Object Classification using L-Fuzzy Concept Analysis

George Tsekpetse Addison

Submitted in partial fulfilment
of the requirements for the degree of

Master of Science

Department of Computer Science
Brock University
St. Catharines, Ontario

Dedication

This dissertation is dedicated to the memory of Mawusi Dogbe and Yao Addison. Although they were always my inspiration, they are unable to see me graduate. This is for them. I also dedicate this dissertation to my brother Reuben Newton Addison who helped me in all things great and small. My final dedication goes to my fiancée Yedem for her continuous support throughout the course of my studies.

Abstract

Object classification and processing have become a coordinated piece of modern industrial manufacturing systems, generally utilized in a manual or computerized inspection process. Vagueness is a common issue related to object classification and analysis such as the ambiguity in input data, the overlapping boundaries among the classes or regions and the indefiniteness in defining or extracting features and relations among them. The main purpose of this thesis is to construct, define and implement an abstract algebraic framework for $\mathcal{L}$-fuzzy relations to represent the uncertainties involved at every stage of the object classification. This is done to handle the proposed vagueness that is found in the process of object classification such as retaining information as much as possible from the original data for making decision at the highest level making the ultimate output or result of the associated system with least uncertainty.

Acknowledgements

First of all, I would like to thank God. Also, I would like to express my deepest appreciation and gratitude to my supervisor Dr. Michael Winter. His patience, continuous guidance and encouragement inspired me to complete this work successfully. I would also like to thank my supervisory committee members Dr. Ke Qiu and Dr. Robson de Grande for their support and guidance. I would also like to thank the entire staff of the Computer Science Department for their help and support throughout my stay and duration of my studies.

Contents

List of Figures

Chapter 1

Introduction

Set theory is a fundamental tool in all of mathematics, computer science and their application. In the classical theory a value x is either element of a set A (usually denoted by $x \in A$) or not. This allows to represent a set by its characteristic function $f_A : A \to \{0,1\}$ returning 1 for x if $x \in A$ and 0 if not [20]. However, the introduction of fuzzy sets published by Zadeh [25], i.e., replacing the set $\{0,1\}$ by the unit interval $[0,1]$ in the characteristic function, brought about a lot of changes in the world of sets. Later on, Goguen then replaced the unit interval $[0,1]$ by a suitable lattice $\mathcal{L}$. This new concept paved the way for further theoretical research by other researchers which was also followed by applications in the field of computer science, engineering and others. Some examples of the application of fuzzy logic can be seen in how washing machines and video cameras operate.

Object classification is an essential procedure in a lot of modern applications of computer science in the industrial environment. Vagueness is a common issue related to object classification and analysis such as the ambiguity in input data, the overlapping boundaries among classes or regions and the indefiniteness in defining or extracting features and relations among them.

Formal Concept Analysis (FCA) deals with the study of data by deriving a concept hierarchy from a collection of objects and their attributes. There have been several attempts to derive a generalized framework which is based on Formal Concept Analysis in the classification of objects. This has led to the development of many frameworks in this research area. For instance, [10] introduced a framework to integrate association rule and classification rule mining based on concept lattice of formal concept analysis. They went on further to suggest an algorithm that builds a concept lattice for each class label. Their algorithm was incremental in nature such that it could handle an increased number of objects and attributes efficiently. We can look at the work of [16] who proposed a method that created concept lattice by using Formal Concept Analysis and generated association rules for classification from concept lattice. These rules were then pruned and sorted which was used in a priority format. In order to achieve proper accuracy in the classification process, the minimum support and minimum confidence had to be adjusted. The minimum support and confidence referred to the measure of how important or valuable a rule was. [15] introduced a model of learning from positive and negative examples described in terms of FCA. This model produced two sets of results. The first being intents of only positive examples and their corresponding extents. The second containing intents such that the corresponding extents contained only negative examples.

Most of the existing works published so far had to perform some form of conceptual scaling, cuts or similar constructions to convert the data from a multi-valued (fuzzy) context to a Boolean context before they applied Formal Concept Analysis.

This brings us to the main contribution of this thesis which is to construct, define and implement an abstract algebraic framework for $\mathcal{L}$-fuzzy relations to represent the uncertainties involved at every stage of object classification. This is done to handle the proposed vagueness that is found in the process of object classification such as retaining information as much as possible from the original data for making a decision at the highest level making the ultimate output or result of the associated system with least uncertainty. Our approach used here is different because we accept the fuzzy contexts (relations) and apply FCA directly on them.

The thesis has been organized into six chapters. Chapter 1 provides the general introduction and information about this work. Chapter 2 provides the relevant mathematical preliminaries with examples such as the basic definitions of relations and $\mathcal{L}$-fuzzy relations as well as various categorical approaches. Chapter 3 mainly discusses the formal concept analysis and object classification. It also introduces the relational algebraic formulas to formulate formal concept analysis as in this context. Chapter 4 talks about the use of hypotheses and classifications using Formal Concept Analysis. This is followed by our concrete implementations of both the algebraic formulas as well as our main framework in Chapter 5. We give our concluding remarks and some useful ideas for future works in Chapter 6.

Chapter 2

Mathematical Preliminaries

In this chapter, we discuss about the mathematical concepts pertaining to this work. We will define the various concepts and provide concrete examples to demonstrate different properties and operations. This is first done using classical set theory and then adapted to the fuzzy case.

2.1 Lattices

A relation $\leq$ on a set that describes a condition of being smaller or equal to is usually called an order relation. The underlying set together with the binary relation $\leq$ is known as a poset. Formally, we get the definition below [6, 22].

Definition 2.1.1. A set A with a binary relation $\leq$ is a poset that satisfies the following axioms for all $x, y, z \in A$

(1) $x \leq x$ Reflexive

(2) if $x \leq y$ and $y \leq z$, then $x \leq z$ Transitive

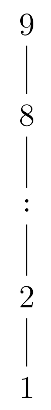
(3) if $x \leq y$ and $y \leq x$, then $x = y$ Antisymmetric

(4) if $x \leq y$ and $y \neq x$, then $x < y$

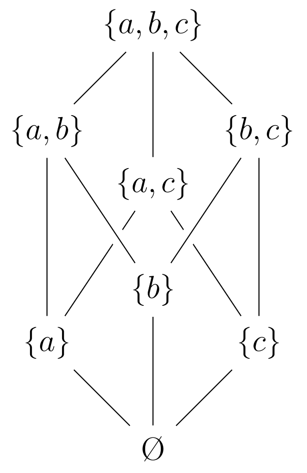
Example 2.1.2. We take the following set of numbers $\{1, 2, ..., 9\}$ together with the usual relation $\leq$. Normally we visualize a poset by a Hasse diagram. The vertices of a Hasse diagram are the elements of the poset. There is an edge between x and y iff $x < y$ and there is no z with $x < z < y$. In other words, the Hasse diagram is the graph of the order where edges that follow from reflexivity and transitivity are removed.

In the Hasse diagram we place larger elements higher than smaller elements. Figure 2.1(a) shows the Hasse diagram of this example.

Example 2.1.3. In this example we want to consider all subsets of the set $\{a, b, c\}$, i.e., the power set $\mathcal{P}(\{a, b, c\}) = \{\varnothing, \{a\}, \{b\}, ...\}$. The smallest element is $\varnothing$ and the greatest element is $\{a, b, c\}$. The Hasse diagram of this example is given in Figure 2.1(b).



(a) Hasse diagram of a set of natural numbers



(b) Hasse diagram of the powerset of {a,b,c}

Figure 2.1: Hasse diagrams of both a set of natural numbers and the powerset of {a,b,c}

A poset$(A, \leq)$ is called linear iff $x \leq y$ or $y \leq x$ holds for all $x, y \in A$. The power set $\mathcal{P}(A)$ of a set A with more than one element together with set-inclusion $\subseteq$ is a standard example of a non-linear poset, i.e., a poset that is not linear [22]. For example, in Figure 2.1(b) we can see that $\{a\}$ is not smaller or equal to $\{b\}$ nor vice versa.

Suppose we have a partially ordered set $(A, \leq)$. If B is a subset of A, then if for every element $x \in A$, $x \geq y$ for all $y \in B$, then A is called an upper bound of B. The least upper bound of B is the least element of the set of upper bounds of B. Dually, a lower bound of B is an element $x \in A$ such that $x \leq y$ for all $x \in B$. Similarly, the greatest lower bound of B is the greatest element of the set of lower bounds of B [9, 22].

Definition 2.1.4. An ordered or partially ordered set $\mathcal{L} = (L, \leq)$ is a lattice, if for any two elements x and y in L the supremum or join given as $x \vee y$ and the infimum or meet given as $x \wedge y$ always exist. $\mathcal{L}$ is called a complete lattice, if the join $\vee X$ and the meet $\wedge X$ exist for any subset of X of $\mathcal{L}$. Furthermore, every lattice $\mathcal{L}$ has a largest element denoted by 1 known as the unit element and a smallest element 0 called the zero element. [6]

We can define a lattice also algebraically, since each pair of elements can have only one upper and lower bound. Finding the bounds can be treated as the result of the use of the meet and join operations $\{\wedge, \vee\}$. For all x, y, z we have the following which have been proved in [9]:

$$x \vee y = y \vee x \text{ and } x \wedge y = y \wedge x \qquad \text{Commutative}$$

$$x \vee (y \vee z) = (x \vee y) \vee z \text{ and}$$

$$x \wedge (y \wedge z) = (x \wedge y) \wedge z \qquad \text{Associative}$$

$$x \vee x = x \text{ and } x \wedge x = x \qquad \text{Idempotency}$$

$$x \wedge y = x \iff x \leq y \text{ and } x \vee y = x \iff x \leq y \qquad \text{Consistency}$$

$$y \leq z \Rightarrow x \wedge y \leq x \wedge z \text{ and } y \leq z \Rightarrow x \vee y \leq x \vee z \qquad \text{Monotonicity}$$

$$x \wedge (x \vee y) = x \text{ and } x \vee (x \wedge y) = x \qquad \text{Absorption.}$$

There could be an instance where we could have a poset which does not qualify to be called a lattice. Although we could have both upper bounds and lower bounds. We can have a look at such an example in Figure 2.2. We realize the upper bounds $\{d, e, f\}$ for b and c but we do not have a least upper bound. The same goes for the lower bounds $\{a, b, c\}$ for d and e which does not also have a greatest lower bound.

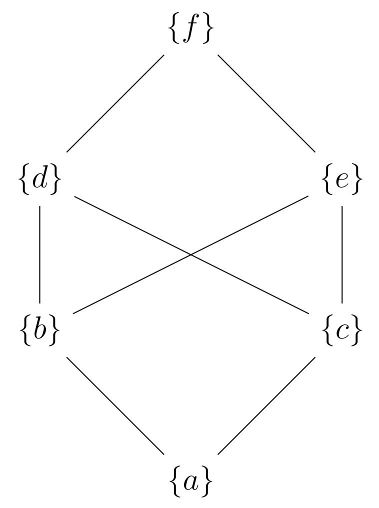


Figure 2.2: An example of a poset which does not qualify to be a lattice

2.1.1 Distributive lattice

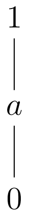
We have many classes of lattices, and taking the meet and join operations into consideration, we realise they may behave like the typical arithmetic multiplication and addition operations where the first distributes over the later. But it is rather unfortunate that not all lattices possess the distributive properties of meet and join.

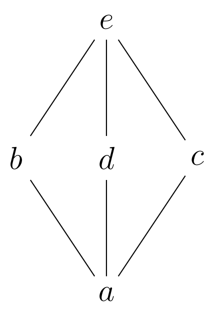
Definition 2.1.5. A lattice $\mathcal{L}$ is said to be distributive iff it satisfies any of the conditions below for all $x, y, z \in \mathcal{L}$.

1. $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
2. $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$

The properties above imply over each. This means that if the first property is true, then the second is also true and vice and versa. The proof can be found in [11, 22].

Figure 2.3(b) illustrates an example of a non distributive lattice known as M_3 which is quite important in the study of lattices since it is able to determine whether a lattice is distributive or not. Another type of a non distributive lattice is the N_5 which has also been illustrated in Figure 2.3(c). A lattice is said to be distributive if none of its sublattices is isomophic to M_3 and N_5. We define a sublattice to be a nonempty subset $M \subseteq \mathcal{L}$ which in itself is a lattice with the same meet and join operations as $\mathcal{L}$. That means for any $a, b, \in M$, we have $a \wedge b$ and $a \vee b \in M$. Two lattices $\mathcal{L}_1 = (L_1, \leq)$ and $\mathcal{L}_2 = (L_2, \leq)$ are said to be isomorphic, and the map $\varphi : L_1 \longrightarrow L_2$ is an isomorphism iff φ is a bijection and $a \leq b$ in $\mathcal{L}_1$ iff $\varphi(a) \leq \varphi(b)$ in $\mathcal{L}_2$ [9].


(a) [0...1] (distributive lattice)


(b) M_3 (non-distributive lattice)

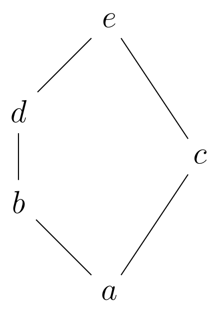

(c) N_5 (non-distributive lattice)

Figure 2.3: A distributive and non distribute lattices

2.1.2 Complete Heyting Algebra

A bounded lattice is a lattice that has a smallest 0 and a greatest element 1. A Heyting algebra is a bounded lattice with an implication operation. This can also be referred to as relative pseudo-complement. A Heyting algebra is said to be complete if it is a complete lattice.

Definition 2.1.6. A bounded lattice $\mathcal{L}$ with the largest element of 1 and the smallest element of 0 and a binary operation $\implies$ is said to be a Heyting algebra iff for all $x, y, z \in \mathcal{L}$ it satisfies the conditions below:

1. $x \implies x = 1$
2. $x \wedge (x \implies y) = x \wedge y$
3. $y \wedge (x \implies y) = y$
4. $x \implies (y \wedge z) = (x \implies y) \wedge (x \implies z)$

where the 4th rule is the distributive property or law for $\implies$. $a \implies b$ is called the relative pseudo-complement of a with respect to b. This operation can be equivalently defined by the following equivalence: $x \leq a \implies b$ iff $a \wedge x \leq b$.

2.2 Boolean Relations

Whenever there is an association between a pair of objects we can term them as binary relations. Normally in mathematics, the operations of Cartesian product forms ordered pairs from two given sets of objects. For instance if A and B are two sets then their Cartesian product is denoted as : $A \times B = \{(a,b) \mid a \in A \text{ and } b \in B\}$. Cartesian products are neither associative nor commutative. This means that for any sets A, B and C.

1. Non-associativity: $(A \times B) \times C \neq A \times (B \times C)$.
2. Non-commutativity: $A \times B \neq B \times A$.

However, please note that the sides in both equations above are isomorphic, i.e., there exists a bijective function between the sets on either side.

1. A relation is a subset of $A \times B$.
2. It can also be represented by the characteristic function.

We can formally define a Boolean or binary relation R as a characteristic function taking into the consideration the sets A and B.

For a Boolean function: $R : A \times B \to \mathbb{B}$ where $\mathbb{B} = \{true, false\}$. This notation maps an ordered pair (a,b) to true if it is related to R, otherwise it is mapped to false. Such a function can be termed as an indicator or characteristic function of the set. Therefore, a pair (a,b) has membership degree of 1 if it is true and a zero if it is false.

The set A is called the source of R and B is known as the target. So writing $R : A \to B$ expresses the fact that there is an $a \in A$ which is related to some $b \in B$ by R. To better understand relations we can visualize them as matrices. In the such a matrix the source elements are set to the labels of the rows and those of the targets are placed as column labels. Whenever we have a *1* in a row and column we say that element in the set A is related to some B by R and it is *0* otherwise.

All the relations that we have looked at so far are usually called heterogeneous Boolean relations, i.e., they are relations between two different sets. A special case of a relation R is when the source and target are the same set. In this case we call R homogeneous [18]. We can consider the factors of 12 $\{1, 2, 3, 4, 6, 12\}$ as a set and the following homogeneous relation on that set.

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

Figure 2.4: Boolean matrix representation of the factors of 12

This is the relation indicating that one number divides another number evenly on the given set of factors of 12. For example, the entry 1 in Row 2 and Column 6 states that 2 divides 6 evenly. Similar, the 0 in Row 4 and Column 3 shows that 3 cannot be divided by 4 evenly.

2.2.1 Basic Operations on Boolean Relations

Let us recall that a relation $R : A \to B$ is a subset of the Cartesian product of A and B, i.e., $R \subseteq A \times B$. Being defined as a subset implies that we can perform some basic operations pertaining to subsets.

Definition 2.2.1. Given two relations $Q, R : A \to B$ of the same type we define the following.

1. **Union:** $R \cup Q = \{(a, b) \in A \times B \mid (a, b) \in R \vee (a, b) \in Q\}$.

2. **Intersection:** $R \cap Q = \{(a, b) \in A \times B \mid (a, b) \in R \wedge (a, b) \in Q\}$.

3. **Complement:** $\overline{\mathrm{R}} = \{(a, b) \in A \times B | (a, b) \notin R\}$ The complement operation also referred to as the negation operation. This operation when applied twice reverts the relation to its initial state before the first operation. This is often referred to as an involution.

4. **Empty relation:** $(\perp_{AB} = \varnothing)$ This relation also known as the empty relation depicts that there is no relation between the relations A and B that is if we want to check whether they equal in terms of the elements in them.

5. **Universal relation:**($\top_{AB} = A \times B$) This relation is refered to as the top relation is the largest relation which relates all elements of A to the elements B.

6. **Identity relation:** $\mathbb{I} = \{(a,a) \in A \times A \mid a \in A\}$. An identity relation is a relation which relates to itself alone.

We will now give an example for each of the operations defined from 1-3 and in 4-6, the definitions will only have the formulas. Using the relations R and Q we have the following.

$$R = \begin{array}{c|cccc} & y_1 & y_2 & y_3 & y_4 \\ \hline x_1 & 1 & 0 & 1 & 0 \\ x_2 & 1 & 1 & 0 & 1 \\ x_3 & 0 & 1 & 1 & 0 \end{array} \qquad Q = \begin{array}{c|cccc} & y_1 & y_2 & y_3 & y_4 \\ \hline x_1 & 1 & 1 & 0 & 1 \\ x_2 & 0 & 1 & 1 & 1 \\ x_3 & 1 & 0 & 0 & 1 \end{array}$$

The intersection and union between R and Q is performed component-wise using the meet and join operations respectively. The meet of 1 and 0 gives us a 0, whereas the join of 1 and 0 gives us a 1.

$$R \cup Q = \begin{array}{c|cccc} & y_1 & y_2 & y_3 & y_4 \\ \hline x_1 & 1 & 1 & 1 & 1 \\ x_2 & 1 & 1 & 1 & 1 \\ x_3 & 1 & 1 & 1 & 1 \end{array} \qquad R \cap Q = \begin{array}{c|cccc} & y_1 & y_2 & y_3 & y_4 \\ \hline x_1 & 1 & 0 & 0 & 0 \\ x_2 & 0 & 1 & 0 & 1 \\ x_3 & 0 & 0 & 0 & 0 \end{array}$$

$$\overline{\mathrm{R}} = \begin{array}{c|cccc} & y_1 & y_2 & y_3 & y_4 \\ \hline x_1 & 0 & 1 & 0 & 1 \\ x_2 & 0 & 0 & 1 & 0 \\ x_3 & 1 & 0 & 0 & 1 \end{array}$$

2.2.2 Relational Operations

In this section we will be talking about additional operations on relations different from the set based operations of the previous section. They are equally important in the use and study of relations. We will demonstrate the examples using the already defined relations R and Q in Section 2.2.1.

Definition 2.2.2. Let $R : A \to B$ and $S : B \to C$ be relations. Then we define

1. the converse $R^{\smile}$ of R by $R^{\smile} = \{(a,b) \mid (b,a) \in R\}$,

2. the composition $R;S$ of R and S by $R;S = \{(a,c) \in A \times C \mid \exists b \in B : (a,b) \in R \wedge (b,c) \in S\}$.

Let us give examples for the converse and composition operations, using relations R and S.

$$R = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{cccc} y_1 & y_2 & y_3 & y_4 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array} \qquad S = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{ccc} y_1 & y_2 & y_3 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \end{array}$$

For the converse, we just exchange the row and column and mirroring the matrix along the diagonal upper left to lower right. Just like the negation operation, the converse operation is said to be involutory as well, i.e., applying the operation twice results in the original relation: $(R^{\smile})^{\smile} = R$.

The composition operation can only occur when the column of the first relation is equal to the row of the second relation. There are left and right unit elements that must exist for composition to take place. The unit elements are referred to as identity relations which vary over the sets: $\mathbb{I}_a, \mathbb{I}_b$ and $\mathbb{I}_c$. They also relate every element to itself [17]. Examples of both operations can be found below:

$$R^{\smile} = \begin{array}{c} \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{array} \begin{array}{c} \begin{array}{ccc} x_1 & x_2 & x_3 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \end{array} \qquad R;S = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array} \begin{array}{c} \begin{array}{ccc} y_1 & y_2 & y_3 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \end{array}$$

The following lemma shows some important properties of the operations introduced so far.

Lemma 2.2.3.

1. Taraski rule : If $X \neq \bot_{DE}$, then $\top_{DD}; X; \top_{EE} = \top_{DE}$.

2. Schröder equivalences: $X;Y \subseteq W \iff X^{\smile}; \overline{\mathrm{W}} \subseteq \overline{\mathrm{Y}} \iff \overline{\mathrm{W}}; Y^{\smile} \subseteq \overline{\mathrm{X}}$ [18].

3. Dedekind rule: $X;Y \cap W \subseteq (X \cap W;Y^{\smile});(Y \cap X^{\smile};W)$.

   The proofs for the above properties are straight forward and can be found in either [17] and [18].

2.2.3 Composite Operations

Similar to multiplication and division on numbers an operation such as composition often induces a quotient operation. Due to this, the concept of residuals are introduced. Since composition is not commutative we have a left and a right residual.

Definition 2.2.4. Let $R : B \to C$ and $S : A \to C$ be relations. Then we define

1. the left residual $R\backslash S$ of R by S is given as $R\backslash S = \overline{\mathrm{R}^{\smile};\overline{\mathrm{S}}}$,
2. the right residual R/S of R and S by $R/S = \overline{\overline{\mathrm{R}};\mathrm{S}^{\smile}}$.

Given the relations R and S, we give the examples of both the left and right residuals.

$$R = \begin{array}{c|cccc} & y_1 & y_2 & y_3 & y_4 \\ \hline x_1 & 1 & 0 & 1 & 0 \\ x_2 & 1 & 1 & 0 & 1 \\ x_3 & 0 & 1 & 1 & 0 \end{array} \qquad S = \begin{array}{c|cccc} & y_1 & y_2 & y_3 & y_4 \\ \hline x_1 & 1 & 1 & 0 & 1 \\ x_2 & 0 & 1 & 1 & 1 \\ x_3 & 1 & 0 & 0 & 1 \\ x_4 & 1 & 1 & 1 & 0 \end{array}$$

The left residual is the largest solution A of $R; A \subseteq S$, i.e., we have $A \subseteq R\backslash S$ iff $R; A \subseteq S$. The proof is straight forward and can be found in [18]. In the same way, we divide R from S on the right in the right residual which equally denotes the greatest of all solutions. From the example given below, we realise the residual R/S sets into a relation a row of R with the rows of S it contains [17].

$$R\backslash S = \begin{array}{c|cccc} & y_1 & y_2 & y_3 & y_4 \\ \hline x_1 & 0 & 0 & 0 & 0 \\ x_2 & 1 & 0 & 1 & 0 \\ x_3 & 0 & 0 & 0 & 0 \end{array} \qquad R/S = \begin{array}{c|cccc} & y_1 & y_2 & y_3 & y_4 \\ \hline x_1 & 0 & 1 & 0 & 1 \\ x_2 & 0 & 0 & 0 & 1 \\ x_3 & 1 & 0 & 0 & 1 \\ x_4 & 0 & 1 & 1 & 1 \end{array}$$

Definition 2.2.5. Symmetric Quotient: Given two relations that is R and S with similar sources, we can also derive their symmetric quotients as: $\mathbf{syq}(R, S) = \overline{\mathrm{R}^{\smile};\overline{\mathrm{S}}} \cap \overline{\overline{\mathrm{R}^{\smile}};\mathrm{S}}$.. This operation has been used successfully in various application fields. Mathematically for the two relations R and S we can deduce the symmetric quotient as a relation $R : A \longrightarrow B$ and $S : A \longrightarrow C$ where there is relation between $b \in B$ to an element $c \in C$ specifically b and c have the same set of "inverse images" with respect to R or S respectively. We can equally get the proof from [18].

$$R = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array}\!\!\begin{array}{c} \begin{array}{cccc} y_1 & y_2 & y_3 & y_4 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \end{array} \qquad S = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \end{array}\!\!\begin{array}{c} \begin{array}{cccc} y_1 & y_2 & y_3 & y_4] \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

$$\mathbf{syq}(\text{R,S}) = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array}\!\!\begin{array}{c} \begin{array}{cccc} y_1 & y_2 & y_3 & y_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

2.3 $\mathcal{L}$-Fuzzy Relations

Since a relation $R : A \to B$ is a set of pairs it has a characteristic function $f_R : A \times B \to \{0,1\}$. An $\mathcal{L}$-fuzzy relation replaces the set $\{0,1\}$ of the Boolean values by a complete Heyting algebra $\mathcal{L}$. The lattice $\mathcal{L}$ usually depicts the degree to which there is a relation between to pairs of sets. The theory of $\mathcal{L}$-fuzzy relations is critical to our study since it allows us to apply relational methods to problems in which classical relations are not appropriate. Please note that $\mathcal{L}$-fuzzy relations generalize the notion of fuzzy relations since the unit interval is a complete Heyting algebra.

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

Figure 2.12: $\mathcal{L}$-fuzzy relation presented as a matrix

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

These are the same kind of operations used for classical relations which are generalized to $\mathcal{L}$-fuzzy relations which has already been defined in Section 2.2.1. For $\mathcal{L}$-fuzzy relations $Q, R : A \longrightarrow B$, $S : B \longrightarrow C$, and $T : D \longrightarrow B$, we have the following: [22]

- **Union:** $(Q \cup R)(x,y) := Q(x,y) \vee R(x,y)$

- **Intersection:** $(Q \cap R)(x,y) := Q(x,y) \wedge R(x,y)$

- **Converse/Transposition:** $Q^{\smile}(x,y) \coloneqq Q(y,x)$

- **Composition:** $(Q;S)(x,y) \coloneqq \bigvee_{y \in B} (Q(x,y) \wedge S(y,z))$

- **Inclusion:** $Q \subseteq R \Longleftrightarrow \forall x \in A, y \in B : Q(x,y) \leq R(x,y)$

- **Empty and Universal relation:** $\bot_{AB} \coloneqq 0$, $\top_{AB} \coloneqq 1$

- **Identity relation:**

$$\mathbb{I}_{AB}(x,y) = \begin{cases} 1 & \text{iff } x = y \\ 0 & Otherwise \end{cases}$$

- **Left residual:** $(Q \backslash T)(x,v) \coloneqq \bigwedge_{y} T(v,y) \to Q(x,y)$ where the $\to$ represents the relative pseudocomplement.

For us to properly define our framework for classification we had to further dive deeper into the properties of the above operations in regards to $\mathcal{L}$-fuzzy relations. The correlative proofs of the following properties can be found in [22].

Theorem 2.3.1. *Let $\mathcal{L}$ be a complete Heyting algebra. Then for all $\mathcal{L}$-fuzzy relations, $Q, Q\prime, Q_i : A \to B, R, R_i : B \to C, S : C \to D$ 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. $(Q \cap Q\prime)^{\smile} = Q^{\smile} \cap Q\prime^{\smile}$
4. $(Q;R)^{\smile} = R^{\smile};Q^{\smile}$
5. $(Q^{\smile})^{\smile} = Q$
6. $Q;(\bigcap_{i \in I} R_i) \subseteq \bigcap_{i \in I}(Q;R_i)$ *and* $(\bigcap_{i \in I} Q_i);R \subseteq \bigcap_{i \in I}(Q_i;R)$

7. $Q;R \cap T \subseteq Q;(R \cap Q^{\smile};T)$

8. $Q;\perp\!\!\!\perp_{BC} = \perp\!\!\!\perp_{AC}$

9. $Q;(\bigcup_{i\in I} R_i) = \bigcup_{i\in I}(Q;R_i)$ *and* $(\bigcup_{i\in I} Q_i);R = \bigcup_{i\in I}(Q_i;R)$

2.3.2 Crispness and Arrow Operations in $\mathcal{L}$-Fuzzy Relations

Since $\mathcal{L}$-fuzzy relations are a generalization of classical or Boolean relations we can say an $\mathcal{L}$-fuzzy relation Q is called $0-1$ crisp iff $Q(x,y) = 0$ or $Q(x,y) = 1$ for all x and y. Crispness is crucial notion to the study of fuzzy theory. The crisp relation can be identified with regular relations that is the regular true and false of $\mathbb{B}$. Taking into consideration that these relations are closed under the operations above [22].

We can also consider another class of relation known as scalar relations. Scalar relations first came to light in [4] and [12]. A relation $\alpha : A \to A$ is called a scalar on A iff $\alpha \subseteq \mathbb{I}_A$ and $\top\!\!\!\top_{AA};\alpha = \alpha;\top\!\!\!\top_{AA}$. Given an element from $\mathcal{L}$ we can define a scalar on any set as follows.

Definition 2.3.2. Let $a \in \mathcal{L}$ and $x, y \in A$. Then we define a scalar by $\alpha : A \to A$

$$\alpha_A^a(x,y) = \begin{cases} a & \text{iff } x = y, \\ 0 & \text{otherwise} \end{cases}$$

Scalar relations are characterized as partial identities with a single element from $\mathcal{L}$ on the diagonal. An example can be seen in Figure 2.14

$$\begin{pmatrix} a & 0 & 0 & 0 \\ 0 & a & 0 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & a \end{pmatrix}$$

(a) Scalar relation with $a \in \mathcal{L}$

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

(b) Scalar relation of $\perp\!\!\!\perp_{AA}$

Figure 2.13: An example of two scalar relations

It is of utmost importance to generate crisp relations in the area of fuzzy relations. An example to consider can be based on the classification process where we will eventually have to conclude to a crisp decision at the end. For us to get to the point where crisp decisions can be made, we will need to introduce certain operations and one of them is known as the α-cut which helps us generate crisp relations. This operation was first introduced in [13].

Definition 2.3.3. Let $\alpha \in \mathcal{L}$ and $R : A \to B$. Then the α-cut of an $\mathcal{L}$-fuzzy relation R is 0-1 crisp relation defined by the following:

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

At this point we can introduce two operations that assist in the process of α-cut operations. These new operations are known as the $up - arrow$ and $down - arrow$. They are defined as follows:

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

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

The up-arrow operation is used to increase the membership degrees which are greater than 0, whereas the down-arrow operation reduces membership degrees which are lesser or smaller than 1. These operations both give off the least and greatest 0-1 crisp relation respectively. The up-arrow and down-arrow are also known as the support and kernel respectively. The figure below shows a relation R with the up-arrow and down-arrow operation applied on the relation.

$$R = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{cccc} y_1 & y_2 & y_3 & y_4 \end{array} \\ \begin{pmatrix} 2 & 1 & 2 & 0 \\ 1 & 0 & 2 & 0 \\ 2 & 0 & 1 & 1 \\ 2 & 2 & 2 & 2 \end{pmatrix} \end{array}$$

$$R^{\uparrow} = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{cccc} y_1 & y_2 & y_3 & y_4 \end{array} \\ \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \end{array} \qquad R^{\downarrow} = \begin{array}{c} \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \begin{array}{c} \begin{array}{cccc} y_1 & y_2 & y_3 & y_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Below are certain useful properties of the operations defined above:

Lemma 2.3.4. Let $\mathcal{L}$ be a complete Brouwerian lattice, and $Q, R : A \to B$ and $S : B \to C$ be $\mathcal{L}$-fuzzy relations. Then we have

1. Q is 0-1 crisp iff $Q^{\uparrow} = Q$ iff $Q^{\downarrow} = Q$
2. $(R^{\smile}; S^{\downarrow})^{\uparrow} = R^{\uparrow\smile}$
3. $(Q \cap R^{\downarrow})^{\uparrow} = Q^{\uparrow} \cap R^{\downarrow}$
4. if $u \neq 0$, then $\alpha_A^{u\uparrow} = \mathbb{I}_A$
5. $Q_u = (\alpha_A^u \backslash Q)^{\downarrow}$

Their corresponding proofs can be found in [22].

Another important property is given by the so-called α-cut theorem. For an $\mathcal{L}$-fuzzy relation $R : A \to B$ and a scalar α then we have $(\alpha\backslash R)^{\downarrow}$ as the α-cut of the relation R. Consequently we have the theorem below.

Theorem 2.3.5. *Let $\mathcal{L}$ be a complete Brouwerian lattice and $R : A \to B$ a $\mathcal{L}$-fuzzy relation. Then we have*

$$R = \bigcup_{u \in \mathcal{L}} (\alpha_A^u; R_u).$$

The proof of the above theorem can also be found in [22]. The relation expression of $(\alpha^u\backslash R)^{\downarrow}$ computes the α-cut of R. Let R be a relation with the scalar α^u, the α-cut of R is given as follows.

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

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

$$(\alpha\backslash R)^{\downarrow} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

2.4 Categories of Relations

In this section, we will discuss the various categorical approaches which would be used to deduce our abstract algebraic framework for $\mathcal{L}$-fuzzy relations.

2.4.1 Categories

A category can be defined as a framework or concepts that is sufficiently expressive through abstraction and describes various structures and formalisms in mathematics. The use of categories allows us to make hidden properties of an object exposed to allow for the applications of new methods on them. As it has been defined in [22].

Definition 2.4.1. A category C consists of

1. a class of objects Objc
2. for every pair of objects A and B a class of morphisms in $C[A,B]$.
3. an associative binary operation ; mapping 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 $g;\mathbb{I}_A = g$

Some common examples of categories are listed below:

1. **Set** - referred to as the category of sets
2. **Rel** - referred to as the category of relations.
3. **$\mathcal{L}$-Rel**- referred to as the category of $\mathcal{L}$-fuzzy relations
4. **Vect** - referred to as the category of vector spaces and linear mapping

We can convey a morphism in $C[A,B]$ as $f : A \to B$ which is equivalent in relational form as $R : A \to B$, since relations will be morphisms. The theory of categories coincides with set theory, in the sense that whiles set theory deals with membership and equality, category theory deals about the composition and equality of abstract functions called morphisms.

2.4.2 Allegories

Allegories are referred to as categories with certain structural elements of category of sets and a binary relation between them. They are often used as abstractions of the categories of relations. As defined in [22], we have the following definition for allegories.

Definition 2.4.2. An allegory $\mathcal{R}$ is a category satisfying the following:

1. For all objects A and B the class $\mathcal{R}[A, B]$ 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: $(Q; S)^{\smile} = S^{\smile}; Q^{\smile}$ and $(Q^{\smile})^{\smile} = Q$.
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 $Q; R \sqcap S \sqsubseteq Q; (R \sqcap Q^{\smile}; S)$ holds.

We realise that the category of $\mathcal{L}$-fuzzy relations that is, **$\mathcal{L}$-Rel** with meet and converse operations as well as the collection of binary relations on fixed sets constitutes a distributive lattice with a smallest element. In spite of that, we intend to define distributive lattice structure as the order in allegories which replaces the lower semilattices. The following definition can be found in [22].

Definition 2.4.3. 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 $Q; (R \sqcup S) = Q; R \sqcup Q; S$.

Based on the above definition, we realise that the allegory **$\mathcal{L}$-Rel** of $\mathcal{L}$-fuzzy relations with the basic set operation union is distributive.

With the addition of the residual(division) operation to a distributive allegory, we derive a division allegory which is defined as follows according to [22].

Definition 2.4.4. A division allegory $\mathcal{R}$ is a distributive allegory such that the composition operation $(;)$ has an upper left adjoint, that is, 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:

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

We can equally have an upper right adjoint which is denoted as $Q\backslash S$ and known as the right residual of S and Q. Having both residuals gives off another form of operation which is known as the symmetric quotient. Below is Lemma presenting some basic properties of this operation. The proof can be found in [22] and another form of it's presentation can also be found here as well [1].

Lemma 2.4.5. Let $\mathcal{R}$ be a division allegory, $Q : A \to B, R : A \to C, S : A \to D$ be relations, and if $f : D \to A$ be a mapping. Then we have

1. $f;\, syq(Q,R) = syq(Q;f^{\smile},R)$
2. $syq(Q,R)^{\smile} = syq(R,Q)$.
3. $syq(Q,R);\, syq\,(R,S) \sqsubseteq syq(Q,S)$.

2.4.3 Dedekind Categories

In this section, we will discuss another crucial categorical approach known as the Dedekind Categories. A Dedekind category is typically a division allegory which is a locally complete distributive allegory or where the distributive lattices are complete Heyting algebras. [3]. Dedekind categories falls as part of our fundamental theory in the development of our framework for classification using $\mathcal{L}$-fuzzy relations. We have the complete definition of Dedekind categories as defined in [21].

Definition 2.4.6. A Dedekind category $\mathcal{R}$ is a category satisfying the following:

1. For all objects A and B the collection $\mathcal{R}[A,B]$ is a complete distributive lattice. Meet, join, the induced ordering and the least and the greatest elements are denoted by $\sqcap, \sqcup, \sqsubseteq, \perp\!\!\!\perp_{AB}, \top\!\!\!\top_{AB}$, respectively.

2. There is a monotone operation $^{\smile}$ (called conversion) such that for all relations $Q : A \to B$ and $R : B \to C$ the following holds: $(Q;R)^{\smile} = R^{\smile};Q^{\smile}$ and $(Q^{\smile})^{\smile} = Q$.

3. For all relations $Q : A \to B, R : B \to C$ and $S : A \to C$ the modular law $Q;R \sqcap S \sqsubseteq Q;(R \sqcap Q^{\smile};S)$ holds.

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

We will go on further to describe some properties of Dedekind categories. The definition is as follows as depicted in [21, 22].

Lemma 2.4.7. Let $\mathcal{R}$ be a Dedekind category. Then for all objects A and B in $\mathcal{R}$ the following are true.

1. $\mathbb{T}^{\smile}_{AB}$=$\mathbb{T}_{BA}$

2. $\mathbb{T}_{AA};\mathbb{T}_{AB}$=$\mathbb{T}_{AB};\mathbb{T}_{BB}$=$\mathbb{T}_{AB}$

3. $\mathbb{T}_{AB}$=$\mathbb{T}_{AB};\mathbb{T}_{BA};\mathbb{T}_{AB}$.

   The proof for the lemma above is straightforward and can be found in [22].

We will go on to the define an important class of relations given by maps as defined in [24].

Definition 2.4.8. Let $\mathcal{R}$ be a Dedekind category. Then a relation $Q : A \to B$ is called

1. univalent(or partial function) iff $Q^{\smile};Q \subseteq \mathbb{I}_B$,

2. total iff $\mathbb{I}_A \subseteq Q;Q^{\smile}$,

3. injective iff $Q^{\smile}$ is univalent,

4. surjective iff $Q^{\smile}$ is total,

5. a map iff Q is total and univalent.

The next lemma contains some other properties which are also valid in Dedekind categories and they are as follows::

Lemma 2.4.9. Let $\mathcal{R}$ be a Dedekind category $Q : A \to B, R : B \to C, S : A \to D$, and $T : D \to C$. Then we have

1. $(Q \sqcap S;\mathbb{T}_{DB});R = Q;R \sqcap S;\mathbb{T}_{DC}$ and $Q;(R\sqcap \mathbb{T}_{BD};T) = Q;R\sqcap \mathbb{T}_{AD};T$

2. $\mathbb{I}_A \sqcap Q; Q^{\smile} = \mathbb{I}_A \sqcap Q; \mathbb{T}_{BA} = \mathbb{I}_A \sqcap \mathbb{T}_{AB}; Q^{\smile}$.

3. Q is total iff $Q; \mathbb{T}_{BC} = \mathbb{T}_{AC}$ for all objects in C.

The Lemma was taken from [21, 22] and their proofs can equally be found in the same text.

There are several other properties in Dedekind categories that reference partial identities. This has been elaborated in the Lemma below and was taken from [22].

Lemma 2.4.10. Let $\mathcal{R}$ be a Dedekind category, $S, T, S_i : A \to A$ for all $i \in I$ partial identities and $R : C \to A, U : A \to B$. Then we have

1. $S = \mathbb{I}_A \sqcap S; \mathbb{T}_{AA} = \mathbb{I}_A \sqcap \mathbb{T}_{AA}; S$.

2. $R; S = R \sqcap \mathbb{T}_{CA}; S$ and $S : U = U \sqcap S; \mathbb{T}_{AB}$.

3. $\prod\limits_{i \in I}(\mathbb{T}_{CA}; S_i) = \mathbb{T}_{CA}; (\prod\limits_{i \in I} S_i)$, and $\prod\limits_{i \in I}(S_i; \mathbb{T}_{AB}) = (\prod\limits_{i \in I} S_i); \mathbb{T}_{AB}$

4. $\prod\limits_{i \in I}(R; S_i) = R; (\prod\limits_{i \in I} S_i)$, and $\prod\limits_{i \in I}(S_i; U) = (\prod\limits_{i \in I} S_i); U$.

The complete proof for the lemma can equally be found in [22].

Obviously, $\mathcal{L}$**-Rel** is a Dedekind category. However, the language of Dedekind categories does not allow to talk about crisps relations. Therefore, we will have to introduce a new type of category known as *arrow categories*. This new form of category introduces the up-arrow and down-arrows to achieve the crispness property of $\mathcal{L}$-fuzzy relations. We will go deeper into them in the next section.

2.4.4 Arrow Categories

In this section we are going to define arrow categories and present some of their properties. Below is a clear definition of arrow categories adopted from [23].

Definition 2.4.11. An arrow category $\mathcal{A}$ is a Dedekind category with $\perp\!\!\!\perp_{AB} \neq \mathbb{T}_{AB}$ for all objects A and B together with two operations $\uparrow$ and $\downarrow$ satisfying the following:

1. $R^{\uparrow}, R^{\downarrow} : A \to B$ for all $R : A \to B$.

2. $(\uparrow, \downarrow)$ is a Galois correspondence.

3. $(R^{\smile}; S^{\downarrow})^{\uparrow} = R^{\uparrow\smile}; S^{\downarrow}$ for all $R : B \to A$ and $S : B \to C$.

4. if $\alpha \neq \mathbb{T}_{AA}$ is a non-zero scalar then $\alpha^{\uparrow} = \mathbb{I}_A$.

5. $(Q \sqcap R^{\downarrow})^{\uparrow} = Q^{\uparrow} \sqcap R^{\downarrow}$ for all $Q, R : A \to B$

The following theorem implies that arrow categories constitutes the theory of $\mathcal{L}$-fuzzy relations

Theorem 2.4.12. *$\mathcal{L}$-**Rel** with an $\uparrow$ and $\downarrow$ is an arrow category. The proof for this theorem can be found in [22].*

We go on further to define crispness in arrow category as follows [22]:

Definition 2.4.13. A relation $R : A \to B$ of an arrow category $\mathcal{A}$ is called crisp iff $R^{\uparrow} = R$. The crisp fragment $\mathcal{A}^{\uparrow}$ of $\mathcal{A}$ is defined as the collection of all crisp relations of $\mathcal{A}$.

The following contains some basic properties of arrow categories as illustrated in [22].

Lemma 2.4.14. Let $\mathcal{A}$ be an arrow category and $Q, R : A \to B, S : B \to C, T : A \to C$. Then the following are true

1. $\mathbb{I}_A^{\uparrow} = \mathbb{I}_A \neq \perp\!\!\!\perp_{AA}$,

2. $R^{\downarrow\uparrow} = R^{\downarrow}$,

3. $R^{\uparrow\downarrow} = R^{\uparrow}$,

4. $\uparrow$ is a closure and $\downarrow$ a kernel operation,

5. $R = R^{\uparrow}\ iff\ R^{\downarrow} = R^{\uparrow}R^{\downarrow} = R$,

6. $\perp\!\!\!\perp_{AB}^{\uparrow}$=$\perp\!\!\!\perp_{AB}$ and $\mathbb{T}_{AB}^{\downarrow}$=$\mathbb{T}_{AB}$,

7. $(R^{\smile}; S^{\uparrow})^{\uparrow} = R^{\uparrow\smile}; S^{\uparrow}$,

8. $R^{\smile\uparrow} = R^{\uparrow\smile}$ and $R^{\smile\downarrow} = R^{\downarrow\smile}$,

9. $(R; S^{\downarrow})^{\uparrow} = R^{\uparrow}; S^{\downarrow}$ and $(R^{\downarrow}; S)^{\uparrow} = R^{\downarrow}; S^{\uparrow}$,

10. $(R; S^{\uparrow})^{\uparrow} = R^{\uparrow}; S^{\uparrow}$ and $(R^{\uparrow}; S)^{\uparrow} = R^{\uparrow}; S^{\uparrow}$,

11. $(Q \sqcap R^{\uparrow})^{\uparrow} = Q^{\uparrow} \sqcap R^{\uparrow}$,

12. $R^{\downarrow} : Q^{\uparrow} = (R : Q^{\uparrow})^{\downarrow} \sqsubseteq (R : Q)^{\downarrow}$ and $(R : Q)^{\uparrow} \sqsubseteq R^{\uparrow} : Q^{\downarrow}$,

13. $Q^{\uparrow}\backslash T^{\downarrow} = (Q^{\uparrow}\backslash T)^{\downarrow} \sqsubseteq (Q\backslash T)^{\downarrow}$ and $(Q\backslash T)^{\uparrow} \sqsubseteq Q^{\downarrow}\backslash T^{\uparrow}$,

14. $T^{\downarrow}/S^{\uparrow} = (T/S^{\uparrow})^{\downarrow} \sqsubseteq (T/S)^{\downarrow}$ and $(T/S)^{\uparrow} \sqsubseteq T^{\uparrow}/S^{\downarrow}$.

The complete proof for the lemma above can be found in [22].

The following lemma includes some closure properties of the class of crisp relations collected from [22], the proof for the lemma can be found in the same text.

Lemma 2.4.15. Let $\mathcal{A}$ be an arrow category and $Q_i, Q, T : A \to B$ for $i \in I$, $R : A \to C$, and $S : B \to C$ crisp relations. Then the following holds:

1. $\bigsqcup_{i\in I} Q_i$ and $\bigsqcap_{i\in I} Q_i$ are crisp.

2. $Q^{\smile}$ is crisp.

3. $Q;S$ is crisp.

4. R/S and $Q\backslash R$ are crisp.

A Boolean arrow category can be defined as follows

Definition 2.4.16. An arrow category $\mathcal{A}$ is said to be a Boolean arrow category if there is a Boolean algebra $\mathcal{B}$ and a mapping $F : \mathcal{A} \to \mathcal{B}$ from objects of $\mathcal{A}$ to elements of $\mathcal{B}$, such that for all objects A and B in $\mathcal{A}$, we have $F(A) \leq F(B)$ in $\mathcal{B}$ iff there is an arrow $f : A \to B$ in $\mathcal{A}$.

The next lemma talks about Boolean arrow categories and it is depicted as follows:

Lemma 2.4.17. Let $\mathcal{A}$ be a Boolean arrow category and $Q, R : A \to B$. Then we have

1. $\overline{\mathrm{Q}^{\downarrow}} = \overline{\mathrm{Q}}^{\uparrow}$ and $\overline{\mathrm{Q}^{\uparrow}} = \overline{\mathrm{Q}}^{\downarrow}$,

2. $Q^{\uparrow} \sqcup R^{\downarrow} = (Q^{\uparrow} \sqcup R)^{\downarrow}$,

3. $Q^{\uparrow} : R^{\downarrow} = (Q : R^{\downarrow})^{\downarrow}$ and $(Q : R)^{\uparrow} = Q^{\downarrow} : R^{\uparrow}$.

The proof for the lemma above can be found in [23].

The next lemma introduces some properties of symmetric quotients in arrow categories

Lemma 2.4.18. Let $\mathcal{A}$ be an arrow category with $Q : A \to B$ and $R : C \to B$ to be crisp relations. if $syq(Q, R)^{\downarrow}$ is surjective, then the following holds:

$$Q : syq(Q, R)^{\downarrow} = R.$$

The proof can be found [22].

In the next lemma we have summarized some other properties of arrow categories as shown in [22] and their corresponding proofs can also be found in the same text.

Lemma 2.4.19. Let $\mathcal{A}$ be an arrow category, $Q, R : A \to B$ relations and $\alpha_A \neq \perp\!\!\!\perp_{AA}$ a scalar. Then we have

1. $(\alpha_A \backslash R^{\uparrow})^{\downarrow} = R^{\uparrow}$,
2. $(\mathbb{I}_A \sqcap \top\!\!\!\top_{AA}; R; \top\!\!\!\top_{BA})^{\uparrow} = \mathbb{I}_A \sqcap \top\!\!\!\top_{AA}; R^{\uparrow}; \top\!\!\!\top_{BA}$,
3. $\top\!\!\!\top_{AB}; \top\!\!\!\top_{BC} = \top\!\!\!\top_{AC}$ for all objects A, B and C, that is $\mathcal{A}$ is uniform.
4. if $R \neq \perp\!\!\!\perp_{AB}$, then $\top\!\!\!\top_{CA}; R^{\uparrow}; \top\!\!\!\top_{BD} = \top\!\!\!\top_{CD}$ for all objects C and D.

2.4.5 Relational Product

This section talks about relational product which is a type of relational construction in Dedekind categories. We provide the definition in the context of arrow categories, which additionally requires the projections to be crisp. The definition below can be found in [22].

Definition 2.4.20. Let A and B be objects of an arrow category. An object $A \times B$, together with two crisp relations $\pi : A \times B \to A$ and $\rho : A \times B \to B$, is called a relational product of A and B iff the following holds:

1. $\pi^{\smile}; \pi \subseteq \mathbb{I}_A$,
2. $\rho^{\smile}; \rho \subseteq \mathbb{I}_B$,
3. $\pi^{\smile}; \rho = \top\!\!\!\top_{AB}$,
4. $\pi; \pi^{\smile} \cap \rho; \rho^{\smile} = \mathbb{I}_{A \times B}$.

π and ρ are mappings, usually referred to as projections. $\mathcal{R}$ has relational products iff for every pair of objects a relational product does exist. We note that the relational product is equivalent to the categorical products of sets and the corresponding set-theoretic projections.

2.4.6 Relational Sum

We will now go ahead to define relational sum which is often called coproduct. This definition can be found in [17].

Definition 2.4.21. Let A and B be objects of an arrow category. An object $A+B$, together with two crisp relations $\iota : A+B \to A$ and $\kappa : A+B \to B$, is called a relational sum or coproduct of A and B iff the following holds:

1. $\iota;\iota^{\smile} \subseteq \mathbb{I}_A$,
2. $\kappa;\kappa^{\smile} \subseteq \mathbb{I}_B$,
3. $\iota^{\smile};\iota \cup \kappa^{\smile};\kappa = \mathbb{I}_{A+B}$,
4. $\iota;\kappa^{\smile} = \perp\!\!\!\perp_{AB}$.

ι and κ are called injective mappings with disjoint value sets in the relational sum. Given their sources, ι and κ are essentially uniquely defined.

2.4.7 Relational Powers

In this section we are going to look at relational or direct product. They are also the relational version of power objects. The definition below was taken from [24].

Definition 2.4.22. An object $\mathcal{P}(A)$ together with a relation $\varepsilon : A \to \mathcal{P}(A)$ is called a relational power iff

1. $\mathrm{syq}(\varepsilon, \varepsilon) = \mathbb{I}_{\mathcal{P}(A)}$
2. and $\mathrm{syq}(R, \varepsilon)$ is total for every $R : A \to B$.

The relational power is an abstract version of the power set of a set. Normally, in a fuzzy context we are usually interested in the $\mathcal{L}$-power set $\mathcal{L}^A$ of a set, i.e., the set of $\mathcal{L}$-characteristic functions from A to $\mathcal{L}$. This set together with the obvious generalization of the "is element of" relation does not necessarily satisfy the axioms of a relational power. The reason is that ε may not be extensional. We will now look at the definition of a relational power in terms of the set $\mathcal{L}$ [24].

Definition 2.4.23. An object $\mathcal{P}_{\mathcal{L}}(A)$ together with a relation $\varepsilon : A \to \mathcal{P}_{\mathcal{L}}(A)$ is called a fuzzy relational power iff

1. $\mathrm{syq}(\varepsilon, \varepsilon)^{\downarrow} = \mathbb{I}_{\mathcal{P}_{\mathcal{L}}(A)}$

2. and $\text{syq}(R, \varepsilon)^{\downarrow}$ is total for every $R : A \to B$.

The next lemma introduces a fuzzy relational power in terms of arrow categories.

Lemma 2.4.24. Let $\mathcal{A}$ be an arrow category. Then the fuzzy relational power is unique up to isomorphism.

The proof can be found in [24].

2.4.8 Splitting and Units

In this section we are going to define an abstract notion of singleton sets called a unit and the concept of splitting. The following definition was taken from [22].

Definition 2.4.25. An object B is called a unit iff $\mathbb{I}_B = \mathbb{T}_{BB}$ and $\mathbb{T}_{AB}$ is total for all objects in A.

Units are distinctive up to isomorphism. Moving further, we can say in $\mathcal{L}$-**Rel** every singleton set is a unit. A subset M of a set N may be described by the canonical injection $f : M \to N$. Additionally, the set of equivalence classes of an equivalence relation is determined by the function mapping each element to its equivalence class. An equivalence relation can be defined as a homogeneous relation that is reflexive, transitive and symmetric. Putting these two concepts together we introduce the notion of splitting. We will go ahead to define splitting in detail. Please note that the definition was adapted to fit in the context of arrow categories. The original definition can be found in [22].

Definition 2.4.26. Let $Q : A \to A$ be a crisp symmetric idempotent relation, i.e.,$Q^{\downarrow} = Q$, $Q^{\smile} = Q$ and $Q;Q = Q$. An object B together with crisp relation $R : B \to A$ is called a splitting of Q (or R splits Q) iff $R;R^{\smile} = \mathbb{I}_B$ and $R^{\smile};R = Q$.

As depicted earlier on, a splitting is also unique up to isomorphism. If Q is a partial identity, the object B of the splitting corresponds to the subset given by Q. Similarly, if Q is an equivalence relation, B corresponds to the set of equivalence classes. This therefore shows that in $\mathcal{L}$-**Rel** every crisp symmetric idempotent splits.

Chapter 3

Formal Concept Analysis

In this chapter, we will talk about formal concept analysis as well as $\mathcal{L}$-fuzzy concept analysis. We will define some basic definitions concerning formal concept analysis. The detailed definitions can be found in [6].

3.1 Formal Concept Analysis

Formal concept analysis can be defined as a mathematical theory of concepts that deals with the analysis of data. The analysis of the data describes the relationship between a particular set of objects and a particular set of attributes. We will continue with the following definitions which can also be found in [6].

3.1.1 Formal Context

Definition 3.1.1. A formal context is a triple $\mathbb{K} = (G, M, I)$, where G is a set of objects, M is referred to as the set of attributes, and $I \subseteq G \times M$ is a binary relation between G and M with $(g, m) \in I$ or gIm depicting that the object g has the attribute m.

Example 3.1.2. We consider the context in Figure 3.1 with results of an expert analysis of some winter chains.This context was taken from this paper [15] which has a more detailed list of items since we are only considering some of the chains and not all of them. The attributes con and dur are short for convenience and durability respectively. Also the attributes S, Q corresponds to steel chain, quick mounting chain and F, B means the ability for a chain to be mounted whether on the front wheels or both front and rear wheels respectively.

chains	system	mount	con	snow	ice	dur	grade
2	S	B	+	+		+	+
5	Q	B	+		+	+	+
8	Q	B	+			+	+
14	Q	B	+			+	
1	R	F	+	+	+	+	+
3	R	F	+	+	+		+
17	R	F				+	
18	R	B	+	+	+	+	
19	R	F	+	+	+		+
20	Q	F	+			+	

Figure 3.1: Context of some expert analysis of winter chains.

Definition 3.1.3. For a set of objects $A \subseteq G$ and a set of attributes $B \subseteq M$ we define the following:

$$A^{\triangle} = \{m \in M \mid gIm \text{ for all } g \in A\}$$
$$B^{\triangledown} = \{g \in G \mid gIm \text{ for all } m \in B\}$$

$A^{\triangle}$ refers to the set of attributes common to all objects in A, and $B^{\triangledown}$ is the set of objects possessing the attributes in B.

3.1.2 Formal Concept

Definition 3.1.4. A formal concept of the formal context (G, M, I) is a pair (A, B) with $A \in G$ and $B \in M$ which satisfies $A' = B$ and $B' = A$. We call A the extent and B the intent of the concept (A, B) respectively. The set of all formal concepts in (G, M, I) is denoted by $\mathfrak{B}(G, M, I)$.

Definition 3.1.5. if (A_1, B_1) and (A_2, B_2) are concepts of a formal context (G, M, I) we define $(A_1, B_1) \leq (A_2, B_2)$ iff $A_1 \subseteq A_2$ which is equivalent to $B_2 \subseteq B_1$. We refer to (A_1, B_1) as a subconcept of (A_2, B_2) whereas (A_2, B_2) is the superconcept of (A_1, B_1). In this case we have $(A_1, B_1) \leq (A_2, B_2)$. The relation $\leq$ is known as the hierarchical order of the concepts. The set of all concepts of (G, M, I) ordered with this $\leq$ is denoted by $\mathfrak{B}(G, M, I)$ and is called the concept lattice of the context (G, M, I).

Example 3.1.6. Figure 3.2 contains the concepts of the context in Example 3.1.2. The line diagram represents the concept lattice of the context in the previous example. The number of concepts generated here are 38. The numbers below represent the objects or the extent of the concepts generated and the attributes or intents for that matter can be found on top of the extents. We can track each extent to its corresponding intent in the diagram. The red lines represent that there is cross path between intents and extents.

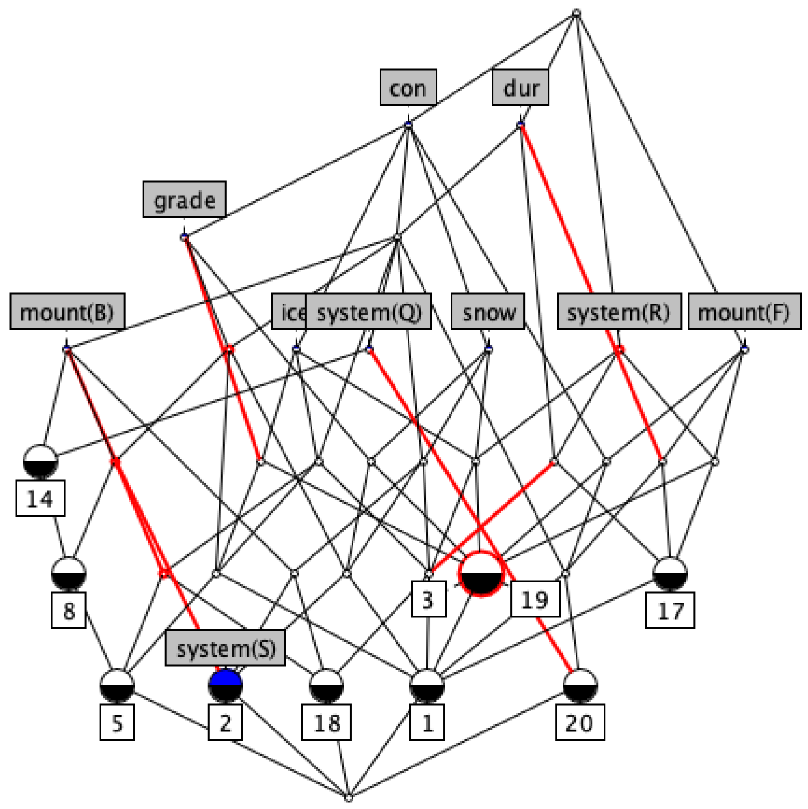


Figure 3.2: Concept lattice of the context in Example 3.1.2.

Definition 3.1.7. For an object $g \in G$ we have g' for the object intent $\{m \in M \mid gIm\}$ of the object g. Similarly, we can also construct $m' = \{g \in G \mid gIm\}$ which is also the attribute extent of the attribute m. In so doing, we tend to have (g'', g') for the object concept and (m', m'') for the attribute concept.

3.2 Relational Algebraic Approach

So far, we have looked at how contexts in formal concept analysis as well as how to generate concepts for a context. We have also looked at some definitions pertaining to formal concept analysis. We will also refer to formal concept analysis as (FCA). Now we would want to formulate the above definitions relation algebraically. Going further in the formulations, we will notice some arrows on the formulas. Since the formulas work for both the Boolean and fuzzy setting the arrows will have much meaning in the fuzzy setting and not the Boolean case. For now we will consider the Boolean case of the formulas and later move to the fuzzy setting. We will begin by defining the partial identity.

Definition 3.2.1. The "subset of" relation $e : \mathcal{P}_L(A) \to \mathcal{P}_L(A)$ is defined by $e = \varepsilon \backslash \varepsilon$. Componentwise we also have the following :

$$e(A,B) = \bigsqcap_x \varepsilon(x,A) \to \varepsilon(x,B)$$

Definition 3.2.2. Let $I : G \to M$ be a relation. Then $\vartriangle : \mathcal{P}_L(G) \to \mathcal{P}_L(M)$ is defined by:

$$\vartriangle\ =\ syq(I^{\smile} / \varepsilon^{\smile}, \varepsilon)^{\downarrow}$$

Deriving the above formula componentwise we have:

$$\vartriangle(A,B) \Leftrightarrow B(x) = \bigsqcap_y \varepsilon(y,A) \to I(y,x)$$

Definition 3.2.3. Let $I : G \to M$ be a relation. Then $\triangledown : \mathcal{P}_L(G) \to \mathcal{P}_L(M)$ is defined by:

$$\triangledown\ =\ syq\,(I/\varepsilon^{\smile}, \varepsilon)^{\downarrow}$$

Correspondingly, we also have the following:

$$\triangledown(A,B) \Leftrightarrow A(y) = \bigsqcap_x \varepsilon(x,B) \to I(y,x)$$

The next lemma shows some function of pairing the above relational operations together into one deriving a whole relational equation which would be used later in this thesis.

Lemma 3.2.4. Considering both the $\vartriangle$ and $\triangledown$ we derive a whole new relational equation given as:

$$\begin{aligned}
\vartriangle ; \triangledown &= syq(I^{\smallsmile}/\varepsilon^{\smallsmile}, \varepsilon)^{\downarrow} ; syq(I/\varepsilon^{\smallsmile}, \varepsilon)^{\downarrow} \\
&= syq((I/\varepsilon^{\smallsmile}) ; syq(\varepsilon, I^{\smallsmile}/\varepsilon^{\smallsmile})^{\downarrow}, \varepsilon)^{\downarrow} \\
&= syq(I/(syq(I^{\smallsmile}/\varepsilon, \varepsilon)^{\downarrow} ; \varepsilon^{\smallsmile}), \varepsilon)^{\downarrow} \\
&= syq(I/(I^{\smallsmile}/\varepsilon^{\smallsmile})^{\smallsmile}, \varepsilon)^{\downarrow} \\
&= syq(I/(\varepsilon\backslash I), \varepsilon)^{\downarrow}
\end{aligned}$$

The Lemma above in addition to the identity relation is given as $\vartriangle ; \triangledown \cap \mathbb{I}$ which is used to generate the partial identity(or concepts) of a relation.

The diagram below shows the diagrammatic view of Lemma of 3.2.4 , and explains the definitions that have been given thus far. As shown in the traditional formal concept analysis, G represents the set of objects as a Relation R and M also represents the set of attributes as a relation. $\mathcal{P}(G)$ and $\mathcal{P}(M)$ represents the relational power set of both relations respectively.

$$\begin{tikzcd}
G \arrow[r, "R"] \arrow[d, "\varepsilon"'] & M \arrow[d, "\varepsilon"] \\
\mathcal{P}_L(G) \arrow[r, shift left, "\vartriangle"] & \mathcal{P}_L(M) \arrow[l, shift left, "\triangledown"]
\end{tikzcd}$$

Figure 3.3: A diagrammatic view of the Lemma 3.2.4

Example 3.2.5. Based on the example given earlier on, we convert the said formal context in Example 3.1.2 into a Boolean relation. We will later on apply a series of equations on them to derive the expected concepts. In the Boolean relation, SS, SQ, SR all represent the attributes specific to systems and MB, MF represent attributes related to the mount options. In the example below, a one means the object has a said attribute and a zero means the object does not have the said attribute.

For the Boolean case of our relations we derive the concepts with the help of the definitions below;

Definition 3.2.6. The relation i_F refers to those pairs of sets (A, B) which are concepts. We have the following: The π and ρ are known as projection relations such

$$\begin{array}{l|cccccccccc}
chains & SS & SQ & SR & MB & MF & con & snow & ice & dur & grade \\
2 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\
5 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\
8 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\
14 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\
3 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\
17 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
18 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
19 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\
20 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0
\end{array}$$

Figure 3.4: An example Boolean relation depicting the context in Example 3.1.2

that $\pi : A \times B \to A$ and $\rho : A \times B \to B$

$$i_F = \pi; \triangle; \rho^{\smile} \cap \rho; \triangledown; \pi^{\smile} \cap \mathbb{I}$$

We equally have the following which further explains the above equation:

$$i_F((A, B), (A, B)) = 1 \Leftrightarrow A^{\triangle} = B \wedge B^{\triangledown} = A$$

The relation i_G refers to only the set of objects that are concepts and is denoted as:

$$i_G = \triangle \ \triangledown \cap \mathbb{I}$$

We equally the following as well:

$$i_G(A, A) = 1 \Leftrightarrow A^{\triangle\triangledown} = A$$

The relation i_M similarly refers to only the set of attributes that are concepts and also has the definition below:

$$i_M = \triangledown; \triangle \cap \mathbb{I}$$

and equally:

$$i_M(B, B) = 1 \Leftrightarrow B^{\triangledown\triangle} = B$$

Figure 3.5 gives a pictorial view of the definition above. The symbols used are similar to that of the ones used so far.

$$\begin{tikzcd}
\mathcal{P}(G)\times\mathcal{P}(M) \arrow[loop above, "i_F"] \arrow[d, "\pi"] \arrow[dr, "\rho"] & \\
\mathcal{P}(G) \arrow[loop left, "i_G"] \arrow[r, shift left, Rightarrow, "\triangle"] & \mathcal{P}(M) \arrow[l, shift left, Rightarrow, "\triangledown"] \arrow[loop right, "i_M"]
\end{tikzcd}$$

Figure 3.5: A visual representation of the Definition 3.2.6

Using the definition above we derive the concepts of the Boolean relation of Example 3.2.5. By applying the equations defined above, we are able to generate a total of 38 concepts of the said example just like the traditional FCA but due to the size of the matrix which is a 1024 by 1024 matrix, we are unable to display the example in details. But the object and attribute pair of the concepts are equally the same as that of the example in Figure 3.2. This goes on to prove that we can equally use a relational algebraic approach to define the Formal concept analysis.

To justify notations in the definitions given in Definition 3.2.6, we show it in the lemma below:

Lemma 3.2.7. With the notation of Definition 3.2.6, we have

$$i_F;\pi;\, e;\pi^{\smile};i_F = i_F;\rho;\, e^{\smile};\, \rho^{\smile};\, i_F$$

Proof. First, we show $\triangle^{\smile};\, e;\, \triangle \subseteq e^{\smile}$

$$\begin{aligned}
e;\, \triangle;\, \varepsilon^{\smile} &= (\varepsilon\backslash\varepsilon);\, syq(R^{\smile}/\varepsilon^{\smile},\varepsilon)^{\downarrow}\, \varepsilon^{\smile} \\
&= (\varepsilon\backslash\varepsilon);\, (R^{\smile}/\varepsilon^{\smile})^{\smile} \\
&= (\varepsilon\backslash\varepsilon)\,(\varepsilon\backslash R) \\
&\subseteq\ \varepsilon\backslash R \\
&= (R^{\smile}/\varepsilon^{\smile})^{\smile} \\
&=\ syq(R^{\smile}/\varepsilon^{\smile},\varepsilon)^{\downarrow};\, \varepsilon^{\smile} \\
&= \triangle\,;\varepsilon^{\smile}
\end{aligned}$$

This implies $\triangle^{\smile};e;\triangle;\varepsilon^{\smile} \subseteq \varepsilon^{\smile}$, and, hence, $\triangle^{\smile};e;\triangle \subseteq \varepsilon^{\smile}\backslash\varepsilon^{\smile} = (\varepsilon\backslash\varepsilon)^{\smile} = e^{\smile}$

Now, we set

$$\begin{aligned} i_F;\, \pi;\, e;\, \pi^{\smile};\, i_F &= i_F;\, i_F;\, \pi;\, e;\, \pi^{\smile};\, i_F;\, i_F \\ &= i_F;\, i_F;\, \pi;\, e;\, \pi^{\smile};\, i_F;\, i_F \\ &\subseteq i_F;\, \rho;\, \triangle^{\smile};\, \pi^{\smile};\, \pi;\, e;\, \pi^{\smile};\, \pi;\, \triangle;\, \rho^{\smile};i_F \\ &\subseteq i_F;\, \rho;\, \triangle^{\smile};\, e;\, \triangle;\, \rho^{\smile};\, i_F \\ &\subseteq i_F;\, \rho;\, e^{\smile};\, \rho^{\smile};\, i_F \end{aligned}$$

□

The opposite inclusion is shown analogously. □

From the above lemma, we realise that i_F gives us those pairs of sets that are concepts. But again we can do that by either taking the attribute part or object part of a context. Lets consider the Lemma below

Based on Definition 2.4.26, we can split the partial identity of i_F, i_G and i_M giving us $i_F = c_F^{\smile};\, c_F,\ i_G = c_G^{\smile};\, c_G$ and $i_M = c_M^{\smile};\, c_M$. Hence giving us $(c_M\, splits i_M) c_M\,; c_M^{\smile} = \mathbb{I}$, $(c_G\ splits\ i_G)\ c_G\,; c_G^{\smile} = \mathbb{I}$ and $(c_F\ splits\ i_F)\ c_F\,; c_F^{\smile} = \mathbb{I}$.

Lemma 3.2.8. Given c_F, c_G and c_M, the following relations

1. $c_G; (\mathbb{I} \otimes \triangle); c_F^{\smile}$
2. $c_M; (\triangledown \otimes \mathbb{I}); c_F^{\smile}$

are bijective maps.

The relations above are bijective because each element in either c_M or c_G is paired exactly with an element in c_F and vice versa.

In Figure 3.6 we have the pairs, G and M giving of the concepts i_F. The split function c_F gives us E representing only those pairs that concepts. Taking into consideration the object and attribute sets, we have the partial identities i_G and i_M representing the objects and attributes which are concepts. Applying the split functions denoted by c_G and c_M gives us D and C respectively which refers to only those objects and attributes that are concepts.

$$\begin{tikzcd}
 & E \arrow[d, "c_F"] & \\
 & \mathcal{P}(G)\times\mathcal{P}(M) \arrow[loop left, "i_F"] & \\
 & \mathcal{P}(G) \arrow[u, "\mathbb{I}\olessthan\vartriangle"] \arrow[loop left, "i_G"] & \mathcal{P}(M) \arrow[ul, "\triangledown\olessthan\mathbb{I}"'] \arrow[loop right, "i_M"] \\
 & D \arrow[u, "c_G"] & C \arrow[u, "c_M"']
\end{tikzcd}$$

Figure 3.6: A visual representation of Lemma 3.2.8

The proof of Lemma 3.2.8 is as follows:

Proof. We only show 2, since 1 follows analogously. Consider the following derivation

$$\begin{aligned}
& c_M;\,(\triangledown\olessthan\mathbb{I});c_F^{\smile};\,c_F;\,(\triangledown\olessthan\mathbb{I})^{\smile};\,c_M^{\smile}\\
&= c_M;\,(\triangledown\olessthan\mathbb{I});\,i_F;\,(\triangledown\olessthan\mathbb{I})^{\smile};\,c_M^{\smile}\\
&= c_M;\,(\triangledown;\pi^{\smile}\cap\rho^{\smile});\,(\pi;\vartriangle;\rho^{\smile}\cap\rho;\triangledown;\pi^{\smile}\cap\mathbb{I});\,(\pi;\triangledown^{\smile}\cap\rho);\,c_M^{\smile}\\
&= c_M;\,((\triangledown;\pi^{\smile}\cap\rho^{\smile});\pi;\vartriangle;\rho^{\smile}\cap(\triangledown;\pi^{\smile}\cap\rho^{\smile});\rho;\triangledown;\pi^{\smile}\cap\triangledown;\pi^{\smile}\cap\rho^{\smile});(\pi;\triangledown^{\smile}\cap\rho);c_M^{\smile}\\
&= c_M;\,(\triangledown;\vartriangle;\rho^{\smile}\cap\triangledown;\pi^{\smile}\cap\triangledown;\pi^{\smile}\cap\rho^{\smile});\,(\pi;\triangledown^{\smile}\cap\rho);c_M^{\smile}\\
&= c_M;\,(\triangledown;\vartriangle;\rho^{\smile}\cap\triangledown;\pi^{\smile}\cap\rho^{\smile});\,(\pi;\triangledown^{\smile}\cap\rho);\,c_M\\
&= c_M;\,(\triangledown;\vartriangle;\rho^{\smile};(\pi;\triangledown^{\smile}\cap\rho)\cap\triangledown;\pi^{\smile};(\pi;\triangledown^{\smile}\cap\rho)\cap\rho^{\smile};(\pi;\triangledown^{\smile}\cap\rho));\,c_M\\
&= c_M;\,(\triangledown;\vartriangle\cap\triangledown;\triangledown^{\smile}\cap\mathbb{I});\,c_M\\
&= c_M;\,(\triangledown;\vartriangle\cap\mathbb{I});\,c_M^{\smile}\\
&= c_M;c_M^{\smile};c_M;c_M^{\smile}\\
&= \mathbb{I},
\end{aligned}$$

i.e., the relation is total and injective.

In order to show that the relation is also univalent, we have

$$\begin{aligned}
&c_F;\,(\triangledown \obslash \mathbb{I})^\smile; c_M^\smile; c_M; (\triangledown \obslash \mathbb{I});\, c_F^\smile \\
&= c_F; (\pi; \triangledown^\smile \cap \rho);\, i_M;\, (\triangledown; \pi^\smile \cap \rho^\smile);\, c_F^\smile \\
&= c_F; (\pi; \triangledown^\smile \cap \rho);\, (\triangledown; \vartriangle \cap \mathbb{I});\, (\triangledown; \pi^\smile \cap \rho^\smile);\, c_F^\smile \\
&\subseteq c_F; (\pi; \triangledown^\smile; \triangledown; \vartriangle \cap \pi; \triangledown^\smile \cap \rho);\, (\triangledown; \pi^\smile \cap \rho^\smile);\, c_F^\smile \\
&\subseteq c_F; (\pi; \triangledown^\smile; \triangledown; \vartriangle; \triangledown; \pi^\smile \cap \pi; \triangledown^\smile; \triangledown; \pi^\smile \cap \rho; \triangledown; \pi^\smile \cap \pi; \triangledown^\smile; \triangledown; \vartriangle; \rho^\smile \cap \pi; \triangledown^\smile; \rho^\smile \cap \rho; \rho^\smile); c_F^\smile \\
&\subseteq c_F; (\pi; \triangledown^\smile; \triangledown; \pi^\smile \cap \rho; \triangledown; \pi^\smile \cap \pi; \triangledown^\smile; \triangledown; \vartriangle; \rho^\smile \cap \rho; \rho^\smile); c_F \\
&\subseteq c_F; (\pi; \pi^\smile \cap \rho; \triangledown; \pi^\smile \cap \pi' \vartriangle; \rho^\smile \cap \rho; \rho^\smile);\, c_F^\smile \\
&= c_F; (\pi; \vartriangle; \rho^\smile \cap \rho; \triangledown; \pi^\smile \cap \mathbb{I});\, c_F^\smile \\
&= c_F; i_F; c_F^\smile \\
&= c_F; c_F^\smile; c_F; c_F^\smile \\
&= \mathbb{I}.
\end{aligned}$$

Finally, surjectivity is shown by

$$\begin{aligned}
\mathbb{I} &= c_F; c_F^\smile; c_F; c_F^\smile \\
&= c_F; i_F; c_F^\smile \\
&= c_F; (\pi; \vartriangle; \rho^\smile \cap \rho; \triangledown; \pi^\smile \cap \mathbb{I});\, c_F^\smile \\
&= c_F; (\pi; \vartriangle; \rho^\smile \cap \rho; \triangledown; \pi^\smile \cap \pi; \pi^\smile \cap \rho; \rho^\smile); c_F^\smile \\
&= c_F; (\pi; \vartriangle; \rho^\smile \cap \pi; \pi^\smile \cap \rho; (\triangledown; \pi^\smile \cap \rho^\smile)); c_F^\smile \\
&\subseteq c_F; ((\pi; \vartriangle; \rho^\smile \cap \pi; \pi^\smile); (\pi; \triangledown^\smile \cap \rho) \cap \rho); (\triangledown; \pi^\smile \cap \rho^\smile); c_F^\smile \\
&= c_F; (\pi; \vartriangle \cap \pi; \triangledown^\smile \cap \rho); (\triangledown \obslash \mathbb{I}); c_F^\smile \\
&\subseteq c_M; (\pi; \triangledown^\smile \cap \rho); ((\triangledown; \pi^\smile; \rho^\smile); \pi; \vartriangle \cap \mathbb{I}); (\triangledown \obslash \mathbb{I}); c_F^\smile \\
&= c_F; (\triangledown \obslash \mathbb{I})^\smile; (\triangledown; \vartriangle \cap \mathbb{I}); (\triangledown \obslash \mathbb{I}); c_F^\smile \\
&= c_F; (\triangledown \obslash \mathbb{I})^\smile; i_M; (\triangledown \obslash \mathbb{I}); c_F^\smile \\
&= c_F; (\triangledown \obslash \mathbb{I})^\smile; c_M^\smile; c_M; (\triangledown \obslash \mathbb{I}); c_F^\smile. \quad \square
\end{aligned}$$

□

The next step is to define the orders on the pairs of which are concepts or just taking an attribute or object set and inducing the ordering on the concepts. The definition is given as:

Definition 3.2.9. We define order relations e_F, e_G, and e_M on $\mathcal{P}(G)\times\mathcal{P}(M)$, $\mathcal{P}(G)$ and $\mathcal{P}(M)$ by

1. $e_F = c_F;\ (\pi; e; \pi^\smile \cap \rho; e^\smile; \rho^\smile); c_F^\smile$
2. $e_G = c_G; e; c_G^\smile$
3. $e_M = c_M; e; c_M^\smile$

Based on the definition given above, with e_F being the strict order of those pairs of sets that are concepts, e_G being the strict order of the object set that are concepts and e_M being the strict order of the attribute set that are concepts. Using the diagram in Figure 3.6, we have c_F, c_G and c_M splitting the sets taking out only those sets that are concepts thereby inducing the subset order on the sets for both pairs or only the object and attribute set respectively. C and D give us only the order of sets that are concepts for both the object and attribute sets. The same applies to E which only gives us the order on those pairs of sets that are concepts.

We have the following lemma based on the above definition since they are subset orderings.

Lemma 3.2.10. We have $e_F = c_F; \pi; e; \pi^\smile; c_F^\smile = c_F; \rho; e^\smile; \rho^\smile; c_F^\smile$.

Proof. We only show the second equality. The other one follows analogously.

$$\begin{aligned}
e_F &= c_F;\ (\pi; e; \pi^\smile \cap \rho; e^\smile; \rho^\smile); c_F^\smile\\
&= c_F; \pi; e; \pi^\smile; c_F^\smile; \cap c_F; \rho; e^\smile; \rho^\smile; c_F^\smile\\
&= c_F; c_F^\smile; c_F; \pi; e; \pi^\smile; c_F^\smile; c_F; c_F^\smile \cap c_F; \rho; e^\smile; \rho^\smile; c_F^\smile\\
&= c_F; i_F; \pi; e; \pi^\smile; i_F; c_F^\smile; \cap c_F; \rho; ^\smile; \rho^\smile; c_F^\smile\\
&= c_F; i_F; \rho; e^\smile; \rho^\smile; i_F; c_F^\smile \cap c_F; \rho; e^\smile; \rho^\smile; c_F^\smile\\
&= c_F; c_F^\smile; c_F; \rho; e^\smile; \rho^\smile; c_F^\smile; c_F; c_F^\smile \cap c_F; \rho; e^\smile; \rho^\smile; c_F^\smile\\
&= c_F; \rho; e^\smile; \rho^\smile; c_F^\smile. \quad \square
\end{aligned}$$

□

Lemma 3.2.11. The maps from Lemma 3.2.8 are monotonic.

Proof. First, we have

$$\begin{aligned}
&c_M; (\triangledown \otimes \mathbb{I}); c_F^{\smile}; e_F^{\smile}; (c_M; (\triangledown \otimes \mathbb{I}); c_F^{\smile})^{\smile} \\
&= c_M; (\triangledown \otimes \mathbb{I}); c_F^{\smile}; e_F^{\smile}; c_F; (\triangledown \otimes \mathbb{I})^{\smile}; c_M^{\smile} \\
&= c_M; (\triangledown \otimes \mathbb{I}); c_F^{\smile}; c_F; \rho; e^{\smile}; \rho^{\smile}; c_F^{\smile}; c_F; (\triangledown \otimes \mathbb{I})^{\smile}; c_M^{\smile} \\
&\subseteq c_M; (\triangledown \otimes \mathbb{I}); \rho; e^{\smile}; \rho^{\smile}; (\triangledown \otimes \mathbb{I})^{\smile}; c_M^{\smile} \\
&= c_M; (\triangledown \otimes \mathbb{I}); \rho; e^{\smile}; ((\triangledown \otimes \mathbb{I}); \rho)^{\smile}; c_M^{\smile} \\
&= c_M; e^{\smile}; c_M^{\smile} \\
&= e_M.
\end{aligned}$$

This implies the assertion. □ □

3.3 $\mathcal{L}$-Fuzzy Concept Analysis

In the previous section we have looked at regular or classical relations and how to generate their respective concepts. In this section we will be looking at fuzzy concept analysis and then dive further into fuzzy context and concepts. Finally we will look at fuzzy relations.

3.3.1 $\mathcal{L}$-Fuzzy Context

Actually nothing changes. The only difference is that I is now a fuzzy relation. The advantage of the relational approach here is that the formulae we used in the Boolean case can also be used in the fuzzy case. First of all, we could define an $\mathcal{L}$-Fuzzy context as two fuzzy sets and the fuzzy relation between them. A fuzzy context can also be defined as found in [19] as:

Definition 3.3.1. An $\mathcal{L}$-fuzzy formal context is a triplet $K = (G, M, I)$ where G is a finite set of objects, M is the finite set of attributes, and I is an $\mathcal{L}$-fuzzy relation on $G \times M$.

3.3.2 $\mathcal{L}$-Fuzzy Relational Approach

In Section 3.2 we introduced some formulas for generating concepts for classical or regular relations. These formulas also work for fuzzy relations as well, hence there is no need to modify them since we get the desired results. We explore the formulas with the examples below

Example 3.3.2. In this example, we consider two fruits as our objects that is an apple and a kiwi fruit. For the attribute, we look at how round and how smooth the fruit is said to be. In this context we use a three element lattice as the interval of degree of membership which $(0, 2, 1)$ where a zero means the object does not have the said attribute, a two meaning the an object has the said attribute to a certain degree which is two and a one means the object completely has the said attribute.

$$\begin{array}{l c} Fruits & \begin{array}{cc} Round & Smooth \end{array} \\ \begin{array}{l} apple \\ kiwi \end{array} & \begin{pmatrix} 1 & \quad 2 \\ 0 & \quad 2 \end{pmatrix} \end{array}$$

Figure 3.7: An example fuzzy relation

We first begin by generating the object concepts by using the $\triangledown\vartriangle$ formulation in Section 3.2. By intersecting the formula with the identity relation we are able to have the formulation below:

$$\triangledown;\vartriangle = syq(R/(\varepsilon\backslash R), \varepsilon)^{\downarrow} \cap \mathbb{I}$$

Where R is the relation and ε refers to the powerset of the relation given as $\mathcal{P}(G)$ since we are working with the object set. By applying the formula above we generate the following concepts for the object set shown below:

$$\begin{array}{l c} Fruits & \\ \begin{array}{l} \{\emptyset\} \\ \{1/kiwi\} \\ \{2/kiwi\} \\ \{1/apple\} \\ \{1/kiwi, 1/apple\} \\ \{2/kiwi, 1/apple\} \\ \{2/apple\} \\ \{1/kiwi, 2/apple\} \\ \{2/kiwi, 2/apple\} \end{array} & \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

Figure 3.8: Concepts generated from the object set

We realise from the above figure that we have four full concepts, that is the diagonals that give us a one. We also have two twos in the diagonal telling as that those pairs of objects are a concept to a degree two.

We now move on to generate the attribute concepts from the same example. To generate the attribute concepts, we use the same equation as used for the object concepts. The only difference is we first have to find the transpose of our relation giving us the new relation below : By finding the transpose of the relation, we now

$$\begin{array}{ll} \textit{Attributes} & \begin{array}{cc} \textit{apple} & \textit{kiwi} \end{array} \\ \begin{array}{l} \textit{round} \\ \textit{smooth} \end{array} & \begin{pmatrix} 1 & 0 \\ 2 & 2 \end{pmatrix} \end{array}$$

have the formula below for generating the attribute concepts of the relation above.

$$\triangle ; \triangledown = syq(R^{\smile}/(\varepsilon \backslash R^{\smile}), \varepsilon)^{\downarrow} \cap \mathbb{I}$$

We realise the equation is the same as before but with just the transposition of the relation R. The attribute concepts are generated below.

$$\begin{array}{ll} \textit{Attributes} & \\ \begin{array}{l} \{\emptyset\} \\ \{1/smooth\} \\ \{2/smooth\} \\ \{1/round\} \\ \{1/smooth, 1/round\} \\ \{2/smooth, 1/round\} \\ \{2/round\} \\ \{1/smooth, 2/round\} \\ \{2/smooth, 2/round\} \end{array} & \begin{pmatrix} 0&0&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \end{pmatrix} \end{array}$$

Figure 3.9: Concepts generated from the attribute set

Based on the definition of a fuzzy concept and comparing both the object concepts and attribute concepts, we realize that the objects have certain attributes to a corresponding membership degree which are all ones. Looking at the sets the other way around, the attributes also have certain objects possessing some attributes to a membership degree with some being ones and others being to a certain degree two. In this section we have shown that, the formulas defined in Section 3.2 works for both regular or classical relations and fuzzy relations.

Chapter 4

Classification using Formal Concept Analysis

In the simplest terms, machine learning is the focus of mathematical models and practical methods for adapting computer systems. A common approach to the idea of learning is the process of learning from both positive and negative examples where the system gets details of positive and negative examples and goes on to create predictions of positive examples which does not represent negative examples. In this chapter, we discuss the logical framework of the JSM-method and Formal Concept Analysis and how classification of objects or data are performed. Since we have already discussed Formal Concept Analysis in Chapter 3, we will go directly to define the JSM-method.

4.1 JSM-Method

This framework was first introduced in the work of Finn [2]. The main purpose for this framework was to serve as a plausible inference on the basis of infinite-valued first-order logical theory with quantifiers over tuples of variable length. Upon careful observation, we realise the application of this method can be used within the scope of machine learning. Conversely, to the majority of existing models used in machine learning where similarity is given a metric or as a relation, the JSM-method uses this similarity defined as an algebraic operation with certain properties [15]. Apart from deducing the decision rule that is used to separate positive from negative examples, the JSM-method constructs a minimal cover or hypothesis of positive examples which has no ties to the negative examples. As a method of data analysis, JSM-method is a system of machine learning from positive and negative examples. This is done by constructing a generalization of positive examples given from positive and negative examples with respect to a ”goal attribute” [15]. We will now show the relationship between Formal Concept Analysis and the JSM-method in the definitions below which can be found in [5, 14, 15].

Definition 4.1.1. Let $K = (G, M, I)$ be a formal context and $w \notin M$ is a certain goal attribute of objects, different from attributes from the set M(referred to as the set of structural attributes). This produces three subsets from the set G of all objects that is: The set G_+ of those objects that are known to have the property w(positive samples), the set G_- of those objects that do not possess the property w(negative samples) and the set G_τ of undetermined examples, i.e., those objects of which it is unknown if they have the property w or not. This brings about three subcontexts of $\mathbb{K} = (G, M, I)$:

1. positive context $K_+ = (G_+, M, I_+)$
2. negative context $K_- = (G_-, M, I_-)$
3. undetermined context $K_\tau = (G_\tau, M, I_\tau)$

The definition below represents the learning and classification contexts which can also be found in [14, 15]

Definition 4.1.2. Let a positive context $K_+ = (G_+, M, I_+)$, negative context $K_- = (G_-, M, I_-)$, and undetermined context $K_\tau = (G_\tau, M, I_\tau)$ be given. The context $K_\pm = (G_+ \cup G_-, M \cup \{w\}, I_+ \cup I_- \cup (G_+ \times \{w\}))$ is referred to as the *learning context.* The context $K_c = (G_+ \cup G_- \cup G_\tau, M \cup \{w\}, I_+ \cup I_- \cup I_\tau \cup G_+ \times \{w\})$ is called the *classification context.*

We will now look at the definition of a positive hypothesis, this definition was taken from [14].

Definition 4.1.3. Consider a positive context $K_+ = (G_+, M, I_+)$ and a negative context $K_- = (G_-, M, I_-)$. A pair (e_+, i_+) is a positive concept if it is a concept of the context K_+. If intent i_+ of a positive concept (e_+, i_+) is not found in the intent of any negative concept i.e., $(\forall g_- \in G_-, i_+ \nsubseteq \{g_-\}')$, then the concept (e_+, i_+) is called a positive hypothesis with respect to the property w. Negative hypotheses are defined in a similar manner.

From the above definition, we can deduce that hypotheses are very critical in the classification of undetermined examples from the object set G_τ to predict whether it has the goal attribute w or not. If an undetermined example $g_\tau \in G_\tau$ has all the attributes from the intent of i_+ then g_τ is classified positively. A negative classification is defined in a similar way. If $\{g_\tau\}^\tau$ contains both positive and negative hypotheses or does not include any of the hypotheses, then no classification is made. This classification is said to be contradictory. We can also have a subset of hypotheses which is equivalent to the set of all hypotheses with respect to all the possible classifications. We can define a minimal positive hypothesis i_+ as a positive hypothesis if no subset $i \subset i_+$ of it is a positive hypothesis [15].

We will consider the example in Example 3.2.5. We will have the positive context having objects $2, 5, 8, 14$, our negative context with objects $1, 3, 7$ and our undetermined context $18, 19, 20$.

$$\begin{array}{l|cccccccccc} chains & SS & SQ & SR & MB & MF & con & snow & ice & dur & grade \\ 2 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 5 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 8 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 14 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{array}$$

Figure 4.1: Positive context

$$\begin{array}{l|cccccccccc} chains & SS & SQ & SR & MB & MF & con & snow & ice & dur & grade \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 3 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 17 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{array}$$

Figure 4.2: Negative context

$$\begin{array}{l|cccccccccc} chains & SS & SQ & SR & MB & MF & con & snow & ice & dur & grade \\ 18 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 19 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 20 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{array}$$

Figure 4.3: Undetermined context

From the above contexts we decided to pick the high cost of chains as our goal attribute that is to say, positive samples conform to expensive chains since they all have the minimal positive properties $\{MB, con, dur\}$ whereas negative samples conform to cheap chains since they also have the negative properties $\{SR, MF\}$. In regards to the classification of the undetermined samples, sample 18 was classified positively since it contained the minimal positive properties or hypothesis and there was none in favor for a negative classification. Sample 19 was also classified negatively since it contained the minimal negative hypothesis and there was no hypothesis conforming to a positive classification. Sample 20 could not be classified since the hypothesis did not fall under any of the classes defined above and hence the classification is said to be contradictory.

4.2 Algebraic Formulation for Classification

So far, we have been able to define both positive and negative hypotheses. In this section we will go over some algebraic formulas pertaining to deriving our respective hypotheses.

In order to model the global decision attribute we assume that the attributes are of the form $M + 1$ where M are the regular attributes and the additional 1 models the global decision attribute.

Before we define our algebraic formulations, we will represent the complement of $\overline{I}$ as a shorthand for $I \rightarrow \perp\!\!\!\perp$.

We will begin by defining the prerequisites to our equation.

Definition 4.2.1. Let I be a relation, then I^+ and I^- are given as:

1. $I^+ = I$
2. $I^- = (I; \iota^{\smile}; \iota) \cup ((\overline{I; \kappa^{\smile}}); \kappa)$

Definition 4.2.2. Given two relations that is I^+ and I^-. We can derive I^+_{M+1}. and I^-_{M+1} as:

1. $I^+ = I^+_{M+1} = syq(I^{+\smile}/(\varepsilon \backslash I^{+\smile}), \varepsilon)^{\downarrow} \cap \mathbb{I}$
2. $I^- = I^-_{M+1} = syq(I^{-\smile}/(\varepsilon \backslash I^{-\smile}), \varepsilon)^{\downarrow} \cap \mathbb{I}$

Now c^+ splits I^+ and c^- splits I^-.

Finally we define our positive and negative hypotheses relational algebraically.

Definition 4.2.3. Consider the following relational equations;

$$e^+ = syq(v^{\smile}, \varepsilon_M)^{\downarrow};\ e^{\smile}_M;\ (syq(\varepsilon_M,\ \iota; \varepsilon_{M+1})^{\downarrow} \cap\ \mathbb{T}_{\mathcal{P}_L(M)1}; \kappa; \varepsilon_{M+1}); c^{+\smile}; \mathbb{T}_{C1}\colon 1 \rightarrow 1$$
$$e^- = syq(v^{\smile}, \varepsilon_M)^{\downarrow};\ e^{\smile}_M;\ (syq(\varepsilon_M,\ \iota; \varepsilon_{M+1})^{\downarrow} \cap\ \mathbb{T}_{\mathcal{P}_L(M)1}; \kappa; \varepsilon_{M+1}); c^{-\smile}; \mathbb{T}_{C1}\colon 1 \rightarrow 1$$

Where: v = the input vector relation (the undetermined example)
$\varepsilon_M, \varepsilon_{M+1}$ = the element relations between M and $\mathcal{P}(M)$ resp $M+1$ and $\mathcal{P}(M+1)$
e_M = The relation e giving as: $\varepsilon \backslash \varepsilon$

Both relations e^+ and e^- are relations between 1 and 1, i.e., 1×1 matrices in the concrete case. Therefore, we can take their only entry as the degree to which there is a positive (negative) hypothesis.

e^+ compares the attributes of the given example v with a smaller attribute set (up to a certain degree) that has the decision attribute and is a concept. The supremum of all these degrees is then computed. Similar e^- compares the attributes of the given example v with any smaller attribute set (up to a certain degree) that does not have a decision attribute but is a concept.

From the definition above, The equation e^+ is used to test for a positive classification and e^- is used to test for a negative classification. So for instance, if an undetermined sample $g_\tau \in G_\tau$ has all the attributes from the intent of the positive examples, then it is classified positively with a degree 1 and negatively with a degree 0. A negative classification occurs with a degree 1 when the said sample has all the attributes from the intent of negative examples and a degree 0 otherwise. A contradiction like sample 20 in Example 3.2.5 occurs when the classification of $\{g_\tau\}^\tau$ is classified having both a positive and a negative hypothesis with a degree 1 or 0. A positive hypothesis with a degree 1 denotes that the sample has exactly all the attributes pertaining to a positive classification and a degree of 0 when it does not have any attribute leading to a positive classification. Negative hypothesis with their appropriate degrees are defined in a similar way.

In the fuzzy case, we could have a situation where an object(undetermined sample) could be classified as a positive hypothesis to a certain degree a and a negative hypothesis to a certain degree b. This normally means that it possesses attributes of both a negative and a positive concept to certain degrees. In this case we look at the two classifications and determine the greatest among them. The sample is then classified positively with a degree of a iff $a > b$ and negatively otherwise.

Chapter 5

Implementation

In this chapter, we will discuss the implementation of the abstract algebraic framework for $\mathcal{L}$-fuzzy relations. The implementation was realised with the use of the Java programming language. We will first of all, give a general overview of the language and then dive further into the library which was designed to help in our implementation of our classification framework and then finally our implementation itself which actually performs the classification.

5.1 Java

Java is a general-purpose, class-based object-oriented language. This language was designed purposely to be as simple as possible for programmers to gain enough fluency in it [8]. Just like C and $C{+}{+}$, Java has features similar to them but the organization of the language is rather made differently. It is easy to create and use classes in this language, hence our choice to use this language for our research. An example is our *Relation* class which represents relations and implements all of the relational operations.

Java is a strong and statically typed language. This means that every single line of expression has to be prefixed with a data type which is to be determined at compile time. The compile time consists of converting the programs into a machine-readable byte code and the run-time processes involves loading and linking classes which are needed for a program to be executed, for optional machine code generation and dynamic optimization of the program, and actual program execution [7]. However, if there is an explicit data type missing, an error points it out to a user of the language.

In addition to the above, Java has an automatic storage management, which is typically known as garbage collection to avoid safety problems of explicit deallocation.

(as in C++'s delete keyword). The language does not contain any unsafe constructs, such as array accesses without any proper indexing, since this issue would cause a program to behave in an undesignated way [8]. The current version of the language is Java 12 which contains a lot of helper functions to aid in this work as well.

5.2 Implementation of $\mathcal{L}$-Fuzzy Relation Library

In this section we will talk about the class *Lattice* and the various functions or methods within, and then we will continue to the class *Relation*.

5.2.1 Class Lattice

Since we know the lattice denotes the membership degrees of the elements, we defined it is an abstract class that contains the main Lattice methods and components which is extended by the various forms of Lattices we implemented.

```java
/**
 *This method returns the bottom element of the lattice.
 *By Default, 0 is the
 *value element of any lattice.
 *@return return bottom element.
 */
public abstract int GetZero();

/**
 *This method returns the top element of lattice.
 *By Default, 1 is the value element of any lattice.
 *@return return top element.
 */
public abstract int GetOne();

/**
 *This method determines the meet of two elements.In other words,
 *it returns the greatest element from the set of elements where
 *each element of that set is smaller than both input variables.
 *@param elem1  the first element of meet operation.
 *@param elem2  the second element of meet operation.
 *@return the greatest lower bound of elem1 and elem2.
 */
public abstract int Meet(int elem1, int elem2);

```

```
  /**
   *This method determines the join or supremum of two elements.In
   *other words, it returns the least element from
   *the set of elements where each element of that set is
   *greater than both input variables.
   *@param elem1 the first element of join operation.
   *@param elem2 the second element of join operation.
   *@return the least greater bound of elem1 and elem2.
   */
  public abstract int Join(int elem1, int elem2);

  /**
   *This method determines the greatest element (x)
   *such that the meet of x and elem1 is less or equal to elem2.
   *Here, x can be any element from lattice.
   *@param elem1 the first element of implication operation.
   *@param elem2 the second element of implication operation.
   *@return implication of elem1 and elem2.
   */
  public abstract int Implication(int elem1, int elem2);
```

From the sample code above, we note the presence of the top element denoted in the function as GetOne() and the bottom element denoted as GetZero(). We also have the meet, join and pseudo-complement(implication) methods.

5.2.2 Relation Class

This class consists of the relational operations as well as the fuzzy concept operations. These two forms of operations both have their individual interfaces which were implemented in the Relation class but our focus will be on the main operations since all the heavy lifting is done in this class. This class creates the relation structure or matrix with the help of a 2D Array with the row and column determining the size of the matrix.

Relational Operations

The relational operations along with their descriptions are given as:

```
/**
   *This method determines the union of two relations with
   *same dimension. Every element of one relation is compared
   *with the corresponding elementfrom the other relation and
   *if one of them is true then it writes true in the output
```

```
 *relation.
 *@param relation is one of the input relation for union method.
 *The other input relation accessed using this operator in java.
 *@return union of two relation.
 */
@Override
public Relation union(Relation relation) {

}

/**
 *This method determines the intersection of two relation with
 *same dimension. Every element of one relation is compared with
 *the corresponding element from the other relation and if both
 *of them is true then it writes true in the output relation.
 *@param relation is one of the input relation for intersection
 *method. The other input relation accessed using this operator
 *in java.
 *@return intersection of two relation
 */
public Relation intersection(Relation relation) {

}



/**
 *This method determines the implication of two relation with
 *same dimension. Every element of one relation is compared with
 *the corresponding element from the other relation. If the
 *implication is true then it writes true in the output relation.
 *@param relation is one of the input relation for
 *implication method.
 *The other input relation accessed using this operator in java.
 *@return implication of two relation
 */
public Relation implication(Relation relation) {

}

/**
 *This method calculates the relative multiplication of two
 *relation. The (x,z) element would be true in the resulting
```

```
   *relation if and only if there exists one element y such that
   *(x,y) is true in the first relation and (y,z) is true in
   *the other relation.
   * @param relation one of the input relation for composition
   *method. The other input relation accessed using this
   *operator in java.
   *@return the composition of two relation.
   */
  public Relation composition(Relation relation) {

  }

  /**
   *This method calculates the right residual of two relation
   *R and S (R\S) which is the greatest of all relation X with
   *(R.X) is the subset of S.
   *@param relation one of the input relation for right residual
   *method. The other input relation accessed using this operator
   *in java.
   *@return the left residue of two relation.
   */
  public Relation rightResidue(Relation relation) {

  }

  /**
   *This method calculates the left residual of two relation
   *R and S (R/S) which is the greatest of all relation X
   *with (X.R) is the subset of S.
   *@param relation one of the input relation for left method.
   *The other input relation accessed using this operator
   *in java.
   *@return right residue of two relation.
   */
  public Relation leftResidue(Relation relation) {

  }





  /**
```

```
 *This method creates an entry for each subset. For example,
 *there are a total of 8 subsets for 3 elements.
 *Then this method creates a relation with dimension 8 by 3
 *where row represents each of the subset and column represents
 *each object. Empty subset contain 0 for every column while
 *subset with one element contain 1 in the column which
 *represents the element in the subset.
 * @param element no of elements in the set.
 * @return the epsilon relation for the input.
 */
public static Relation epsilon(int element) {

}

/**
 *This method calculates the intersection of left residue
 *and right residue to two relation R and S. Let define
 *R:X-Y and S:X-Z. Then the symmetric quotient relates an
 *element y from Y to an element z from Z when y and z
 *have the same set of inverse image with respect
 *to R and S respectively.
 *@param relation one of the input relation for
 *symetricQuotient method. The other input
 *relation accessed using this operator in java.
 *@return symmetric quotient of two relation.
 */
public Relation symetricQuotient(Relation relation) {

}

/**
 *This static method returns an identity relation
 *i.e. a square relation with ones on the main
 *diagonal and zero elsewhere.
 *@param size the dimension of matrix. This size
 *is set as the row and column of the identity relation.
 *@return the identity relation with specified dimension.
 */
public static Relation identity(int size) {

}

/**
```

```
    *This method flips the relation, that is it
    *switches the row and column by
    *producing another relation.
    *@return it returns transposed relation
    */
   public Relation transpose() {

   }


   /**
    * This method create a relation with 1 and 0.
    *The relation dimension depends on the input parameters.
    *For example, if m and n two parameters in the function
    *then relation consist of (m) rows and (m+n) columns.
    *The result relation contains 1 for those indexes where
    *row and column represents same element
    *from m and 0 elsewhere.
    *@param m the first parameter.
    *@param n the second parameter.
    *@return the iota relation consist of 1 and 0.
    */
   public static Relation Iota(int m, int n) {

   }

   /**
    *This method create a relation with 1 and 0.
    *The relation dimension depends on the input parameters.
    *For example, if m and n two parameters in the function
    *then relation consist of (n) rows and (m+n) columns.
    *The result relation contains 1 for those indexes where
    *row and column represents same element from n and 0
    *elsewhere.
    *@param m the first parameter.
    *@param n the second parameter.
    *@return the kappa relation consist of 1 and 0.
    */
   public static Relation Kappa(int m, int n) {

   }

   /**
```

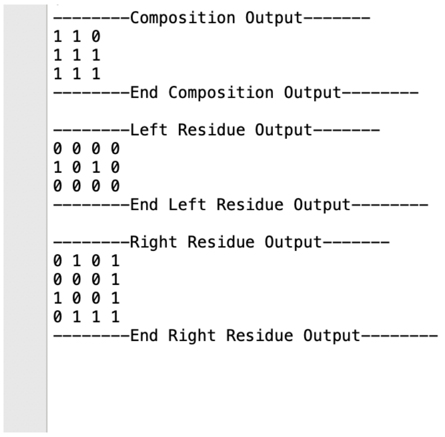
```
     *This method create a relation with 1 and 0.
     *The relation dimension depends on the input parameters.
     *For example, if m and n two parameters in the function
     *then relation consist of (m*n) rows and (m) columns.
     *The result relation contains 1 for those indexes
     *which represents the row element and 0 elsewhere.
     *@param m the first parameter.
     *@param n the second parameter.
     *@return the pi relation consist of 1 and 0.
     */
    public static Relation pi(int m, int n) {

    }

    /**
     *This method create a relation with 1 and 0.
     *The relation dimension depends on the input parameters.
     *For example, if m and n two parameters in the function
     *then relation consist of (m*n) rows and (n) columns.
     *The result relation contains 1 for those indexes which
     *represents the row element and 0 elsewhere.
     *@param m  the first parameter.
     *@param n the second parameter.
     *@return the rho relation consist of 1 and 0.
     */
    public static Relation rho(int m, int n) {

    }

    /**
     *This function takes relation as a input.
     *If the relation entry getter than 0 then
     *this makes the entry 1 otherwise keep the same.
     *@return modified matrix with 1 and 0 only
     */
    public Relation up() {

    }

    /**
     *This function takes relation as a input.
     *If the relation entry equal to 1 then this
     *makes the entry 1 otherwise keep the same.
     *@return modified matrix with 1 and 0 only
     */
    public Relation down() {

    }

    /**
     *This function removes all rows in a relation
     *that whose entry does not contain a 1.
     *It relies on the removeZero functions to take out
     *all rows with zero as their only entry
     *This is also refered to as the split function in
     *the previous chapter example - C_f splits I_f
     *@return modified matrix with rows containing 1 only
     */
    public Relation getEquivalence() {

    }
```

We will now look at some examples of implementations of the relational operations. In the following examples we will be using a Boolean lattice (two element lattice that consists of 1s and 0s.

```


//2D array to hold the elements of R
int [][] R = {{1,0,1,0},
              {1,1,0,1},
              {0,1,1,0}};

//2D array to hold the elements of S
int [][] S = {{1,1,0},
              {0,1,1},
              {1,0,0},
              {1,1,1}};
//2D array to hold the elements of Q
int [][] Q = {{1,1,0,1},
              {0,1,1,1},
              {1,0,0,1},
              {1,1,1,0}};



//This is our relations R,S and Q which is set
//by the elements in the 2D array above
```

```
//The lattice value we are using for this example
//is a two element lattice( or Boolean)
//hence the name of the class BoolLattice
Relation relationR = new Relation(3, 4, new BoolLattice());
relationR.setMatrix(R);

Relation relationS = new Relation(4, 3, new BoolLattice());
relationS.setMatrix(S);

Relation relationQ = new Relation(4, 4, new BoolLattice());
relationQ.setMatrix(Q);


//The result for finding the composition of R;S
Relation result1 = relationR.composition(relationS);
// The result for finding the left residual of R\S
Relation result2 = relationR.leftResidue(relationQ);
// The result for finding the left residual of R/S
Relation result3 = relationR.rightResidue(relationQ);


//This this a function that prints the results
//The first argument outputs the results to the screen
//The second argument lables the results for us
Utility.PrintArray(result1, "Composition Output");
Utility.PrintArray(result2, "Left Residue Output");
Utility.PrintArray(result3, "Right Residue Output");
```

The output of the above code is given in Figure 5.1 below



```
--------Composition Output-------
1 1 0
1 1 1
1 1 1
--------End Composition Output--------

--------Left Residue Output-------
0 0 0 0
1 0 1 0
0 0 0 0
--------End Left Residue Output--------

--------Right Residue Output-------
0 1 0 1
0 0 0 1
1 0 0 1
0 1 1 1
--------End Right Residue Output--------
```

Figure 5.1: Output for relational example implementation

$\mathcal{L}$-Fuzzy Concept Operations

For the implementation of the concept generation we designed the following method:

```java
/**
   *This method derives all concepts from a relational data set.
   *@return it returns all concepts in the form of matrix
   *where true represents concept.
   */
  private Relation generateConcepts() {

  }


/**
   *This method derives the object concepts from
   *a relational data set.
   *@return it returns all object concepts in the
   *form of matrix where true represent concept.
   */
  public Relation generateObjectConcepts() {

  }


/**
   *This method derives the attribute concepts from a
   *relational data set.
   *@return it returns all attribute concepts in the
   *form of matrix where true represent concept.
   */
  public Relation generateAttributeConcepts() {

  }
```

The methods in the previous page follow the algebraic formula which can be found in Lemma 3.2.4. From the generateConcepts method, we derived two other methods, one which was used to generate only the object concepts (i_G) and the other function to generate only the attribute concepts (i_M). The attribute concepts were generated by finding the transpose of the relation.

5.2.3 Algebraic Implementation for Classification

Finally, we introduce the algebraic formulation and method for the purpose of our classification implementation for $\mathcal{L}$–fuzzy relations as ePositive(e^{+}) and eNegative(e^{-}). Their algebraic formulas are given as follows:

$$e^{+} = syq(v^{\smile}, \varepsilon_M)^{\downarrow}; e_M^{\smile}; (syq(\varepsilon_M, \iota; \varepsilon_{M+1})^{\downarrow} \cap \pi_{\mathcal{P}_L(M)1}; \kappa; \varepsilon_{M+1}); c^{+\smile}; \pi_{C1}\colon 1 \to 1$$
$$e^{-} = syq(v^{\smile}, \varepsilon_M)^{\downarrow}; e_M^{\smile}; (syq(\varepsilon_M, \iota; \varepsilon_{M+1})^{\downarrow} \cap \pi_{\mathcal{P}_L(M)1}; \kappa; \varepsilon_{M+1}); c^{-\smile}; \pi_{C1}\colon 1 \to 1$$

For the classification to work, we had to have a group of trained samples namely I^{+} which was denoted as the positive samples and I^{-} which denoted the negative samples. These samples are also referred to as our training set. The implementation for the classification methods are given below:

```java

/**
   *This method takes the trained sample relation as input
   *and returns a true if the input vector relation is
   *found in the positive class and 0 otherwise
   *@param relation one of the input relation for
   *positive classification method. The other input
   *relation accessed using this operator in java.
   *@return class of input vector relation.
   */
public Relation ePositive(Relation r) {

}


/**
   *This method takes the trained sample relation as input
   *and returns a true if the input vector relation is
   *found in the negative class and 0 otherwise
   *@param relation one of the input relation for
   *negative classification method. The other input
   *relation accessed using this operator in java.
   *@return class of input vector relation.
   */
public Relation eNegative(Relation r) {

}
```

We will start by using the example in Example 3.2.5. This example was taken from [15]. The right column has the decision of the objects with 1 signifying the object as a positive sample and a 0 signifying a negative sample. Objects 18,19 and 20 were used as our test samples.

$$\begin{array}{c|ccccccccccc}
chains & SS & SQ & SR & MB & MF & con & snow & ice & dur & grade & decision \\
\hline
2 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\
5 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\
8 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\
14 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
3 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\
17 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0
\end{array}$$

From the figure above, we have objects 2,5,8,14 denoting the positive samples since they have the minimal positive properties $\{MB, con, dur\}$ and objects 1,3,17 denoting the negative samples since they also have the common minimal properties $\{SR, MF\}$. The table below shows our undetermined or test samples which will be used to test our methods.

$$\begin{array}{c|cccccccccc}
chains & SS & SQ & SR & MB & MF & con & snow & ice & dur & grade \\
\hline
18 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\
19 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\
20 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0
\end{array}$$

Using each object as our vector input we came to a conclusion that object 18 was classified positively since it had the minimal positive properties and object 19 was classified negatively since it also had the minimal properties of the negative class. object 20 could not be classified since it did not fall under any of the classes denoted above hence the classification is contradictory.

We will now have an example in the fuzzy case, by classifying objects as either being a fruit or not. For this example we will be using a four element lattice, i.e., a lattice made up of four elements $(1, 3, 2, 0)$ with 1 being the greatest element and 0 being the smallest element. The features include how firm the object is, the form or shape of the object broken into three forms(round, cubic and oval) and how smooth the object feels. Just like the previous example the decisions for the training samples are shown as well with a one signifying a positive classification and a zero showing the object is a negative sample.

In the above figure, we have an apple and kiwi shown as fruits which means they are classified as positive samples. The subsequent examples are classified negatively since they are not fruits.

$$\begin{array}{lcccccc} fruits & firm & FR & FC & FO & smooth & decision \\ apple & 0 & 1 & 0 & 0 & 1 & 1 \\ kiwi & 0 & 0 & 0 & 1 & 3 & 1 \\ egg & 1 & 2 & 0 & 1 & 1 & 0 \\ toycube & 1 & 0 & 1 & 0 & 2 & 0 \end{array}$$

$$\begin{array}{lccccc} fruits & firm & FR & FC & FO & smooth \\ mango & 2 & 0 & 0 & 1 & 1 \\ tennisball & 1 & 1 & 0 & 0 & 2 \end{array}$$

Using the above objects as our vector input, we have mango being classified as a positive sample since it has the minimal positive properties of a fruit whereas the tennis ball was also classified negatively though it was classified as a fruit to a certain degree due to it possessing some features of a fruit.

From the above examples, we realise that our algebraic framework works for both classical and fuzzy relations.

Chapter 6

Conclusion

This chapter summarizes the work that has been completed in this thesis as well as how our current research can be extended in future works.

In this thesis we have introduced an abstract algebraic framework for $\mathcal{L}$-fuzzy relations to help us in the process of classifying objects. Our framework is capable of processing and managing crisp or regular relations, linear ordered as well as incomparable values. These processes were generalized through concepts and properties of a complete Heyting lattice $\mathcal{L}$. As shown in our examples so far, we have realised that every entry in our generated concepts is a fuzzy set. This goes on further to prove that crisp relations are a special case of fuzzy relations which involves membership degrees of either 0 or 1.

Throughout the course of the research we presented some machine learning models from the concept lattice point of view. We also tried to prove each formulation and the processes involved in achieving our framework for the classification process which can be found in Chapter 3 of this work. We also looked at how classification is done using Formal Concept Analysis in Chapter 4. In addition to that, we have developed an implementation of the algebraic formula using the Java programming language. The implementation also includes various categorical structures such as arrow categories. Our research allows to classify any kind of object depending on the criteria given, just as it was shown in some of our examples in Chapter 5 where we applied our formula to classify chains based on how good and durable a chain can be.

We propose that further research can be done on our work to optimize the formula to decrease the computation time. Also, the implementation can be improved upon by allowing the input to be given in a different format than a relation. For example, the input could be an image from which we extract features first before we finally perform the classification. Currently we would have to convert our images into a relation before performing classifications on them.

Bibliography

[1] Rudolf Berghammer and Michael Winter. Decomposition of relations and concept lattices. *Fundamenta Informaticae*, 126(1):37–82, 2013.

[2] VK Finn, MI Zabezhailo, and OM Anshakov. On a computer-oriented formalization of plausible reasoning in f. bacon-js mill's style (main principles and computer experiments). *IFAC Proceedings Volumes*, 16(20):351–363, 1983.

[3] Peter J Freyd and Andre Scedrov. *Categories, allegories*, volume 39. Elsevier, 1990.

[4] Hitoshi Furusawa. A representation theorem for relation algebras: Concepts of scalar relations and point relations. 1998.

[5] Bernhard Ganter and Sergei O Kuznetsov. Formalizing hypotheses with concepts. In *International Conference on Conceptual Structures*, pages 342–356. Springer, 2000.

[6] Bernhard Ganter and Rudolf Wille. *Formal concept analysis: mathematical foundations.* Springer Science & Business Media, 2012.

[7] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. The java language specification.

[8] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java language specification.* Addison-Wesley Professional, 2000.

[9] George Grätzer. *General lattice theory.* Birkhäuser, 2nd edition, 2007.

[10] Anamika Gupta, Naveen Kumar, and Vasudha Bhatnagar. Incremental classification rules based on association rules using formal concept analysis. In *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 11–20. Springer, 2005.

[11] Nation J.B. *Notes on lattice theory.* Un. of hawaii, web draft edition, 2008.

[12] Yasuo Kawahara and Hitoshi Furusawa. Crispness and representation theorem in dedekind categories. 1997.

[13] Leonid Kitainik. *Fuzzy decision procedures with binary relations: towards a unified theory*, volume 13. Springer Science & Business Media, 2012.

[14] Sergei O Kuznetsov. Learning of simple conceptual graphs from positive and negative examples. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 384–391. Springer, 1999.

[15] Sergei O Kuznetsov. Machine learning on the basis of formal concept analysis. *Automation and Remote Control*, 62(10):1543–1564, 2001.

[16] Yong Liu and Xueqing Li. Application of formal concept analysis in association rule mining. In *2017 4th International Conference on Information Science and Control Engineering (ICISCE)*, pages 203–207. IEEE, 2017.

[17] Gunther Schmidt. *Relational mathematics*, volume 132. Cambridge University Press, 2011.

[18] Gunther Schmidt and Thomas Ströhlein. *Relations and graphs: discrete mathematics for computer scientists.* Springer Science & Business Media, 2012.

[19] Prem Kumar Singh and Ch Aswani Kumar. A method for reduction of fuzzy relation in fuzzy formal context. In *International Conference on Mathematical Modelling and Scientific Computation*, pages 343–350. Springer, 2012.

[20] Michael Smithson and Jay Verkuilen. *Fuzzy set theory: Applications in the social sciences.* Number 147. Sage, 2006.

[21] Michael Winter. A new algebraic approach to L-fuzzy relations convenient to study crispness. *Information Sciences*, 139(3-4):233–252, 2001.

[22] Michael Winter. *Goguen categories: a categorical approach to L-fuzzy relations*, volume 25. Springer Science & Business Media, 2007.

[23] Michael Winter. Arrow categories. *Fuzzy Sets and Systems*, 160(20):2893–2909, 2009.

[24] Michael Winter. Type-n arrow categories. In *International Conference on Relational and Algebraic Methods in Computer Science*, pages 307–322. Springer, 2017.

[25] Lotfi A Zadeh. Fuzzy sets. *Information and control*, 8(3):338–353, 1965.