.2.4 The WHERE Clause

The WHERE clause is the central part of most SQL queries. In our language $\mathcal{L}$FSQL it is used to specify a condition in a SELECT statement as we will see shortly. In classical SQL the WHERE clause is also used in other DML operations, for example the DELETE and the UPDATE operations, as well as in building more complex queries.

A primitive comparison in such a WHERE clause has the form $S\ LFC\ S'$ where S, S' are either some attributes of the form $R.A$ or $\mathcal{L}$-fuzzy sets and LFC is an $\mathcal{L}$-fuzzy comparator which might be a necessity fuzzy (NFC) comparator or a possibility fuzzy comparator (FC) based on some binary comparator C. Note that, for a comparison $S\ LFC\ S'$ to be syntactically correct it is required that the domain of S, S', and C are the same. If successfully evaluated, such a comparison returns a degree from $\mathcal{L}$ for each tuple of R indicating up to which it satisfies the condition.

In $\mathcal{L}$FSQL a comparison can be equipped with a threshold if required. Such a comparison, C THOLD l for example, returns the degree d_C if $d_C \geqslant l$ in $\mathcal{L}$, otherwise it returns a 0.

Compound comparison can be formed off primitive comparisons by using logical connectives AND or OR. In $\mathcal{L}$FSQL these refer to the meet and join operation on $\mathcal{L}$, respectively. For example, if C_1 AND C_2 is such a comparison with d_{C_1} and d_{C_2} being the degrees of the individual comparisons, then the final outcome is the degree $(d_{C_1} \wedge d_{C_2}) \in \mathcal{L}$. In addition, both AND or OR can be based on a t-norm like or t-conorm like operation, respectively.

Finally, a WHERE clause consists of the keyword WHERE followed by a comparison, either primitive or compound. Two examples are given below.

WHERE $R.Distance\ NF = \$Close$ THOLD l AND $R.Age\ F < \$Young$,

WHERE $R.Age\ F = S.Age$ OR$(*)$ $R.Height\ F > \$Short$,

For each tuple of R the former returns a degree which is the meet of the two individual degrees resp. indicating up to which its Distance entry is necessarily *Close* with at least l and its Age entry is possibly *Young*.

3.2.5 The DELETE Statement:

As in FSQL we use the DELETE statement in $\mathcal{L}$FSQL to remove tuples from a table. It has the following simple form.

DELETE FROM R WHERE wh;,

where R is the name of an existing table and wh is a WHERE clause discussed earlier. Such a syntactically correct DELETE statement when executed, deletes all tuples from R for which wh produces a non-zero degree.

3.2.6 The SELECT Statement:

The SELECT statement is the primary Data Manipulation Language (DML) operation used to retrieve information from a database. It has the following general form.

SELECT $A_1, \ldots, A_m$ FROM $R_1, \ldots, R_n$ WHERE wh;

Here $A_1, \ldots, A_m$ are the attributes to be selected from the tables $R_1, \ldots, R_n$ and wh is a condition. The syntactic requirement for a SELECT statement to be executed properly is that each of the attributes uniquely identify the table it is selected from. In that case it returns a new table with attributes $A_1, \ldots, A_m$ which is made up off the old ones. Only those tuples from the combined table that satisfy the condition wh with degree not equal to 0, qualify in the new table. Note that each column of a tuple in the resultant table takes its value from the respective column in one of these old tables.

3.2.7 An $\mathcal{L}$FSQL Query Example

For demonstrating the operations in the remaining of this chapter, we consider this example on our $\mathcal{L}$-database: suppose Peter needs someone's quick help in lifting some household stuffs

(referring to the Database Table 3.1). In other words, he might be helped by someone who is young and lives close to him. A SELECT query for this in our $\mathcal{L}$FSQL might be

SELECT Name, Phone no.

FROM CONTACT

WHERE Distance NF=\$*Close* THOLD b AND Age $F<$ \$Young;

This query when executed returns the names and phone numbers of those who lives close to Peter, i.e., within 5km of his place, and also whose age is 20 or less. The resultant table is shown in Figure 3.4.

Name	**Phone no.**
Kevin	+13333333333

FIGURE 3.4: Query output

From the Table 3.1 we find that although John lives close to Peter (3km), he is aged enough not to qualify the query condition. Kevin, on the other hand, fulfils both the conditions and so is included in the result. The distance between Peter's and Linda's place is unknown and also she is over aged to be considered. Finally, Richy and Tijo are not listed as their entries in the table do not satisfy the distance requirement of the query, however, although they are young.

3.2.8 Inner Joins

Database join operations, as the name suggest, are used to combine tables together, unsually within a SELECT statement. An INNER JOIN in $\mathcal{L}$FSQL has the the following basic form.

$$R_1 \text{ INNER JOIN } R_2 \text{ ON } R_1.A_{1i} = R_2.A_2 j,$$

where R_1 and R_2 are two database tables and A_{1i} and A_{2j} are attributes of R_1 and R_2, respectively, that have the same domain. Therefore, an INNER JOIN creates a new table combining all the attributes from the two tables and only the tuples having the same value for A_{1i} and A_{2j} in the original tables qualify to be in the new table.

One could easily verify that an INNER JOIN is essentially the same as a SELECT statement. For example, the above INNER JOIN can be expressed by the following $\mathcal{L}$FSQL SELECT

statement.

$$\begin{aligned}&\text{SELECT } R_1.A_{11}, \ldots, R_1.A_{1m}, R_2.A_{21}, \ldots, R_2.A_{1n}\\ &\quad \text{FROM } R_1, R_2\\ &\text{WHERE } R_1.A_{1i}\ F =\ R_2.A_{2j};,\end{aligned}$$

3.3 Semantics of $\mathcal{L}$FSQL

At this point we are ready to define the semantics of $\mathcal{L}$-fuzzy query language $\mathcal{L}$FSQL in the abstract theory of arrow categories. In doing so we will require that all injections, projections, and splittings used are crisp relations. Note that crisp versions of these relational constructions do exist in most cases [31, 33], and so this type of requirements do not cause any major restriction in our study. As an example let us assume that the projections are not crisp and we have a database where some attribute A has a single crisp value in it. Now, as the projections are not crisp when we project on A we would get a non-crisp value although the actual value it has is crisp. This means that the whole process introduces some sort of fuzziness to a non-fuzzy context which is not acceptable. The same concept applies to injections as they are simply the converse of projections in our study.

Now, while defining the semantics for $\mathcal{L}$FSQL, we would need some sort of interpretations for the lattice $\mathcal{L}$, domains of the attributes, the metadatabase, and so on. Therefore, we require the followings. For the illustrations we use the arrow category of concrete D_6-fuzzy relations as an example.

1. The algebraic theory that is going to be used is an arrow category, written as $\mathcal{A}$. We require that $\mathcal{A}$ have relational products, relational sums, splittings, a zero object and a unit, all of which are crisp.

2. From Chapter 2 we know that scalar relations help identify the underlying lattice. In $\mathcal{A}$ the complete Heyting algebra of scalar elements is isomorphic to $\mathcal{L}$. This is because the corresponding Dedekind category is uniform meaning that $\pi;\pi = \pi$ irrespective of the source and the destination. As a result, there is a scalar $I(l)$ in $\mathcal{A}$ for every $l \in \mathcal{L}$. We say that $I(l)$ is the interpretation of l. For example, if we consider the lattice element $c \in D_6$, then the corresponding scalar $I(c) : A \to A$ on some object $A = \{a_1, a_2, \ldots, a_n\}$ has the

following form.

$$\begin{array}{c} \\ a_1 \\ a_2 \\ \dots \\ a_n \end{array} \begin{array}{c} \begin{array}{cccc} a_1 & a_2 & \dots & a_n \end{array} \\ \begin{pmatrix} c & 0 & 0 & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & c \end{pmatrix} \end{array}$$

3. A domain D is interpreted by an object $I(D)$ in $\mathcal{A}$ and an element $d \in D$, by a crisp point $I(d) : 1 \to I(D)$. Note that in our study we denote the unit object 1 by $\{*\}$. For instance the domain of the attribute *Name* and a value *Richy* $\in$ *Name* would be interpreted as follows.

   $I(Name) = \{John, Kevin, Linda, Richy, Tijo\}$

   $$I(Richy) = {}_* \begin{array}{c} \begin{array}{ccccc} John & Kevin & Linda & Richy & Tijo \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

   In addition, we have the followings for domains.

   - For an ordered domain D the associated order ($\leqslant$) is interpreted by $I(\leqslant) : I(D) \to I(D)$ so that $d_x \leqslant d_y$ iff $I(d_x); I(\leqslant); I(d_y)^\smile = \top_{11}$. As an example let us consider $\{1, 2, 3, 4, 5\}$ as the domain for *Distance* and we want to check if $3 \leqslant 5$.

   $$I(3); I(\leqslant); I(5)^\smile = {}_* \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}; \begin{array}{c} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}; \begin{array}{c} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \\ \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \end{array}$$

   $$= {}_* \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \end{pmatrix} \end{array}; \begin{array}{c} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \\ \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \end{array} = {}_* \begin{array}{c} \\ \begin{pmatrix} 1 \end{pmatrix} \end{array} = \top_{11}$$

   - If the domain D comes with an approximate equality $\equiv$, then we have in $\mathcal{A}$ a relation $I(\equiv) : I(D) \to I(D)$ and we say that an element d_x is approximately equal to another element d_y with degree $l \in \mathcal{L}$, i.e., $d_x \equiv d_y = l$ iff $I(d_x); I(\equiv); I(d_y)^\smile = I(l)$. For the illustration we continue using the same *Distance* example with the approximate equality begin defined to be distances that very by less than 1km. If we are interested

in the approximation $3 \equiv 4$, then we have the followings.

$$I(3); I(\equiv); I(4)^{\smile} = {}_{*}\begin{pmatrix} {\scriptstyle 1} & {\scriptstyle 2} & {\scriptstyle 3} & {\scriptstyle 4} & {\scriptstyle 5} \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}; \begin{matrix} {\scriptstyle 1} \\ {\scriptstyle 2} \\ {\scriptstyle 3} \\ {\scriptstyle 4} \\ {\scriptstyle 5} \end{matrix}\begin{pmatrix} {\scriptstyle 1} & {\scriptstyle 2} & {\scriptstyle 3} & {\scriptstyle 4} & {\scriptstyle 5} \\ 1 & c & 0 & 0 & 0 \\ c & 1 & d & 0 & 0 \\ 0 & d & 1 & c & 0 \\ 0 & 0 & c & 1 & d \\ 0 & 0 & 0 & d & 1 \end{pmatrix}; \begin{matrix} {\scriptstyle 1} \\ {\scriptstyle 2} \\ {\scriptstyle 3} \\ {\scriptstyle 4} \\ {\scriptstyle 5} \end{matrix}\begin{pmatrix} {\scriptstyle *} \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$$= {}_{*}\begin{pmatrix} {\scriptstyle 1} & {\scriptstyle 2} & {\scriptstyle 3} & {\scriptstyle 4} & {\scriptstyle 5} \\ 0 & d & 1 & c & 0 \end{pmatrix}; \begin{matrix} {\scriptstyle 1} \\ {\scriptstyle 2} \\ {\scriptstyle 3} \\ {\scriptstyle 4} \\ {\scriptstyle 5} \end{matrix}\begin{pmatrix} {\scriptstyle *} \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = {}_{*}\begin{pmatrix} {\scriptstyle *} \\ c \end{pmatrix} = I(c)$$

Note that, as the Heyting algebra of scalar elements is isomorphic to the lattice $\mathcal{L}$, it is required that for every t-norm and t-conorm like operation included in the metadatabase, there is a corresponding operation defined on the scalars. We will see these operations later on.

Now if a database table has n rows or tuples, the corresponding relation in $\mathcal{A}$ should have a source which has n elements. We write $I(n)$ for the object with n elements and we get this by taking the relational sum of the unit object n times, i.e.,

$$I(n) = \underbrace{1 + \cdots + 1}_{n-\text{times}}.$$

Recall that a unit 1 is an object in $\mathcal{A}$ for which $\mathbb{I}_1 = \pi_{11}$ and π_{A1} is total for any other object A. Note that $I(0) = 0$ which is the zero object (i.e., $\perp\!\!\!\perp_{00} = \pi_{00}$). Also note that the object $I(m \cdot n)$ is isomorphic to $I(m) \times I(n)$ as relational products distribute over relational sums, i.e., $A \times (B + C) \cong A \times B + A \times C$.

3.3.1 Semantics of $\mathcal{L}$-Fuzzy Sets

As we know $\mathcal{L}$-fuzzy sets are used to represent the entries of a table and also in the WHERE clause to form conditions. If m is an $\mathcal{L}$-fuzzy set defined on the domain D, then the semantics of m is given by the vector $[\![m]\!] : 1 \to I(D)$. Also, the part of the metadatabase that stores $\mathcal{L}$-fuzzy sets is modelled by a function σ_s which takes the name of an $\mathcal{L}$-fuzzy set, let us say $\$m$, and returns the corresponding vector relation, i.e., $\sigma_s(\$m) : 1 \to I(D)$. If such a function is given, the semantics of basic $\mathcal{L}$-fuzzy sets are given as follows.

- For explicitly given sets:

$$[\![\{l_1/d_1, \ldots, l_n/d_n\}]\!](\sigma_s) = \bigsqcup_{i=1}^{n} I(l_i); I(d_i)$$

Example:

$$\begin{aligned}
[\![\{c/4, 1/5, d/6\}]\!](\sigma_s) &= {}_{*}\begin{pmatrix} * \\ c \end{pmatrix}; {}_{*}\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & \ldots \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \sqcup \\
&\quad {}_{*}\begin{pmatrix} * \\ 1 \end{pmatrix}; {}_{*}\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & \ldots \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \sqcup \\
&\quad {}_{*}\begin{pmatrix} * \\ d \end{pmatrix}; {}_{*}\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & \ldots \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \\
&= {}_{*}\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & \ldots \\ 0 & 0 & 0 & c & 1 & d & 0 & 0 \end{pmatrix}
\end{aligned}$$

- For linguistic labels: $[\![\$m]\!](\sigma_s) = \sigma_s(\$m)$
  Example:

$$[\![\$Close]\!](\sigma_s) = \sigma_s(\$Close) = {}_{*}\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & \ldots \\ 1 & 1 & 1 & 1 & 1 & c & a & 0 \end{pmatrix}$$

- For predefined functions:

$$[\![\#f]\!](\sigma_s) = \bigsqcup_{d \in D} I(f(d)); I(d)$$

  Here, we first use the function f to get the degree of membership l for some $d \in D$. It then follows the same procedure stated above.

As we know, intensifying and weakening modifiers can be used to define new $\mathcal{L}$-fuzzy sets. In relation algebra, we use residuals and composition respectively, to compute these modifies [31].

$$\begin{aligned}
[\![\text{extremely}(\equiv, m)]\!](\sigma_s) &= ([\![m]\!](\sigma_s) \backslash I(\equiv)) \backslash I(\equiv), \\
[\![\text{very}(\equiv, m)]\!](\sigma_s) &= [\![m]\!](\sigma_s) \backslash I(\equiv), \\
[\![\text{more_or_less}(\equiv, m)]\!](\sigma_s) &= [\![m]\!](\sigma_s); I(\equiv), \\
[\![\text{roughly}(\equiv, m)]\!](\sigma_s) &= [\![m]\!](\sigma_s); I(\equiv); I(\equiv),
\end{aligned}$$

As an example we consider the $\mathcal{L}$-fuzzy set $m = \{d/4, 1/5, c/6\}$ on the domain of *Distance* with the approximate equality being defined as:

$$I(\equiv) = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ \dots \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & \dots \end{array} \\ \begin{pmatrix} 1 & d & 0 & 0 & 0 & 0 & 0 & \dots \\ d & 1 & c & 0 & 0 & 0 & 0 & \dots \\ 0 & c & 1 & c & 0 & 0 & 0 & \dots \\ 0 & 0 & c & 1 & d & 0 & 0 & \dots \\ 0 & 0 & 0 & d & 1 & c & 0 & \dots \\ 0 & 0 & 0 & 0 & c & 1 & d & \dots \\ 0 & 0 & 0 & 0 & 0 & d & 1 & \dots \\ 0 & 0 & 0 & 0 & 0 & \dots & \dots & \dots \end{pmatrix} \end{array}$$

Now, the "more or less" modifier for m can be computed as follows.

$$[\![m]\!](\sigma_s); I(\equiv) = {}_* \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & \dots \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & c & 1 & d & 0 & 0 \end{pmatrix} \end{array}; \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ \dots \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & \dots \end{array} \\ \begin{pmatrix} 1 & d & 0 & 0 & 0 & 0 & 0 & \dots \\ d & 1 & c & 0 & 0 & 0 & 0 & \dots \\ 0 & c & 1 & c & 0 & 0 & 0 & \dots \\ 0 & 0 & c & 1 & d & 0 & 0 & \dots \\ 0 & 0 & 0 & d & 1 & c & 0 & \dots \\ 0 & 0 & 0 & 0 & c & 1 & d & \dots \\ 0 & 0 & 0 & 0 & 0 & d & 1 & \dots \\ 0 & 0 & 0 & 0 & 0 & \dots & \dots & \dots \end{pmatrix} \end{array}$$

$$= {}_* \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & \dots \end{array} \\ \begin{pmatrix} 0 & 0 & c & 1 & 1 & 1 & d & 0 \end{pmatrix} \end{array}$$

However, as mentioned before, if a t-norm like operation is used, then the corresponding residual and the composition based on that operation is used instead.

3.3.2 Semantics of Tables

Before you dive into the semantic details for an $\mathcal{L}$-fuzzy table we first demonstrate the usual way we think of a database table using a simple example. Let us say our table has two attributes A and B with the corresponding domains being $\{a, b\}$ and $\{c, d\}$, respectively. We take $\mathcal{L}$ to be $\{0, m, 1\}$ for membership values. As each entry is actually an $\mathcal{L}$-fuzzy subset of the corresponding domain, for attribute A a tuple can have one of the $|\mathcal{L}|^{|A|} = 3^2 = 9$ different combinations (subsets). The set of all the subsets on A is called its $\mathcal{L}$-fuzzy powerset and is written as $\mathcal{P}(A)$ or $\mathcal{L}^A$. Here,

$$\mathcal{L}^A = \{\{0/a, 0/b\}, \{0/a, m/b\}, \{0/a, 1/b\}, \{m/a, 0/b\}, \{m/a, m/b\}, \{m/a, 1/b\}, \{1/a, 0/b\}, \{1/a, m/b\}, \{1/a, 1/b\}\}$$

Similarly $\mathcal{L}^B$ has $|\mathcal{L}|^{|B|} = 3^2 = 9$ elements. Note that in case of classical sets, i.e., when $\mathcal{L}$ is $\{0, 1\}$ or $\{true, false\}$, $|2^A| = 2^2 = 4$ and the same for 2^B.

Now, the usual way of modelling a database table is to use a subset of the Cartesian product of its attributes. For example, the classical database in Figure 3.5a could be represented by the subset $\{(a, d), (b, c)\} \subseteq 2^A \times 2^B$. Similarly, for an $\mathcal{L}$-fuzzy table it would be a subset of $\mathcal{L}^A \times \mathcal{L}^B$ which is essentially the relational product of the two objects $\mathcal{L}^A$ and $\mathcal{L}^B$ in our arrow category $\mathcal{A}$. Figure 3.5c shows such a representation for the first tuple of Figure 3.5b. Therefore, in our example together for the two attributes an arbitrary tuple might be one of $|\mathcal{L}^A| \times |\mathcal{L}^B| = 9 * 9 = 81$ possible $\mathcal{L}$-fuzzy subsets. However, in the algebra of relations, such a table can be modelled as a crisp function that maps an object or tuple to one of these 81 possibilities. Figure 3.5d provides a conceptual view for such a function where the source indicates the tuple numbers and the target contains all the subsets, to be precise the pairs of $\mathcal{L}^A \times \mathcal{L}^B$, each of which has the form of Figure 3.5c. Note that in every row of the relation in Figure 3.5d there is exactly one 1 indicating the $\mathcal{L}$-fuzzy subset for that tuple. We assume that $S_3 = \{m/a, 1/b, 0/c, m/d\}$ be the subset for the first tuple.

A	**B**
a	d
b	c

(A) A classical database

A	**B**
{m/a,1/b}	{0/c,m/d}
{0/a,m/b}	{1/c,0/d}

(B) An $\mathcal{L}$-fuzzy database

$$\begin{array}{c} \mathcal{L}^A \quad \times \quad \mathcal{L}^B \\ \left(\begin{array}{ccc} \{m/a, & & \{0/c, \\ 1/b\} & , & m/d\} \end{array} \right) \end{array}$$

(C) $A \times B$ $\mathcal{L}$-subset

$$\begin{array}{c} \quad S_1 \; S_2 \; S_3 \; S_4 \; S_5 \; S_6 \; \ldots \; S_{80} \; S_{81} \\ \begin{array}{c} 1 \\ 2 \end{array} \left(\begin{array}{ccccccccc} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{array} \right) \end{array}$$

(D) The table as a crisp relation (to $\mathcal{L}^A \times \mathcal{L}^B$)

$$\begin{array}{c} \mathcal{L}^{A+B} \\ \left\{ \begin{array}{c} m/a \\ 0/c \\ 1/b \\ m/d \end{array} \right\} \end{array}$$

(E) $A + B$ $\mathcal{L}$-subset

$$\begin{array}{c} \quad T_1 \; T_2 \; T_3 \; T_4 \; T_5 \; T_6 \; \ldots \; T_{80} \; T_{81} \\ \begin{array}{c} 1 \\ 2 \end{array} \left(\begin{array}{ccccccccc} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right) \end{array}$$

(F) The table as a crisp relation (to $\mathcal{L}^{A+B}$)

$$\begin{array}{c} \quad a \; b \; c \; d \\ \begin{array}{c} 1 \\ 2 \end{array} \left(\begin{array}{cccc} m & 1 & 0 & m \\ 0 & m & 1 & 0 \end{array} \right) \end{array}$$

(G) The table as an $\mathcal{L}$-relation

FIGURE 3.5: Modelling a database table

At this point, we become interested in expressing the same $\mathcal{L}$-fuzzy table using relational sum. As each attribute has two values in our database, the construction $A + B$ has four elements as it computes the disjoint union. For example, Figure 3.5e shows the first tuple of Figure 3.5b in the new notation. Now, from [27] we know that $\mathcal{L}^A \times \mathcal{L}^B \cong \mathcal{L}^{A+B}$ meaning that the pair of subsets of Figure 3.5c is isomorphic to one subset of the new type shown in Figure 3.5e. However, the set of all such $\mathcal{L}$-fuzzy subsets will be $\mathcal{L}^{A+B}$ which has a total of $|\mathcal{L}|^{|A|+|B|} = 3^4 = 81$ elements in it. Eventually the final relation for the whole table is still a crisp function as shown in Figure

3.5f. Notice that the $A \times B$ $\mathcal{L}$-subset S_3 in Figure 3.5d for example, is isomorphic to the $A + B$ $\mathcal{L}$-subset T_6 in Figure 3.5f.

In the final step, we would like to get rid of the $\mathcal{L}$ from the construction $\mathcal{L}^{A+B}$ which also removes the property that the relation in Figure 3.5f is a crisp function. The result is an arbitrary $\mathcal{L}$-relation from the same source to $A + B$ as shown in 3.5g. Notice that the new relation is equivalent to the previous two but comparatively simpler and serves our purpose much better.

Now we summarize the whole idea. Let us say R is a table in our database which has r rows and n-attributes $A_1, A_2, \ldots, A_n$ with domains $D_1, D_2, \ldots, D_n$. If R is non fuzzy, we can view R as a finite subset of the product of the corresponding domains which is $D_1 \times D_2 \times, \ldots, \times D_n$. Relation algebraically this can be modelled in one of the following three ways.

1. *As a point relation*: $[\![R]\!] : 1 \to \mathcal{P}(I(D_1) \times \cdots \times I(D_n))$, where $\mathcal{P}(X)$ is an abstract version of a power set construction and therefore the whole table is basically a single element of $\mathcal{P}(X)$.
2. *As a vector*: $[\![R]\!] : 1 \to I(D_1) \times \cdots \times I(D_n)$.
3. *As a function*: $[\![R]\!] : I(r) \to I(D_1) \times \cdots \times I(D_n)$ as we deal with finite database.

However, if R is an $\mathcal{L}$-fuzzy table, then each attribute stores sets for tuples. As a result, the target object with the last option changes to $\mathcal{P}(I(D_1)) \times \cdots \times \mathcal{P}(I(D_n))$. We already know that this object is isomorphic to $\mathcal{P}(I(D_1) + \cdots + I(D_n))$ [27]. This eventually leads to the fact that having a function of the form $[\![R]\!] : I(r) \to \mathcal{P}(I(D_1) + \cdots + I(D_n))$ is equivalent to having a relation of the form $[\![R]\!] : I(r) \to I(D_1) + \cdots + I(D_n)$. This final relation thus constitutes the semantics for the table R. Notice that the n-ary sum can be obtained by iterating binary sums. However, we denote the injection from $I(D_1)$ into $I(D_1) + \cdots + I(D_n)$ by ι_i. As projections are converse of injections in our study, we have $[\![R.A_i]\!] = [\![R]\!]; \iota_i^\smile$.

3.3.3 Semantics of $\mathcal{L}$-Fuzzy Comparators

In Section 3.2.1 we have defined the two types of fuzzy comparator: possibility and necessity. Relation algebraically they can be computed using composition and residual operations, respectively. In order to demonstrate this, we continue using the same "Distance" example. Recall that our interest was to check if Tijo's home is fuzzy (possibly and necessarily) *Close* to Peter's place. We model the entry for "Distance" in Tijo's record by the $\mathcal{L}$-fuzzy set $\{a/4, d/5, b/6\}$. It is well known that this $\mathcal{L}$-fuzzy as well as the one for the label $\$Close$ will be represented by some vectors in the semantics, let us denote them by T and C, respectively. At first we would

like to consider the non-fuzzy case only. Therefore, for the possibility comparator we have:

$$
\begin{aligned}
&Tijo.Distance\ F = \$Close \\
&= T; C^{\smile} \\
&= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & \cdots \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & \cdots \end{pmatrix}; \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & \cdots \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & \cdots \end{pmatrix}^{\smile} \\
&= (1)
\end{aligned}
$$

As the result is a 1, we can say that Tijo's place is possibly *Close* to Peter's place. Now, for the necessity comparator we have the followings.

$$
\begin{aligned}
&Tijo.Distance\ NF = Close \\
&= (C/T)^{\smile} \\
&= \left(\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & \cdots \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & \cdots \end{pmatrix} / \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & \cdots \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & \cdots \end{pmatrix} \right)^{\smile} \\
&= (0)
\end{aligned}
$$

The generalization of these two operations for $\mathcal{L}$-fuzzy case can be found in [31]. For example, the composition operation will compute the least upper bound of all $\mathcal{L}$ values obtained as the membership degrees of those elements belonging to both sets. This means that, if v_1 and v_2 are two vectors on some domain D, then we have

$$v_1; v_2^{\smile} = \bigsqcup_{x \in D} v_1(x) \sqcap v_2(x).$$

In our example,

$$
\begin{aligned}
T; C^{\smile} &= \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & \cdots \\ 0 & 0 & 0 & a & d & b & 0 & 0 & \cdots \end{pmatrix}; \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & \cdots \\ 1 & 1 & 1 & 1 & 1 & c & a & 0 & \cdots \end{pmatrix} \\
&= \bigsqcup\{a \sqcap 1, d \sqcap 1, b \sqcap c\} = \bigsqcup\{a, d, b\} = 1
\end{aligned}
$$

On the other hand, the residual operation which represents the necessity equality, computes the greatest lower bound of all $\mathcal{L}$-values obtained as the maximal degree of which an element belongs to the first set implies that it also belongs to the second set. Therefore, we have

$$v_1; v_2^{\smile} = \bigsqcap_{x \in D} v_1(x) \rightarrow v_2(x).$$

If, however, the underlying binary comparison is not $=$, then the corresponding relation has to be added in the composition as well as in the residual operation.

At this point, we define the semantics of a comparison. As discussed in Section 3.2.4, we write $S\ LFC\ S'$ for a general comparison where S, S' are either some attributes of the form $R.A$ or some $\mathcal{L}$-fuzzy sets and LFC is either a possibility or a necessity comparator based on the binary comparator C. Recall that in Section 3.3.1 we defined σ_s to be the function that takes an $\mathcal{L}$-fuzzy set $\$m$ as input and returns the corresponding vector, i.e., $\sigma_s(\$m) : 1 \to I(D)$. But if the selection is some attribute $R.A$, then we need another construction that maps a table name to its semantics, i.e., a relation of the form $[\![R]\!] : I(r) \to I(D_1) + \cdots + I(D_n)$. Let us name it σ_t. Note that σ_t represents the whole $\mathcal{L}$-fuzzy database in the semantics. We write $\sigma_t[Q/R]$ to denote the update of σ_t at table R by the relation Q. We will see an example in Section 3.3.5. Finally, the semantics of a selection S written as $[\![S]\!](\sigma_s, \sigma_t) : I(r) \to I(D_i)$, is defined by the following two cases.

- *S is an attribute ($R.A$)*: $[\![R.A_i]\!](\sigma_s, \sigma_t) = [\![R.A_i]\!](\sigma_t) = \sigma_t(R); \iota_i^\smile$
  From the example in Figure 3.5

$$
\begin{aligned}
[\![R.A_i]\!](\sigma_t) &= \sigma_t(R); \iota_i^\smile \\
&= \begin{array}{c} \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} m & 1 & 0 & m \\ 0 & m & 1 & 0 \end{pmatrix} \end{array}; \begin{array}{c} \\ a \\ b \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \end{array}^\smile \\
&= \begin{array}{c} \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} m & 1 & 0 & m \\ 0 & m & 1 & 0 \end{pmatrix} \end{array}; \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \end{array} \\
&= \begin{array}{c} \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{pmatrix} m & 1 \\ 0 & m \end{pmatrix} \end{array}.
\end{aligned}
$$

- *S is an $\mathcal{L}$-fuzzy set*: $[\![m]\!](\sigma_s, \sigma_t) = \pi_{I(n)1}; [\![m]\!](\sigma_s)$
  As an example, let us deduce semantics for the $\mathcal{L}$-fuzzy set $m = \{m/a, m/b\}$ on attribute

A. Therefore,

$$\begin{aligned}
[\![m]\!](\sigma_s,\sigma_t) &= \pi_{I(n)1};[\![m]\!](\sigma_s)\\
&= {}^{1}_{2}\begin{pmatrix} {\scriptstyle *}\\ 1\\ 1 \end{pmatrix}; {}_{*}\begin{pmatrix} {\scriptstyle a} & {\scriptstyle b}\\ m & m \end{pmatrix}\\
&= {}^{1}_{2}\begin{pmatrix} {\scriptstyle a} & {\scriptstyle b}\\ m & m\\ m & m \end{pmatrix}.
\end{aligned}$$

Having the interpretations for individual selections, we can now proceed to have a semantics for a complete $\mathcal{L}$-fuzzy comparison. If $S\ LFC\ S'$ is a general comparison, then we define its semantics $[\![S\ LFC\ S']\!](\sigma_s,\sigma_t)$ to be a relation of the form $I(r) \to I(r)$ which has either of the following two forms:

$$\begin{aligned}
[\![S\ FC\ S']\!](\sigma_s,\sigma_t) &= [\![S]\!](\sigma_s,\sigma_t);I(C);[\![S']\!](\sigma_s,\sigma_t)^{\smile} \sqcap \mathbb{I}_{I(r)},\\
[\![S\ NFC\ S']\!](\sigma_s,\sigma_t) &= (([\![S']\!](\sigma_s,\sigma_t);I(C)^{\smile})/[\![S]\!](\sigma_s,\sigma_t))^{\smile} \sqcap \mathbb{I}_{I(r)}.
\end{aligned}$$

Note that the constructions on the left of $\sqcap$ above compare each row of $[\![S]\!](\sigma_s,\sigma_t)$ with every row of $[\![S']\!](\sigma_s,\sigma_t)$. Therefore, in order to make sure that only the corresponding rows are matched we intersect the result with the identity $\mathbb{I}_{I(r)}$.
Now, let us see how it works with our example for the comparison $R.A\ F = \{m/a, m/b\}$.

$$\begin{aligned}
[\![R.A\ F=\{m/a,m/b\}]\!](\sigma_s,\sigma_t) &= {}^{1}_{2}\begin{pmatrix} {\scriptstyle a} & {\scriptstyle b}\\ m & 1\\ 0 & m \end{pmatrix}; {}^{a}_{b}\begin{pmatrix} {\scriptstyle a} & {\scriptstyle b}\\ 1 & 0\\ 0 & 1 \end{pmatrix}; {}^{1}_{2}\begin{pmatrix} {\scriptstyle a} & {\scriptstyle b}\\ m & m\\ m & m \end{pmatrix}^{\smile} \sqcap {}^{1}_{2}\begin{pmatrix} {\scriptstyle 1} & {\scriptstyle 2}\\ 1 & 0\\ 0 & 1 \end{pmatrix}\\
&= {}^{1}_{2}\begin{pmatrix} {\scriptstyle a} & {\scriptstyle b}\\ m & 1\\ 0 & m \end{pmatrix}; {}^{a}_{b}\begin{pmatrix} {\scriptstyle 1} & {\scriptstyle 2}\\ m & m\\ m & m \end{pmatrix} \sqcap {}^{1}_{2}\begin{pmatrix} {\scriptstyle 1} & {\scriptstyle 2}\\ 1 & 0\\ 0 & 1 \end{pmatrix}\\
&= {}^{1}_{2}\begin{pmatrix} {\scriptstyle 1} & {\scriptstyle 2}\\ m & m\\ m & m \end{pmatrix} \sqcap {}^{1}_{2}\begin{pmatrix} {\scriptstyle 1} & {\scriptstyle 2}\\ 1 & 0\\ 0 & 1 \end{pmatrix}\\
&= {}^{1}_{2}\begin{pmatrix} {\scriptstyle 1} & {\scriptstyle 2}\\ m & 0\\ 0 & m \end{pmatrix}
\end{aligned}$$

The resultant matrix indicates that both the tuples satisfy the condition with degree m. Note that the semantics of an $\mathcal{L}$-fuzzy comparison is a partial identity, i.e., a relation which is smaller or

equal to $\mathbb{I}_{I(r)}$.

We want to conclude this section by showing relation algebraically how possibility comparators generalize necessity comparators. For Q *LFC* R, we shorthand the semantic expressions for the two comparators to $Q;C;R^{\smile}$ and $((R;C)^{\smile}/Q)^{\smile}$, respectively. Now, we have

$$
\begin{aligned}
X \sqsubseteq ((R;C)^{\smile}/Q)^{\smile} &\Leftrightarrow X^{\smile} \sqsubseteq R;C^{\smile}/Q && \text{Taking converse}\\
&\Leftrightarrow X^{\smile};Q \sqsubseteq R;C^{\smile}\\
&\Leftrightarrow (X^{\smile};Q)^{\smile} \sqsubseteq (R;C^{\smile})^{\smile} && \text{Taking converse}\\
&\Leftrightarrow Q^{\smile};X \sqsubseteq C;R^{\smile}\\
&\Leftrightarrow Q;Q^{\smile};X \sqsubseteq Q;C;R^{\smile} && \text{Multiplying by } Q\\
\text{Therefore, } X \sqsubseteq Q;Q^{\smile};X &\sqsubseteq Q;C;R^{\smile} && \text{If } Q \text{ is total}\\
&\Leftrightarrow X \sqsubseteq Q;C;R^{\smile}\\
\text{Also, } ((R;C)^{\smile}/Q)^{\smile} &\sqsubseteq Q;C;R^{\smile}
\end{aligned}
$$

From the proof above it is evident that possibility comparisons are more general than necessity comparisons.

3.3.4 Semantics of WHERE Clause

As mentioned before in $\mathcal{L}$FSQL one or more primitive comparisons can be combined by using logical AND or OR which are based on $\sqcap$ and $\sqcup$, respectively. However, if a t-norm like or t-conorm like operation $*$ is used instead of the two logical connectives, then we use $\sqcap_*$ that covers both the cases. At the end of Chapter 2 we saw how to compute $\sqcap$ for a complete lattice-ordered semigroup operation $*$ componentwisely. In fact, restricting $\sqcup$ to scalar relations and using it as a t-conorm like operation yields $\sqcap_{\sqcup} = \sqcup$ [31].

Also, we know that primitive comparisons in $\mathcal{L}$FSQL may have threshold values specified with them. In Chapter 2 we have seen that α-cut produces relations that associate elements with membership degrees of at least α. As this is what a threshold in a comparison operation requires, we therefore, model such a comparison using α-cut as follows.

$$[\![Com \text{ THOLD } l]\!](\sigma_s, \sigma_t) = (I(l) \backslash [\![Com]\!](\sigma_s, \sigma_t))^{\downarrow}$$

Using the running example let us evaluate the comparison $R.A\ F= \{m/a, m/b\}$ THOLD m.

$$
\begin{aligned}
[\![R.A\ F= \{m/a, m/b\} \text{ THOLD } m]\!](\sigma_s, \sigma_t) &= (I(m) \backslash [\![R.A\ F= \{m/a, m/b\}]\!](\sigma_s, \sigma_t))^{\downarrow} \\
&= \left(\begin{array}{cc} & \begin{array}{cc} 1 & 2 \end{array} \\ \begin{array}{c} 1 \\ 2 \end{array} & \begin{pmatrix} m & 0 \\ 0 & m \end{pmatrix} \end{array} \backslash \begin{array}{c} \\ \begin{array}{c} 1 \\ 2 \end{array} \end{array} \begin{array}{c} \begin{array}{cc} 1 & 2 \end{array} \\ \begin{pmatrix} m & 0 \\ 0 & m \end{pmatrix} \end{array} \right)^{\downarrow} \\
&= \begin{array}{c} \\ \begin{array}{c} 1 \\ 2 \end{array} \end{array} \begin{array}{c} \begin{array}{cc} 1 & 2 \end{array} \\ \begin{pmatrix} 1 & m \\ m & 1 \end{pmatrix} \end{array}^{\downarrow} \\
&= \begin{array}{c} \\ \begin{array}{c} 1 \\ 2 \end{array} \end{array} \begin{array}{c} \begin{array}{cc} 1 & 2 \end{array} \\ \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{array}
\end{aligned}
$$

From the final matrix it is evident that both tuple 1 and 2 satisfy the condition with the threshold m.

As because partial identities are closed under meets, joins (including the t-norm and t-conorm based versions) and α-cuts, it is evident that the semantics of a WHERE clause is also a partial identity.

3.3.5 Semantics of Statements

As mentioned before $\sigma_t[Q/R]$ represents the update of σ_t at a table R by the relation Q. Mathematically,

$$\sigma_t[Q/R](X) = \begin{cases} Q, \text{ if } X = R, \\ \sigma_t(X), \text{ otherwise.} \end{cases}$$

This is necessary particularly for the CREATE and INSERT statements. Such a statement when executed produces a new relation which is then used to update the database.

3.3.5.1 Semantics of CREATE Statement

The CREATE statements adds a new table to the database. Therefore, the semantics of a CREATE statement produces a relation which is then added to the database, i.e., modifies the existing database. We define the semantics of the CREATE statement as follows.

$$[\![\text{CREATE TABLE } R(A_1 : D_1, \ldots, A_n : D_n);]\!](\sigma_t) = \sigma_t[\perp\!\!\!\perp_{0(I(D_1)+\ldots+I(D_n))}/R]$$

It says that we update σ_t by adding the empty relation $\perp\!\!\!\perp_{0(I(D_1)+\ldots+I(D_n))}$ as R does not already exist. Graphically this relation $[\![R]\!] : I(r) \to I(D_1) + \cdots + I(D_n)$ would be an empty matrix with no rows in it.

3.3.5.2 Semantics of INSERT Statement

As we know every tuple of a table in an $\mathcal{L}$-fuzzy database has $\mathcal{L}$-fuzzy sets for each attribute. Let us say $\{m_1, \ldots, m_n\}$ are $\mathcal{L}$-fuzzy subsets of the domains $(D_1, \ldots, D_n)$. We define the semantics for a tuple of that table by

$$[\![(m_1, \ldots, m_n)]\!](\sigma_s) = \bigsqcup_{i=1}^{n} [\![m_i]\!](\sigma_s); \iota_i.$$

Suppose we want to insert a tuple into our example database that has $\{1/a, m/b\}$ for attribute A and $\{0/c, 1/d\}$ for attribute B. Now,

$$[\![(\{1/a, m/b\}, \{0/c, 1/d\})]\!](\sigma_s) = [\![\{1/a, m/b\}]\!](\sigma_s); \iota_1 \sqcup [\![\{0/c, 1/d\}]\!](\sigma_s); \iota_2$$

$$= * \begin{pmatrix} a & b \\ 1 & m \end{pmatrix}; \begin{matrix} a \\ b \end{matrix} \begin{pmatrix} a & b & c & d \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \sqcup * \begin{pmatrix} c & d \\ 0 & 1 \end{pmatrix}; \begin{matrix} c \\ d \end{matrix} \begin{pmatrix} a & b & c & d \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= * \begin{pmatrix} a & b & c & d \\ 1 & m & 0 & 0 \end{pmatrix} \sqcup * \begin{pmatrix} a & b & c & d \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

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

In order to deduce the semantics for the final table where this tuple has been added, we refer to the following figure.

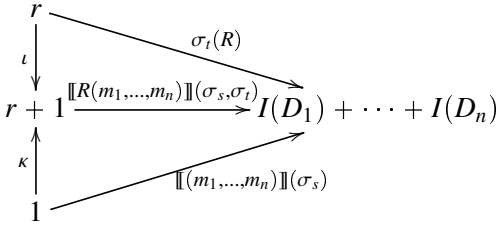


FIGURE 3.6: Modelling the INSERT statement

In the relational diagram above the table has r rows and thus is interpreted by the relation $I(r) \to I(D_1) + \cdots + I(D_n)$. As we have already seen in the example, a vector $1 \to I(D_1) + \cdots + I(D_n)$ represents the tuple to be inserted. The resultant table therefore, has $r + 1$ tuple and maps from

the object $r+1$ to $I(D_1)+\cdots+I(D_n)$ in $\mathcal{A}$. Finally, from Figure 3.6 we get the followings for the semantics of the resultant table.

$$[\![R(m_1,\ldots,m_n)]\!](\sigma_s,\sigma_t) = \iota^{\smile};\sigma_t(R) \sqcup \kappa^{\smile};[\![(m_1,\ldots,m_n)]\!](\sigma_s)$$

As an illustration, let us insert the tuple $(\{1/a, m/b\}, \{0/c, 1/d\})$ into the table of Figure 3.5.

$$[\![R(\{1/a,m/b\},\{0/c,1/d\})]\!](\sigma_s,\sigma_t) = \iota^{\smile};\sigma_t(R) \sqcup \kappa^{\smile};[\![(\{1/a,m/b\},\{0/c,1/d\})]\!](\sigma_s)$$

$$= \begin{array}{c} \\ 1 \\ 2 \end{array}\!\!\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}^{\smile} \end{array}; \begin{array}{c} \\ 1 \\ 2 \end{array}\!\!\begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} m & 1 & 0 & m \\ 0 & m & 1 & 0 \end{pmatrix} \end{array} \sqcup_* \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & \end{array} \\ \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^{\smile} \end{array}; * \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 1 & m & 0 & 1 \end{pmatrix} \end{array}$$

$$= \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\!\!\begin{array}{c} \begin{array}{cc} 1 & 2 \end{array} \\ \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} \end{array}; \begin{array}{c} \\ 1 \\ 2 \end{array}\!\!\begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} m & 1 & 0 & m \\ 0 & m & 1 & 0 \end{pmatrix} \end{array} \sqcup \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\!\!\begin{array}{c} \begin{array}{c} * \end{array} \\ \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \end{array}; * \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 1 & m & 0 & 1 \end{pmatrix} \end{array}$$

$$= \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\!\!\begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} m & 1 & 0 & m \\ 0 & m & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \sqcup \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\!\!\begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & m & 0 & 1 \end{pmatrix} \end{array}$$

$$= \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\!\!\begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} m & 1 & 0 & m \\ 0 & m & 0 & 1 \\ 1 & m & 0 & 1 \end{pmatrix} \end{array}$$

Note that the first injection ι maps from the r-ary relational sum of the unit object to the $(r+1)$-ary relational sum. In our example, as the table already contains 2 tuples and so the final table would have 3 tuples, ι injects to the object $I(3)$.

Based on these definitions we define the semantics of an INSERT statement as follows:

$$\begin{aligned} &[\![\text{INSERT INTO } R \text{ VALUES } (m_1,\ldots,m_n);]\!](\sigma_s,\sigma_t) \\ &= \sigma_t[[\![R(m_1,\ldots,m_n)]\!](\sigma_s,\sigma_t)/R]. \end{aligned}$$

From the definition of update function it is clear that the new relation replaces the existing one for the particular table.

3.3.5.3 Semantics of DELETE Statement

As we know the DELETE operation removes tuples from a database table that satisfy certain condition specified by a WHERE clause. In the semantics of the DELETE statement this means that we have to filter out the rows satisfying the condition. For this we can use the splitting of the semantics of WHERE clause, let us denote it by X for now, i.e., $X = [\![wh]\!](\sigma_s, \sigma_t)^{\uparrow}$. In [2] it was shown that this splitting can be computed as $S = \bigsqcup_{i \in A} \iota_i^{\smile}; \iota_i : I(|A|) \to I(r)$ where $A = \{i \in \{1, \ldots, r\} \mid \iota_i; X; \iota_i^{\smile} = \perp\!\!\!\perp_{11}\}$. Therefore, we consider injections from the object of splitting to only those elements of $I(r)$ (tuples of the table-relation) that do not satisfy the WHERE clause "wh". With these definitions, we model the complete DELETE statement as follows [2]

$$[\![\text{DELETE FROM } R \text{ WHERE wh;}]\!](\sigma_s, \sigma_t) = \sigma_t[S; \sigma_t(R)/R].$$

As an example we would like to delete those tuples from our final table which satisfy the condition $R.A\ F = \{m/a, m/b\}$THOLD m. Therefore, the object A of the splitting only contains 2. So, we get the following.

$$S = \bigsqcup_{i \in A} \iota_i^{\smile}; \iota_i = \iota_1^{\smile}; \iota_1 = \begin{matrix} \\ 2 \end{matrix} \overset{1}{\begin{pmatrix} 1 \end{pmatrix}}; \begin{matrix} 1 \end{matrix} \overset{1}{\underset{}{\begin{pmatrix} 0 \end{pmatrix}}} \overset{2}{\underset{}{\begin{pmatrix} 1 \end{pmatrix}}} \overset{3}{\underset{}{\begin{pmatrix} 0 \end{pmatrix}}}$$

$$= \begin{matrix} \\ 2 \end{matrix} \overset{1}{\underset{}{\begin{pmatrix} 0 \end{pmatrix}}} \overset{2}{\underset{}{\begin{pmatrix} 1 \end{pmatrix}}} \overset{3}{\underset{}{\begin{pmatrix} 0 \end{pmatrix}}}$$

Now, for the semantics of the complete DELETE statement, we get

$$[\![\text{DELETE FROM } R \text{ WHERE } R.A\ F = \{m/a, m/b\};]\!](\sigma_s, \sigma_t) = \sigma_t[S; \sigma_t(R)/R]$$

$$= \sigma_t[\begin{matrix} \\ 2 \end{matrix} \overset{1}{\underset{}{\begin{pmatrix} 0 \end{pmatrix}}} \overset{2}{\underset{}{\begin{pmatrix} 1 \end{pmatrix}}} \overset{3}{\underset{}{\begin{pmatrix} 0 \end{pmatrix}}}; \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \overset{a}{\underset{}{\begin{pmatrix} m \\ 0 \\ 1 \end{pmatrix}}} \overset{b}{\underset{}{\begin{pmatrix} 1 \\ m \\ m \end{pmatrix}}} \overset{c}{\underset{}{\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}}} \overset{d}{\underset{}{\begin{pmatrix} m \\ 1 \\ 1 \end{pmatrix}}} /R]$$

$$= \sigma_t[\begin{matrix} \\ 2 \end{matrix} \overset{a}{\underset{}{\begin{pmatrix} 0 \end{pmatrix}}} \overset{b}{\underset{}{\begin{pmatrix} m \end{pmatrix}}} \overset{c}{\underset{}{\begin{pmatrix} 0 \end{pmatrix}}} \overset{d}{\underset{}{\begin{pmatrix} 1 \end{pmatrix}}} /R].$$

Therefore, we replace the existing relation for R with the new relation.

3.3.5.4 Semantics of SELECT Statement

As we know the SELECT statement is the basic DML operation found in all types of SQL. In $\mathcal{L}$FSQL it has the following form.

$$\text{SELECT } S_1, \ldots, S_m \text{ FROM } R_1, \ldots, R_n \text{ WHERE wh;}$$

In order to define the semantics for this SELECT statement we first generate an intermediate table consisting of all attributes from all the participating tables. If T is the corresponding relation for the intermediate table, then we have

$$T = \prod_{i=1}^{n} \pi_i; [\![R_i]\!](\sigma_t); \iota_i : I(\coprod_{i=1}^{n} r_i) \to \sum_{i=1}^{n} I(D_{i_1}) + \cdots + I(D_{i_{k_i}})$$

where $D_{i_1}, \cdots, D_{i_{k_i}}$ are the domains for the attributes of table R_i and ι_i is the injection from $I(D_{i_1}) + \ldots + I(D_{i_{k_i}})$ into $\sum_{i=1}^{n} I(D_{i_1}) + \cdots + I(D_{i_{k_i}})$.

Whereas the DELETE statement filters out the tuples that satisfy the WHERE condition and replaces the original table with the resultant one, the SELECT statement does not update the table, rather just generate a new table from multiple parent tables consisting of tuples satisfying the WHERE clause. Thus, as with the DELETE statement, we will be using splitting of the semantics of the WHERE clause in computing the semantics of the SELECT statement. Let us denote the splitting of $[\![\text{wh}]\!](\sigma_s, \sigma_t)^{\uparrow}$ by $S : I(r') \to I(\coprod_{i=1}^{n} r_i)$. At this point we apply the splitting to the intermediate table by evaluating the expression $Q = S; T$ which then produces a new relation $Q : I(r') \to \sum_{i=1}^{n} I(D_{i_1}) + \cdots + I(D_{i_{k_i}})$ containing the qualified tuples only. Note that this table contains the selections $S_1, \ldots, S_m$ as well as the other attributes of the participating tables. So, in order to get the final result P we do the following.

$$P = \bigsqcup_{i=1}^{m} Q; \iota_i^{\smile}; \iota_i : I(r') \to I(D_1) + \cdots + I(D_m).$$

Note that in the expression $Q; \iota_i^{\smile}; \iota_i$ the injection in the middle maps from $I(D_i)$ to $\sum_{i=1}^{n} I(D_{i_1}) + \cdots + I(D_{i_{k_i}})$ whereas the one on the right has the form $\iota_i : I(D_i) \to I(D_1) + \cdots + I(D_m)$.

Therefore, we define the semantics of the whole SELECT statement by

$$[\![\text{SELECT } S_1, \ldots, S_m \text{ FROM } R_1, \ldots, R_n \text{ WHERE wh;}]\!](\sigma_s, \sigma_t) = P.$$

For the demonstrations, we use the tables in Figure 3.7a and 3.7b. Table R_1 has two attributes $A = \{a, b\}$ and $B = \{c, d\}$ whereas table R_2 has the attributes $C = \{c, d\}, D = \{e, f, g\}$ and $E = \{h, i\}$. Note that attribute B in table R has the same domain as the attribute C of table T.

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array}\!\!\begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{pmatrix} m & 1 & 0 & m \\ 0 & m & 0 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

(A) Relation for R_1

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\!\!\begin{array}{c} \begin{array}{ccccccc} c & d & e & f & g & h & i \end{array} \\ \begin{pmatrix} m & 0 & m & 1 & m & 0 & m \\ 0 & 1 & 0 & 1 & m & 1 & 0 \\ 1 & 1 & 1 & 1 & m & m & m \\ 0 & m & 1 & m & 0 & 0 & 1 \end{pmatrix} \end{array}$$

(B) Relation for R_2

SELECT $R_1.A, R_1.B, R_2.D$
FROM R_1, R_2
WHERE $R_1.A \;\; F = \{m/a, m/b\}$
AND $(R_1.B \; F = R_2.C$ THOLD 1)

(C) A query

FIGURE 3.7: Modelling SELECT statement

As stated above we first generate the temporary table T of all attributes from all the participating tables as follows. Notice that the attributes inherit their names in the form $R.A$ (for some table R with an attribute A) from their mother table to the new table T. This will be useful while computing semantics of the WHERE clause, specially if two or more tables share attribute names.

$$T = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \end{array}\!\!\begin{array}{c} \begin{array}{cccccccccc} a & b & c & d & c & d & e & f & g & h & i \end{array} \\ \begin{pmatrix} m & 1 & 0 & m & m & 0 & m & 1 & m & 0 & m \\ m & 1 & 0 & m & 0 & 1 & 0 & 1 & m & 1 & 0 \\ m & 1 & 0 & m & 1 & 1 & 1 & 1 & m & m & m \\ m & 1 & 0 & m & 0 & m & 1 & m & 0 & 0 & 1 \\ 0 & m & 0 & 1 & m & 0 & m & 1 & m & 0 & m \\ 0 & m & 0 & 1 & 0 & 1 & 0 & 1 & m & 1 & 0 \\ 0 & m & 0 & 1 & 1 & 1 & 1 & 1 & m & m & m \\ 0 & m & 0 & 1 & 0 & m & 1 & m & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & m & 0 & m & 1 & m & 0 & m \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & m & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & m & m & m \\ 0 & 0 & 0 & 1 & 0 & m & 1 & m & 0 & 0 & 1 \end{pmatrix} \end{array}.$$

Now, for the semantics of the WHERE clause we get the followings.

$$[\![T.(R_1.A)\ F=\{m/a, m/b\}]\!](\sigma_s,\sigma_t) = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \end{array} \\ \begin{pmatrix} m & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & m & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & m & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & m & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & m & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

and $[\![T.(R_1.B)\ F= T.(R_2.C)\ \text{THOLD}\ 1]\!](\sigma_s,\sigma_t) =$

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \end{array} \downarrow \\ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & m & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & m \end{pmatrix} \end{array}$$

$$= \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \end{array} \\ \left(\begin{array}{cccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right) \end{array}$$

Therefore, for the complete WHERE clause we get

$$\begin{aligned} &[\![R_1.A \ F= \{m/a, m/b\} \text{ AND } R_1.B \ F= R_2.C \text{ THOLD } 1]\!](\sigma_s, \sigma_t) \\ &= [\![T.(R_1.A) \ F= \{m/a, m/b\} \text{ AND } T.(R_1.B) \ F= T.(R_2.C) \text{ THOLD } 1]\!](\sigma_s, \sigma_t) \end{aligned}$$

$$= \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \end{array} \\ \left(\begin{array}{cccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & m & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & m & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \right) \end{array}.$$

Now, the splitting would be

$$S = \begin{array}{c} \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{cccccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \end{array} \\ \left(\begin{array}{cccccccccccc} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{array} \right) \end{array}.$$

At this point we apply the splitting S to the temporary table T and get the following relation Q.

$$Q = S;T$$

$$= \begin{array}{c} \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{cccccccccc} a & b & c & d & c & d & e & f & g & h & i \end{array} \\ \begin{pmatrix} 0 & m & 0 & 1 & 0 & 1 & 0 & 1 & m & 1 & 0 \\ 0 & m & 0 & 1 & 1 & 1 & 1 & 1 & m & m & m \end{pmatrix} \end{array}$$

Finally, we select the appropriate selections by

$$P = \bigsqcup_{i=1}^{m} Q;\iota_i{}^{\smile};\iota_i = Q;\iota_1{}^{\smile};\iota_1 \sqcup Q;\iota_2{}^{\smile};\iota_2 \sqcup Q;\iota_4{}^{\smile};\iota_4$$

$$= \begin{array}{c} \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{cccccccccc} a & b & c & d & c & d & e & f & g & h & i \end{array} \\ \begin{pmatrix} 0 & m & 0 & 1 & 0 & 1 & 0 & 1 & m & 1 & 0 \\ 0 & m & 0 & 1 & 1 & 1 & 1 & 1 & m & m & m \end{pmatrix} \end{array}; \begin{array}{c} \\ a \\ b \\ c \\ d \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \end{array}; \begin{array}{c} \\ a \\ b \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

$$\sqcup\, Q;\iota_2{}^{\smile};\iota_2 \sqcup Q;\iota_4{}^{\smile};\iota_4$$

$$= \begin{array}{c} \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{pmatrix} 0 & m \\ 0 & m \end{pmatrix} \end{array}; \begin{array}{c} \\ a \\ b \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \sqcup Q;\iota_2{}^{\smile};\iota_2 \sqcup Q;\iota_4{}^{\smile};\iota_4$$

$$= \begin{array}{c} \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{pmatrix} 0 & m & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \end{pmatrix} \end{array} \sqcup \begin{array}{c} \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array} \sqcup \begin{array}{c} \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix} \end{array}$$

$$= \begin{array}{c} \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{pmatrix} 0 & m & 0 & 1 & 0 & 1 \\ 0 & m & 0 & 1 & 1 & 1 \end{pmatrix} \end{array}.$$

Therefore,

⟦SELECT $R_1.A, R_1.B, R_2.D$

FROM R_1, R_2

WHERE $R_1.A\ F = \{m/a, m/b\}$ AND $(R_1.B\ F = R_2.C$ THOLD $1)$; $\rrbracket(\sigma_s, \sigma_t)$

$$= \begin{matrix} \\ 6 \\ 7 \end{matrix} \begin{matrix} \begin{matrix} a & b & c & d & e & f \end{matrix} \\ \begin{pmatrix} 0 & m & 0 & 1 & 0 & 1 \\ 0 & m & 0 & 1 & 1 & 1 \end{pmatrix} \end{matrix}.$$

Chapter 4

Implementations

In the previous chapter we have defined a semantics for the query language $\mathcal{L}$FSQL using the abstract notion of an arrow category. This chapter includes an implementation of those concepts in the programming language Haskell. We start with an informal overview of Haskell and its features that are of interests in our implementation.

4.1 Haskell

Haskell is a purely-functional programming language named after logician Haskell Curry [14, 15]. A functional programming language differs from its imperative counterpart in that rather than performing operations in sequence it evaluates expressions. Haskell has some advanced features which have made it an efficient and flexible choice for science and research. In the basics, a Haskell program is a series of high-level generalizable functions and each function syntactically is inspired by mathematical notation. In our research from concepts to constructions, everything is very much mathematical, for example the $\mathcal{L}$-fuzzy relations. This is one of the main reasons lying behind our choice of it.

Haskell is a language with strong static typing. This means that every single expression has a type determined at the compile time. However, if an explicit type definition is missing, Haskell system infers the type automatically. Types in Haskell not only guarantee correctness but also contributes to the clarity and efficiency of the programs.

Lazy evaluation is another important feature inherent in Haskell. It means that expressions in Haskell are not evaluated unless and until their result is needed by some computations. Laziness significantly affect the way we write programs in Haskell. Although it is difficult to predict memory usage in lazy evaluation, it is indeed a very powerful way to write compact and modular

code. Note that compared to most traditional programming languages, an equivalent program in Haskell often has less code with fewer bugs and requires less time to develop.

Last but not the least, the latest stable release of Haskell comes with a huge collection of libraries and packages [14]. Many of these libraries contain type classes with algebraic or category-theoretic underpinnings. However, we are particularly interested in the library *Parsec*. Parsec is a very powerful parser combinator library that has a rich set of basic parsing functions. More importantly it includes mechanisms by which one can built more complex parsers using those simple functions.

4.2 Implementation of $\mathcal{L}$-Fuzzy Relations

This section solely describes our implementation with the related concepts in Haskell. We also include code snippets as required.

4.2.1 Data Types

The most common way of declaring a type in Haskell is by using the `data` statement. Types defined in this way are called algebraic types and has the following general form.

```
data [context =>] Typename tv1 ... tvi =  Con1 c1t1 c1t2 ... c1tn
                                | ...
                                | Conm cmt1 cmt2 ... cmtq
                    [deriving]
```

The `data` keyword here defines the new type `TypeName` with an optional context and a number of type variables $tv1 \ldots tvi$. The definition then includes a variable number of constructors where each constructor $Coni$ has a name followed by a list of type variables or type constants $cit1 \ldots cit j$. Finally the `deriving` keyword allows the newly created type a part of some predefined typeclasses.

In our implementation we have the following major types defined.

```
data LSet l a = LSet [(a,l)] deriving (Show)

type LRel l a b = LSet l (a,b)

data RelTerm =   Comp RelTerm RelTerm     -- composition
             | Conv RelTerm               -- converse
             | LeftRes RelTerm RelTerm    -- left residuals
             | RightRes RelTerm RelTerm   -- right residuals
             | Var String
             deriving (Show)
```

The first statement declares $\mathcal{L}$-fuzzy set to be a list of pairs with 'a' being the type of the elements and 'l' the corresponding membership degrees from some arbitrary Browerian lattice. The second statement however, uses the keyword `type` which gives some types new names. Therefore, we create a new type for $\mathcal{L}$-fuzzy relations with three type variables: 'l' for lattices values and 'a' and 'b' are for elements of the participating sets. Essentially an $\mathcal{L}$-fuzzy relation is an $\mathcal{L}$-fuzzy set with each element in the pair being a pair itself.

Finally, the type `RelTerm` defines the grammar for relational terms. Note that a term might be a single variable or the result of some relational operations such as composition, converse, etc. The expression `deriving(Show)` makes the type `RelTerm` representable as a character string.

4.2.2 Type Classes

A type class in Haskell differs from the concept of class in Object Oriented Programming. In contrast, it is like an interface that defines a set of behaviour for the member types. As an example, the standard "Eq" class as defined in the "Prelude" library is given below.

```
class Eq a where
    (==) :: a -> a -> Bool
    (/=) :: a -> a -> Bool
    x == y = not (x /= y)
    x /= y = not (x == y)
```

This definition includes two functions to test equality and inequality each of which takes two arguments and produces a Boolean output. The last two lines, however, are the default definition for the two functions in the class.

Once a class is defined we can make a type an instance of the class by using the `instance` statement where we define those signature functions for this type. For example, the basic Haskell type Bool can be made an instance of the equality class as follows:

```
instance Eq Bool where
   True == True      = True
   False == False    = True
   _ == _            = False
```

In our study we limit our implementation to finite cases only. Note that this restriction immediately follows from the fact that a real-world database is always finite in nature. That is, it has a finite number of tables with a finite number of tuples in each table, each attribute has a finite domain, and so on. As a result, we define a class for a finite set with a finite list of elements.

```
class FiniteSet a where
   elements :: [a]

data C = A | B
```

```
instance FiniteSet C where
   elements = [A,B]

instance (FiniteSet a, FiniteSet b) => FiniteSet(a,b) where
   elements = [(x,y) | x <- elements, y <- elements]
```

The first two lines define the class `FiniteSet` which has a set of elements of type 'a'. Next we define a type for the finite set $C = \{A, B\}$ which is then made an instance of the `FiniteSet` class. Lastly, we make an instance for the pairs from two finite sets. This is necessary for defining $\mathcal{L}$-fuzzy relations.

As we know any lattice which would be used for the membership degrees of the elements, has several components and operations which resembles the structure of a class. So, we also define it as a class as follows.

```
class Lattice l where
  bot :: l
  top :: l
  (&&&)  :: l -> l -> l    -- meet
  (|||)  :: l -> l -> l    -- join
  (-->)  :: l -> l -> l    -- relative pseudocomplement


membership :: (Eq a, Lattice l) => LSet l a -> a -> l
membership (LSet l) a = maybe bot id (lookup a l)

instance (FiniteSet a, Eq a, Eq l, Lattice l) => Lattice (LSet l a) where
  bot = LSet  []
  top = LSet  (map (\x -> (x,top)) elements)
  (LSet l) &&& set = LSet [ (x,d) | (x,d1) <- l, let d = d1 &&& membership set x, d /= bot ]
```

As we can see in the above code segment the `Lattice` class consists of a top element, a bottom element, and three binary functions namely, meet, join, and the relative pseudocomplement. The membership function takes an $\mathcal{L}$-fuzzy set and an element as the input and returns the corresponding l-value if exists in the set, otherwise, it returns the bottom element.

We then make `FiniteSet` an instance of the `Lattice` class where the bottom element `bot` is the empty $\mathcal{L}$-fuzzy set and `top` being the set of all elements from the corresponding finite set, each assigned the top element of the lattice. Finally, the last line is a list comprehension defining meet over two $\mathcal{L}$-fuzzy sets. Note that the expression `d/ = bot` forces an `LSet` to contain elements with a degree greater than the bottom element.

4.2.3 $\mathcal{L}$-Fuzzy Relational Operations

Once we `FiniteSet` and `LRel` types are defined, we proceed to have an implementation of the different operations on $\mathcal{L}$-fuzzy relations. However, we are going the include just the composition here. Recall from Chapter 2 that the composition of two relations $R_1 : A \to B$ and $R_2 : B \to C$ is another relation $R_1;R_2 : A \to C$ which is defined by

$$R_1;R_2 \;=\; \{(a,c) \in A \times C \mid \exists b \in B \,:\, (a,b) \in R_1 \text{ and } (b,c) \in R_2\}$$

In our implementation we define composition as follows.

```
compos :: ( FiniteSet a, Eq a,
            FiniteSet b, Eq b,
            FiniteSet c, Eq c,
            Eq l, Lattice l) => LRel l a b -> LRel l b c -> LRel l a c
compos r1 r2 = LSet [ ((x,z),d) | x <- elements, z <- elements, let d = foldr (|||) bot
     [membership r1 (x,y) &&& membership r2 (y,z) | y <- elements], d /= bot ]
```

Notice that the context in the type signature of `compos` requires that the types ‘a’, ‘b’, and ‘c’ are all instances of both `FiniteSet` and `Eq` and ‘l’ an instance of `Eq` and `Lattice` classes. Once again we use list comprehension to compute composition of two `LRel`s. In doing so we first make pairs of type $(\mathtt{a},\mathtt{c})$ and assign membership degrees produced by taking join of the individual meets between the pairs $(\mathtt{a},\mathtt{b})$ and $(\mathtt{b},\mathtt{c})$ for every element of type `b`. However, if the final degree is something other than `bot` only then we include it in the resultant `LSet`.

4.3 Parser

As mentioned earlier, Parsec is fast, simple, and easy to use monadic parser combinator library for Haskell. Therefore, we can sequence together primitive parsers using the `do` notation. As an example let us consider a parser to parse a sentence.

```
sentence :: Parser [String]
sentence = do {   words <- sepBy1 word separator
            ; oneOf ".?!" <?> "end of sentence"
            ; return words
            }
```

As shown above this parser if successful, produces a list of words which are of course some strings. This is done by the combination of three primitive parsers: `sepBy1`, `word`, and `separator`. It then checks if the next character is one of the legal ending characters for a sentence. If so, it returns the list of words parsed successfully, otherwise prints the error message “end of sentence”.

In our implementation we need a parser to translate a valid relational expression into the construction `RelTerm` as defined above. An example of valid expressions is $R; S\check{}/T$ which is supposed to perform the converse on S first, then the composition with R and finally the left residual of the result with T. Note that we define all the binary operators to be left associative. A part of the parser is presented below.

```
opChar = ";/\^"

reservedOp2 :: String -> CharParser st ()
reservedOp2 name = try (string name >> notFollowedBy (oneOf opChar))

relexpr :: Parser RelTerm
relexpr = buildExpressionParser table term <?> "expression"

table = [ [postfix "^" Conv]
      , [binary ";" Comp AssocLeft]
      , [binary "/" LeftRes AssocLeft, binary "\ " RightRes AssocLeft]
        ]

binary  name fun = Infix   (do { reservedOp2 name; return fun })
postfix name fun = Postfix (do { reservedOp2 name; return fun })

term =  parens relexpr
    <|> Var <$> word
    <?> "term"
```

The first line defines the symbols for the four operators. The parser `reservedOp2` parses successfully a primitive expression if it is not followed by one of those characters. Otherwise, it pretends like it did not consume any input. We call our parser `relexpr` which if successful, produces a parser of type `RelTerm`.

The function `buildExpressionParser` builds an expression parser. It has two parameters. The first one is a table of operators with decreasing priority, meaning that the higher an operator is in the list, the higher is its priority. Associativity is defined by the following data type.

```
data Assoc =  AssocNone
            | AssocLeft
            | AssocRight
```

The second argument of `buildExpressionParser` is the basic expression term which might be just a variable or another expression enclosed by parenthesis. For details of parsec we refer to [21].

4.4 The `eval` Function and the Semantics

Finally, we need a function in order to evaluate a relation algebraic expression into our implementation of $\mathcal{L}$-fuzzy relations. We define it as follows.

```
eval :: (Eq l, Lattice l) => RelTerm -> (String -> MyLRel l) -> MyLRel l
eval t f = case t of
         Var x                -> f x
         Comp exp1 exp2       -> composMyLRel (eval exp1 f) (eval exp2 f)
         Conv exp1            -> convMyLRel (eval exp1 f)
         LeftRes exp1 exp2    -> lresMyLRel (eval exp1 f) (eval exp2 f)
         RightRes exp1 exp2   -> rresMyLRel (eval exp1 f) (eval exp2 f)
```

This function takes a relational term and a function (called the environment, `env`) that maps a String variable-name into the actual relation. It then performs the operations in the expression and produce an $\mathcal{L}$-fuzzy relation. Note that the type of the output relation here depends on the types of the relational terms. Similarly, `env` should be a dependent type because depending on which value is provided, the result type is different. Unfortunately the type system in Haskell is not flexible enough to define this kind of type dependency. Therefore, we fix some specific types for our implementation and embed their all possible combinations into a new type and thus making it untyped in essence. As an example let us say we choose to use `Char`, `Int`, and `Float` only. Then the new type, we call it `MyRel` would look like as follows.

```
data MyLRel l =   II (LRel l Int Int)
                | IC (LRel l Int Char)
                | CI (LRel l Char Int)
                | CC (LRel l Char Char)
                | IF (LRel l Int Float)
                | FI (LRel l Float Int)
                | FC (LRel l Float Char)
                | CF (LRel l Char Float)
                | FF (LRel l Float Float)
            deriving (Show)
```

Now the system is able to infer the type for ‘a’ and ‘b’ in `LRellab` for an arbitrary relational term. This type restriction also requires us to have a `MyRel` version of all the $\mathcal{L}$-fuzzy operations. Here we include the code for converse as an example.

```
convMyLRel :: (Eq l, Lattice l) => MyLRel l -> MyLRel l
convMyLRel mlr = case mlr of
                 II (r)   -> II(conv r)
                 IC (r)   -> CI(conv r)
                 CI (r)   -> IC(conv r)
                 CC (r)   -> CC(conv r)
                 IF (r)   -> FI(conv r)
                 FI (r)   -> IF(conv r)
                 FC (r)   -> CF(conv r)
                 CF (r)   -> FC(conv r)
                 FF (r)   -> FF(conv r)
```

From the above definition it is evident that if a relation from `Int` to `Char` is provided as the argument, the converse of it would be a relation from `Char` to `Int`, so on and so forth.

Now, using the semantics presented in Chapter 3 we can actually execute such a relational expression resultant from a $\mathcal{L}$FSQL query. However, as because Haskell does not allow us to have dependent types which was already a problem with the `eval` function for the relational terms, we decide not to proceed doing the semantic implementation.

Chapter 5

Conclusion and Future Works

Databases have become an indispensable part in almost all software applications nowadays. Their usage can be as vast as the customer database of a bank, at the same time as compact as the contact list in a cell phone. Fuzzy databases, a generalization of classical databases, provide a convenient way to deal with imprecision in data. In this thesis we have introduced lattice based database called $\mathcal{L}$-fuzzy database, which further generalizes a fuzzy database by replacing the unit interval of $[0 \ldots 1]$ by a complete Brouwerian lattice $\mathcal{L}$. Although a Dedekind category forms a suitable theory to abstractly work on $\mathcal{L}$-fuzzy relations, it is unable to express the fundamental notion of 0-1 crispness. We proceed to use an extension of it, called the Arrow category. In this research, we have presented a semantics for the query language of $\mathcal{L}$-fuzzy database, called $\mathcal{L}$FSQL using the abstract notion of an arrow category. In doing so we have explained one DDL statement namely the CREATE statement and three most common DDL operations, SELECT, INSERT, and DELETE. For the demonstrations, examples were included wherever required.

In addition to that we have developed an implementation of the $\mathcal{L}$-fuzzy relational operations in the functional programming language, Haskell. The implementation also includes a parser which translates a relational expression into an abstract data type which could then be executed using the underlying implementation.

Our work could be extended in multiple directions. The followings are some of them.

1. The formal semantics presented in our study includes four DDL and DML operations. However, in the future we will extend it to include the other operations like DELETE, UPDATE, ALTER, DROP and so on. Attempts can also be taken to introduce classical-SQL clauses like ORDER BY, GROUP BY, etc. into $\mathcal{L}$FSQL which would eventually strengthen it as a complete query language.

2. An important design concern for modern relational database is to define functional dependency. Studying functional dependencies based on the semantics we have presented here might be a potential research direction from here. Note that in case of fuzzy database we have several generalizations of the notion of a functional dependency. Exploring each of these possibilities and have a complete set of associated axioms could also be a standard contribution in the future.

3. As mentioned in the previous chapter our prototype implementation did not include the semantics part. This is because working with dependent types is not allowed in Haskell. By using a language with dependent types, the implementation would be much nicer and easy. In addition, a real implementation in terms of a programming language or even using a database system much more efficient and faster than Haskell might be considered a standard next step.

4. Furthermore, it would be very useful to introduce features of practical significance to the language we have presented here. One such direction might be to investigate compatibility degree in $\mathcal{L}$FSQL. In fuzzy-SQL this is achieved by the CDEG function that applies to attributes and computes the compatibility degree of conditions involving these attributes [2]. As the compatibility degree is already available in the semantics of the WHERE clause in our thesis, its semantics follows immediately.

Bibliography

[1] Adjei E.: L - Fuzzy Structural Query Language, MSc Thesis, Brock University 2015.

[2] Adjei E., Chowdhury W., and Winter M.: L-Fuzzy Databases in Arrow Categories. 15th International Conference on Relational and Algebraic Methods in Computer Science, 2015.

[3] Birkhoff G.: Lattice Theory. American Mathematical Society Colloquium Publications Vol. XXV, 3rd edition (1940).

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

[5] Fodor J., Roubens M.: Fuzzy Preference Modelling and Multicriteria Decision Support. Kluwer Academic Publishers (1994)

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

[7] Goguen J.A.: L-fuzzy sets. J. Math. Anal. Appl. 18, 145-157 (1967).

[8] Galindo J., Urrutia A., Piattini, M.: Fuzzy Databases: Modeling, Design and Implementation. Idea Group Publishing Hershey, USA, (2006).

[9] Galindo J., Medina J.M., Pons O., Cubero J.C.: A Server for Fuzzy SQL Queries. In: Andreasen T., Christiansen H., Larsen H.L.(Eds.): Flexible Query Answering Systems. LNAI 1495, 164-174 (1998).

[10] Galindo J.: New Characteristics in FSQL, a Fuzzy SQL for Fuzzy Databases. WSEAS Transactions on Information Science and Applications 2(2), 161-169 (2005).

[11] H. Furusawa: A representation theorem for relation algebras: Concepts of scalar relations and point relations, DOI Technical Report DOI-TR-139, 1997.

[12] H. Furusawa: Algebraic formalisations of fuzzy relations and their representation theorems, Ph.D. Thesis, Kyushu University, 1998.

[13] Heyting, A.: "Die formalen Regeln der intuitionistischen Logik. I, II, III", Sitzungsberichte Akad. Berlin: 42–56, 57–71, 158–169 (1930).

[14] Haskell homepage: https://www.haskell.org/

[15] Haskell wiki site: https://wiki.haskell.org/Haskell

[16] Jónsson B., Tarski A.: Boolean algebras with operators, I, II, Amer. J. Math. 73, 891-939 (1951), 74, 127-162 (1952)

[17] Kawahara, Y., Furusawa H.: Crispness and Representation Theorems in Dedekind Categories. DOI-TR 143, Kyushu University (1997).

[18] Kawahara, Y., Furusawa H.: An algebraic formalization of fuzzy relations. Elsevier, Fuzzy Sets and Systems, Volume 101, Issue 1, 125–135 (1999).

[19] Kitainik, L.: Fuzzy Decision Procedures with Binary Relations - Towards a United Theory. Kluwer Academic Press (1993).

[20] Lejeune-Dirichlet, P.G., Vorlesungen über Zahlentheorie, fourth edition, edited and with supplements by R. Dedekind; Vieweg: Braunschweig (1893); reprinted by Chelsea: New York, 1968.

[21] Leijen D.: Parsec, a fast combinator parser. Dept. of Computer Science, University of Utrecht, PO.Box 80.089, 3508 TB Utrecht, The Netherlands. October 4, 2001.

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

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

[24] Peirce C. S.: "Description of a Notation for the Logic of Relatives, Resulting from an Amplification of the Conceptions of Boole's Calculus of Logic", Memoirs of the American Academy of Arts and Sciences 9 (1870), 317–378

[25] Richard J. W. Dedekind: Sur la Théorie des Nombres Entiers Algébrique, Gauthier-Villars: Paris (1877); reprinted in Dedekind (1930–32), Vol. 3, pp. 262–296; English trans. Dedekind (1996b).

[26] R. Hirsch and I. Hodkinson.: Relation Algebras by Games. Volume 147 of Studies in Logic and the Foundations of Mathematics, North Holland, 2002.

[27] Schmidt G.: Relational Mathematics. Encyclopaedia of Mathematics and Its Applications (2011).

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

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

[30] Terano T., Asai K., Sugeno M.: Fuzzy Systems Theory and Its Applications. Academic Press, Inc. (1992).

[31] Winter M.: Goguen Categories - A Categorical Approach to *L*-fuzzy relations. Trends in Logic 25, Springer (2007).

[32] Winter M.: A new Algebraic Approach to *L*-Fuzzy Relations Convenient to Study Crispness. INS Information Science 139, 233-252 (2001).

[33] Winter M.: Relational Constructions in Goguen Categories. *in:* de Swart, H. (Eds.): Relational Methods in Computer Science, 6th Int. Conf. RelMiCS. LNCS 2561, 212-227 (2002).

[34] Winter M.: Derived Operations in Goguen Categories. TAC Theory and Applications of Categories 10(11), 220-247 (2002).

[35] Winter, M.: Arrow Categories. Fuzzy Sets and Systems 160, 2893-2909 (2009).

[36] Zadeh L.A.: Fuzzy sets. Information and Control 8, 338-353 (1965).