Fuzzy Preference Queries to Relational Databases
Fuzzy Preference Queries to Relational Databases

Olivier Pivert & Patrick Bosc

IRISA/ENSSAT, University of Rennes 1, France

Imperial College Press

www.allitebooks.com
To Ewa (O.P.)
Foreword

Databases are pervasive in our information-focused world. While in many cases, such as corporate databases, they are apparent to the user, in many other cases they are the hidden backbone of applications. This is particularly the case in web applications. Many of the widely used popular web applications make considerable use of databases. This is true for websites such as Amazon.com, Facebook and even Google. As we move into a future world in which there will be even more demand for smart user-friendly computing, the requirement for intelligent manipulation of databases has become extremely important. Successful websites will be required to allow users to interface in what can be called “computing with words”. The concept of a fuzzy set, introduced by L.A. Zadeh of the University of California at Berkeley, provides an ideal framework for the digital representation of words and their associated semantics. It allows for the capturing of the nuances and grayness of human language. A further feature, as the name “fuzzy sets” implies, is that, like database theory, it is a set-based theory. Thus the joining of fuzzy set theory with databases, and particularly relational databases, is very natural. Together they become a very rich framework on which to build the type of intelligent information technologies that will underlie future user-friendly smart applications. The authors of this book, Patrick Bosc and Olivier Pivert, are internationally recognized leaders in the field of fuzzy relational database theory. They have worked for many years in this field and have made many important contributions to its development. They have a deep understanding of the subject matter. Their team at Enssat in Lannion, France, is perhaps the best in the world in fuzzy relational database theory. Over the years I have always been impressed by, and enjoyed, the lucidity and clarity of the authors’ writing style. This book manifests this notable feature. Their presentations have always been,
for the most part, self-contained: they will tell you everything you need from beginning to end. This book displays the same characteristic. In addition to providing the formal theoretical underpinnings, the authors provide clear examples illustrating the ideas.

I enjoyed the book and I thank the authors for the effort put into providing such a pleasant and useful book.

Ronald R. Yager
New York, December 2011
Acknowledgments

The authors wish to thank their colleagues Ludovic Liétard on the one hand, and Allel Hadjali and Grégory Smits on the other hand, for their contributions to Chapters 6 and 10 respectively.
This page intentionally left blank
Contents

Foreword vii

Acknowledgments ix

1. Introduction 1
   1.1 Databases and their Evolution .................. 1
   1.2 Preferences and Fuzzy Sets ..................... 2
   1.3 Overview of the Book ........................... 3

2. Reminders on Relational Databases 5
   2.1 Basic Notions and Vocabulary ................... 5
   2.2 Algebraic Operations ........................... 8
      2.2.1 Set operations ............................ 8
      2.2.2 Relational operations ....................... 11
      2.2.3 Properties ................................. 20
   2.3 An Overview of SQL ............................. 21
      2.3.1 The base block ............................ 22
      2.3.2 Combining base blocks ....................... 25
      2.3.3 Partitioning ................................ 27
      2.3.4 Expressing division and antidiision ........ 29

3. Basic Notions on Fuzzy Sets 31
   3.1 Introduction ................................ 31
   3.2 Definitions and Notations ...................... 33
   3.3 Composition of Fuzzy Sets ...................... 36
3.3.1 Intersection and union of fuzzy sets .......... 36
3.3.2 Difference between fuzzy sets .................. 41
3.3.3 Cartesian product of fuzzy sets .................. 43
3.3.4 Trade-off operators ......................... 44
3.3.5 Nonsymmetric operators .......................... 45
3.4 Inclusions and Implications ........................ 50
3.4.1 Fuzzy implications .......................... 50
3.4.2 Inclusions .......................... 58
3.5 Fuzzy Measures and Integrals ....................... 68
3.5.1 Introduction .......................... 68
3.5.2 Fuzzy measures .......................... 68
3.5.3 Fuzzy integrals .......................... 70
3.6 The Extension Principle .......................... 71
3.7 Fuzzy Quantified Propositions ....................... 73
3.7.1 Fuzzy linguistic quantifiers ..................... 73
3.7.2 Quantified propositions ..................... 73

4. Non-Fuzzy Approaches to Preference Queries: A Brief Overview 77
4.1 Introduction ................................ 77
4.2 Quantitative Approaches ........................ 78
4.2.1 Distances and similarity ..................... 78
4.2.2 Linguistic preferences ........................ 79
4.2.3 Explicit scores attached to entities ............ 80
4.2.4 Top-k queries .......................... 81
4.2.5 Outranking .......................... 81
4.3 Qualitative Approaches .......................... 82
4.3.1 Secondary preference criterion .................. 82
4.3.2 Pareto-order-based approaches ............... 83
4.3.3 CP-nets .......................... 85
4.3.4 Domain linearization ....................... 87
4.3.5 Possibilistic-logic-based approach ............ 88
4.4 Conclusion .......................... 89

5. Simple Fuzzy Queries 91
5.1 Introduction .......................... 91
5.2 An Extended Relational Algebra ............... 93
5.3 An Overview of a Basic Version of SQLf .......... 94
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3.1</td>
<td>Introduction</td>
<td>94</td>
</tr>
<tr>
<td>5.3.2</td>
<td>The multiple relation base block</td>
<td>95</td>
</tr>
<tr>
<td>5.3.3</td>
<td>Subqueries</td>
<td>96</td>
</tr>
<tr>
<td>5.3.4</td>
<td>Set-oriented operators</td>
<td>102</td>
</tr>
<tr>
<td>5.3.5</td>
<td>Relation partitioning</td>
<td>104</td>
</tr>
<tr>
<td>5.4</td>
<td>Interface for User-Defined Terms and Operators</td>
<td>106</td>
</tr>
<tr>
<td>5.5</td>
<td>Contextual Queries</td>
<td>108</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Queries with one level of context</td>
<td>110</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Queries with several levels of context</td>
<td>113</td>
</tr>
<tr>
<td>5.6</td>
<td>Evaluation of Simple Fuzzy Queries</td>
<td>114</td>
</tr>
<tr>
<td>5.6.1</td>
<td>Derivation principle</td>
<td>114</td>
</tr>
<tr>
<td>5.6.2</td>
<td>Derivation-based processing of SQLf queries</td>
<td>121</td>
</tr>
<tr>
<td>5.7</td>
<td>Conclusion</td>
<td>128</td>
</tr>
<tr>
<td>6.1</td>
<td>Introduction</td>
<td>131</td>
</tr>
<tr>
<td>6.2</td>
<td>Quantified Statements</td>
<td>132</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Introduction</td>
<td>132</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Quantified statements and fuzzy integral theory</td>
<td>133</td>
</tr>
<tr>
<td>6.2.3</td>
<td>Interpretation of statements of the type “Q X are A”</td>
<td>138</td>
</tr>
<tr>
<td>6.2.4</td>
<td>Integration into SQLf queries</td>
<td>149</td>
</tr>
<tr>
<td>6.2.5</td>
<td>Evaluation of SQLf queries involving quantified statements</td>
<td>152</td>
</tr>
<tr>
<td>6.3</td>
<td>Aggregates</td>
<td>160</td>
</tr>
<tr>
<td>6.3.1</td>
<td>Introduction</td>
<td>160</td>
</tr>
<tr>
<td>6.3.2</td>
<td>The case of monotonic predicates and aggregates</td>
<td>161</td>
</tr>
<tr>
<td>6.3.3</td>
<td>Dealing with the general case</td>
<td>164</td>
</tr>
<tr>
<td>6.3.4</td>
<td>SQLf queries involving aggregates</td>
<td>169</td>
</tr>
<tr>
<td>6.3.5</td>
<td>Evaluation of SQLf queries involving aggregates</td>
<td>176</td>
</tr>
<tr>
<td>6.4</td>
<td>Conclusion</td>
<td>182</td>
</tr>
</tbody>
</table>
7. Division and Antidivision of Fuzzy Relations 185
   7.1 Introduction ............................................. 185
   7.2 Division of Fuzzy Relations .......................... 186
       7.2.1 Principles ........................................... 186
       7.2.2 On the choice of implication .................... 189
       7.2.3 Primitivity of the extended division operator .......... 190
       7.2.4 Expressing extended division in SQLf ............ 192
   7.3 Tolerant Division ....................................... 193
       7.3.1 Exception-based tolerant division .................. 193
       7.3.2 Resemblance-based tolerant division .............. 196
   7.4 Stratified Division ..................................... 198
       7.4.1 Introduction ...................................... 198
       7.4.2 The queries ....................................... 200
       7.4.3 Quotient property of the result delivered .......... 205
   7.5 Queries Mixing Division and Antidivision .............. 207
       7.5.1 Motivation ......................................... 207
       7.5.2 Mixed stratified queries ......................... 208
   7.6 Evaluation of Division Queries ......................... 211
       7.6.1 Processing the division of fuzzy relations ......... 211
       7.6.2 Processing the tolerant divisions of fuzzy relations .... 213
       7.6.3 Processing the conjunctive stratified division .......... 215
   7.7 Conclusion ............................................... 219

8. Bipolar Fuzzy Queries 221
   8.1 Introduction ............................................ 221
   8.2 Preliminaries .......................................... 222
       8.2.1 About bipolarity .................................. 222
   8.3 Extended Algebraic Operators ......................... 224
       8.3.1 Intersection ....................................... 224
       8.3.2 Union .............................................. 225
       8.3.3 Cartesian product .................................. 226
       8.3.4 Negation .......................................... 226
       8.3.5 Difference ......................................... 232
       8.3.6 Selection .......................................... 235
       8.3.7 Projection ......................................... 237
8.3.8 Join .................................................. 238
8.3.9 Division ............................................. 239
8.4 Implementation Aspects .............................. 246
8.5 Conclusion ............................................. 248

9. Fuzzy Group By ........................................... 251
9.1 Introduction ............................................ 251
9.2 An Extended Group By Clause ......................... 252
  9.2.1 Use of a crisp partition .......................... 252
  9.2.2 Use of a fuzzy partition .......................... 253
9.3 Having Clause .......................................... 255
  9.3.1 Inclusion constraint .............................. 256
  9.3.2 Aggregate_1 \theta aggregate_2 .................... 256
  9.3.3 Aggregate is \psi .................................. 257
9.4 Application to Association Rule Mining ............... 258
  9.4.1 Rules of the type A is \text{L}_i \rightarrow B is \text{L}_i' ........... 259
  9.4.2 Rules of the type A is \text{L} \rightarrow B is \text{L}_i' .......... 261
9.5 Evaluation of a Fuzzy Group By ....................... 262
9.6 Related Work ........................................... 262
  9.6.1 Extended group by ............................... 262
  9.6.2 Fuzzy OLAP ........................................ 263
  9.6.3 Fuzzy database summarization techniques ........ 263
  9.6.4 Mining association rules with SQL ................ 264
9.7 Conclusion ............................................. 264

10. Empty and Plethoric Answers .......................... 267
10.1 Introduction .......................................... 267
10.2 Empty Answer Problem ................................ 268
  10.2.1 Query relaxation ................................ 268
  10.2.2 Relaxation by predicate weakening ............... 269
  10.2.3 Case-based reasoning approach ................... 279
10.3 Plethoric Answer Problem ............................ 287
  10.3.1 Introduction ..................................... 287
  10.3.2 Approach based on predicate strengthening ....... 288
  10.3.3 Approach based on query expansion ............... 294
10.4 Conclusion ............................................. 306
xvi

Fuzzy Preference Queries to Relational Databases

11. Conclusion

Bibliography

Index
Chapter 1

Introduction

1.1 Databases and their Evolution

Databases cover the domain of computer science dedicated to the management of huge quantities of permanent data. A database represents the computerized image of a given universe (e.g., sales, deployed products, employees of a company, etc.). Its main use is the retrieval of information from the data stored inside it, but this implies updates to be performed as changes happen in the real world to make the database as faithful to reality as possible. The birth of databases can be situated in the sixties as a response to the growing need for the automation of various data management tasks in the industry, for overcoming the limitations of both manual procedures and/or the use of file structures associated with dedicated programs. The first commercial systems were simple, fairly empirical (i.e., without any formal basis) and their efficiency was highly dependent on programming skills. However, they had the merit of providing developers with more advanced tools than those previously available.

In 1970, E.F. Codd, a researcher from IBM Research Labs, proposed the foundations of a formal approach to database management systems (DBMSs) relying on the mathematical concept of relation. In his view, it was possible to express queries in a high-level language, namely relational algebra, leading to formulations much more compact than traditional programs. This characteristic remained in the SQL language designed in the middle of the seventies, such that the user has to focus on what he or she is interested in, not on how to get it, thus hiding the programming aspects to a large extent. This tendency, aimed at offering high-level concepts rather than interacting with computers through basic programming languages, is not specific to databases, and has widely developed
in many other computer science areas, thus contributing to the definition of “intelligent systems”. As to relational database management systems, the challenge resides in their ability to translate “what” into “how” efficiently, in other words to produce querying programs running sufficiently fast compared to ad hoc programs. This issue was controversial for a while, but finally, technological evolutions (both of processors and disks) made the performances reached by relational systems credible; relational DBMSs became the standard and have been widely deployed since the mid-eighties.

Initially motivated by business applications, databases have spread to new domains such as CAD, office automation, and geographic systems, which has led to extensions to the model in order to capture richer structures than “simple” relational tables. Object orientation has been an important step, and database systems have incorporated facilities to deal with objects, rather than relations, since the end of the nineties. Many other evolutions have occurred, for instance data warehouses and data marts to support decision applications based on “online analytical processing” (OLAP). Here the underlying model is a multi-dimensional one and related operations apply to a hypercube of data, in particular aggregations over dimensions. But once again, relational systems are at the heart of this technology since they serve to feed and to interface data warehouses. One of most recent trends is the accounting for web data with XML models and the development of associated query languages.

1.2 Preferences and Fuzzy Sets

It is worth noting that, in most research works and commercial database management systems, it is assumed that the aim of a query is the delivery of a (multi-)set of answers where all elements are implicitly “equally” satisfactory. Behind such a view is the use of Boolean predicates and operators supporting the notions of acceptance and rejection. Except for some pioneering works, the idea of preference queries has received attention only recently in the database community. Various approaches have been proposed, but most often the objective of efficient implementation constrains the functionalities the user is provided with. For instance, specifying preferences is considered a complementary step after some Boolean selection, and/or is restricted to the selection operation. In contrast, the suggestion advocated in this book is to envisage preferences as a central concern in
database queries and to call on fuzzy sets, a concept initially introduced by L.A. Zadeh in 1965, instead of regular sets. By its very nature, a fuzzy set is strongly related to preferences, inasmuch as a membership function is nothing but the description of the values that are more or less acceptable regarding a given concept, i.e., more or less preferred values. By doing so, a sound foundation of queries is obtained and preferences may appear wherever it makes sense, thus designing a fully fledged preference query language. The key idea is that of graduality, i.e., the fact that a given statement is more or less true, which naturally extends the usual Boolean framework of truth/falsehood. This point of view somewhat differs from that of the binary relations used in Pareto-based approaches (e.g., Skyline queries) where only a partial order is provided which leads to incomparable pairs of elements/answers. In addition, fuzzy predicates are generally expressed as natural language expressions and, as such, they convey a "human touch" to a query language. Last, the fuzzy-set-based framework makes it possible to deal with two levels of preferences, since one can assign different importances to fuzzy predicates intervening in a given expression. In this respect, such languages have a greater expression power as they go far beyond what Boolean languages offer.

1.3 Overview of the Book

Numerous aspects of preferences are tackled in this book, after some reminders on databases (Chapter 2) and a presentation of some basic notions about fuzzy sets (Chapter 3) that are necessary for the forthcoming chapters. We should emphasize that, although presented in the context of relational databases, the topics tackled could apply to other data models (especially object-oriented ones).

The two main families of approaches to preferences in database queries are reviewed in Chapter 4: those based on scores, called quantitative, and those relying on preference relations, called qualitative.

Chapters 5 and 6 are devoted to a presentation of an extended relational algebra on the one hand, and SQLf — an SQL-like language offering a wide range of querying capabilities including preference features — on the other hand. It is shown how the various concepts of SQL (the base block as well as nesting and partitioning) can be enhanced to incorporate preferences at the limited expense of defining fuzzy terms (mainly predicates, binary operators, and quantifiers used in a query).
Chapter 7 is entirely dedicated to the division operation which has the distinction of being an operator with very rich semantics. Its original version allows for the retrieval, inside a relation called dividend, of the values associated with all the elements of a given set (the divisor). Several basic cases of extension can be imagined, among which are those defining the division: i) when the dividend and/or divisor relations involve preferences over the tuples, ii) in a tolerant way if we look for the values associated with “almost all” of the values of the divisor, iii) when the domain common to the dividend and the divisor is provided with a similarity relation, iv) in the presence of a divisor made up of a hierarchy of subsets.

Next, the concept of bipolarity comes into play. Chapter 8 deals with fuzzy bipolar queries in the sense that a query involves two (possibly fuzzy) components, one describing a constraint and another where the condition plays the role of a wish, as in the query “I am interested in a low-consuming car around $9,000, if possible grayish with good comfort”. The basic interpretation principle relies on the fact that the two poles are treated separately and never aggregated; it is shown that the relational algebra can be extended to handle such queries.

Chapter 9 is about an extension of the “group by” clause present in SQL in order to define a fuzzy partition of the tuples instead of a crisp one. Beyond the interest of this feature for end-user queries, it turns out that it makes it possible to write queries intended for mining association rules “on demand”.

The issue of empty and over-abundant answers to fuzzy preference queries is at the heart of Chapter 10. It may seem reasonable to consider empty answers on the one hand, and over-abundant ones on the other hand, as dual problems. With this point of departure, dual transformations of the queries (in terms of modification of the involved fuzzy predicates) lead to dual solutions. Nevertheless, it is also shown that some specific solution to each of these problems can be devised which breaks their apparent duality.

The conclusion part (Chapter 11) summarizes and highlights the principal contributions of the book and discusses some directions for future work, for instance preference queries in the presence of imperfect data.
Chapter 2

Reminders on Relational Databases

This chapter is mainly devoted to recalling the notions of the relational model of data which are essential for understanding most of the aspects dealt with later in this book. It may be omitted by readers who are familiar with the domain, but the reader who would like to go deeper is encouraged to refer to Date (1995), Korth and Silberschatz (1998), and Ullman (1989).

The data model grounded in the notion of relation is first described, then the algebraic operators which constitute the basis for manipulating relations are presented. The next subsection is devoted to querying relational databases using the SQL language, which is available in commercial DBMSs and used by a large number of application developers.

2.1 Basic Notions and Vocabulary

The relational model, proposed by E.F. Codd in 1970 (Codd, 1970), has its foundations in the mathematical theory of relations. The principle is to consider a database as a set of relations, each of which describes a part of the underlying universe (personnel of an enterprise, real-estate transactions, scientific observations, management of a stock of components, etc.). In this framework, a relation $r$ corresponds to a subset of a Cartesian product of possibly non-distinct domains $D_i$, $i \in [1, n]$, therefore:

$$r \subseteq D_1 \times \cdots \times D_n.$$  

This subset is called the extension of relation $r$. From a concrete point of view, a relation gathers the associations of values (tuples) describing the elements of the considered universe. For instance, a relation about services (or departments) of a company with an identifying number, a name, and
a budget, includes only triples corresponding to the actual services of the company, like (149, sales, 20.5).

The schema (also called intention) $R$ of a relation $r$ is defined over a set of distinct attributes $A_j$ associated with each of the domains. Every attribute $A_k$ represents the role played by its corresponding domain and $R(A_1, \ldots, A_n)$ denotes a relation schema. For example, one may have the attributes l-city and place built over the domain of cities, the former (resp. latter) standing for the city where an employee lives (resp. the city where a company is located). Two different attributes are not a priori comparable, except if they share the same domain, in which case these attributes are said to be compatible. Such attributes may then appear in comparison operations (equality/inequality, or even superiority if their underlying domain is provided with an order relation; see the selection and join operations in Section 2.2).

The number of attributes of a relation is called its degree and its number of tuples (elements) is its cardinality (as for sets). Any relation involves at least one identifier called a key, i.e., a minimal set of attributes allowing for distinguishing among any two tuples. Such a set does exist, since the whole set of attributes is a potential (perhaps not minimal) identifier due to the fact that a relation does not contain duplicates. One of the principal interests of the notion of key is from the query-processing point of view, since a key may serve to build an index, thus providing efficient access to tuples (provided that the key value is known as an input).

Example 2.1. Consider the schema $S(sid, ss-number, name, birth-date)$ about students of a university with a number given by the university ($sid$), a national social security number ($ss-number$), a family name ($name$), and a date of birth ($birth-date$). It may be reasonably assumed that a relation built over $S$ has two keys provided that student IDs and social security numbers are uniquely assigned to students. A more sophisticated case is the following, where the schema $ZC(zcode, city, street)$ is considered for the description of zip codes in a given country. No isolated attribute constitutes a key, but there are possibly two keys: {city, street} on the one hand, if it is assumed that a given street in a given city is not “split” over several codes, and {zcode, street} on the other hand.

Usually, a relation is represented as a bi-dimensional array (or table) whose columns are the attributes and whose lines are the tuples of the considered extension. The order of both lines and columns does not matter, since the notion of set (of attributes and tuples here) does not convey any order over the elements.
As can be seen, there are only a few concepts in the relational model, which emphasizes its simplicity and ease to learn. However, this also reflects some semantic poorness, mainly because in this model, everything is either an attribute or a relation. In particular, no distinction is made between the representation of an entity (or object) of the real world with an intrinsic existence (e.g., employee, service, customer, vehicle) and that of an association between several entities (e.g., an order connecting a customer and a product) whose existence depends on that of the referenced entities. Thus, the assignment of an employee to a given service supposes the prior existence of that employee and that service. This is also the case for the purchase of a car by a customer. Indeed, complementary concepts do exist, among them the important notion of integrity constraints, which represent properties of the data that must be enforced both at their creation and when updates are performed. From a practical point of view, in most of the DBMSs, these can be checked through “triggers”, i.e., programs which are activated when needed. In practice, the analysis aiming at building up a database relies on a richer semantic model (typically the entity–relationship model) that offers the tools appropriate for modeling the concepts and properties of interest. In this approach, the relational data model is the target of a translation of the schema thus designed.

**Example 2.2.** Consider a company willing to have an employee database. Each employee is described by an identifier (e-id), his/her name (e-name), surname, birth and living cities, as well as the department of the company he/she works for. Then, the schema of the considered relation is \( \text{EMPLOYEE}(e\text{-id}, e\text{-name}, \text{surname}, \text{birth-city, living-city, work-serv}) \) whose key is e-id, where the attributes birth-city and living-city share the same domain, namely names of cities. An extension of this relation is given in Table 2.1; each line (tuple) describes the data related to a given employee of the company. Adding a relation with schema \( \text{SERVICE}(s\text{-id, s-name, ...}) \)
Fuzzy Preference Queries to Relational Databases

budget), whose key is s-id, will enable the name and budget of each service of the company to be taken into account. We remark that although similarly named, e-name and s-name are not compatible attributes, in contrast to s-id (in relation ser) and work-serv (in relation emp).

2.2 Algebraic Operations

As was seen before, the relational model has a double characteristic, since a relation is a set (of tuples) on the one hand, and the elements of this set are structured on the other hand. As a consequence, operators allowing for the manipulation of relations take advantage of both aspects. The relational algebra constitutes the formal basis for manipulating relations, although it is not the language in use from a practical point of view (see Section 2.3 devoted to the SQL language). This algebra is made of set-oriented and relational operators which have relations as inputs and which also deliver relations as results. So it is a closed calculus system allowing for the composition of operators using expressions such as:

\[ \text{op}_2(\text{op}_1(r), s) \] or \[ \text{op}_1(\text{op}_2(\text{op}_3(r, s), t)) \],

where r, s and t are relations, \( \text{op}_1 \) is a unary operator, and \( \text{op}_2 \) and \( \text{op}_3 \) are binary operators. These operators are presented below, and X, Y and Z stand for sets of attributes appearing in schemas of relations.

2.2.1 Set operations

The union, intersection, and difference of two relations r and s, whose attributes must be pairwise compatible, return a relation with the same schema. They are defined as follows:

- \( \text{union}(r, s) = \{ t | t \in r \text{ or } t \in s \} \),
- \( \text{inter}(r, s) = \{ t | t \in r \text{ and } t \in s \} \),
- \( \text{differ}(r, s) = \{ t | t \in r \text{ and } t \notin s \} \).

The cardinality of the results is such that:

- \( \text{max}(\text{card}(r), \text{card}(s)) \leq \text{card}(\text{union}(r, s)) \leq \text{card}(r) + \text{card}(s) \),
- \( 0 \leq \text{card}(\text{inter}(r, s)) \leq \min(\text{card}(r), \text{card}(s)) \),
- \( 0 \leq \text{card}(\text{differ}(r, s)) \leq \text{card}(r) \).

These operations are illustrated in Figures 2.1, 2.2, and 2.3. The Cartesian product of two relations r and s (unconstrained as to their schemas \( R(X) \))
and $S(Y)$, delivers a relation defined as:

$$\text{cartprod}(r, s) = \{ t = \langle u, v \rangle \mid u \in r \text{ and } v \in s \}.$$

Formally, the schema of the resulting relation is $T(X \cup Y)$ assuming that $X \cap Y = \emptyset$, which can always be achieved using the renaming of attributes if needed. The cardinality of the Cartesian product is known exactly:

$$\text{card} (\text{cartprod}(r, s)) = \text{card}(r) \ast \text{card}(s),$$

which expresses the combinatorial nature of this operator and the computational complexity which may then occur. Figure 2.4 illustrates the principle
 Except for the difference operation, these operations are associative and commutative. Finally, it must be remarked that the difference is useful for answering certain queries involving negations. The answer to this type of query calls on the general scheme:

\[ \text{searched items} = \text{differ(} \text{referential of elements}, \text{unsatisfactory elements}) \]

knowing that these two sets are generally fairly easy to specify. In the previous example, this procedure produces:

\[ \text{differ(} \text{all the known services}, \text{those services where at least one engineer works}) \]
2.2.2 Relational operations

The relational operators exploit the fact that relations possess a structure in terms of attributes. The four most common operators, namely selection, projection, theta-join, and division, are presented in turn.

Selection is a unary operator aimed at extracting a subrelation from an input relation by means of a condition (predicate or constraint) applying to every tuple. Such a condition is a logical expression where elementary conditions are built with constants, comparators and the sole attributes of the concerned relation (which may be either a base relation, i.e., part of the schema of the database, or an intermediate relation resulting from preceding operations). The atomic conditions may be connected by conjunctions and disjunctions, and parentheses are used to force priority if needed. Formally, if \( r \) is a relation with schema \( R(X) \) and \( p \) is a predicate referring to attributes of \( X \), one has:

\[
select(r, p) = \{ t \mid t \in r \text{ and } p(t) \}.
\]
The operating principle of this operation, aimed at keeping only those tuples that comply with the condition, is graphically represented in Figure 2.5.

The second unary operator is projection, whose goal is to retain a subset of desired attributes from those initially present in the input relation. It removes some columns and may be seen as the counterpart of the selection, which discards lines. If $r$ is a relation with schema $R(X)$ and $Z$ is a proper subset of $X$ ($Z \subset X$), the projection of $r$ onto $Z$ is defined as:

$$\text{project}(r, Z) = \{ z | \exists t, \ t \in r \ \text{and} \ t.Z = z \},$$
Fig. 2.5 Principle of the selection operation

Fig. 2.6 Principle of the projection operation

where \( t.Z \) denotes the value of the constituent \( Z \) in tuple \( t \). Figure 2.6 illustrates the functioning principle of the projection. Note that, a relation being a set, the result of a projection does not contain any “duplicate”. So, in the case where two initially distinct tuples share the same value on \( Z \), they collapse and a single tuple is generated in the result.

www.allitebooks.com
Example 2.3. Consider the database composed of the relations \textit{emp} and \textit{ser} with respective schemas EMPLOYEE(\textit{e-id}, \textit{e-name}, \textit{age}, \textit{birth-city}, \textit{living-city}, \textit{work-serv}) with key \textit{p-id}, and SERVICE(\textit{s-id}, \textit{s-name}, \textit{budget}) with key \textit{s-id}, where \textit{work-serv} and \textit{s-id} are compatible attributes. The query aimed at the retrieval of any service (number) whose budget is over $20M or where there is at least one employee less than 35 years old living in his birth city may be written as:

\[
\text{union}(\text{project}(\text{select}(\text{ser}, \text{budget} > 20), \text{s-id}),
\text{project}(\text{select}(\text{emp}, \text{living-city} = \text{birth-city} \text{ and } \text{age} < 35),
\{\text{work-serv}\})).
\]

With the extensions of Tables 2.2 and 2.3, the result is made up of the two values 5 and 9.

The two remaining operators are binary, and correspond to particular (useful) combinations of previously defined operators. Theta-join stems from the Cartesian product, but differs from it by keeping only the pairs of tuples matching a given (joining) property according to the formula:

\[
\text{join}(r, s, \theta, A, B) = \{(u, v) \mid u \in r \text{ and } v \in s \text{ and } \theta(u.A, v.B)\}
= \text{select}(\text{cartprod}(r, s), \theta(A, B))
\]

<table>
<thead>
<tr>
<th>e-id</th>
<th>e-name</th>
<th>age</th>
<th>birth-city</th>
<th>living-city</th>
<th>work-serv</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>Dupond</td>
<td>32</td>
<td>Paris</td>
<td>Nantes</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Mesnard</td>
<td>47</td>
<td>Lyon</td>
<td>St Nazaire</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>Robin</td>
<td>29</td>
<td>Rennes</td>
<td>Rennes</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>Perrin</td>
<td>52</td>
<td>Angers</td>
<td>Angers</td>
<td>10</td>
</tr>
<tr>
<td>12</td>
<td>Marival</td>
<td>34</td>
<td>Lens</td>
<td>Paris</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s-id</th>
<th>s-name</th>
<th>budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>sales</td>
<td>17</td>
</tr>
<tr>
<td>1</td>
<td>manufacturing</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>secretariat</td>
<td>30</td>
</tr>
<tr>
<td>10</td>
<td>marketing</td>
<td>12</td>
</tr>
</tbody>
</table>
where \( R \) (resp. \( S \)), the schema of \( r \) (resp. \( s \)), is defined over \( X \) (resp. \( Y \)), \( A \) and \( B \) are two compatible respective subsets of \( X \) and \( Y \), \( \theta \) is a comparator, and \( \langle u, v \rangle \) denotes the tuple gathering the elements \( u \) and \( v \). This operation, very often used in practice, allows for combining relations with a semantic link conveyed by the presence (in their schema) of one of several attributes sharing the same domains. The principle of the join is illustrated by Figure 2.7. In practice, the comparison operator is often equality, and it

![Fig. 2.7 Principle of the join operation](image-url)
is possible to define a variant called equijoin in the following way:

\[
equijoin(r, s, A, B) = \{ (u, v') \mid u \in r \text{ and } v \in s \text{ and } (u.A = v.B) \text{ and } v' = v.(Y - B) \},
\]

or:

\[
equijoin(r, s, A, B) = \text{project}\left(\text{join}(r, s, A, B, =), X \cup Y - B\right),
\]

where \( X \cap Y = \emptyset \). A side-effect is to avoid generating useless (duplicate) information, since in every tuple of the output relation the \( B \)-value would be equal to the \( A \)-value (with the previous definition). The interest of the equijoin is shown in the next example.

**Example 2.4.** The two relations of Example 2.3 have a semantic link due to the presence of the compatible attributes \( \text{work-serv} \) (in \( \text{emp} \)) and \( \text{s-id} \) (in \( \text{ser} \)). The query to look for any pair \( (e-name, s-name) \) such that \( e-name \) works in \( s-name \) and \( s-name \) corresponds to a service which has a budget less than $15M can be written thanks to an equijoin, as follows:

\[
\text{project}\left(\text{equijoin}(\text{emp}, \text{select}(\text{ser}, \text{budget} < 15), \{\text{work-serv}\}, \{\text{s-id}\}), \{e-name, s-name\}\right).
\]

With the extensions given in Tables 2.2 and 2.3, services numbered 1 and 10 fulfill the budget requirement, and the final result is the binary relation of Table 2.4.

<table>
<thead>
<tr>
<th>Table 2.4: Result of Example 2.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>e-name</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>Dupond</td>
</tr>
<tr>
<td>Mesnard</td>
</tr>
<tr>
<td>Perrin</td>
</tr>
</tbody>
</table>

Several other specific forms of join exist, among which are:

- **self-join**, where the same relation is used twice, which enforces the renaming of the attributes in the resulting relation,
- **natural join**, which corresponds to the equijoin of two relations over the set of attributes having the same \( name \) in both. With the schemas
Reminders on Relational Databases

$R(X, Y)$ and $S(X, Z)$ with $Y \cap Z = \emptyset$, the natural join of $r(R)$ and $s(S)$ is defined as:

$$\text{natjoin}(r, s) = \{(x, y, z) \mid \exists t \in r, \exists u \in s \text{ such that } t.XY = xy \text{ and } u.XZ = xz\},$$

• antijoin, which delivers those tuples of the first relation which match none of those of the second relation, i.e.:

$$\text{antijoin}(r, s, \theta, A, B) = \{u \mid u \in r \text{ and } \nexists v \in s \text{ such that } \theta(u.A, v.B)\},$$

• semijoin, which may be seen as a join followed by a projection over the sole attributes of the first relation, i.e.:

$$\text{semijoin}(r, s, \theta, A, B) = \text{project}(\text{join}(r, s, \theta, A, B)), X$$

$$= \{u \mid u \in r \text{ and } \exists v \in s \text{ such that } \theta(u.A, v.B)\}. $$

The last operator is called division. If $r$ and $s$ are two relations with respective schemas $R(A, X)$ and $S(B, Y)$ where $A$ and $B$ are compatible (sets of) attributes, the division is defined as follows:

$$\text{div}(r, s, A, B) = \{x \mid \forall a, (a \in \text{project}(s, B)) \Rightarrow ((a, x) \in r)\},$$

or equivalently:

$$\text{div}(r, s, A, B) = \{x \mid \text{project}(s, B) \subseteq \Gamma_r^{-1}(x)\},$$

with $\Gamma_r^{-1}(x) = \{a \mid (a, x) \in r\}$. In other words, a value $x$ belongs to the result of the division if and only if it is associated with at least all the values of the (set of) attribute(s) $B$ of relation $s$, as illustrated in Figure 2.8. It is worth noting that the set of attributes $Y$ in relation $s$ does not play any role and can be empty. The justification of the term “division” assigned to this operation relies on the fact that a property similar to that of the quotient of integers holds. Indeed, the resulting relation $\text{res}$ obtained with Expression (2.1) or (2.2) is called a “quotient” and has the double characteristic:

$$\forall t \in \text{res}, \text{cartprod}(s, \{t\}) \subseteq r,$$

$$\forall t \notin \text{res}, \text{cartprod}(s, \{t\}) \nsubseteq r.$$  

**Example 2.5.** Consider the relation $\text{emp}$ given in Table 2.2 and the relation $\text{stay}$ with schema $ST(s\text{-num, emp\text{-num, duration}})$ describing how long
employees have stayed in various services. The query looking for any service where all the employees older than 40 have stayed for more than three months, is expressed as:

$$\text{div}(\text{project}(\text{select(stay, duration \geq 3)}, \{s\text{-num, emp\text{-num}}\}),$$

$$\text{select(emp, age > 40)},$$

$$\{\text{emp\text{-num}}, \{\text{e-id}\}\}).$$

With the extensions of emp and stay given in Tables 2.2 and 2.5, the result of the previous query contains only the service number 5 where both
employees 3 and 7 (who are older than 40) have stayed more than three months. We remark that it is necessary to discard attribute duration (by means of a projection) so as to get the correct result. Otherwise, the result would be a binary relation composed of pairs (service number, duration) such that all the employees older than 40 have stayed for this same duration (over three months).

It can be easily shown that the division may be written as:

\[ \text{div}(r, s, A, B) = \text{differ}(\text{project}(r, X), \text{project}(\text{differ}(r, \text{cartprod}(\text{project}(r, X), \text{project}(s, B))), X)). \]  

(2.3)

This expression can be justified as follows:

\begin{itemize}
  \item the sub-expression \( r_1 = \text{cartprod}(\text{project}(r, X), \text{project}(s, B)) \) computes the pairs \((a, x)\) which should be in relation \(r\) for all \(x\)'s to be qualified,
  \item the sub-expression \( x_1 = \text{project}(\text{differ}(r, r_1), X) \) delivers the values \(x\) for which at least one association with the \(B\)-values of \(s\) is missing in \(r\),
  \item the final difference \( \text{differ}(\text{project}(r, X), x_1) \) keeps the values \(x\) of \(r\) for which no association is missing.
\end{itemize}

Closely related to division is antidivision. This operator aims at selecting the \(X\)-values present in relation \(r\) which are associated with none of the \(B\)-values present in \(s\). It is defined as follows:

\[ \text{antidiv}(r, s, A, B) = \{ x \in \text{project}(r, X) \mid \forall a, (a \in \text{project}(s, B)) \Rightarrow (a, x) \not\in r \}. \]  

(2.4)
or equivalently:

\[
\text{antidiv}(r, s, A, B) = \{ x \in \text{project}(r, X) \mid \text{project}(s, B) \subseteq \text{cp}(r^{-1}(x)) \} ,
\]

\[(2.5)\]

cp(rel) denoting the complement of rel. Here also, relation r (resp. s) is called the dividend (resp. divisor) and the result obtained by Expression (2.4) or (2.5), called an “anti-quotient”, has the property:

\[
\forall t \in \text{res, } \text{cartprod}(s, \{t\}) \subseteq \text{cp}(r),
\]

\[
\forall t \notin \text{res, } \text{cartprod}(s, \{t\}) \not\subseteq \text{cp}(r).
\]

### 2.2.3 Properties

The operators of the relational algebra possess a number of properties. It is, in particular, possible to design a minimal set of operators from which any other one can be built. This is the case with \{union, difference, selection, projection, Cartesian product\}. Then, the intersection can be written as:

\[
\text{inter}(r, s) = \text{differ}(\text{union}(r, s), \text{union}(\text{differ}(r, s), \text{differ}(s, r))) .
\]

Some properties express equivalences between algebraic formulas, for instance due to the distributivity of one operator with respect to another. So, if r and s are two relations with respective schemas R(X) and S(X), one has:

- \(\text{union}(\text{select}(r, p), \text{select}(s, p)) = \text{select}(\text{union}(r, s), p)\),
- \(\text{inter}(\text{select}(r, p), \text{select}(s, p)) = \text{select}(\text{inter}(r, s), p)\),
- \(\text{differ}(\text{select}(r, p), \text{select}(s, p)) = \text{select}(\text{differ}(r, s), p)\).

Moreover, if Y denotes a subset of X, the following properties hold:

- \(\text{union}(\text{project}(r, Y), \text{project}(s, Y)) = \text{project}(\text{union}(r, s), Y)\),
- \(\text{inter}(\text{project}(r, Y), \text{project}(s, Y)) \supseteq \text{project}(\text{inter}(r, s), Y)\),
- \(\text{differ}(\text{project}(r, Y), \text{project}(s, Y)) \subseteq \text{project}(\text{differ}(r, s), Y)\).

The following example shows their validity using a simple case.

**Example 2.6.** Consider the relations r and s with schema (A, B) and their extensions given by:

\[
r = \{(a, b_1)\}, \quad s = \{(a, b_2)\}.
\]

Let \(X = \{A, B\}\) and \(Y = \{A\}\). The projections \(\text{project}(r, Y)\) and \(\text{project}(s, Y)\) yield:

\[
\text{project}(r, Y) = \{a\}; \quad \text{project}(s, Y) = \{a\}.
\]
Moreover:

\[
\text{union}(r, s) = \{(a, b_1), (a, b_2)\}; \quad \text{inter}(r, s) = \emptyset; \quad \text{differ}(r, s) = \{(a, b_1)\}.
\]

The results, corresponding to the left- and right-hand sides of the previous formulas, are synthesized as follows:

\[
\text{union}(\text{project}(r, Y), \text{project}(s, Y)) = \{a\}; \quad \text{project}(\text{union}(r, s), Y) = \{a\}; \\
\text{inter}(\text{project}(r, Y), \text{project}(s, Y)) = \{a\}; \quad \text{project}(\text{inter}(r, s), Y) = \emptyset; \\
\text{differ}(\text{project}(r, Y), \text{project}(s, Y)) = \emptyset; \quad \text{project}(\text{differ}(r, s), Y) = \{a\}.
\]

Finally, we point out the following property concerning selection and join:

\[
\text{join}(\text{select}(r, p), s, \theta, A, B) = \text{select}(\text{join}(r, s, \theta, A, B), p),
\]

where \(p\) is a predicate bearing only on the attributes of relation \(r\).

The interest of these properties resides in the possibility for: i) the user to translate his or her natural language need into different correct expressions, and ii) the DBMS to carry out some transformations for optimization purposes.

### 2.3 An Overview of SQL

The SQL language is the most commonly used tool to manipulate relational databases. It can be used either directly to submit queries, or through programming languages (e.g., C and Java) which provide appropriate interfaces. The foundations of the language were laid in the mid-seventies with SEQUEL, developed in the context of a prototype called System-R (Chamberlin et al., 1976). The set of operations offered by SQL includes the design and modification of the schema of a database, the management of access rights, querying and data updates. It involves extensions with respect to the relational algebra, in particular in order to ease the formulation of complex queries. It is a de facto standard which is found in most (not to say all) commercial DBMSs. The principal components of SQL related to querying capabilities are reviewed in the rest of this section.
2.3.1 The base block

The SQL base block “select … from … where …” is a key element which allows for the expression of several relational operators. One of its commonly used forms is written as:

\[
\text{select} \ [\text{distinct}] \ \text{attributes from relations where condition}
\]

which is interpreted the following way (regardless of the effective processing strategy of any DBMS):

1. compute the Cartesian product of the relations in the “from” clause, which are either base or temporary relations,
2. select the tuples agreeing with the condition of the “where” clause,
3. retain the sole attributes appearing in the “select” clause and remove duplicates if “distinct” is mentioned; when * is used instead of a list of attribute names (star convention), all attributes are kept.

This interpretation scheme “covers” several operators of the relational algebra. Table 2.6 gives the correspondences between algebraic operators and forms of the base block.

Example 2.7. Consider the relations \( \text{emp}(e\text{-id}, e\text{-name}, age, birth\text{-city}, \text{living\text{-city}}, \text{work\text{-serv}}, \text{resp}, \text{salary}) \) and \( \text{ser}(s\text{-num}, s\text{-name}, \text{budget}) \), whose respective keys are \( e\text{-num} \) and \( s\text{-num} \). Attribute \( \text{resp} \) represents the employee’s senior; 0 means that this employee has no senior. The query aimed at the retrieval of any pair (employee’s name, service name) such that the employee works in the service, this employee does not live in his/her birth-city, and this service has a budget less than $15M, may be written as:

\[
\text{select } e\text{-name}, s\text{-name from emp, ser where budget < 15 and birth\text{-city} \neq living\text{-city} and work\text{-serv} = s\text{-num}}.
\]

Here, the join condition is integrated inside the “where” clause and the distinction between selection and join conditions made in relational algebra

<table>
<thead>
<tr>
<th>operator</th>
<th>algebraic expression</th>
<th>SQL expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>selection</td>
<td>select(r, p)</td>
<td>select * from r where p</td>
</tr>
<tr>
<td>projection</td>
<td>project(r, {A_1, \ldots, A_n})</td>
<td>select distinct A_1, \ldots, A_n from r</td>
</tr>
<tr>
<td>Cartesian product</td>
<td>cartprod(r, s)</td>
<td>select * from r, s</td>
</tr>
<tr>
<td>theta-join</td>
<td>join(r, s, \theta, A, B)</td>
<td>select * from r, s where A \theta B</td>
</tr>
</tbody>
</table>
vanishes. With the extensions of \textit{emp} and \textit{ser} shown in Tables 2.7 and 2.8, this query delivers the result reported in Table 2.9.

When a relation is used several times (for instance twice as is the case in a self-join), it is necessary to know which occurrence an attribute refers to. To this end, a variable is associated with every occurrence of the relation; this variable serves as a prefix for attribute names. We will see later other situations where variables are of interest.

\textbf{Example 2.8.} With the relation \textit{emp} of Example 2.7, the query to look for the pairs (employee’s name, senior’s name) such that the employee earns more than his senior, requires two occurrences of relation \textit{emp}, and it written as:

\begin{verbatim}
select E.name, S.name from emp E, emp S
where E.senior = S.e-id and E.salary > S.salary,
\end{verbatim}

where \textit{E} and \textit{S} refer to two copies of relation \textit{emp} (more specifically, \textit{S} plays the role of the seniors of the elements of \textit{E}).

The two preceding examples illustrate some of the predicates that usually appear in a “where” clause, i.e., founded on a conjunctive or disjunctive

\begin{table}[h]
\centering
\caption{Relation \textit{emp} of Example 2.7}
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
\textit{e-id} & \textit{e-name} & \textit{age} & \textit{birth-city} & \textit{living-city} & \textit{work-serv} & \textit{resp} & \textit{salary} \\
\hline
14 & Durand & 37 & Paris & Nantes & 5 & 3 & 9000 \\
5 & Milon & 49 & Lyon & St Nazaire & 1 & 34 & 21500 \\
34 & Rocca & 28 & Le Croisic & Rennes & 1 & 3 & 12000 \\
3 & Perrol & 34 & Lannion & Redon & 5 & 0 & 7600 \\
7 & Robin & 30 & Nantes & Nantes & 5 & 3 & 11000 \\
\hline
\end{tabular}
\end{table}

\begin{table}[h]
\centering
\caption{Relation \textit{ser} of Example 2.7}
\begin{tabular}{|c|c|c|}
\hline
\textit{s-num} & \textit{s-name} & \textit{budget} \\
\hline
1 & sales & 19 \\
5 & marketing & 13 \\
\hline
\end{tabular}
\end{table}

\begin{table}[h]
\centering
\caption{Result of the query of Example 2.7}
\begin{tabular}{|c|c|}
\hline
\textit{e-name} & \textit{s-name} \\
\hline
Durand & marketing \\
Perrol & marketing \\
\hline
\end{tabular}
\end{table}
combination of terms (predicates) of the form “attribute \( \theta \) constant” or “attribute-1 \( \theta \) attribute-2” where \( \theta \) is a comparator. There are also many dedicated predicates (e.g., “attribute is null”) which are not dealt with here.

A variant of the base block allows the delivery of a result made of a single tuple using one or more aggregate functions according to the syntax:

\[
\text{select } ag-1(\text{att-1}), \ldots, ag-p(\text{att-p}) \text{ from relations where condition,}
\]

where \( ag-i(\text{att-i}) \) denotes an aggregate (i.e., a set-oriented) function, \( ag-i \) applying to attribute \( \text{att-i} \). This is interpreted as follows:

1. perform the Cartesian product of the relations of the “from” clause,
2. select the tuples complying with the condition of the “where” clause,
3. compute the value of each aggregate function of the “select” clause over the tuples resulting from the previous step.

The most common aggregate functions, generally predefined in available DBMSs, are: maximum \((\max)\), minimum \((\min)\), average \((\avg)\), sum \((\sum)\), variance \((\var)\), standard deviation \((\std)\), median \((\med)\), and cardinality \((\text{count})\). With the except of the last function, it is assumed that the argument used is numeric.

**Example 2.9.** With the previous relations \( emp \) and \( ser \), the query to look for the number of employees working in a service having a budget less than $15M, along with their average salary, may be written as:

\[
\text{select count}(\ast), \text{avg}(\text{salary}) \text{ from } emp, ser
\]

\[
\text{where } \text{work-serv} = s-\text{num and budget} < 15.
\]

Here, “\( \text{count}(\ast) \)” returns the cardinality of the tuples resulting from steps (1) and (2), while “\( \text{avg}(\text{salary}) \)” computes the average of the attribute \( \text{salary} \) over those tuples. With the data of the extensions given previously in Tables 2.7 and 2.8, the result obtained is the tuple: \((3, 9200)\). If one wants to know how many employees of service 5 play the role of a senior, it is necessary to use “\( \text{count} \)” applied to the attribute \( \text{senior} \), together with the keyword “distinct” so that any senior is taken into account only once. Such a query would be formulated as:

\[
\text{select count(different senior) from } emp
\]

\[
\text{where } \text{senior} \neq 0 \text{ and work-serv} = 5.
\]

With the extension of \( emp \) in Table 2.7, this query returns 1 as a result, while 2 would have been delivered in the absence of the keyword “distinct”.

\[\blacksquare\]
2.3.2 Combining base blocks

Several blocks may appear inside an SQL query by means of two major constructs: predicates based on *subqueries* and *set-oriented combinations*.

Besides those mentioned previously, SQL allows for more sophisticated predicates with a nested block, also called a “subquery”; this can even be used recursively if needed. The principal forms of nested predicates are the following:

1. \( \text{att-1} \ [\text{not}] \ \text{in} \ (\text{select att-2 from ...}) \) where **att-1** and **att-2** are compatible attributes; this predicate is true if the value of **att-1** in the current tuple of the outer block belongs (resp. does not belong when “not” is used) to the set of values returned by the inner block (the subquery),

2. \( [\text{not}] \ \text{exists} \ (\text{select * from ...}) \); this predicate is true if the result returned by the inner block is not empty (resp. empty if “not” is used),

3. \( \text{att-1} \ \theta \ {\text{any}} | {\text{all}} \ (\text{select att-2 from ...}) \) where **att-1** and **att-2** are compatible attributes; with “any” (resp. “all”), this predicate is true if the value of **att-1** in the current tuple of the outer block is in relation \( \theta \) with at least one (resp. all) the values of **att-2** returned by the inner block,

4. \( \text{att-1} \ \theta \ (\text{select ag(att-2) from ...}) \) where **att-1** and **att-2** are compatible attributes; this predicate is true if the value of **att-1** in the current tuple of the outer block is in relation \( \theta \) with the single value (aggregate) returned by the inner block.

It is worth noting that some equivalences hold between these expressions, as illustrated in Example 2.10.

**Example 2.10.** Consider relations **emp** and **ser** as defined previously. The query to look for the name of any service whose budget is over $17M where there is no employee born in Nantes may be written as:

```sql
select s-name from ser where budget > 17 and s-num not in
   (select work-serv from emp where living-city = "Nantes")
```

or:

```sql
select s-name from ser where budget > 17 and s-num \neq\ all
   (select work-serv from emp where living-city = "Nantes")
```
or:

```
select s-name from ser where budget > 17 and not exists
   (select * from emp
    where living-city = "Nantes" and work-serv = ser.s-num).
```

In the last expression, the subquery refers to the outer block thanks to the predicate “work-serv = ser.s-num” where the dot notation “relation.attribute” is used. This method enhances the readability of the formulation, even if it is not mandatory here since one could also write “work-serv = s-num”.

It is also possible to combine base blocks by means of compositions corresponding to the set operators presented in Section 2.2.1. The general syntax is:

```
(selct att1-1, ..., att1-n from relations1 where condition1)
{union|intersect|except}
(selct att2-1, ..., att2-n from relations2 where condition2)
```

where the attributes from att1 and att2 are pairwise compatible. Except in the case of the union, which is mandatory since it builds a superset of each incoming relation, this kind of expression may be replaced by a construct with a subquery. So the intersection:

```
(selct A, B from r) intersect (selct A, B from s)
```

may be formulated as:

```
select A, B from r where exists
   (select * from s where A = r.A and B = r.B)
```

and the difference:

```
(select A, B from r) except (select A, B from s)
```

may be replaced by:

```
select A, B from r where not exists
   (select * from s where A = r.A and B = r.B).
```

**Example 2.11.** Consider the preceding relations `emp` and `ser`, and the query to look for any service (number) with a budget over $6M where there
is at least one employee living in Nantes. Such a query may be expressed either as:

\[
(\text{select } s\text{-num from } \text{ser where } \text{budget} > 6) \\
\text{intersect} \\
(\text{select } \text{work-serv from } \text{emp where } \text{living-city} = \text{“Nantes”})
\]

or as:

\[
\text{select } s\text{-num from } \text{ser where } \text{budget} > 6 \text{ and exists} \\
(\text{select } * \text{ from } \text{emp} \\
\text{where } \text{living-city} = \text{“Nantes”} \text{ and } \text{work-serv} = \text{ser.s-num}).
\]

### 2.3.3 Partitioning

Consider a user who would like to know the average salary in each service. Using the constructs presented so far, it would be necessary to submit first a query to get the number of any service where at least one employee works, then to process as many queries delivering the average salary of the corresponding service as there are services. Such a process would be tedious, and can be avoided using the partitioning facility of SQL.

The principle consists of partitioning a relation on the basis of one or more attributes sharing the same value, according to the schema of Figure 2.9. From now on, the result of the query concerns the subsets generated by the partitioning, and not tuples individually. The syntax of a query is:

\[
\text{select } \text{att-set-1, ag-set from relations where } \text{ind-cond} \\
\text{group by } \text{att-set-2 having } \text{set-cond}
\]

where:

- \text{ind-cond} is a condition for the selection of tuples of the relation to be partitioned,
- \text{att-set-2} is the set of attributes onto which the partitioning is done,
- \text{set-cond} is a condition for the selection of subsets coming from the partitioning,
- \text{att-set-1} is a subset of the attributes pertaining to \text{att-set-2} since only attributes of \text{att-set-2}, which have a same value in a subset, make sense at the subset level,
- \text{ag-set} stands for a list of aggregate functions applying to attributes of the partitioned relation.
The interpretation of a partitioning query is the following (regardless of any actual processing in a DBMS):

1. perform the Cartesian product of the relations of the “from” clause,
2. select the tuples complying with the condition (ind-cond) of the “where” clause,
3. from the relation resulting from the previous step, build subsets of tuples with the same value over the attribute(s) mentioned in the “group by” clause (see the schema of Figure 2.9),
4. remove the subsets which do not comply with the condition (set-cond) of the clause “having” (which makes use of aggregate functions),
5. for each subset, retain the value of the attributes appearing in the “select” clause (att-set-1) and compute the value of the aggregates of this same clause.

**Example 2.12.** The query to look for any service where the average salary of its employees born in Nantes is over $12K may be written as:

```
select s-num from emp where birth-city = "Nantes"
group by s-num having avg(salary) > 12.
```

The query returning the number of employees, the average salary, the lowest and highest salary of any service where more than 15 employees work is
expressed as:

```sql
select s-num, count(*), avg(salary), min(salary), max(salary)
from emp
group by s-num having count(*) > 15.
```

### 2.3.4 Expressing division and antidivision

No specific feature is available in SQL to express division queries. Such a need is covered, thanks to partitioning. Recall that the division of relation \( r \) with schema \( R(A, X) \) by relation \( s \) with schema \( S(B, Y) \) retains a value \( x \) appearing in attribute \( X \) of \( r \) if it is associated with at least all the values of attribute \( B \) in \( s \). It turns out that partitioning enables the computation of the set of values associated with every \( x \) (using \( X \) as the partitioning attribute). However, it is still necessary to overcome the lack of set containment (or any other set comparison operation) in SQL. Fortunately, it is possible to replace this operation by a comparison over cardinalities of sets, although not the original ones of course, once one remarks that:

\[
A \subseteq B \iff A = A \cap B \iff \text{card}(A) = \text{card}(A \cap B).
\]

From this observation, the dividend relation is seen to be restricted to tuples whose value of the division attribute \( (A) \) pertains to the divisor relation. Now, a value \( x \) is acceptable if it is associated with a set whose cardinality is exactly that of the divisor. This results in the following generic expression:

```sql
select X from r where A in (select B from s)
group by X
having count(distinct A) = (select count(distinct B) from s).
```

An alternative expression of division is possible using subqueries (nested predicates) based on a convenient rewriting of Expression (2.1), since one has:

\[
\{ x \mid \forall a, (a \in \text{project}(s, B)) \Rightarrow ((a, x) \in r) \} =\]

\[
\{ x \mid \exists a, (a \in \text{project}(s, B)) \text{ and } ((a, x) \notin r) \}.
\]
This second expression leads to the following possible expression for the division of \( r \) by \( s \):

\[
\text{select distinct } X \text{ from } r \text{ r1 where not exists}
\]

\[
(\text{select } * \text{ from } s \text{ where } B \text{ not in}
\]

\[
(\text{select } A \text{ from } r \text{ where } X = r1.X)).
\]

**Example 2.13.** Consider the query to look for the number of any service where all the employees over 40 have stayed for at least three months (Example 2.5). Using partitioning, its expression is:

\[
\text{select } s\text{-num from stay where duration } \geq 3 \text{ and emp-num in}
\]

\[
(\text{select e-id from emp where age } > 40)
\]

\[
\text{group by s-num having count(distinct emp-num) =}
\]

\[
(\text{select count(}*) \text{ from emp where age } > 40).
\]

The introduction of the condition “\( \text{emp-num in (select e-id from emp where age } > 40) \)” guarantees the correctness of the statement.

The other possible formulation (using subqueries) is:

\[
\text{select } s\text{-num from stay s1 where duration } \geq 3 \text{ and not exists}
\]

\[
(\text{select } * \text{ from emp where age } > 40 \text{ where e-id not in}
\]

\[
(\text{select emp-num from stay}
\]

\[
\text{where duration } \geq 3 \text{ and s-num } = s1.s\text{-num}).
\]

The use of a variable (s1) in the outermost block is mandatory for the reference to attribute \( s\text{-num} \) made in the innermost block (\( s\text{-num} = s1.s\text{-num} \)).

The expression of an antidivision of relation \( r(A, X) \) by relation \( s(B, Y) \) (see Section 2.2.2) is much easier, and calls on a difference. Indeed, an element \( x \) is satisfactory if it does not belong to the set of values from \( X \) for which at least one association with a value \( B \) of \( s \) exists. This gives the formulation:

\[
\text{select } X \text{ from } r \text{ where } X \text{ not in (select } X \text{ from } r, s \text{ where } A = B).
\]
Chapter 3

Basic Notions on Fuzzy Sets

3.1 Introduction

Set theory is used in various domains of computer science (for instance principles of programming) and is one of the foundations of relational databases, as seen in the previous chapter. It is of interest to note that this theory has both a set-oriented side and a logical side, each set operation having its counterpart in logic. Several extensions of set theory have been studied in recent years, among which rough sets (Pawlak, 1982) and fuzzy sets (Zadeh, 1965) are of particular interest in the database area. Rough sets are used for data mining (see, e.g., Quafafou and Boussouf, 1997) and are not dealt with later on. Fuzzy sets were initially created for the design of automated systems, and they constitute a central topic in this book.

The key idea behind fuzzy sets is their ability to represent categories or classes of elements where the boundary between membership and rejection is not clear-cut (or abrupt), but on the contrary gradual. This approach is especially suitable for dealing with numerous concepts in everyday life (and terms of natural languages) for which there is a progressive transition between full- and non-membership, in the sense that the concept of interest associated with the fuzzy set (high temperature, inexpensive price, fair color, etc.) is observed to a greater or lesser extent. If the interpretation of such terms is clearly contextual, it is generally hard to represent them using regular sets, as illustrated by Figure 3.1. In this case, one can observe an abrupt transition between the values which pertain to the concept “high temperature” and those which do not. Thus, a temperature of 25°C is definitely not a “high temperature”, while 25.1°C definitely is a “high temperature”, which is not in accordance with common
sense. The history of fuzzy sets has been rich; its main events are listed below:

- 1965: Zadeh’s seminal paper is published (Zadeh, 1965),
- beginning of the seventies: birth of the “expert system” approach in control with fuzzy rules,
- 1973: first book about fuzzy sets written by A. Kaufmann,
- 1975: first results of the use of fuzzy rules by Mamdani and Assilian,
- 1978: paper introducing possibility theory which is closely related to fuzzy sets; this theory is introduced by Zadeh (1978) as a new theoretical framework for dealing with uncertainty,
- 1978: first issue of the international journal *Fuzzy Sets and Systems*,
- second half of the seventies: intensive activity in fuzzy control in Europe (United Kingdom, Denmark, Netherlands especially) with numerous applications to industrial processes,
- 1985: first congress of IFSA (International Fuzzy Systems Association) with a biennial occurrence ever since,
- end of the eighties: fuzzy boom in Japan with a wide range of applications such as the Sendai subway, washing machines, photocopiers, rice cookers, hi-fi sets, camcorders, and cameras,
- beginning of the nineties: growing interest in fuzzy sets in Germany (with the strong support of the authorities), France, Spain and North America; many national and international scientific conferences are launched,
- 1992: first international conference of the IEEE (FUZZ-IEEE) in San Diego, which has since become a yearly event,
- 1993: first issue of the international journal *IEEE Transactions on Fuzzy Systems*,
The initial application of fuzzy sets was in the area of control, since L.A. Zadeh was an automatician. However, fuzzy sets received huge interest in many other communities, even if the applications have not been as spectacular as in fuzzy control. The use of fuzzy sets has been, and is still, a topic of research and development in mathematics, image processing, classification, data analysis, artificial intelligence, operations research and optimization, decision making, information retrieval, and database systems, this last topic being the main concern of this book.

This chapter is limited to introducing those notions strictly necessary for understanding the following chapters. The reader interested in complementary aspects of fuzzy sets is invited to refer to Dubois and Prade (1980, 2000); Klir and Yuan (1995); Leondes (1999); Ruspini et al. (1998). After presenting some definitions and notations, the principal operations over fuzzy sets (composition and inclusion) are detailed. Then, the notions of fuzzy measures and integrals are given, as well as the extension principle and some key aspects of fuzzy quantified propositions.

3.2 Definitions and Notations

In the following, regular sets are denoted by $A$, $B$ and fuzzy sets by $E$, $F$. In the usual case, a set $A$ is associated with a characteristic function $f_A$ taking its value from the pair $\{0, 1\}$. For any value $x$ of the domain $X$ (also called the universe or referential), $f_A(x)$ indicates whether $x$ belongs to $A$ or not. This view extends to a fuzzy set which is associated with a characteristic function denoted by $\mu$ valued in the interval $[0, 1]$ of the real line $\mathbb{R}$, also called the unit interval. The value $\mu_E(x)$ expresses the extent to which the element $x$ of $X$ is a member of the fuzzy set $E$:

$$\mu_E : X \rightarrow [0, 1]$$

$$x \rightarrow \mu_E(x).$$

If $\mu_E(x)$ equals 0, $x$ does not belong to $E$ at all; if it is 1, $x$ is a full member of $E$. The closer $\mu_E(x)$ is to 1 (resp. 0), the more $x$ belongs (resp. does not belong) to $E$. A regular set is a special case of a fuzzy one, where the characteristic function takes only the two values 0 and 1.

In practice, a fuzzy set is often used to model a linguistic term, in particular an adjective, as illustrated in Figures 3.2 and 3.3. A fuzzy set provides a symbolic/numeric (and vice versa) interface, between the symbolic aspect
of the term and the numeric aspect of the characteristic function valued in the unit interval.

Most often, the modeled fuzzy concepts correspond to functions that are either monotonic, or increasing–decreasing represented by a trapezium (possibly left or right open), or a triangle. The part of the function corresponding to transitions is then linear, which is generally sufficient for coping with most requirements. For convenience, it may happen that a discrete function (for instance over the variable \(\text{age}\)) is represented in a continuous fashion. In such a case, the function makes sense only for the effective values of the variable (for instance, integers between 18 and 70 for the variable \(\text{age}\)).

In the case of a finite discrete fuzzy set \(E\) (which will be considered in what follows since it is sufficient in the database context), we use:

\[
E = \{\mu_1/x_1, \ldots, \mu_n/x_n\},
\]

which means that \(E\) has the associated characteristic function:

\[
E : \{x_1, \ldots, x_n\} \rightarrow [0, 1]
\]

where \(\mu_E(x_i) = \mu_i\) is the degree of membership of \(x_i\) to \(E\). By convention, the associations for which the degree equals 0 are omitted.
Basic Notions on Fuzzy Sets

The support of a fuzzy set \( E \), denoted by \( \text{supp}(E) \), is the regular set made of the elements with a strictly positive degree, i.e., which belong somewhat to \( E \):

\[
\text{supp}(E) = \{ x \mid x \in X \text{ and } \mu_E(x) > 0 \}.
\]

The core of a fuzzy set \( E \), denoted by \( \text{core}(E) \), is defined as the regular set of the elements whose degree is 1, i.e., which are full members of \( E \):

\[
\text{core}(E) = \{ x \mid x \in X \text{ and } \mu_E(x) = 1 \}.
\]

These two notions collapse in the case of a regular set, since if \( x \) is somewhat in \( A \) it belongs (totally) to \( A \).

A fuzzy set \( E \) may be described as a collection of nested regular sets using the concept of \( \alpha \)-level cut or \( \alpha \)-cut defined as the regular set \( E_\alpha \) of elements whose degree is greater than or equal to \( \alpha \):

\[
E_\alpha = \{ x \mid x \in X \text{ and } \mu_E(x) \geq \alpha \}.
\]

The strict \( \alpha \)-cut of \( E \) denoted by \( E_\alpha^\prec \) is obtained by replacing the symbol \( \geq \) by \( > \) in the above formula.

The following properties hold:

- \( \alpha_1 > \alpha_2 \Rightarrow E_{\alpha_1} \subseteq E_{\alpha_2} \).
- \( E_1 = \text{core}(E) \).
- \( E_\emptyset = \text{supp}(E) \).

The notion of \( \alpha \)-cut establishes a natural link between fuzzy and regular sets, in the sense that a fuzzy set is expressed by more or less demanding interpretations in terms of \( \alpha \)-cuts. Moreover, a fuzzy set may be recovered from its \( \alpha \)-cuts by considering the proper elements \( P_E(\alpha) \) of each \( \alpha \)-cut, i.e., the elements of \( E \) that belong to none of the \( \alpha \)-cuts with a degree strictly higher than \( \alpha \). The initial fuzzy set \( E \) is then made of the elements of each \( P_E(\alpha) \) which are assigned their associated degree \( \alpha \).

The cardinality of a regular set extends to fuzzy sets as follows:

\[
\text{card}(E) = \sum_{x \in X} \mu_E(x).
\]

The height of a fuzzy set \( E \) is given by the degree of the element of maximal membership. \( E \) is said to be normalized if its height is 1.
In the following, the **complement** of fuzzy set $E$ is defined as:

$$\forall x \in X, \mu_{\overline{E}}(x) = 1 - \mu_E(x).$$

This operation is involutive, i.e., $\overline{\overline{E}} = E$ ($\mu_{\overline{E}}(x) = \mu_E(x)$).

**Example 3.1.** Let $X = \{a, b, \ldots, j, k\}$ and $E = \{1/a, 0.7/c, 0.2/e, 0.3/f, 0.6/h, 0.7/j, 0.9/k\}$. The height of $E$ equals 1 ($E$ is normalized) and:

- $\overline{E} = \{1/b, 0.3/c, 1/d, 0.8/e, 0.7/f, 1/g, 0.4/h, 1/i, 0.3/j, 0.1/k\}$,
- $E_{0.3} = \{a, c, f, h, j, k\}; E_{0.5} = \{a, c, h, j, k\}$,
- $E_{0.6} = \{a, c, j, k\}; E_{0.8} = \{a, k\}$,
- $\text{supp}(E) = \{a, c, e, f, h, j, k\}$,
- $\text{core}(E) = E_1 = \{a\}$,
- $\text{card}(E) = 4.4.$

### 3.3 Composition of Fuzzy Sets

The operations combining fuzzy sets extend the usual set operations; they are presented along with their logical counterpart.

#### 3.3.1 Intersection and union of fuzzy sets

**Norm** and **co-norm** operators constitute the foundation of the extension of intersection and union for fuzzy sets whose logical counterpart is the pair conjunction/disjunction (and/or). In the unit interval, the norm $\top(x, y)$ is a binary operator with the following properties:

- **associativity**: $\top(x, \top(y, z)) = \top(\top(x, y), z)$,
- **monotonicity** (non-decreasing with respect to each argument):
  
  \[
    \text{if } x_1 < x_2 \text{ then } \top(x_1, y) \leq \top(x_2, y),
    \text{if } y_1 < y_2 \text{ then } \top(x, y_1) \leq \top(x, y_2),
  \]

- **existence of the neutral element 1**: $\top(1, x) = \top(x, 1) = x$.

Similarly, a **co-norm** $\perp(x, y)$ is associative, commutative, monotonic, and admits 0 as a neutral element. The following inequalities hold for norms and co-norms:

$$\forall y, \top(x, y) \leq x \leq \perp(x, y) \quad \text{and} \quad \forall x, \top(x, y) \leq y \leq \perp(x, y),$$
which look like the triangular inequality and lead to the terms \textit{triangular norm} and \textit{triangular co-norm}. Any pair norm/co-norm ($\top/\perp$) is tied by duality relationships:

$$\top(x, y) = 1 - \perp(1 - x, 1 - y) \quad \text{and} \quad \perp(x, y) = 1 - \top(1 - x, 1 - y),$$

which correspond to the De Morgan laws in the usual set-theoretic framework, i.e.:

$$(a \text{ and } b) = (\overline{a} \text{ or } \overline{b}) \quad \text{and} \quad (a \text{ or } b) = (\overline{a} \text{ and } \overline{b}).$$

The intersection and union of fuzzy sets are defined as:

- $\forall x \in X, \mu_{E \cap F}(x) = \top(\mu_E(x), \mu_F(x))$,
- $\forall x \in X, \mu_{E \cup F}(x) = \perp(\mu_E(x), \mu_F(x))$

where $\top(x, y)$ (resp. $\perp(x, y)$) stands for a norm (resp. the dual co-norm).

Norms have a \textit{minimal element} called \textit{Weber’s norm} defined as:

$$\top_W(x, y) = y, \quad \top_W(x, 1) = x \quad \text{otherwise}.$$  

Indeed, any norm $\top$ is such that:

$$\top(x, 1) = x, \quad \top(1, y) = y; \quad \text{in any other case } \top(x, y) \geq 0.$$  

They also admit the \textit{minimum} ($\min(x, y)$) as a \textit{maximal element}.

\textbf{Proof.} Assume that $\min(x, y) = x$. Due to the properties of norms, one has:

$$\top(x, y) \leq \top(x, 1) \quad \text{and} \quad \top(x, 1) = x.$$  

Then, $\top(x, y) \leq \min(x, y)$. ■

Due to the relationship between co-norms and norms, the minimal co-norm is the maximum ($\max(x, y)$) and the maximal co-norm is \textit{Weber’s co-norm} defined as:

$$\perp_W(0, y) = y, \quad \perp_W(x, 0) = x \quad \text{and} \quad \perp_W(0, y) = 1 \quad \text{otherwise}.$$  

In addition, 0 (resp. 1) turns out to be an \textit{absorbing element} for any norm (resp. co-norm).

\textbf{Proof.} From the fact that $\min(x, y)$ is the largest norm, any norm $\top$ complies with:

$$\forall x \in [0, 1], \top(x, 0) \leq \min(x, 0).$$
But \( \min(x, 0) = 0 \) and then: \( \forall x \in [0, 1], \top(x, 0) = 0 \). In an analogous way, for any co-norm \( \bot: \forall x \in [0, 1], \bot(x, 1) \geq \max(x, 1) \) and \( \max(x, 1) = 1 \) leads to the conclusion that \( \forall x \in [0, 1], \bot(x, 1) = 1 \). \( \blacksquare \)

Norms and co-norms generalize the usual conjunction and disjunction operators:

- \( \top(0, 1) = \top(1, 0) = \top(0, 0) = 0 \) since 0 is absorbing for any norm \( \top \),
- \( \top(1, 1) = 1 \) since 1 is the neutral element of any norm \( \top \),
- \( \bot(0, 1) = \bot(1, 0) = \bot(1, 1) = 1 \) since 1 is absorbing for any co-norm \( \bot \),
- \( \bot(0, 0) = 0 \) since 0 is the neutral element of any co-norm \( \bot \).

**Example 3.2.** Let \( E \) and \( F \) be the fuzzy sets:

\[
E = \{1/a, 0.3/b, 0.7/c, 0.2/e\}
\]

\[
F = \{0.3/a, 1/c, 1/d, 0.6/e\}.
\]

With min/max, the intersection and union of \( E \) and \( F \) are:

\[
E \cap F = \{0.3/a, 0.7/c, 0.2/e\}
\]

\[
E \cup F = \{1/a, 0.3/b, 1/c, 1/d, 0.6/e\},
\]

whereas with the probabilistic norm and co-norm (see Table 3.1) one has:

\[
E \cap F = \{0.3/a, 0.7/c, 0.12/e\}
\]

\[
E \cup F = \{1/a, 0.3/b, 1/c, 1/d, 0.68/e\},
\]

and with Łukasiewicz’s norm and co-norm (see Table 3.1):

\[
E \cap F = \{0.3/a, 0.7/c\}
\]

\[
E \cup F = \{1/a, 0.3/b, 1/c, 1/d, 0.8/e\}. \diamond
\]

The number of norms (and co-norms) is infinite; the most common ones are given in Table 3.1. Example 3.2 shows that the result of the intersection and union of two fuzzy sets may vary widely according to the choice made for the norm and co-norm. Moreover, it is impossible to find a couple \((\top, \bot)\) for which all the properties of Boolean algebra hold. In particular, idempotency and non-contradiction/excluded-middle laws are incompatible.

**Proof.** Consider an idempotent norm \( \top \) such that \( \forall x \in X, \top(x, x) = x \). Moreover, assume that \( E \cap \overline{E} = \emptyset \), i.e.: \( \forall x \in X, \top(x, 1 - x) = 0 \). We should have, simultaneously: \( \top(0.5, 0.5) = 0.5 \) and \( \top(0.5, 0.5) = 0 \), which is impossible.
Table 3.1 Principal norms and co-norms

<table>
<thead>
<tr>
<th>norm: ( \top(x, y) )</th>
<th>associated co-norm: ( \bot(x, y) )</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \min(x, y) )</td>
<td>( \max(x, y) )</td>
<td>Zadeh</td>
</tr>
<tr>
<td>( xy )</td>
<td>( x + y - xy )</td>
<td>probabilistic</td>
</tr>
<tr>
<td>( \max(x + y - 1, 0) )</td>
<td>( \min(x + y, 1) )</td>
<td>Lukasiewicz</td>
</tr>
<tr>
<td>( xy ) ( \gamma \in [0, 1] )</td>
<td>( \frac{x + y - xy - (1 - \gamma)xy}{1 - (1 - \gamma)xy} ), ( \gamma \in [0, 1] )</td>
<td>Hamacher</td>
</tr>
<tr>
<td>( \begin{cases} x &amp; \text{if } y = \text{yes} \ y &amp; \text{if } x = \text{yes} \ 0 &amp; \text{otherwise} \end{cases} )</td>
<td>( \begin{cases} x &amp; \text{if } y = \text{no} \ y &amp; \text{if } x = \text{no} \ 1 &amp; \text{otherwise} \end{cases} )</td>
<td>Weber</td>
</tr>
</tbody>
</table>

Similarly, an idempotent co-norm \( \bot \) is such that \( \forall x \in X, \bot(x, x) = x \). Moreover, if the excluded-middle law holds: \( E \cup \overline{E} = X \), i.e.: \( \forall x \in X, \bot(x, 1 - x) = 1 \). Here, we must have both: \( \bot(0.5, 0.5) = 0.5 \) and \( \bot(0.5, 0.5) = 1 \), which is impossible.

Double distributivity is not guaranteed for a given pair \((\top, \bot)\). It turns out that the couple \((\min, \max)\) is of particular interest, because it preserves most of the properties usually valid:

- computational simplicity,
- applicability to a purely ordinal scale, coarser than the unit interval, where symbolic values are provided for the grades. For instance, with the ordered symbols \( \text{bad} \prec \text{weak} \prec \text{small} \prec \text{medium} \prec \text{large} \prec \text{huge} \), it is possible to compare any pair of symbols \((a, b)\) and then to evaluate \( \min(a, b) \) and \( \max(a, b) \),
- preservation of both idempotency and double distributivity,
- distributivity of the \( \alpha \)-cut operation with respect to intersection and union:

\[
(E \cap F)_{\alpha} = E_{\alpha} \cap F_{\alpha} \quad \text{and} \quad (E \cup F)_{\alpha} = E_{\alpha} \cup F_{\alpha}.
\]

A drawback of the minimum (resp. maximum) lies in its inability to discriminate between two situations where vectors of grades are compared and share the same minimal (resp. maximal) value. So, \( \min(0.8, 0.3, 0.5) = \min(1, 0.3, 0.9) = 0.3 \). Of course, other norm/co-norm couples have better discrimination power, but generally they do not offer the ordinal feature of minimum and maximum. Several refinements of min/max have been proposed (Dubois et al., 1997) in order to stay within an ordinal framework and to reach a better discrimination power. These refinements no longer correspond to norm/co-norm pairs. We now illustrate two representatives:
the *discrim* and the *leximin* operators (the case of *discrimax* and *leximax* being quite similar).

We consider two vectors \( x \) and \( y \) over \([0, 1]^p\). We use the notations: \( x_i \) (resp. \( y_i \)) for the \( i \)th component of \( x \) (resp. \( y \)), \( D(x, y) = \{ i \mid x_i \neq y_i \} \). The minimum, *discrim* and *leximin* operations over \( x \) and \( y \) are defined as:

\[
\begin{align*}
  x >_{\text{min}} y &\iff \min_{i \in [1,p]} x_i > \min_{i \in [1,p]} y_i, \\
  x >_{\text{disc}} y &\iff \min_{i \in D(x,y)} x_i > \min_{i \in D(x,y)} y_i, \\
  x >_{\text{lex}} y &\iff \exists k \in [1,p] \text{ such that } \forall i < k, x^*_i = y^*_i \text{ and } x^*_k > y^*_k,
\end{align*}
\]

where \( x^*_i \) stands for the \( i \)th component of the vector \( x^* \) obtained from \( x \) by ranking its components in increasing order. *Leximin* refines *discrim*, which in turn refines the minimum, i.e.:

\[
x >_{\text{min}} y \Rightarrow x >_{\text{disc}} y \Rightarrow x >_{\text{lex}} y.
\]

For instance, if \( x = (0.1, 0.2, 0.7, 0.3) \) and \( y = (0.2, 0.2, 1, 0.1) \), we have:

\[
\begin{align*}
  x^* &= (0.1, 0.2, 0.3, 0.7), \\
  y^* &= (0.1, 0.2, 0.2, 1), \\
  D(x, y) &= \{1, 3, 4\}, \\
  \min_{i \in D(x,y)} x_i &= 0.1, \\
  \min_{i \in D(x,y)} y_i &= 0.1,
\end{align*}
\]

then \( x \) and \( y \) are incomparable using *discrim*, just as they are for the minimum, but \( x >_{\text{lex}} y \) \((k = 3)\). Now if we take: \( x = (0.1, 0.2, 0.7, 0.3) \) and \( y = (0.2, 0.2, 1, 0.8) \), we obtain:

\[
\begin{align*}
  x^* &= (0.1, 0.2, 0.3, 0.7), \\
  y^* &= (0.2, 0.2, 0.8, 1), \\
  D(x, y) &= \{1, 3, 4\}, \\
  \min_{i \in D(x,y)} x_i &= 0.1, \\
  \min_{i \in D(x,y)} y_i &= 0.2,
\end{align*}
\]

then \( y >_{\text{disc}} x, y >_{\text{min}} x \) and \( y >_{\text{lex}} x \) \((k = 1)\).

It is worth noting that the only case where two vectors \( x \) and \( y \) cannot be ranked according to *leximin* is when they have exactly the same values (regardless of their order in the vectors).
Another pair of operators stemming from the min/max couple is their weighted versions (denoted by \(w\text{min}\) and \(w\text{max}\)), intended for preserving the ordinal feature and most of its properties (in particular associativity, commutativity, neutral element). So if the importances \(w_1, \ldots, w_n\) (with \(\max(w_i) = 1\); i.e., at least one set \(E_j\) is completely important) are associated with the fuzzy sets \(E_1, \ldots, E_n\), one has:

\[
\begin{align*}
\mu_{w\text{min}}(E_1, \ldots, E_n, w_1, \ldots, w_n)(x) &= \min(\max(1 - w_1, \mu_{E_1}(x)), \ldots, \max(1 - w_n, \mu_{E_n}(x))), \\
\mu_{w\text{max}}(E_1, \ldots, E_n, w_1, \ldots, w_n)(x) &= \max(\min(w_1, \mu_{E_1}(x)), \ldots, \min(w_n, \mu_{E_n}(x))).
\end{align*}
\]

The normalization condition over the weights is imposed by the fact that, in its absence, the maximal (resp. minimal) degree of satisfaction 1 (resp. 0) cannot be reached for \(w\text{max}\) (resp. \(w\text{min}\)) even though all the input degrees \(\mu_{E_i}(x)'s\) equal 1 (resp. 0). The usual minimum and maximum are recovered from the previous expressions by assigning the maximal level of importance 1 to every set, i.e., \(w_1 = \ldots, w_n = 1\). Finally, the properties:

\[
\begin{align*}
\mu_{w\text{min}}(E_1, \ldots, E_n, w_1, \ldots, w_n) &= \mu_{w\text{max}}(E_1, \ldots, E_n, w_1, \ldots, w_n), \\
\mu_{w\text{max}}(E_1, \ldots, E_n, w_1, \ldots, w_n) &= \mu_{w\text{min}}(E_1, \ldots, E_n, w_1, \ldots, w_n)
\end{align*}
\]

hold.

### 3.3.2 Difference between fuzzy sets

An initial approach to the difference, whose logical counterpart is the operator “except” (or “and not”), consists of extending the usual definition over two regular sets \(A\) and \(B\):

\[
A - B = \{x \mid x \in A \text{ and } x \notin B\}. \tag{3.1}
\]

More generally, from the definition of an operator given as:

\[
\text{op}(<\text{parameters}> = \{t \mid \text{formula } \Phi \text{ defining the acceptable elements } t\},
\]

the characteristic function \(\mu_{\text{op}}(t)\) is obtained by making the following replacements in formula \(\Phi\):

- a conjunction (resp. disjunction) by a norm \(\top\) (resp. co-norm \(\bot\),
Fuzzy Preference Queries to Relational Databases

• the universal (resp. existential) quantifier by a generalized conjunction (resp. disjunction); the n-ary version of min/max is taken later on (except otherwise stated),
• the negation by the complement to 1,
• membership \((x \in E)\) by a degree of membership \((\mu_E(x))\).

In the specific case of the difference, letting \(E\) and \(F\) be two fuzzy sets, one obtains:

\[
\forall x \in X, \mu_{E - F}(x) = \top(\mu_E(x), \mu_F(x)) = \top(\mu_E(x), 1 - \mu_F(x)),
\]

which, with particular choices for the norm \(\top\), leads to:

\[
\mu_{E - F}(x) = \min(\mu_E(x), 1 - \mu_F(x)) \text{ with } \top(x, y) = \min(x, y),
\]

\[
\mu_{E - F}(x) = \max(\mu_E(x) - \mu_F(x), 0) \text{ if } \top(x, y) = \max(x + y - 1, 0) \text{ is chosen.}
\]

Example 3.3. Consider the fuzzy sets:

\[
E = \{1/a, 0.3/b, 0.7/c, 0.2/e\} \quad \text{and} \quad F = \{0.3/a, 1/c, 1/d, 0.6/e\}.
\]

Using the minimum for the conjunction, one obtains \(\{0.7/a, 0.3/b, 0.2/e\}\) for the difference \(E - F\), while \(E - F = \{0.7/a, 0.3/b\}\) with the other choice.

Other approaches may be envisaged based on different points of view about the formulation of the usual difference between regular sets, which, of course, are equivalent to Expression (3.1). For instance, one may take:

\[
E - F = \text{the smallest set to add to } E \cap F \text{ to obtain } E,
\]

as a starting point. Then, it follows that:

\[
E - F = \cap \{G \mid G \cup (E \cap F) \supseteq E\},
\]

and the characteristic function of the difference between the fuzzy sets \(E\) and \(F\) is defined as:

\[
\mu_{E - F}(x) = \inf \{u \mid \bot(u, \top(\mu_E(x), \mu_F(x)) \geq \mu_E(x))\},
\]

where \(\top\) (resp. \(\bot\)) is a triangular norm (resp. co-norm).

Example 3.4. With the fuzzy sets \(E\) and \(F\) of Example 3.3, the norm “minimum” and the co-norm “maximum”, one obtains the difference \(E - F = \{1/a, 0.3/b\}\). This result is quite different from those obtained previously (see Example 3.3), which highlights the variety of behaviors observed according to the choice of difference operator.
This kind of approach is particularly appropriate when it is impossible to call on the notion of complement, for instance if one wants to deal with fuzzy bags (see Rocacher and Bosc, 2002; Rocacher, 2002). A somewhat similar view is based on:

\[ E - F = \text{the smallest set to add to } F \text{ to obtain } E \cup F, \]

which leads to:

\[ \mu_{E - F}(x) = \inf \{ u \mid \perp(u, \mu_F(x)) \geq \perp(\mu_E(x), \mu_F(x)) \}, \]

where \( \perp \) is a triangular co-norm.

### 3.3.3 Cartesian product of fuzzy sets

The Cartesian product of the two regular sets \( A \) and \( B \), defined as:

\[ A \times B = \{ xy \mid x \in A \text{ and } y \in B \}, \]

where \( xy \) denotes the concatenation of \( x \) and \( y \), extends to fuzzy sets as soon as a triangular norm \( \top \) is taken for the conjunction, which yields:

\[ \mu_{E \times F}(xy) = \top(\mu_E(x), \mu_F(y)). \]

**Example 3.5.** Let \( E \) and \( F \) be the fuzzy sets:

\[ E = \{ 1/a, 0.3/b, 0.7/c, 0.2/e \}, \]
\[ F = \{ 1/y, 0.6/z \}. \]

Their Cartesian product \( E \times F \) is:

\[ \{ 1/(a,y), 0.6/(a,z), 0.3/(b,y), 0.3/(b,z), \]
\[ 0.7/(c,y), 0.6/(c,z), 0.2/(e,y), 0.2/(e,z) \} \]

if the minimum is taken for \( \top \), while one would obtain:

\[ \{ 1/(a,y), 0.6/(a,z), 0.3/(b,y), 0.18/(b,z), \]
\[ 0.7/(c,y), 0.42/(c,z), 0.2/(e,y), 0.12/(e,z) \} \]

with the product.

www.allitebooks.com
3.3.4 Trade-off operators

The preceding operators correspond to an adaptation of the operators working on regular sets. However, there are operators which make sense with fuzzy sets but which do not in the usual context, for instance because their result would not be in the pair \{0, 1\}. This is the case in particular for mean operators which perform a trade-off over their arguments. This view is obviously quite orthogonal to the use of the minimum (resp. maximum) which keeps only the worst (resp. best) value among the arguments. It is possible to combine several fuzzy sets by means of diverse simple mean operators (e.g., arithmetic, geometric, or harmonic). If a weighted mean is used, a specific importance is given to each of the input sets. So, if \( W \) denotes a normalized set of weights (whose elements add up to 1), the weighted mean of the fuzzy sets \( E_1, \ldots, E_n \) returns the fuzzy set characterized by:

\[
\mu_{\text{wem}}(E_1, \ldots, E_n, W)(x) = \sum_{i=1}^{n} (\mu_{E_i}(x) \ast W[i]).
\]

Using this operator, weight assignment is static. Yager (1988) has introduced a variant, called ordered weighted averaging (OWA), where the assignment is dynamic. Instead of combining the values by assigning \( W[1] \) to \( E_1 \), \( W[2] \) to \( E_2 \), and so on, the principle relies on a weight assignment once the values of the vector \( E_1(x), \ldots, E_n(x) \), are ranked decreasingly. By doing this, the weight \( W[1] \) applies to the largest value of the vector, \( W[2] \) to the second largest, \ldots, and \( W[n] \) to the smallest. The resulting definition is:

\[
\mu_{\text{OWA}}(E_1, \ldots, E_n, W)(x) = \sum_{i=1}^{n} (\mu_{E'_i}(x) \ast W[i])
\]

where \( \mu_{E'_i}(x) \) is the \( i \)th largest value among \( \{\mu_{E_1}(x), \ldots, \mu_{E_n}(x)\} \). It has been shown that:

- this operator is monotonically non-decreasing with respect to each entry \( (\mu_{E_i}(x)) \),
- its result lies between the minimum and the maximum of the input values,
- the minimum, maximum, and average of the entries are recovered when appropriate weights are chosen, namely

\[
W_{\text{min}} = (0, \ldots, 0, 1),
\]

\[
W_{\text{max}} = (1, 0, \ldots, 0),
\]

and

\[
W_{\text{avg}} = \left( \frac{1}{n}, \ldots, \frac{1}{n} \right).
\]
Example 3.6. Consider the fuzzy sets $E_1 = \{0.4/a, 0.8/b\}$, $E_2 = \{1/a, 0.2/b\}$, $E_3 = \{1/b\}$, and the weight vector $W = (0.2, 0.5, 0.3)$. The regular weighted mean over $E_1, E_2, \text{and} E_3$ delivers the set $F_1 = \{0.58/a, 0.56/b\}$, while $F_2 = \{0.4/a, 0.66/b\}$ is obtained with the OWA aggregation.

The relative position of mean operators with respect to norms and co-norms is given in Figure 3.4. There are many other trade-off operators: see in particular Dubois and Prade (1985b, 2000); Fodor and Yager (2000); Yager (1993).

3.3.5 Nonsymmetric operators

Here, we focus on three operators that have a counterpart in natural language, and can be seen as a way to define a sophisticated interaction between predicates in order to build three new types of condition:

- $P_1 \text{ and if possible} P_2$, which is related to conjunction,
- $P_1 \text{ or else} P_2$, which is connected with disjunction,
- $P_1 \text{ all the more as} P_2$, which expresses a reinforcement of $P_1$ when $P_2$ is more and more satisfied.

These constructs can be nested to consider, for instance, statements like $P_1 \text{ and if possible} (P_2 \text{ and if possible} P_3)$, or combined with each other, as in the statement $P_1 \text{ or else} (P_2 \text{ all the more as} P_3)$, but the scope of this section is restricted to binary statements. A more complete presentation of these operators can be found in Bosc and Pivert (2011).

3.3.5.1 “And if possible”

The expression “$P_1 \text{ and if possible} P_2$”, where $P_1$ and $P_2$ are two (possibly complex) predicates, expresses both a weak and asymmetric conjunction, in the sense that $P_2$ is less important than $P_1$ and does not have a full compensation power. An example of its use is the query “find houses with
four rooms, a price around $300k and if possible a small garden and a garage”. In such a context, the presence of a small garden and a garage intervenes only for houses having four rooms and a price somewhat around $300k. The basis for defining this operator, denoted by \( a_{ip} \), is the following set of axioms (where \( \mu_i \) stands for the degree of satisfaction of predicate \( P_i \) for a given element onto which “\( P_1 \) and if possible \( P_2 \)” applies):

- **AIP1**: “\( P_1 \) and if possible \( P_2 \)” is less drastic than “\( P_1 \) and \( P_2 \)”, i.e.:
  \[
a_{ip}(\mu_1, \mu_2) \geq \min(\mu_1, \mu_2),
\]

- **AIP2**: “\( P_1 \) and if possible \( P_2 \)” is more demanding than \( P_1 \), i.e.:
  \[
a_{ip}(\mu_1, \mu_2) \leq \mu_1,
\]

- **AIP3**: \( a_{ip} \) must have asymmetric behavior, i.e., it is noncommutative and:
  \[
  \exists x, y : a_{ip}(x, y) \neq a_{ip}(y, x),
\]

- **AIP4**: \( a_{ip} \) is increasing in its first argument, i.e.:
  \[
x \geq y \Rightarrow a_{ip}(x, z) \geq a_{ip}(y, z),
\]

- **AIP5**: \( a_{ip} \) is increasing in its second argument, i.e.:
  \[
y \geq z \Rightarrow a_{ip}(x, y) \geq a_{ip}(x, z),
\]

- **AIP6**: “\( P_1 \) and if possible \( P_2 \)” is equivalent to “\( P_1 \) and if possible (\( P_1 \) and \( P_2 \))”, i.e.:
  \[
a_{ip}(\mu_1, \mu_2) = a_{ip}(\mu_1, \min(\mu_1, \mu_2)).
\]

It follows that when \( P_1 \) is not at all satisfied, neither is “\( P_1 \) and \( P_2 \)”, which makes it impossible to call on a mean operator to model \( a_{ip} \). Moreover, the effect of “if possible” intervenes when \( \mu_2 \) is less than \( \mu_1 \) and one wants the value returned to be greater than \( \mu_2 \), which is the result in the presence of a regular conjunction. In other words, the result must be upgraded with respect to \( \mu_2 \) without going beyond \( \mu_1 \). Several ways to perform this upgrade can be envisaged; we suggest the following modeling whose behavior is illustrated in Table 3.2, and which complies with axioms AIP1 to AIP6:

\[
\alpha_1(\mu_1, \mu_2) = \min(\mu_1, k \cdot \mu_1 + (1 - k) \cdot \mu_2)
\]

with \( k \in [0, 1] \). For instance, if \( k = 0.5 \) is chosen, one gets:

\[
a_{ip1}(\mu_1, \mu_2) = \min \left( \mu_1, \frac{\mu_1 + \mu_2}{2} \right).
\]

### 3.3.5.2 “Or else”

The expression “\( P_1 \) or else \( P_2 \)”, where \( P_1 \) and \( P_2 \) are two predicates, expresses both a strong and asymmetric disjunction, in the sense that \( P_2 \)
Basic Notions on Fuzzy Sets

| Table 3.2 Behavior of “P₁ and if possible P₂” |
|-------|-------|-------|-------|
| µ₁    | µ₂    | k = 0.3 | k = 0.5 | k = 0.8 |
| 0.3   | 0     | 0.09   | 0.15   | 0.24   |
| 0.3   | 0.4   | 0.3    | 0.3    | 0.3    |
| 0.3   | 0.7   | 0.3    | 0.3    | 0.3    |
| 0.3   | 1     | 0.3    | 0.3    | 0.3    |
| 0.6   | 0     | 0.18   | 0.3    | 0.48   |
| 0.6   | 0.4   | 0.46   | 0.5    | 0.56   |
| 0.6   | 0.7   | 0.6    | 0.6    | 0.6    |
| 0.6   | 1     | 0.6    | 0.6    | 0.6    |
| 1     | 0     | 0.3    | 0.5    | 0.8    |
| 1     | 0.4   | 0.58   | 0.7    | 0.88   |
| 1     | 0.7   | 0.79   | 0.85   | 0.94   |
| 1     | 1     | 1      | 1      | 1      |

is not considered at the same level as P₁ and then is not a “true alternative” for P₁. As with “and if possible”, this operator, denoted by oe, can be defined on the basis of the following axioms:

- OE1: “P₁ or else P₂” is more drastic than “P₁ or P₂” since, as said above, P₂ is not as good as P₁: oe(µ₁, µ₂) ≤ max(µ₁, µ₂).
- OE2: “P₁ or else P₂” is softer than P₁ since P₂ constitutes a “choice”, i.e.: oe(µ₁, µ₂) ≥ µ₁.
- OE3: oe must have asymmetric behavior like aip, i.e., it is noncommutative (cf. axiom AIP3).
- OE4: oe is increasing in its first argument, i.e.: x ≥ y ⇒ oe(x, z) ≥ oe(y, z).
- OE5: oe is increasing in its second argument, i.e.: y ≥ z ⇒ oe(x, y) ≥ oe(x, z).
- OE6: “P₁ or else P₂” is equivalent to “P₁ or else (P₁ or P₂)”, i.e.: oe(µ₁, µ₂) = oe(µ₁, max(µ₁, µ₂)).

From axiom OE2, it follows that when P₁ is fully satisfied, so is “P₁ or else P₂”. In addition, the effect of “or else” intervenes when µ₂ is greater than µ₁ and one wants the value returned to be less than µ₂, which is the result in the presence of a regular disjunction. In other words, the result must be discounted with respect to µ₂ without going below µ₁. In the spirit of the definition of aip, that proposed for oe is:

$$oe(µ₁, µ₂) = max(µ₁, k · µ₁ + (1 - k) · µ₂)$$
Fuzzy Preference Queries to Relational Databases

with $k \in [0, 1]$. For instance, if $k = 0.5$ is taken:

$$oe_1(\mu_1, \mu_2) = \max(\mu_1, \frac{\mu_1 + \mu_2}{2}).$$

It can easily be proven that this definition conforms to axioms OE1 to OE6 (see Bosc and Pivert, 2011). Moreover, interestingly enough, De Morgan’s laws hold $\forall k \in [0, 1]$ between operators $aip$ and $oe$, which makes them dual as conjunctions and disjunctions based on norms and co-norms:

$$1 - aip(1 - \mu_1, 1 - \mu_2) = oe(\mu_1, \mu_2),$$

$$1 - oe(1 - \mu_1, 1 - \mu_2) = aip(\mu_1, \mu_2).$$

Table 3.3 illustrates the behavior of operator $oe$.

3.3.5.3 “All the more as”

We now deal with the modeling of conditions of the type “$P_1$ all the more as $P_2$”. An example of such a condition is: “find a Honda Accord with a low price, and the price must be all the lower as the mileage is higher”, which can be written as:

$$low(price) \text{ all the more as high(mileage)}.$$  

Once again, different interpretations of such a condition can be devised. The idea advocated here is to strengthen predicate $P_1$ in a way proportional to the satisfaction of $P_2$. In other words, for an element $t$,

$\mu P_1 \text{ all the more as } P_2(t)$ ranges from $\mu P_1(t)$ when $\mu P_2(t) = 0$ to $\mu_{(very \ P_2)}(t)$.

<table>
<thead>
<tr>
<th>$\mu_1$</th>
<th>$\mu_2$</th>
<th>$k = 0.3$</th>
<th>$k = 0.5$</th>
<th>$k = 0.8$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>0</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>0.3</td>
<td>0.4</td>
<td>0.37</td>
<td>0.35</td>
<td>0.32</td>
</tr>
<tr>
<td>0.3</td>
<td>0.7</td>
<td>0.58</td>
<td>0.5</td>
<td>0.38</td>
</tr>
<tr>
<td>0.3</td>
<td>1</td>
<td>0.79</td>
<td>0.65</td>
<td>0.54</td>
</tr>
<tr>
<td>0.6</td>
<td>0</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>0.6</td>
<td>0.4</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>0.6</td>
<td>0.7</td>
<td>0.67</td>
<td>0.65</td>
<td>0.62</td>
</tr>
<tr>
<td>0.6</td>
<td>1</td>
<td>0.88</td>
<td>0.8</td>
<td>0.68</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.4</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0.7</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
when $\mu_{P_1}(t) = 1$. According to this, the following axioms are used to construct the operator $atma$:

- ATMA1: $atma$ is decreasing in its second argument, i.e.: $y \leq z \Rightarrow atma(x, y) \geq atma(x, z)$,
- ATMA2: $atma$ is increasing in its first argument, i.e.: $x \leq y \Rightarrow atma(x, z) \leq atma(y, z)$,
- ATMA3: when $P_1$ is not at all satisfied, so is “$P_1$ all the more as $P_2$”, i.e.: $\forall y, atma(0, y) = 0$,
- ATMA4: when $P_2$ is not at all satisfied, “$P_1$ all the more as $P_2$” is equivalent to $P_1$, i.e.: $\forall x, atma(x, 0) = x$,
- ATMA5: when $P_1$ is partly satisfied and $P_2$ is somewhat satisfied, “$P_1$ all the more as $P_2$” is a “true” strengthening of $P_1$, i.e.: $\forall x \in ]0, 1[, \forall y > 0, atma(x, y) < x$.

From these properties and the choice of:

$$\mu_{very P_1}(x) = (\mu_{P_1}(x))^2$$

as the powering strengthening modifier supporting “very”, several definitions of $atma$ can be derived, among which are:

$$atma_1(\mu_1, \mu_2) = \mu_1^{(\mu_2 + 1)}$$
$$atma_2(\mu_1, \mu_2) = (\mu_1^2 - \mu_1) \cdot \mu_2 + \mu_1,$$

whose behavior is illustrated in Table 3.4. By analogy with the operator “$P_1$ all the more as $P_2$”, one may think of a predicate “$P_1$ all the less as $P_2$” for which predicate $P_1$ is weakened. Thus, $\mu_{P_1}$ all the less as $P_2(t)$ varies from $\mu_{P_1}(t)$ when $\mu_{P_2}(t)$ equals 0 to $\mu_{more or less P_1}(t)$ when $\mu_{P_2}$ is completely true.

<table>
<thead>
<tr>
<th>$\mu_1$</th>
<th>$\mu_2$</th>
<th>$\mu_1^{\mu_2+1}$</th>
<th>$(\mu_1^2 - \mu_1) \cdot \mu_2 + \mu_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6</td>
<td>0</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0.25</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0.8</td>
<td>0.5</td>
<td>0.715</td>
<td>0.72</td>
</tr>
<tr>
<td>0.6</td>
<td>1</td>
<td>0.36</td>
<td>0.36</td>
</tr>
<tr>
<td>0.2</td>
<td>0.75</td>
<td>0.06</td>
<td>0.08</td>
</tr>
</tbody>
</table>
3.4 Inclusions and Implications

The usual definition of the inclusion of \( A \) in \( B \) relies on the expression:

\[
(A \subseteq B) \iff (\forall x \in X, (x \in A) \Rightarrow (x \in B))
\]

which shows the close relationship between inclusion and implication. For this reason, we first present how the usual (material) implication extends to a fuzzy framework, before moving to several types of extension of the inclusion.

3.4.1 Fuzzy implications

3.4.1.1 Introduction

Usually, the implication \((P \Rightarrow Q)\) expresses a link between the values \( p \) and \( q \) of the propositions \( P \) and \( Q \). It returns “true” when proposition \( Q \) is true or when proposition \( P \) is false, according to the truth table reported in Table 3.5. This table is also associated with the expression \( ((\neg P) \lor Q) \). Several approaches to extending the implication can be envisaged when propositions \( P \) and/or \( Q \) are no longer Boolean, but instead take their values in the unit interval. Any fuzzy implication is a function defined as:

\[
\Rightarrow_\ell: [0, 1] \times [0, 1] \rightarrow [0, 1]
\]

\[
(p, q) \rightarrow (p \Rightarrow_\ell q),
\]

and it is increasingly true (resp. false) as its result approaches 1 (resp. 0). Fuzzy implications can be presented according to various classifications; in what follows they are divided into three families covering the most frequent and useful ones. Their principal characteristics are discussed along with their semantics, in order to highlight their rational usage.

<table>
<thead>
<tr>
<th>( P )</th>
<th>( Q )</th>
<th>( P \Rightarrow Q )</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>
3.4.1.2 S-implications

This name stands for **strong** implications. Their starting point is the expression of the material implication as:

\((\neg P) \lor Q\),

and the class of S-implications (denoted by \(\Rightarrow_S\)) is defined as follows:

\[ P \Rightarrow_S Q = \perp(1 - p, q). \]  

(3.3)

where the disjunction (“or”) is generalized by a co-norm \(\perp\). There is an infinity of S-implications: the three most common are given in Table 3.6. These implications are their own contraposition, since from Expression (3.3):

\[ \neg Q \Rightarrow_S \neg P = \perp(1 - q, 1 - p) = \perp(q, 1 - p) = \perp(1 - p, q) = P \Rightarrow_S Q. \]

As for the material implication, one has:

\[ (P \Rightarrow_S \text{false}) = \perp(1 - p, 0) = 1 - p. \]

Furthermore, the S-implications of Table 3.6 can be ranked as follows:

\[ (P \Rightarrow_{KD} Q) \leq (P \Rightarrow_{Rb} Q) \leq (P \Rightarrow_{Lu} Q). \]

and the Kleene–Dienes implication is the smallest among S-implications, because it is built using the smallest co-norm. The largest S-implication, denoted by \(\Rightarrow_{SM}\), is found when Weber’s co-norm is taken in Expression (3.3), which yields:

\[ P \Rightarrow_{SM} Q = \begin{cases} 
(1 - p) & \text{if } q = 0, \\
q & \text{if } p = 1, \\
1 & \text{otherwise.}
\end{cases} \]

All the S-implications \(P \Rightarrow_S Q\) are lower bounded by \((1 - p)\).

<table>
<thead>
<tr>
<th>S-implication</th>
<th>Definition</th>
<th>Underlying co-norm (\perp)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kleene–Dienes</td>
<td>(P \Rightarrow_{KD} Q = \max(1 - p, q))</td>
<td>(\perp(u, v) = \max(u, v))</td>
</tr>
<tr>
<td>Reichenbach</td>
<td>(P \Rightarrow_{Rb} Q = 1 - p + pq)</td>
<td>(\perp(u, v) = u + v - uv)</td>
</tr>
<tr>
<td>Lukasiewicz</td>
<td>(P \Rightarrow_{Lu} Q = \min(1 - p + q, 1))</td>
<td>(\perp(u, v) = \min(u + v, 1))</td>
</tr>
</tbody>
</table>
3.4.1.3 R-implications

The second class of fuzzy implications contains the R-implications, whose name derives from the residuation principle. Usually:

\[(P \text{ and } (P \Rightarrow Q)) \Leftrightarrow (P \text{ and } (\neg P \text{ or } Q)) \Leftrightarrow ((P \text{ and } \neg P) \text{ or } (P \text{ and } Q)) \Leftrightarrow (P \text{ and } Q),\]

and then the truth value of the proposition \((P \text{ and } (P \Rightarrow Q))\) is upper-bounded by that of proposition \(Q\), i.e., \((p \text{ and } (p \Rightarrow q)) \leq q\). In the fuzzy case, this inequality serves as the basis for searching for the maximal element for which it holds:

\[(P \Rightarrow R Q) = \sup \{u \in [0,1] | \top(p,u) \leq q\}, \quad (3.4)\]

\(\top\) denoting a triangular norm. Of course, there is an infinite number of R-implications: they can be rewritten as:

\[P \Rightarrow R Q = \begin{cases} 1 & \text{if } p \leq q, \\ f(p,q) & \text{otherwise}, \end{cases}\]

where \(f(p,q)\) is specific to the considered R-implication. The commonest R-implications are given in Table 3.7, along with their associated generative norms. The Rescher–Gaines implication may be considered as a specific R-implication which takes fuzzy propositions as inputs and delivers a Boolean result. It derives from a formula close to Expression (3.4), namely:

\[(P \Rightarrow_{RG} Q) = \sup \{u \in \{0,1\} | \top(p,u) \leq q\} = 1 \text{ if } p \leq q, 0 \text{ otherwise}\]

whatever the norm \(\top\). These implications can be ranked as follows:

\[(P \Rightarrow_{RG} Q) \leq (P \Rightarrow_{Gd} Q) \leq (P \Rightarrow_{Gg} Q) \leq (P \Rightarrow_{Lu} Q).\]

Gödel’s implication is the smallest proper R-implication (i.e., excluding that of Rescher–Gaines), because it is built using the largest norm (the “minimum”). It is worth noting that Lukasiewicz’s implication is both an

<table>
<thead>
<tr>
<th>name</th>
<th>definition</th>
<th>underlying norm (\top)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gödel</td>
<td>(P \Rightarrow_{Gd} Q = 1 \text{ if } p \leq q, q \text{ otherwise})</td>
<td>(\top(u,v) = \min(u,v))</td>
</tr>
<tr>
<td>Goguen</td>
<td>(P \Rightarrow_{Gg} Q = 1 \text{ if } p \leq q, q \text{ otherwise})</td>
<td>(\top(u,v) = u \ast v)</td>
</tr>
<tr>
<td>Lukasiewicz</td>
<td>(P \Rightarrow_{Lu} Q = 1 \text{ if } p \leq q, 1 - p + q \text{ otherwise})</td>
<td>(\top(u,v) = \max(u + v - 1,0))</td>
</tr>
</tbody>
</table>
R-implication and an S-implications, since its expression given in Table 3.7 is such that:

\[ P \Rightarrow_{Lu} Q = \begin{cases} 1 & \text{if } p \leq q, \\ 1 - p + q & \text{otherwise} \end{cases} = \min(1 - p + q, 1), \]

this latter expression being the one given in Table 3.6. Any R-implication may be reformulated in a format close to that of an S-implication by replacing the co-norm appearing in Expression (3.3) by an appropriate operator. This format is used in Chapter 6. As an illustration, we now deal with Gödel’s and Goguen’s implications. Gödel’s implication \((P \Rightarrow_{Gd} Q = 1 \text{ if } p \leq q, q \text{ otherwise})\) must be rewritten:

\[ P \Rightarrow_{Gd} Q = \text{op}(1 - p, q). \]

Then:

\[ \text{op}(1 - p, q) = \begin{cases} 1 & \text{if } p \leq q, \\ q & \text{otherwise}, \end{cases} \]

which leads to:

\[ \text{op}(u, v) = \begin{cases} 1 & \text{if } u + v \geq 1, \\ v & \text{otherwise}. \end{cases} \]

This operator returns 0 if and only if \(u\) and \(v\) both equal 0 and 1 when \(u\), or \(v\), or both equal 1, which is identical to the behavior of the regular disjunction. Of course, it is noncommutative; this is the reason it is called a noncommutative disjunction \((\text{ncd}_{Gd})\). A dual (in the sense of the De Morgan laws) noncommutative conjunction \((\text{ncc}_{Gd})\) can be associated with this noncommutative disjunction, allowing the following to be stated:

\[ P \Rightarrow_{Gd} Q = \text{ncd}_{Gd}(1 - p, q) = 1 - \text{ncc}_{Gd}(p, 1 - q), \]

where:

\[ \text{ncc}(u, v) = 0 \text{ if } u + v \leq 1, \text{ otherwise}. \]

The same holds for any other R-implication (Dubois and Prade, 1984), for instance with Goguen’s implication:

\[ P \Rightarrow_{Gg} Q = \text{ncd}_{Gg}(1 - p, q) = 1 - \text{ncc}_{Gg}(p, 1 - q), \]
where:

\[ ncd_{G_R}(u, v) = \begin{cases} 
1 & \text{if } u + v \geq 1, \\
\frac{v}{u} & \text{otherwise,}
\end{cases} \]

\[ ncc_{G_R}(u, v) = \begin{cases} 
0 & \text{if } u + v \leq 1, \\
\frac{u + v - 1}{u} & \text{otherwise.}
\end{cases} \]

Similarly, any S-implication can be rewritten in the format of an R-implication, by replacing the norm with the appropriate noncommutative conjunction in Expression (3.4). In fact, the close connections established in Dubois and Prade (1984) are summarized in Figure 3.5.

### 3.4.1.4 Contraposition of R-implications

The third family of implications correspond to the contraposition of R-implications which turn out not to be their own contrapositions (except Lukasiewicz’s implication, which is also an S-implication). Their general definition is:

\[ (P \Rightarrow_{\text{contrapR}} Q) = ((\neg Q) \Rightarrow_{R} (\neg P)). \]

![Fig. 3.5 Connections between norms, co-norms, implications, noncommutative conjunctions, and disjunctions](image)
In particular:

\[ P \Rightarrow_{\text{contrapGd}} Q = \begin{cases} 1 & \text{if } p \leq q, \\ (1 - p) & \text{otherwise}, \end{cases} \]

\[ P \Rightarrow_{\text{contrapGg}} Q = \begin{cases} 1 & \text{if } p \leq q, \\ 1 - q & \text{otherwise}, \end{cases} \]

Notice that the Rescher–Gaines implication is its own contraposition, since:

\[ (\neg Q) \Rightarrow_{\text{RG}} (\neg P) = \begin{cases} 1 & \text{if } p \leq q, \\ 0 & \text{otherwise} \end{cases} = (P \Rightarrow_{\text{RG}} Q). \]

3.4.1.5 Some characteristics of fuzzy implications

All of the fuzzy implications presented above generalize the usual material implication; i.e., when the input values are 0 or 1, their result coincides with that of the regular implication. Indeed, since 1 is an absorbing element and 0 a neutral element, for any S-implication one has:

- \((0 \Rightarrow_S 0) = \bot(1, 0) = 1,\)
- \((1 \Rightarrow_S 1) = \bot(0, 1) = 1,\)
- \((1 \Rightarrow_S 0) = \bot(0, 0) = 0,\)
- \((0 \Rightarrow_S 1) = \bot(1, 1) = 1.\)

In a similar way, for R-implications and their contrapositions:

- \((0 \Rightarrow_R 0) = (0 \Rightarrow_R 1) = (1 \Rightarrow_R 1) = 1 \text{ since the first argument is less than (or equal to) the second one,}\)
- \((1 \Rightarrow_R 0) = \sup \{ u \in [0, 1] \mid \top(1, u) \leq 0 \} = \sup \{ u \in [0, 1] \mid u \leq 0 \} = 0.\)

These last two properties also hold for the Rescher–Gaines implication.

The three classes of fuzzy implications considered so far possess the following properties according to the requirements described in Yager (1980):

- \(\forall p \in [0, 1]: p \Rightarrow_t 1 = 1,\)
- \(\forall q \in [0, 1]: 0 \Rightarrow_t q = 1,\)
- \(\forall q \in [0, 1]: 1 \Rightarrow_t q = q,\)
- \(\forall p, q_1, q_2 \in [0, 1]: \text{if } q_1 > q_2, (p \Rightarrow_t q_1) \geq (p \Rightarrow_t q_2),\)
- \(\forall p_1, p_2, q \in [0, 1]: \text{if } p_1 < p_2, (p_1 \Rightarrow_t q) \geq (p_2 \Rightarrow_t q).\)
Some fuzzy implications can operate in a purely ordinal setting, i.e., it is not mandatory to be provided with numeric truth values in the unit interval and an ordered scale suffices. This is the case for Kleene–Dienes’, Gödel’s and Rescher–Gaines’ implications (see Example 3.7), but not for those of Goguen, Reichenbach and Lukasiewicz. Indeed, the first three implications depend only on comparison operations and scale reversing (associated with the negation), while the other three need arithmetic operations.

Example 3.7. Consider the following ordered symbols $A < B < C < D < E$, where $A$ (resp. $E$) is the minimal (resp. maximal) element corresponding to 0 (resp. 1) in the unit interval. For the sake of simplicity, we use $A \Rightarrow f B$ to represent the implication $P \Rightarrow f Q$ when $A$ is the value of $P$ and $B$ is that of $Q$. Table 3.8 gives the result obtained for a sample of situations.

<table>
<thead>
<tr>
<th></th>
<th>$A \Rightarrow f D$</th>
<th>$B \Rightarrow f E$</th>
<th>$C \Rightarrow f D$</th>
<th>$B \Rightarrow f A$</th>
<th>$D \Rightarrow f B$</th>
<th>$E \Rightarrow f B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kleene–Dienes</td>
<td>$E$</td>
<td>$E$</td>
<td>$D$</td>
<td>$D$</td>
<td>$B$</td>
<td>$B$</td>
</tr>
<tr>
<td>Gödel</td>
<td>$E$</td>
<td>$E$</td>
<td>$E$</td>
<td>$A$</td>
<td>$B$</td>
<td>$B$</td>
</tr>
<tr>
<td>Rescher–Gaines</td>
<td>$E$</td>
<td>$E$</td>
<td>$E$</td>
<td>$A$</td>
<td>$A$</td>
<td>$A$</td>
</tr>
</tbody>
</table>

3.4.1.6 Semantic aspects of fuzzy implications

It is important to pay attention to the meaning which can be associated with the various fuzzy implications. This allows in particular for an appropriate choice in the context of a given domain of application (such as the division operation tackled later in Chapter 6). S-implications (in particular, those of Kleene–Dienes, Reichenbach and Lukasiewicz) have the characteristic of guaranteeing a minimal level of satisfaction ($P \Rightarrow_S Q$ returns a value at least equal to $(1 - p)$) as illustrated in Figures 3.6 and 3.7. Moreover, in the case of the Kleene–Dienes implication, the antecedent $P$ acts as an important level assigned to the conclusion $Q$. Indeed, when $p$, the value of $P$, is low, only the high values of $Q$ are “seen” (i.e., have an impact on the result of the implication), in contrast to the case of a high value of $p$ for which $Q$ is “seen” over a wide range of values (the interval $[1 - p, 1]$ which is large if $p$ is small). As to R-implications, the value of the antecedent $P$ has the role of a threshold, and full satisfaction is reached when the value
of the conclusion $Q$ is greater than (or equal to) $p$. Otherwise, the level of satisfaction is less than 1, and:

- depends on the sole conclusion with Gödel’s implication,
- varies with the ratio between the value of the conclusion $Q$ and that of the antecedent $P$ ($\frac{q}{p}$) using Goguen’s implication,
- is all the lower as the difference between the values of the antecedent and the conclusion increases with Łukasiewicz’s implication.

**Example 3.8.** Table 3.9 summarizes the results obtained with the Rescher–Gaines, Gödel, Goguen, and Łukasiewicz implications for a sample of situations. It may be seen that these results may be close (or even equal), or on the contrary quite different values. Moreover, it can be observed that they comply with the properties described previously, in particular the order over these four implications given in Section 3.4.1.3.
3.4.2 Inclusions

3.4.2.1 Introduction

It may happen that sets are compared in terms of their inclusion or equality. Usually, this type of comparison leads to a Boolean answer, and the fuzzy set framework suggests a triple approach:

- to define these operations when the operands are fuzzy sets,
- to deliver a graded result instead of a binary one, whatever the type (fuzzy or not) of the operands,
- to allow for some tolerance in the inclusion (or equality) mechanism, in the sense that exceptions may be more or less acceptable.

3.4.2.2 Regular inclusion between fuzzy sets

The usual definition of the inclusion of $A$ in $B$ given by Expression (3.2) can also be written in terms of a constraint on the characteristic functions of $A$ and $B$ as:

$$(A \subseteq B) \Leftrightarrow (\forall x \in X, f_A(x) \leq f_B(x)). \quad (3.5)$$

This expression extends in a canonical way to two fuzzy sets $E$ and $F$ by:

$$(E \subseteq F) \Leftrightarrow (\forall x \in X, \mu_E(x) \leq \mu_F(x)), \quad (3.6)$$

which is often called “inclusion in the sense of Zadeh”. Figures 3.8 and 3.9 illustrate the cases of inclusion and non-inclusion according to Expression (3.6). It is notable that Expression (3.6) may serve as a basis for defining the crisp equality of two fuzzy sets $E$ and $F$:

$$(E = F) \Leftrightarrow (\forall x \in X, \mu_E(x) = \mu_F(x)). \quad (3.7)$$

Starting from Expression (3.2), which calls on an implication, one may define a more demanding inclusion, namely:

$$(E \subseteq F) \Leftrightarrow (\forall x \in X, (x \in \text{supp}(E)) \Rightarrow (x \in \text{core}(F))). \quad (3.8)$$
Figure 3.10 illustrates the inclusion of $E$ in $F$ according to Expression (3.8). The major drawback of such definitions stems from the fact that a negative answer is obtained when the inclusion of $E$ in $F$ (for regular or fuzzy set) is “obviously” violated, but also when it almost holds. More generally, this phenomenon may be interpreted as the inability to distinguish between quite different situations. This is obviated by using a graded inclusion.

**Example 3.9.** The fuzzy sets $E_1 = \{1/a, 1/b\}$ and $E_2 = \{0.5/a, 0.7/b\}$ are not included in the fuzzy set $F = \{0.4/a, 0.7/b\}$ using Expression (3.6). However, intuition may suggest that it is definitely more obvious for $E_1$, which certainly contains $F$, than for $E_2$, which is “almost included” in $F$.\(\diamond\)
3.4.2.3 Graded inclusion

Devising an inclusion whose result is a matter of degree allows for finer accounting than with an “all or nothing” view. By using a degree, a graded inclusion extends the notion of inclusion of one set (fuzzy or not) in another. Here again, several approaches may be taken.

Cardinality-based view of graded inclusion. Taking the following definition of the regular inclusion of $A$ in $B$:

$$(A \subseteq B) \iff ((A \cap B) = A),$$

it follows that:

$$(A \subseteq B) \iff (\text{card}(A \cap B) = \text{card}(A)).$$

It is then possible to define a graded inclusion through the degree of inclusion, defined as:

$$deg(A \subseteq B) = \frac{\text{card}(A \cap B)}{\text{card}(A)}.$$

In this way, the inclusion becomes gradual by evaluating the proportion of elements in the intersection of $A$ and $B$ with respect to those present in $A$. More generally, if $E$ and $F$ are two fuzzy sets, the degree of inclusion of $E$ in $F$ is given by:

$$deg(E \subseteq F) = \frac{\text{card}(E \cap F)}{\text{card}(E)} = \frac{\sum_{x \in X} \top(\mu_E(x), \mu_F(x))}{\sum_{x \in X} \mu_E(x)}, \quad (3.9)$$

where $\top$ stands for a triangular norm. Expression (3.9) confers a quantitative (as opposed to logical) nature on the obtained degree. When the
“minimum” is taken as the norm $\top$ in Expression (3.9), the grades associated with $E$ play the role of a threshold. If an element $x$ attains this value in $F$ ($\mu_F(x) \geq \mu_E(x)$), the contribution of $x$ to the overall degree of inclusion is maximal (the case of elements $a$ and $b$ in Example 3.10). Analogously, the choice of the “product” leads to:

$$\text{deg}(E \subseteq F) = \frac{\sum_{x \in X} \mu_E(x) \cdot \mu_F(x)}{\sum_{x \in X} \mu_E(x)},$$

which corresponds to a weighted mean where the degrees $\mu_E(x)$ are the weights and play the role of a level of importance assigned to $x$.

In general, if $E$ is included in $F$ in the sense of Zadeh (Expression (3.6)), the degree of inclusion does not equal 1 except if the norm used in Expression (3.9) is the minimum (see Example 3.10).

**Example 3.10.** Consider the fuzzy sets $E = \{0.4/a, 0.6/b\}$ and $F = \{0.5/a, 1/b\}$, which satisfies $E \subseteq F$. Using Expression (3.9) with the norm “minimum”, one has:

$$\text{deg}(E \subseteq F) = \min(0.4, 0.5) + \min(0.6, 1) = 0.4 + 0.6 = 1,$$

whereas with the product, one obtains:

$$\text{deg}(E \subseteq F) = (0.4 \cdot 0.5) + (0.6 \cdot 1) = 0.8.$$  

**Logical view of graded inclusion.** It is also possible to adopt a logical approach to graded inclusion starting with Expression (3.2), which extends to fuzzy sets as follows:

$$\text{deg}(E \subseteq F) = \min_{x \in X} (\mu_E(x) \Rightarrow \mu_F(x)),$$

⇒ being a fuzzy implication (see Section 3.4.1).

It turns out that the nature of the inclusion obtained differs depending on the type of fuzzy implication used in Expression (3.10). In particular, if $E$ is included in $F$ according to Zadeh (Expression (3.6)), the degree of inclusion is 1 for any R-implication (or its contraposition), but it is less than 1 when an S-implication is used. For instance, with the Kleene–Dienes or Reichenbach implication, degree 1 is obtained if and only if the support of $E$ is included in the core of $F$, as it is for a non-graded inclusion defined in Expression (3.8).
Example 3.11. Consider the following fuzzy sets:

\[ E_1 = \{0.4/a, 0.7/b\}, E_2 = \{1/a, 0.9/b, 0.2/c\}, F = \{1/a, 1/b, 0.9/c\} \]

According to Expression (3.10), for \( i = 1 \) or \( 2 \), we have:

\[
\text{deg}(E_i \subseteq F) = \min(\mu_{E_i}(a) \Rightarrow_{\ell} \mu_F(a), \mu_{E_i}(b) \Rightarrow_{\ell} \mu_F(b), \mu_{E_i}(c) \Rightarrow_{\ell} \mu_F(c)),
\]

which, for the Kleene–Dienes implication, leads to:

\[
\text{deg}(E_1 \subseteq F) = \min(0.4 \Rightarrow_{KD} 1, 0.7 \Rightarrow_{KD} 1) = 1,
\]

\[
\text{deg}(E_2 \subseteq F) = \min(1 \Rightarrow_{KD} 1, 0.9 \Rightarrow_{KD} 1, 0.2 \Rightarrow_{KD} 0.9) = 0.9,
\]

while using Gödel’s implication, one obtains:

\[
\text{deg}(E_1 \subseteq F) = \min(0.4 \Rightarrow_{Gd} 1, 0.7 \Rightarrow_{Gd} 1) = 1,
\]

\[
\text{deg}(E_2 \subseteq F) = \min(1 \Rightarrow_{Gd} 1, 0.9 \Rightarrow_{Gd} 1, 0.2 \Rightarrow_{Gd} 0.9) = 1.0
\]

Difference-based view of graded inclusion. A third type of expression for the inclusion of \( A \) in \( B \) relies on the set difference operation:

\[ A \subseteq B \Leftrightarrow (A - B) = \emptyset, \]

which extends to two fuzzy sets \( E \) and \( F \) as follows:

\[ \text{deg}(E \subseteq F) = 1 - h(E - F), \quad (3.11) \]

where \( h(G) \) denotes the height of the fuzzy set \( G \), i.e., the highest degree of membership attached to its elements. If one uses the “standard” definition of the difference given in Expression (3.1), one obtains:

\[
\text{deg}(E \subseteq F) = 1 - \max_{x \in X} (\mu_E(x), 1 - \mu_F(x))
\]

\[
= \min_{x \in X} 1 - (\mu_E(x), 1 - \mu_F(x))
\]

\[
= \min_{x \in X} (1 - \mu_E(x), \mu_F(x)).
\]

This last expression is exactly that of the degree of inclusion given by Expression (3.10) when \( \Rightarrow_{\ell} \) is replaced by an S-implication. This proves that the logical approach based on S-implications also captures the semantics of a degree of inclusion built from the “standard” difference between fuzzy sets.

Example 3.12. Consider the two fuzzy sets \( E = \{1/a, 0.6/b, 0.3/c\} \) and \( F = \{0.7/a, 0.4/b, 1/c\} \). The degree of inclusion of \( E \) in \( F \) using
Basic Notions on Fuzzy Sets

Reichenbach’s implication $(P \Rightarrow_{R_b} Q = 1 - p + pq)$ in Expression (3.10) is:

$$\text{deg}(E \subseteq F) = \min(1 \Rightarrow_{R_b} 0.7, 0.6 \Rightarrow_{R_b} 0.4, 0.3 \Rightarrow_{R_b} 1)$$

$$= \min(0.7, 0.64, 1)$$

$$= 0.64.$$  

The difference of $E$ and $F$ with the norm “product” in Expression (3.1) is $\{0.3/a, 0.36/b\}$ and $h(E - F) = 0.36$. The degree of inclusion founded on Expression (3.11) then equals $(1 - 0.36) = 0.64$ as obtained before. □

3.4.2.4 Graded equality of fuzzy sets

The equality of two sets is usually expressed as either the equality of their respective characteristic functions or their mutual inclusion. Based on Expression (3.7), the first approach extends to fuzzy sets, but its result is binary (0/1 or alternatively false/true). As we did with the inclusion, it may be convenient to define the notion of degree of equality of two sets as:

$$\text{deg}(E = F) = \top(\text{deg}(E \subseteq F), \text{deg}(F \subseteq E)),$$

where $\top$ stands for a norm, and the two degrees of inclusion are obtained using Expressions (3.9) or (3.10).

Example 3.13. Let $E$ and $F$ be the sets $E = \{1/a, 0.4/b, 0.6/c\}$ and $F = \{0.4/a, 0.7/b, 0.5/c\}$. Using Expression (3.9) with the norm “minimum”, one has:

$$\text{deg}(E \subseteq F) = \frac{\min(1, 0.4) + \min(0.4, 0.7) + \min(0.6, 0.5)}{1 + 0.4 + 0.6} = 0.65$$

$$\text{deg}(F \subseteq E) = \frac{\min(1, 0.4) + \min(0.4, 0.7) + \min(0.6, 0.5)}{0.4 + 0.7 + 0.5} = 0.8125$$

and finally:

$$\text{deg}(E = F) = \min(0.65, 0.8125) = 0.65.$$  

With the norm “product”, the grades would be:

$$\text{deg}(E \subseteq F) = \frac{(1 \ast 0.4) + (0.4 \ast 0.7) + (0.6 \ast 0.5)}{1 + 0.4 + 0.6} = 0.49$$

$$\text{deg}(F \subseteq E) = \frac{(1 \ast 0.4) + (0.4 \ast 0.7) + (0.6 \ast 0.5)}{0.4 + 0.7 + 0.5} = 0.6125$$

$$\text{deg}(E = F) = \min(0.49, 0.6125) = 0.38.$$
The fact that elements \( a \) and \( c \) somewhat violate the inclusion of \( E \) in \( F \) has a limited impact in both cases, due to the compensation which takes place in these formulas. This would hold even with elements in total contradiction with the inclusion. On the contrary, the use of Expression (3.10), where a norm occurs, will lead to a quite different behavior. Indeed, a low value resulting from the implication for a given element \( x \) will become an upper bound for the degree of inclusion, then that of equality. For instance, using Gödel’s implication in Expression (3.10), one obtains:

\[
\begin{align*}
\text{deg}(E \subseteq F) &= \min(1 \Rightarrow_{\text{Gd}} 0.4, 0.1 \Rightarrow_{\text{Gd}} 0.6, 0.6 \Rightarrow_{\text{Gd}} 0.5) \\
&= \min(0.4, 1, 0.5) = 0.4, \\
\text{deg}(F \subseteq E) &= \min(0.4 \Rightarrow_{\text{Gd}} 1, 0.6 \Rightarrow_{\text{Gd}} 0.1, 0.5 \Rightarrow_{\text{Gd}} 0.6) \\
&= \min(1, 0.1, 1) = 0.1, \\
\text{deg}(E = F) &= \min(0.4, 0.1) = 0.1. \\
\end{align*}
\]

### 3.4.2.5 Tolerant inclusion

An alternative extension of inclusion is based on some tolerance of exceptions. This can be understood in two different ways depending on the nature of the exceptions, which can be either quantitative or qualitative. More details on tolerant inclusions can be found in Bosc and Pivert (2006, 2008), especially regarding their properties.

**Quantitative tolerant inclusion.** Some tolerance may be introduced on the basis of a number of exceptions with respect to the universal quantifier implicitly present in Expression (3.10). The definition of a quantitative exception-tolerant inclusion is based on ignoring the fact that some elements of \( E \) are not sufficiently, or even not at all, included in \( F \) (depending on the fuzzy implication taken). In other words, a certain number of values of \( E \) can be more or less ignored depending on the desired level of relaxation. The principle adopted is to weaken the universal quantifier into a relaxed quantifier “almost all”; see Kerre and Liu (1998), Zadeh (1983). In the following, the quantifier “almost all” induces grades defined as follows:

- \( \mu_{\text{almost all}}(0) = w_n = 0 \),
- \( \mu_{\text{almost all}}(1) = w_0 = 1 \),
- \( \mu_{\text{almost all}}(1 - \frac{i}{n}) = w_i \).
Grade $w_i$ expresses the degree of satisfaction when $i$ out of the $n$ elements are ignored. If we denote:

$$k_1 = \max\{j \mid w_j = 1\}, k_2 = \max\{j \mid w_j > 0\},$$

the quantifier “almost all” allows for the total ignoring of $k_1$ exceptions and the partial ignoring of up to $k_2$ exceptions. Basically, the idea is to search for the best compromise such that $k$ elements of $E$ are included in $F$ and $k$ is compatible with “almost all”, which leads to the following definition:

$$\deg(E \subseteq_{\text{qttol}} F) = \max_{k \in [1, n]} \min(\alpha_k, w_{k-1})$$

$$= \min_{k \in [1, n]} \max(\alpha_k, w_k), \quad (3.12)$$

where $\alpha_k$ is the $k$th smallest implication degree ($\mu_E(x) \Rightarrow \mu_F(x)$) and $w_k = \mu_{\text{almost all}}(1 - \frac{k}{n})$ is the degree of ignoring generated by the quantifier “almost all” for $E$ whose support has $n$ elements, i.e., $n$ implication values intervene in Expression (3.12). It is worth noting that:

- Expression (3.10) is recovered from Expression (3.12) when the universal quantifier is used ($w_0 = 1, w_1 = \cdots = w_n = 0$), i.e., when no exception is admitted,
- as expected the result obtained is at least equal to that returned by the non-tolerant inclusion,
- Expression (3.12) establishes a clear connection with Expression (3.10) since, when $w_i$ is 1, total ignoring takes place, whereas for $w_i = 0$, the associated element is completely taken into account as it stands.

**Example 3.14.** Consider the quantifier “almost all” defined as:

$$\mu_{\text{almost all}}(f) = \begin{cases} 
0 & \text{if } f \in [0, 0.75], \\
1 & \text{if } f \in [0.95, 1], \\
\text{linearly increasing} & \text{if } f \in [0.75, 0.95].
\end{cases}$$

Moreover, let

$$E = \{1/a, 0.9/b, 0.9/c, 0.9/d, 0.9/f, 0.8/g, 0.7/h, 0.4/i, 0.2/j, 0.1/k\}$$

and

$$F = \{0.1/b, 0.2/c, 0.5/d, 0.7/e, 0.9/f, 1/g, 1/h, 0.2/i, 0.5/j\}.$$ 

The degrees generated by the above quantifier are $w_0 = 1, w_1 = 0.75, w_2 = 0.25, w_3 = \cdots = w_{10} = 0$, since $E$ contains 10 tuples. The non-tolerant graded inclusion of $E$ in $F$ using Gödel’s implication leads to the degree 0,
since a is missing in F. On the contrary, when the tolerant inclusion is performed with Expression (3.12), the grade obtained is:

\[
\min(\max(0, 0.75), \max(0.1, 0.25), \max(0.2, 0), \max(0.5, 0), \\
\max(0.7, 0), \max(1, 0), \ldots, \max(1, 0)) \\
= \max(\min(0, 1), \min(0.1, 0.75), \min(0.2, 0.25), \min(0.5, 0), \\
\min(0.7, 0), \min(1, 0), \ldots, \min(1, 0)) = 0.2.
\]

The two lowest implication values 0 and 0.1 are ignored thanks to \(w_1\) and \(w_2\).

**Qualitative tolerant inclusion.** In the previous case, the quantitative inclusion of \(E\) in \(F\) expresses the idea that “almost all elements of \(E\) are included in \(F\) according to the chosen implication”. If a qualitative view is adopted, the inclusion operator expresses the idea that “all elements of \(E\) are almost included in \(F\) according to the chosen implication”. Exceptions are then taken into account according to the idea of “almost inclusion”. In other words, the idea is to (more or less) compensate for the initial value of the implication when it expresses a sufficiently low-intensity exception.

The principle is to consider exceptions with respect to the inclusion in the following sense: if one looks for the inclusion of \(E\) in \(F\), compensation takes place for an element \(x\) such that \(\mu_E(x)\) and \(\mu_F(x)\) are sufficiently close to the case of “full” inclusion. It seems reasonable to consider that closeness is a matter of degree; for instance, one may think that if we must be close to \(a\), \(a \pm 0.1\) is totally acceptable, but a shift beyond 0.3 cannot be tolerated, and the compensation is linear in between. Of course, this does not make sense for regular sets, for which exceptions correspond to the case where \(\mu_E(x)\) equals 1 and \(\mu_F(x)\) is 0: this is definitely not a case of low-intensity exception.

Using any R-implication, the intensity of an exception depends on the difference between \(\mu_E(x)\) and \(\mu_F(x)\). Intuitively, if this difference is positive but “small enough”, we are in the presence of a low-intensity exception, which is somewhat tolerable. Here, the compensation is based on a change in the computation of the implication degree itself, where the conclusion part may be upgraded in the presence of a low-intensity exception. The definition of the qualitative exception-tolerant inclusion is then very similar in form to Expression (3.10), namely:

\[
\text{deg}(E \subseteq_{\text{qtolR}} F) = \min_{x \in X} (\mu_E(x) \Rightarrow_R (\mu_F(x)) + \delta) \quad (3.13)
\]
where
\[ \delta = \begin{cases} 
0 & \text{if } \mu_E(x) - \mu_F(x) \geq \beta, \beta \in [0, 1], \\
\mu_E(x) - \mu_F(x) & \text{if } \mu_E(x) - \mu_F(x) \leq \alpha, \alpha \in [0, 1] \text{ and } \alpha < \beta, \\
\text{linear in between.} & 
\end{cases} \]

**Example 3.15.** Consider \( E = \{1/a, 0.1/b, 0.6/c\} \) and \( F = \{0.7/a, 0.4/c\} \).

The degree of inclusion of these two sets using Gödel’s implication in Expression (3.10) is 0, due to the absence of element \( b \) in \( F \). If the qualitative exception-tolerant inclusion (Expression (3.13)) is performed with \( \alpha = 0.1 \) and \( \beta = 0.3 \), the result is:

\[
\min(1 \Rightarrow_{Gd} 0.7, 0.1 \Rightarrow_{Gd} 0 + 0.1, 0.6 \Rightarrow_{Gd} 0.4 + 0.05) = \min(0.7, 1, 0.45) = 0.45.
\]

With Goguen’s implication, the degree would be \( \min \left( 0.7, 1, \frac{0.45}{0.6} \right) = 0.7 \).

In the case of an S-implication \( (P \Rightarrow_S Q) \), the idea is to say that if \( p \) is “small enough” or \( q \) is “high enough”, then a low-intensity exception occurs and the implication value can be upgraded to 1. On the other hand, if \( p \) is “not at all small enough” and \( q \) is “not at all high”, then no compensation should take place, since the element is too big an exception. In between these two extreme situations, some partial compensation can take place.

If \( \alpha \) and \( \beta \) denote two thresholds \( (0 < \alpha < \beta) \), the interval \( [0, \alpha] \) represents “small enough” and the interval \( [\alpha, \beta] \) corresponds to “somewhat small”, then \( [1 - \alpha, 1] \) represents “high enough” and \( [1 - \beta, 1 - \alpha] \) represents “somewhat high”.

In this context, the definition of the qualitative exception-tolerant inclusion is analogous to Expression (3.13), i.e.:

\[
de\text{deg}(E \subseteq_{qtolS} F) = \min_{x \in X} ((\mu_E(x) - \delta_1) \Rightarrow_S (\mu_F(x)) + \delta_2),
\]

where
\[
\delta_1 = \begin{cases} 
\mu_E(x) & \text{if } \mu_E(x) \leq \alpha, \\
0 & \text{if } \mu_E(x) \geq \beta, \\
\left(\frac{\alpha}{\beta - \alpha}\right) \ast (\beta - \mu_E(x)) & \text{in between}, 
\end{cases} \]
\[ \delta_2 = \begin{cases} 1 - \mu_F(x) & \text{if } \mu_F(x) \geq 1 - \alpha, \\ 0 & \text{if } \mu_F(x) \leq 1 - \beta, \\ \left( \frac{\alpha}{\beta - \alpha} \right) \times (\mu_F(x) + \beta - 1) & \text{in between.} \end{cases} \]

**Example 3.16.** Take the two sets \( E = \{0.7/a, 0.9/b, 0.1/c, 0.3/d\} \) and \( F = \{0.8/a, 0.85/b, 0.2/c, 0.75/d\} \). With the Kleene–Dienes implication, the degree of inclusion of \( E \) in \( F \) according to Expression (3.10) is:

\[
\min(\max(0.3, 0.8), \max(0.1, 0.85), \max(0.9, 0.2), \max(0.7, 0.75)) = 0.75.
\]

Using the qualitative tolerant inclusion in Expression (3.14), based on the Kleene–Dienes implication, and the thresholds \( \alpha = 0.2, \beta = 0.45 \), one obtains the degree:

\[
\min(\max(1 - (0.7 - 0), 0.8 + 0.2), \max(1 - (0.9 - 0), 0.85 + 0.15), \\
\max(1 - (0.1 - 0.1), 0.2 + 0), \max(1 - (0.3 - 0.12), 0.75 + 0.16)) = \min(\max(0.3, 1), \max(0.1, 1), \max(1, 0.2), \max(0.82, 0.91)) = 0.91.
\]

### 3.5 Fuzzy Measures and Integrals

#### 3.5.1 Introduction

This section deals with the notion of measure of the subsets of a finite universe \( X \). In the usual theory, it is considered that measure is an additive notion (e.g., the measure of probability). Furthermore, the Lebesgue integral allows for the extension of the notion of measure to a function \( f \) defined over \( X \). This integral calculates the surface delimited by a stepwise function; its value represents the extension of the concept of distance to the set of points defined by the function. However, it is possible to call the additivity hypothesis into question, since this may sometimes be too strong. This leads to the introducing of fuzzy measures (which should better be called *non-additive measures*), then fuzzy integrals which make use of such measures.

#### 3.5.2 Fuzzy measures

A fuzzy measure over a finite universe \( X \) is a function defined as:

\[ m : \mathcal{P}(X) \rightarrow [0, 1], \]
Basic Notions on Fuzzy Sets

where $\mathcal{P}(X)$ is the power set of $X$. It complies with the following three axioms:

- $m(X) = 1$,
- $m(\emptyset) = 0$,
- $\forall A, B$ such that $A \subseteq B, m(A) \leq m(B)$ (monotonicity).

The last axiom weakens the additivity property, which may be written as:

$$m(A \cup C) = m(A) + m(C).$$

Indeed, letting $B = A \cup C$ ($C \neq \emptyset$), it follows that $m(B) > m(A)$, which states that an additive measure is a fuzzy measure while the converse is not true.

**Example 3.17.** Consider the work associated with five tasks denoted respectively by $a, b, c, d$ and $e$, assumed to be independent and without any constraint as to their order of execution. The work is complete when the five tasks are done. The measure of the work that has been accomplished at a given time depends on the tasks that are finished (taken in the universe $X = \{a, b, c, d, e\}$) and is given by:

$$m : \mathcal{P}(X) \rightarrow [0, 1].$$

Thus, $m(A)$ is the state of the work if the tasks belonging to $A$ are finished. For instance, $m(\{b, c\}) = 0.7$ means that 70% of the total work is executed if tasks $b$ and $c$ are over. It is easy to check that $m$ is a fuzzy measure (i.e., that it complies with the previous three axioms) since the work is finished when all the tasks are done ($m(X) = 1$) and is not at all executed as long as a task is not done ($m(\emptyset) = 0$). In addition, when the number of finished tasks increases, the satisfaction cannot decrease then $A \subseteq B$ entails $m(A) \leq m(B)$. If all the tasks are equally important and they contribute equally to the final result, the measure is additive and the measure $m_1$ is defined as:

- $m_1(\{a\}) = m_1(\{b\}) = m_1(\{c\}) = m_1(\{d\}) = m_1(\{e\}) = 0.2$,
- $m_1(A) = \sum_{x \in A} m_1(\{x\})$.

Conversely, if all the tasks contribute to the final result in a non-additive fashion, the following measure $m_2$ can be used:

- for singletons:
  $$m_2(\{a\}) = m_2(\{b\}) = 0.3,$$
  $$m_2(\{c\}) = m_2(\{d\}) = m_2(\{e\}) = 0.2,$$
for subsets of two elements:
\[
\begin{cases}
\text{if } A = \{a, b\}, m_2(A) = 0.6, \\
\text{otherwise } \{\text{if } a \text{ or } b \text{ belongs to } A, m_2(A) = 0.4, \\
\text{otherwise } m_2(A) = 0.3,
\end{cases}
\]

for subsets of three elements:
\[
\begin{cases}
\text{if } \{a, b\} \subseteq A, m_2(A) = 0.8, \\
\text{otherwise } \{\text{if } a \text{ or } b \text{ belongs to } A, m_2(A) = 0.6, \\
\text{otherwise } m_2(A) = m_2(\{c, d, e\}) = 0.5,
\end{cases}
\]

for subsets of four elements:
\[
\begin{cases}
\text{if } \{a, b\} \subseteq A, m_2(A) = 0.9, \\
\text{otherwise } m_2(A) = 0.7.\nonumber
\end{cases}
\]

3.5.3 Fuzzy integrals
Integration with respect to a fuzzy measure may be performed using several operators (i.e., several integrals), the most usual being the Choquet fuzzy integral (Murofushi and Sugeno, 1989) and the Sugeno fuzzy integral (Sugeno, 1974). The calculus of the Choquet integral consists of using the fuzzy measure in the same way as an additive measure, which leads to:
\[
C_m(\mu_A) = \sum_{i=1}^{n} \mu_A(x_i) * (m(A_i) - m(A_{i+1})),
\]
where the elements \(\{x_1, \ldots, x_n\}\) are ranked in increasing order of their degree of membership of \(A (\mu_A(x_1) \leq \ldots \leq \mu_A(x_n))\), \(A_i = \{x_i, \ldots, x_n\}\) and \(A_{n+1} = \emptyset\). This corresponds to the Lebesgue integral when the measure is actually an additive one.

The Sugeno integral of function \(\mu\) from \(X\) to \([0,1]\) is given by:
\[
S_m(\mu_A) = \max_{\alpha \in [0,1]} \min(\alpha, m(A_\alpha)),
\]
where \(A_\alpha\) denotes the cut of level \(\alpha\) of the fuzzy set \(A\). Letting \(X\) be the set of elements \(\{x_1, \ldots, x_n\}\) ranked in increasing order of their degree of membership to \(A\), one obtains:
\[
S_m(\mu_A) = \max_{i \in [1,n]} \min(\mu_A(x_i), m(A_i)),
\]
with \(A_i = \{x_i, \ldots, x_n\}\). When \(A\) represents a property (or predicate) and \(m\) is a measure of satisfaction, the value \(S_m(\mu_A)\) expresses the extent to
which there is a satisfactory set $E$ (in the sense of $m$) such that all of its elements have the property:

$$S_m(\mu_A) = \max_{E \in \mathcal{P}(X)} \min(m(E), t(E)),$$

with $t(E) = \min_{x \in E} \mu_A(x)$ expressing the fact that every element of $E$ satisfies property $A$.

**Example 3.18.** Consider again the context of the previous example. Function $\mu$ from \{a, b, c, d, e\} to the unit interval expresses the percentage of the overall work performed by each task, and the measure $m$ states the percentage of the overall work completed depending on the tasks that have been finished. The assessment of the overall work achieved so far takes into account both $\mu$ and $m$, considering the most favorable set of tasks:

$$\max_{E \in \mathcal{P}(X)} \min(m(E), t(E)),$$

with $t(E) = \min_{x \in E} \mu_A(x)$. This assessment is exactly the Sugeno integral $S_m(\mu_A)$. Taking the values $\mu(a) = 0.8, \mu(b) = 0.7, \mu(c) = 0.3, \mu(d) = 0.2, \mu(e) = 0.1$, one obtains:

$$S_m(\mu) = \max(\min(\mu(c), m(e, d, c, b, a)), \min(\mu(d), m(d, c, b, a)),$$

$$\min(\mu(e), m(c, b, a)), \min(\mu(b), m(b, a)), \min(\mu(a), m(a))).$$

If the measure $m_1$ is chosen (tasks equally important), it follows that:

$$S_{m_1}(\mu) = \max(\min(0.1, 1), \min(0.2, 0.8), \min(0.3, 0.6),$$

$$\min(0.7, 0.4), \min(0.8, 0.2))$$

$$= \max(0.1, 0.2, 0.3, 0.4, 0.2) = 0.4,$$

whereas with measure $m_2$, the result is:

$$S_{m_2}(\mu) = \max(\min(0.1, 1), \min(0.2, 0.9), \min(0.3, 0.8),$$

$$\min(0.7, 0.6), \min(0.8, 0.3))$$

$$= \max(0.1, 0.2, 0.3, 0.6, 0.3) = 0.6.$$

### 3.6 The Extension Principle

In some situations, a fuzzy set may be considered to be an imperfect representation of a given quantity. Although manipulating a quantity by means
of operations is fairly easy when it is perfectly known, it raises the question of using these operators in the presence of an imprecise value. A typical example is that of real numbers which can be added: one can ask what happens for the addition of two imprecise numbers, also called fuzzy numbers (see Dubois and Prade, 1980). The extension principle defined by Zadeh (1975b,c) proposes an answer to this question.

Consider an application from a set $X$ to a set $Y$, which maps an element $x$ of $X$ onto an element $y$ of $Y$. This application is used to define the image of a fuzzy set $A$ of $X$ according to the following intuitive idea: if $x$ belongs to the fuzzy set $A$ of $X$ with the degree $d$, one associates the fuzzy set $B$ of $Y$ in such a way that $y$ (the image of $x$) belongs to $B$ with a degree at least equal to $d$. Formally, if $f$ is an application from $X$ to $Y$, the extension principle maps a fuzzy set $A$ of $X$ to the fuzzy set $B$ of $Y$ as follows:

$$\forall y \in Y, \mu_B(y) = \begin{cases} \sup_{x \in X \mid y=f(x)} \mu_A(x) & \text{if } f^{-1}(y) \neq \emptyset, \\ 0 & \text{otherwise}. \end{cases}$$

In the particular case where $A$ reduces to the singleton $x_0$, the extension principle maps it into the non-fuzzy set $B$ composed of the sole element $f(x_0)$.

**Example 3.19.** Let $f_1$ be the function defined as $f_1(x) = y = |3x - 6|$ where $|v|$ stands for the absolute value of $v$. The calculus of that function extends to imprecise (fuzzy) values according to Expression (3.15). The characteristic function of the image of any fuzzy set $A$ equals 0 for any negative value $y$, since it cannot be an absolute value. For instance, with the input set $A = \{0.4/(-1), 1/1.5, 0.7/2.4, 0.9/3, 0.2/3.3, 0.6/5\}$, the image of $A$ through $f_1$ is $B = \{0.6/9, 1/1.5, 0.7/1.2, 0.9/3, 0.2/3.9\}$. The value 9 has two antecedents ($-1$ and 5) whose degrees of membership are 0.4 and 0.6 respectively, and 9 is assigned the degree $max(0.4, 0.6)$.

The extension principle may also apply to the case where $X$ is a Cartesian product of sets, as shown in Example 3.20 related to (fuzzy) arithmetics.

**Example 3.20.** Consider the addition of two imprecise integers represented as two fuzzy sets $n_1$ and $n_2$. The regular addition extends to such numbers using the extension principle, and Expression (3.15) becomes:

$$\forall z \in \mathbb{N}, \mu_{n_3}(z) = \sup_{x \in \mathbb{N}, y \in \mathbb{N}, x+y=z} \min(\mu_{n_1}(x), \mu_{n_2}(y)).$$

(3.16)
With $n_1 = \{0.2/5, 0.6/6, 0.8/7, 1/8\}$ and $n_2 = \{1/2, 0.4/3\}$, applying Expression (3.16) returns $n_3$, the sum of $n_1$ and $n_2$ defined as $n_3 = \{0.2/7, 0.6/8, 0.8/9, 1/10, 0.4/11\}$. Here, values 8, 9, 10 are obtained in two ways and their associated degree is the largest of the grades obtained in each case. For instance, the degree $\max(\min(0.8, 1), \min(0.6, 0.4)) = 0.8$ is assigned to value 9. The resulting fuzzy set is normalized, inasmuch as the fuzzy sets representing the inputs are normalized, as happens with the data in this example.

3.7 Fuzzy Quantified Propositions

3.7.1 Fuzzy linguistic quantifiers

Fuzzy linguistic quantifiers were introduced by Zadeh (1983) (see also Kerre and Liu, 1998) in order to extend the usual quantifiers, among which the most used are the universal quantifier ($\forall$) and the existential quantifier ($\exists$). It is possible to distinguish between absolute quantifiers expressing a quantity (a number) such as “a dozen” or “about 6”, and relative quantifiers which refer to a proportion, such as “a small half” or “almost all”. An absolute quantifier $QA$ is modeled as:

$$\mu_{QA} : \mathbb{N} (\text{or } \mathbb{R}) \rightarrow [0, 1],$$

whereas a relative $QR$ is represented as:

$$\mu_{QR} : [0, 1] \rightarrow [0, 1].$$

The value $\mu_{QA}(n)$ (resp. $\mu_{QR}(p)$) expresses the adequation of number $n$ (resp. proportion $p$) with respect to the considered quantifier. Quantifiers can be divided into three classes depending on the type of their characteristic function:

- increasing (e.g., “at least about 5, almost all, most of”),
- decreasing (e.g., “at most about 10, a few”),
- increasing-decreasing (e.g., “around 7, more or less a dozen”).

3.7.2 Quantified propositions

Fuzzy quantifiers act inside expressions where they aggregate a set of values generated by fuzzy predicates; they are called fuzzy quantified propositions. The scope of this section is restricted to propositions involving a single increasing absolute quantifier and a fuzzy set $A$, in the context of
the evaluation of the proposition “$Q X$ are $A$” which is used in Chapter 6. The objective is to assess the extent to which the number of elements of the referential $X$ satisfying the (fuzzy/gradual) property $A$ is in adequation with the quantifier $Q$.

The presentation given here is limited to a general view, with the aim of providing an intuitive interpretation for a casual user. Determining the truth value of “$Q X$ are $A$” relies on the idea that this expression has as many interpretations as there are distinct $\alpha$-cuts for $A$. Each level-cut plays the role of a more or less demanding representative of $A$. It is worth noting that this approach does not seek to compute the cardinality of $A$, but rather to assess the truth value of “$Q X$ are $A$” when $A$ is a regular set. Due to the nesting of $\alpha$-cuts, and to the monotonicity property of the considered quantifier $Q$ (which is increasing), when $\alpha$ diminishes, $\mu_Q(\text{card}(A_\alpha))$ cannot decrease. So it seems reasonable to look for the best trade-off between the quality of the representative chosen for $A$ and its adequation with respect to $Q$. This leads to the following truth value $v$ for “$Q X$ are $A$”:

$$v = \max_{\alpha} \min(\alpha, \mu_Q(\text{card}(A_\alpha))),$$

which ensures that: $\forall \alpha \in [0, v], \mu_Q(\text{card}(A_\alpha)) \geq v$. It appears that this expression corresponds to a Sugeno integral (see Section 3.5.3) of the characteristic function of set $A$ with the fuzzy measure $Q(\text{card}(\cdot))$. This measure is non-additive in general, i.e.:

$$(\text{card}(A \cup B)) \neq Q(\text{card}(A)) + Q(\text{card}(B)),$$

as illustrated by Example 3.21. Nevertheless, it obeys the monotonicity requirement of fuzzy measures, since the quantifier $(Q)$ is increasing.

**Example 3.21.** Let $Q$ be the absolute quantifier “at least 5” defined as:

$$\mu_{\text{at least 5}}(v) = 0 \text{ if } v < 3, \mu_{\text{at least 5}}(3) = 0.4, \mu_{\text{at least 5}}(4) = 0.7,$$

$$\mu_{\text{at least 5}}(v) = 1 \text{ if } v > 4.$$

Assume that $A = \{a, b\}$ and $B = \{c, d, e\}$ are two fuzzy sets. Then:

$$\mu_{\text{at least 5}}(\text{card}(A)) = \mu_{\text{at least 5}}(2) = 0$$

$$\mu_{\text{at least 5}}(\text{card}(B)) = \mu_{\text{at least 5}}(3) = 0.4$$

$$\mu_{\text{at least 5}}(\text{card}(A \cup B)) = \mu_{\text{at least 5}}(5) = 1.$$
Thus:

\[ \mu_{\text{at least } 5 (\text{card}(A \cup B))} \neq \mu_{\text{at least } 5 (\text{card}(A))} + \mu_{\text{at least } 5 (\text{card}(B))}, \]

\[ \mu_{\text{at least } 5 (\text{card}(A \cup B))} > \mu_{\text{at least } 5 (\text{card}(A))}, \]

\[ \mu_{\text{at least } 5 (\text{card}(A \cup B))} > \mu_{\text{at least } 5 (\text{card}(B))}. \]

In addition, this approach generalizes the following two special cases:

- if \( A \) is a regular set, the truth value of “\( Q X \) are \( A \)” is given by \( \mu_Q(\text{card}(A)) \),
- if \( Q \) is a non-fuzzy quantifier such as “at least \( p \)” (which is increasing), by analogy with the existential quantifier, the truth value of “\( Q X \) are \( A \)” is defined as the \( p \)th largest value among the membership degrees to set \( A \).

Furthermore, everything that has just been said for an absolute increasing quantifier can also be adapted for an absolute decreasing quantifier. The case of an increasing–decreasing quantifier \( Q \) is dealt with by decomposing it into an increasing part \( QI \) and a decreasing part \( QD \), and then aggregating the truth values obtained for “\( QI X \) are \( A \)” and “\( QD X \) are \( A \)”. Finally, this approach also applies to relative quantifiers.
This page intentionally left blank
Chapter 4

Non-Fuzzy Approaches to Preference Queries: A Brief Overview

4.1 Introduction

The past two decades have witnessed an increasing interest in expressing preferences inside database queries. The motivations for this are manifold; see Hadjali et al. (2008). First, it has become desirable to offer more expressive query languages that are more faithful to what a user intends to say. Second, the introduction of preferences in queries provides a basis for rank-ordering the retrieved items, which is especially valuable in cases where large sets of items satisfy a query. Third, while a classical query may have an empty set of answers, a relaxed (and thus less restrictive) version of the query might be matched by items in the database.

The approaches to database preference queries may be classified into two categories according to their qualitative or quantitative nature (Hadjali et al., 2008). In the latter case, preferences are expressed quantitatively by a monotone scoring function (the overall score is positively correlated with partial scores), often taken as a weighted linear combination of attribute values (which therefore have to be numerical). Since the scoring function associates each tuple with a numerical score, tuple $t_1$ is preferred to tuple $t_2$ if the score of $t_1$ is higher than the score of $t_2$. In the qualitative approach, preferences are defined through binary preference relations.

In this chapter, we briefly review the main approaches proposed in the literature.
4.2 Quantitative Approaches

4.2.1 Distances and similarity

In this approach, preferences are integrated into elementary conditions by means of a similarity (or resemblance) operator, denoted by \( \approx \), which extends strict equality. An elementary condition \( A \approx v \) (resp. \( A \approx B \)) is interpreted in the context of a distance defined over the domain of attribute \( A \) (resp. of attributes \( A \) and \( B \)). For a condition of the type \( A \approx v \), the idea is that \( v \) is the ideal value, but that other values are acceptable even though they are not as preferred. The greater the distance between the values of \( A \) and \( v \), the less the satisfaction, and if the distance between \( A \) and \( v \) is over a predefined threshold, the condition is considered not at all fulfilled. This type of approach can be seen as a refinement of the nearest-neighbors technique (Friedman et al., 1975) where the similarity operator is implicit. The systems ARES (Ichikawa and Hirakawa, 1986) and VAGUE (Motro, 1988) are both founded on such an approach.

Example 4.1. In ARES, the user attaches a threshold to every preference condition. The different atomic conditions may be combined only in a conjunctive way. The interpretation of the query:

\[
(A_1 \approx v_1, t_1) \text{ and } \cdots \text{ and } (A_n \approx v_n, t_n)
\]

involves two steps:

1. expression of a Boolean selection condition involving distances and the user-specified thresholds:

\[
dist(A_1, v_1) \leq t_1 \text{ and } \cdots \text{ and } dist(A_n, v_n) \leq t_n,
\]

2. aggregation of the distances in order to compute a global distance which is used to rank-order the previously selected elements:

\[
g-dist = \sum_i \text{dist}(A_i, v_i) \left( \leq \sum_i t_i \right).
\]

Consider relation \( \text{Emp} \) depicted in Table 4.1, and the query:

\[
(salary \approx 2500, \text{threshold} = 2) \text{ and } (age \approx 41, \text{threshold} = 1).
\]

With the distance relations specified in Table 4.2, employees Dupont, Martin and Dubois are selected (Durant and Lorant are discarded because
Non-Fuzzy Approaches to Preference Queries: A Brief Overview

4.2.2 Linguistic preferences

In this approach, the idea is to associate preference terms with the Boolean conditions involved in a query. In the system MULTOS (Rabitti and Savino, 1990), it is also possible to use linguistic weights to express the importance of the elementary conditions. For instance, to specify that the conditions on age and salary can be more or less satisfied, and that the condition on age is more important than that on salary, one may write:

\[
((\text{age} = 40) \text{ ideal, (age} = 41 \text{ or age} = 39) \text{ good)},
\]
\[
((\text{age} \leq 46 \text{ and age} \geq 42) \text{ or}
\]
\[(\text{age} \geq 34 \text{ and age} \leq 38) \text{ acceptable}) \text{ high];}
\]
\[
((\text{salary} \leq 2500) \text{ ideal,}
\]
\[(\text{salary} > 2500 \text{ and salary} < 3300) \text{ acceptable}) \text{ medium}].
\]

The terms “ideal”, “good”, and “acceptable” define preferences over the values taken by the attributes, whereas “high” and “medium”
reflect importances. The evaluation of such a query relies on a two-step process:

1. a selection based on the more-or-less acceptable values for each attribute,
2. an ordering of the elements selected by the previous step, based on a numerical translation of the linguistic labels into grades in the unit interval.

Each elementary condition
\[(X_k = v_{k,1})^{p_{k,1}}, (X_k = v_{k,2})^{p_{k,2}}, \ldots, (X_k = v_{k,n_k})^{p_{k,n_k}}\] 

where \(X_k\) denotes an attribute, \(v_{k,m}\) is a value whose preference is \(p_{k,m}\), and \(i_k\) is the importance of the term concerning attribute \(X_k\), is assigned a selection value: true if \(X_k\) takes one of the values \(v_{k,j}\), false otherwise, and a ranking value: \(p_{k,j} \times i_k\) if \(X_k = v_{k,j}\), 0 otherwise. A disjunction (resp. conjunction) of such terms is associated with the highest value (resp. the sum of the values) obtained for the elementary conditions involved.

4.2.3 Explicit scores attached to entities

The approach proposed by Agrawal and Wimmers (2000) allows a user to express his/her preference for an entity, either by attaching a score in \([0, 1]\) to that entity, or by expressing a veto (by means of the special symbol \(\natural\)), or by an indifference statement (default case) regarding that entity. An entity is described by a tuple in which the value of a field either belongs to the domain of the corresponding attribute, or equals \(\ast\) (a special symbol which represents every domain value). In order to illustrate these notions, consider a relation \(\text{fridge} (\text{model}, \text{color}, \text{price})\) describing different models of refrigerators. A user expressing the preferences \{\((123, \text{red}), 0.4\), \((123, \ast), \natural\), \((234, \text{green}), \natural\), \((456, \text{white}), 0.8\)\} indicates that he/she has a strong preference for the model 456 in white, a much more moderate preference for the model 123 in red, and absolutely rejects the model 234 in green as well as every fridge of the model 123 in a color other than red. The authors also introduce a generic operator which makes it possible to combine preferences originating from different users.

The approach proposed by Koutrika and Ioannidis (2004) follows the same general philosophy but extends the approach introduced by Agrawal and Wimmers (2000) by considering a more general format for the user’s preference profiles. Notably, it enables the expression of negative preferences.
Non-Fuzzy Approaches to Preference Queries: A Brief Overview

4.2.4 Top-k queries

The top-k query approach (Bruno et al., 2002; Chaudhuri and Gravano, 1999), even though it uses the same general principles as systems based on the notions of distance and similarity, has some points in common with the fuzzy-set-based approach. In the top-k query framework, a user specifies ideal values for certain attributes as well as a number $k$ of desired tuples. The distance between an attribute value and the ideal value is computed by means of a difference (absolute value), after a normalization step (which yields domain values between 0 and 1). The overall distance is calculated by aggregating the elementary distances using a function which can be the minimum, the sum, or the Euclidean distance. The steps in the computation are the following:

1. Using $k$, and taking into account both the chosen aggregation function and statistics about the considered relation, a threshold $\alpha$ over the global distance is deduced,
2. A Boolean query computing the desired $\alpha$-cut — i.e. the set of items whose score is at least equal to $\alpha$ — or a superset of this $\alpha$-cut is determined,
3. This query is processed and the score (the global distance) attached to every element of the result is calculated,
4. If at least $k$ tuples with a score greater than or equal to $\alpha$ have been obtained, the $k$ best are returned to the user; otherwise the procedure is run again (from step 2) using a lower threshold $\alpha$.

4.2.5 Outranking

Bosc et al. (2010d,e) introduce an approach to database preference queries based on the notion of outranking, initially proposed by Roy (1991) in a decision-making context. This approach, which uses a weighted majority rule, constitutes an alternative to using a Pareto-order-based approach (see Section 4.3.2) for handling incommensurable partial preferences. It assumes that a set of scoring functions translating the different partial preferences is available. The approach relies on an outranking measure which aggregates the numbers (weighted by importances) of partial preferences which are either concordant with, discordant with, or indifferent to a certain ordering
between two tuples. Even though outranking does not define an order in
the strict sense of the term, the authors describe a technique which yields a
complete preorder, based on a global aggregation of the outranking degrees
computed for each pair of tuples.

In Bosc et al. (2009e), a variant is presented, which aims at comparing
each tuple with the profiles of predefined classes. In this way, instead of
being compared pairwise as in a Pareto-order-based approach, tuples are
compared with the acceptability profiles associated with the different user-
defined classes, which is of course less computationally expensive. According
to the satisfaction of the different partial preferences specified in the user
query, a tuple is classified into a given class with a certain degree. An
algorithm implementing this classification with linear data complexity is
described in Bosc et al. (2009e).

4.3 Qualitative Approaches

4.3.1 Secondary preference criterion

In this type of approach, a query is made up of two parts: a Boolean selection
condition $C$ and a complementary component $P$ aimed at specifying the
preferences. A query is written as: “$C; P$” and its interpretation is: “find
the elements which satisfy $C$ and rank-order them according to $P$”.

In the system called Preferences (Lacroix and Lavency, 1987), a pref-
erence is expressed by a set of Boolean conditions which can be combined
by means of two operators: nesting (hierarchy of conditions) and juxtapo-
sition (combining conditions which have the same priority). Starting from
$r_C$, the subset gathering the tuples of a relation $r$ satisfying condition $C$,
the nesting of the preference clauses $P_1$ to $P_n$ leads to construction of the
sets $H_i$ gathering the elements of $r_C$ which satisfy the preference clauses
$P_1$ to $P_i$ but not $P_{i+1}$. $H_0$ is the subset of $r_C$ whose elements do not satisfy
$P_1$, and $H_n$ is the subset of $r_C$ whose elements satisfy all of the preference
clauses $P_1$ to $P_n$. In a similar way, juxtaposing the preference clauses $P_1$ to
$P_n$ leads to construction of the sets $J_i$ gathering the elements of $r_C$ which
satisfy $i$ preference clauses ($i \in [0, n]$). The answer initially returned to
the user is the non-empty set $H_i$ (resp. $J_i$) whose index $i$ is the highest
(which corresponds to the best elements). The user can then browse less
satisfactory sets if he/she wants to do so.

The major advantage of this system is that it avoids numerous successive
formulations in order to reach the desired number of answers. However,
one may find that the discrimination power is somewhat limited, since the number of satisfaction levels equals the number of preference clauses (since at most \((n + 1)\) distinct subsets are built).

**Example 4.2.** Consider again relation \(\text{Emp}\) from Table 4.1 and the query aimed at retrieving the employees who live in Lyon, if possible having a salary less than $3,000 \((P_1)\), and if possible older than 38 \((P_2)\). This query involves the preference hierarchy \((P_1; P_2)\) and, with the extension from Table 4.1, one obtains:

\[
H_0 = \{\text{Lorant, Dupont}\}, \quad H_1 = \{\text{Durant}\}, \quad H_2 = \{\text{Dubois}\}.
\]

With the opposite hierarchy \((P_2; P_1)\), the result would be:

\[
H'_0 = \{\text{Lorant, Durant}\}, \quad H'_1 = \{\text{Dupont}\}, \quad H'_2 = \{\text{Dubois}\}.
\]

If the juxtaposition \((P_1, P_2)\) were used instead of a hierarchy, the result would be:

\[
J_0 = \{\text{Lorant}\}, \quad J_1 = \{\text{Dupont, Durant}\}, \quad J_2 = \{\text{Dubois}\}.
\]

Note that the second tuple from \(\text{Emp}\) does not belong to the result, since it does not satisfy the condition on attribute \(\text{city}\). □

### 4.3.2 Pareto-order-based approaches

In the past decade, many algorithms have been proposed for efficient computation of the non-dominated results of a query, in the sense of Pareto order. Seen as points in a multi-dimensional space, these answers constitute a so-called skyline. Some pioneering works in this domain have been done by Börzsönyi et al. (2001). First, we recall the general principle of Pareto-order-based approaches.

Let \(\{G_1, G_2, \ldots, G_n\}\) be a set of atomic partial preferences. We denote by \(t \succ G_i t'\) (resp. \(t \succeq G_i t'\)) the statement “tuple \(t\) satisfies preference \(G_i\) better than (resp. at least as well as) tuple \(t'\).” In the sense of Pareto order, a tuple \(t\) dominates another tuple \(t'\) if and only if

\[
\forall i \in [1, n], \quad t \preceq G_i t' \land \exists k \in [1, n], \quad t \succ G_k t'.
\]

In other terms, \(t\) dominates \(t'\) if and only if it is at least as good as \(t'\) regarding each preference, and it is strictly better than \(t'\) with respect to at least one preference.
Clearly, Pareto-order-based approaches do not require any assumption of commensurability between the satisfaction degrees pertaining to the different elementary preferences, in contrast to, for instance, the fuzzy-set-based approach. As a consequence, some points from the skyline (i.e., some tuples from the result) may represent very poor answers with respect to certain atomic conditions, whereas they are excellent regarding others; the skyline approach only yields a strict partial order while the fuzzy-set-based approach leads to a complete preorder. Kießling (2002) and Kießling and Köstler (2002) described the basis of a preference model founded on Pareto order for database systems. A preference algebra including an operator called winnow has also been proposed by Chomicki (2003) in order to integrate preference formulas inside a relational framework (and SQL). See also the work by Torlone and Ciaccia (2002), who introduced an operator called Best aimed at returning the non-dominated tuples of a relation. The following example uses the syntax of the language Preference SQL (Kießling and Köstler, 2002), which is a typical representative of a Pareto-order-based approach.

Example 4.3. Consider a relation car of schema (id, make, category, price, color, mileage) whose extension is given in Table 4.3, and the query:

```sql
select * from car
where color \neq "purple"
preferring (make = "Opel" else make = "Ford")
and (mileage \approx 40,000);
```

The preference condition on attribute mileage implies that a car whose mileage is 34,000 is better than a car whose mileage is 48,000, since 34,000 is closer to 40,000 than is 48,000. Tuple t_7 is discarded since it does not satisfy

<table>
<thead>
<tr>
<th>id</th>
<th>make</th>
<th>category</th>
<th>price</th>
<th>color</th>
<th>mileage</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_1</td>
<td>Opel</td>
<td>roadster</td>
<td>4,500</td>
<td>blue</td>
<td>19,000</td>
</tr>
<tr>
<td>t_2</td>
<td>Ford</td>
<td>SUV</td>
<td>4,000</td>
<td>red</td>
<td>35,000</td>
</tr>
<tr>
<td>t_3</td>
<td>Fiat</td>
<td>roadster</td>
<td>5,000</td>
<td>red</td>
<td>43,000</td>
</tr>
<tr>
<td>t_4</td>
<td>VW</td>
<td>roadster</td>
<td>4,500</td>
<td>red</td>
<td>19,000</td>
</tr>
<tr>
<td>t_5</td>
<td>VW</td>
<td>roadster</td>
<td>5,000</td>
<td>red</td>
<td>40,000</td>
</tr>
<tr>
<td>t_6</td>
<td>Ford</td>
<td>sedan</td>
<td>5,500</td>
<td>blue</td>
<td>51,000</td>
</tr>
<tr>
<td>t_7</td>
<td>Opel</td>
<td>sedan</td>
<td>2,500</td>
<td>purple</td>
<td>72,000</td>
</tr>
</tbody>
</table>
the selection condition from the \texttt{where} clause. Tuple $t_4$ is dominated by $t_1$ and is thus discarded. Tuple $t_3$ is dominated by $t_5$ whereas $t_6$ is dominated by $t_2$. On the other hand, $t_1$, $t_2$ and $t_5$ are incomparable and the final answer is $\{t_1, t_2, t_5\}$.

With this type of approach, when a query involves conditions on many attributes, there is a high risk of obtaining many incomparable tuples. Several techniques have been proposed to order two tuples $t_1$ and $t_2$ which are incomparable in the sense of skyline. These techniques exploit, for instance:

- the number of tuples that $t_1$ and $t_2$ (respectively) dominate (notion of $k$-representative dominance proposed by (Lin \textit{et al.}, 2007)) or
- a preference order over the attributes; see for instance the notions of $k$-dominance introduced by Chan \textit{et al.} (2006a), and $k$-frequency proposed by the same authors (Chan \textit{et al.}, 2006b).

The issue of efficiently processing skyline queries has given rise to many research works, in particular Bartolini \textit{et al.} (2008) and Chomicki \textit{et al.} (2005).

\subsection{CP-nets}

The use of the structure called CP-net (network of conditional preferences) for modeling database preference queries has been suggested by Brafman and Domshlak (2004) — but this framework was initially developed in artificial intelligence (Boutilier \textit{et al.}, 2004). A CP-net is a graphical representation of statements which express conditional preferences of the \textit{ceteris paribus} type. The premise is that user preferences generally express the idea that, in a given context, a partially described state of affairs is strictly preferred to another partially described state of affairs, both states of affairs being mutually exclusive, in accordance with the \textit{ceteris paribus} semantics, i.e., everything else being equal in the description of these states of affairs.

By means of a CP-net, a user may describe how his/her preferences for the values of a variable depend on the values of other variables. For instance, a user may formulate the following statements:

$s_1$: I prefer minivans cars to sedan cars;
$s_2$: as to minivans, I prefer Chrysler to Ford;
$s_3$: for sedans, I prefer Ford to Chrysler;
$s_4$: in Ford cars, I prefer the black ones to the white ones;
$s_5$: in Chrysler cars, I prefer the white ones to the black ones.
In the CP-net approach applied to database preference queries (Brafman and Domshlak, 2004), a preference is represented by a binary relation over a relation schema (where the attributes are assumed to be binary). If \( r \) is a relation schema, then a preference query \( Q \) on \( r \) consists of a set \( Q = \{ s_1, \ldots, s_m \} \) of statements (usually between subtuples of \( r \), relying on the ceteris paribus semantics). From \( Q \), one induces a set of preference relations \( \{ >_{CP(1)}, \ldots, >_{CP(m)} \} \), from which one may derive a global preference relation \( >_{CP(Q)} \) which defines a strict partial preorder between the tuples from \( r \).

**Example 4.4.** Assume that black (\( b \)) jackets (resp. pants) are preferred to white (\( w \)) jackets (resp. pants), and that in case of jackets and pants of the same color, red (\( r \)) shirts are preferred to white ones; otherwise, white shirts are preferred. Only the colors mentioned are assumed to be available. Thus, we have three binary variables \( J, P, S \) corresponding to the colors of the jacket, the pants, and the shirt. See Figure 4.1 for the CP-net built on these variables. Thus, there are eight tuples corresponding to the relational schema jackets–pants–shirts: \( bbr, bbw, bwr, bww, wbr, wbw, wwr, www \). The preferences associated with the induced CP-net, applying the ceteris paribus principle, are:

- \( bPS >_{CP} \neg bPS, \forall P \in \{ b, \neg b \}, \forall S \in \{ r, \neg r \}; \)
- \( JbS >_{CP} J \neg bS, \forall J \in \{ b, \neg b \}, \forall S \in \{ r, \neg r \}; \)
- \( bbr >_{CP} bbw, wwr >_{CP} www, bbw >_{CP} bwr, wbw >_{CP} wbr. \)

Then the following partial order holds under the ceteris paribus assumption:

\[
\begin{align*}
    bbr & >_{CP} bbw >_{CP} bwr >_{CP} wwr >_{CP} wbr >_{CP} www \\
    bbr & >_{CP} bbw >_{CP} wbr >_{CP} wwr >_{CP} www.
\end{align*}
\]

![Fig. 4.1 CP-net for “evening dress: jacket, pants and shirt”](image)
It is worth mentioning that the ceteris paribus semantics strongly differs from the totalitarian semantics, which is implicitly favored by authors from the database community (including those who advocate approaches based on Pareto order). Under the totalitarian semantics, when evaluating the preference clause of a query, the attributes which are not concerned with that clause are ignored. Obviously, with the ceteris paribus semantics, the number of incomparable tuples is in general much higher than with the totalitarian semantics.

4.3.4 Domain linearization

The approach proposed in Georgiadis et al. (2008) considers preferences defined as preorders over relational attributes and their respective domains. Let $R(\#i, A, F, L)$ be a table describing part of the content of a digital library, where $\#i$ is the identifier of a document, $A$ corresponds to an author’s name, $F$ describes the format of the document, and $L$ is the language in which the document is written. Then, an example of a preference query in the sense of Georgiadis et al. (2008) is made of the following statements:

(1) Joyce is preferred to Proust or Mann (preference $P_A$);
(2) ODT and DOC formats are preferred to PDF ($P_F$);
(3) English is preferred to French, which is itself preferred to German ($P_L$);
(4) Writer ($W$) is as important as format ($F$), whereas the writer–format combination is more important than language ($L$).

Such statements define binary preference relations: (1), (2) and (3) are defined over attribute domains, whereas (4) is defined over the set of attributes. Preference relations are required to be reflexive and transitive, i.e., they must be preorders. The authors propose a technique for linearizing the domains associated with these partial preorders (recall that a domain, in the sense of domain theory, is a partially ordered set). This way, one induces a block sequence (i.e., an ordered partition) of the result of the query. In such a sequence, each block contains preference-wise incomparable elements. The first block contains the most preferred elements, and in every other block, for each element, there exists a more preferred element in the preceding block.

The algorithms proposed in Georgiadis et al. (2008) compute the block sequence that constitutes the answer to a preference query, without constructing the induced ordering of tuples. This is achieved by exploiting the
semantics of a preference expression and, in particular, by linearizing the Cartesian product of all attribute terms appearing in the expression. In reality, one passes from a set of statements expressing partial preferences to a query lattice, from there to a lattice of query results, and finally to a block sequence that constitutes the answer.

With respect to Pareto-order-based approaches, the main originality of the approach described in Georgiadis et al. (2008) lies in its use of partial preorders instead of strict orders for modeling positive independent preferences. This makes it possible to distinguish between the notions of “equally preferred tuples” and “incomparable tuples”.

4.3.5 Possibilistic-logic-based approach

In Hadjali et al. (2008), the authors present an approach to database preference queries based on possibilistic logic (Dubois and Prade, 2004) where the queries involve symbolic weights on a linearly ordered scale. These weights can be processed without assessing their precise values, which leaves the user free to not specify any priority among the preferences (in contrast to CP-nets, where a default priority ordering is induced by the structure of the preference graph). However, the user may also enforce a (partial) ordering between the priorities, if necessary.

In the case of binary preferences, the possibilistic encoding of the conditional preference “in context $c$, $a$ is preferred to $b$” is a pair of possibilistic formulas: $\{(\neg c \lor a \lor b, 1), (\neg c \lor a, 1 - \alpha)\}$. In other words, if $c$ is true, one should have $a$ or $b$ (the choice is only between $a$ and $b$), and in context $c$, it is somewhat imperative to have $a$ true. This encodes a constraint of the form $N(\neg c \lor a) \geq 1 - \alpha$, itself equivalent to a constraint on a conditional necessity measure $II(\neg a|c) \leq \alpha$, where $II$ is the dual possibility measure associated with $N$. This constraint says that the possibility of not having $a$ is upper-bounded by $\alpha$, i.e., $\neg a$ is all the more impossible as $\alpha$ is small. To switch from necessity degrees to satisfaction levels (or possibility degrees), the authors use an order-reversing operator denoted by $1 - (.)$. Thus, the level of priority $1 - \alpha$ associated with a preference is changed into a satisfaction degree $\alpha$ when this preference is violated. Even if the values of the weights are unknown, a partial order between the different choices may be induced by a leximin ordering (Dubois et al., 1997; see also Chapter 3).

This approach can be related to that based on fuzzy sets where the components of a query may be conditionally weighted in terms of importance. In
Non-Fuzzy Approaches to Preference Queries: A Brief Overview

In fact, these approaches can be interfaced, which makes it possible to handle non-binary preferences.

In Hadjali et al. (2008), it is shown that the possibilistic-logic-based approach refines that proposed by Lacroix and Lavency (1987) (see also Section 4.3.1) as well as the operator winnow proposed by Chomicki (2003) and the operator Best introduced by Torlone and Ciaccia (2002) (see also Section 4.3.2).

4.4 Conclusion

Aspects related to implementation, in particular efficient query processing, have not been dealt with here since they are beyond the scope of this chapter, but they are of course crucial in a database context (where the volume of data handled is in general huge).

The interested reader may also find details of the way in which some quantitative approaches (as well as the approach that uses secondary preference criteria) may be modeled in a fuzzy-set-based framework, in Bosc and Pivert (1992).
This page intentionally left blank
Chapter 5

Simple Fuzzy Queries

5.1 Introduction

In this chapter, we present the main features of two languages aimed at expressing fuzzy queries to relational databases. First, we describe a complete extension of relational algebra, then we focus on the expression of simple fuzzy queries in an SQL-like language called SQLf (Bosc and Pivert, 1995b). By “simple queries”, we mainly mean queries that do not involve any quantified statements or aggregates applied to fuzzy relations. These more complex queries, as well as bipolar fuzzy queries and division queries, are dealt with in subsequent chapters (Chapters 6, 7, and 8).

We first recall that gradual predicates (or conditions) can be associated with fuzzy sets (Zadeh, 1965; see also Chapter 3) aimed at describing classes of objects with vague boundaries, just as regular sets allow for the definition of Boolean predicates. Often, elementary fuzzy predicates correspond to adjectives of natural language, such as young, tall, cheap or well-paid. An elementary predicate can also compare two attributes using the usual operators (equality, superiority, etc.), but also gradual relational comparison operators such as “more or less equal” or “somewhat greater than”.

It is possible to alter (weaken or strengthen) the meaning of a given predicate using a modifier which is generally associated with an adverb (e.g., very, more or less, relatively, really). For instance, “very cheap” is more restrictive than “cheap” and “fairly high” is less demanding than “high”. It is possible to define the meaning of the predicate $mod P$ (where $mod$ is a modifier and $P$ is a fuzzy predicate) in a compositional way.
Different approaches have been advocated, among which are:

- $\mu_{\text{mod}P}(x) = \mu_P(x)^n$,
- $\mu_{\text{mod}P}(x) = \mu_P(x) \top \cdots \top \mu_P(x)$, where $\top$ is a non-idempotent triangular norm or co-norm,
- $\mu_{\text{mod}P}(x) = \mu_P(x \pm a)$, where $a$ is a constant.

The interested reader can find more detail about this topic in Bouchon-Meunier and Yao (1992) and Kerre and de Cock (1999).

As mentioned in Chapter 3, atomic and modified predicates can be involved in compound predicates which go far beyond those used in regular queries. Conjunction (resp. disjunction) is interpreted by means of a triangular norm $\top$ (resp. co-norm $\bot$), i.e., an operator which is associative, commutative, monotonically increasing with respect to both arguments, and whose neutral element is 1 (resp. 0), for instance the minimum or the product (resp. the maximum or the probabilistic sum):

- $\forall x, \mu_{P_1 \land P_2}(x) = \top(\mu_{P_1}(x), \mu_{P_2}(x))$
- $\forall x, \mu_{P_1 \lor P_2}(x) = \bot(\mu_{P_1}(x), \mu_{P_2}(x))$

Negation is interpreted by the complement to 1 of the degree:

- $\forall x, \mu_{\neg P}(x) = 1 - \mu_P(x)$.

The weighted conjunction and disjunction as well as weighted mean or OWA (Yager, 1988) can be used to assign a different importance to each of the predicates (see in particular Fodor and Yager, 2000, for a more complete presentation).

The basic idea behind the extensions of both relational algebra and SQLf is thus to authorize the use of (possibly compound) fuzzy predicates in selection or join conditions. The principal concept is that of a fuzzy relation, i.e., a relation $r$ where each tuple $t$ is attached a degree $\mu$ in the unit interval which expresses the extent to which $t$ satisfies the fuzzy concept associated with $r$.

In this chapter, we first give definitions of the extended relational algebraic operators used in this framework (Section 5.2), then we describe the basic features of the SQLf language (Section 5.3). In Section 5.4, we deal with the definition of fuzzy terms and operators through an appropriate interface. Section 5.5 is devoted to a particular type of fuzzy query involving so-called contextual fuzzy predicates, i.e., predicates whose definition can be automatically derived from a certain context. Query processing aspects are tackled in the last section (5.6) of the chapter.
5.2 An Extended Relational Algebra

In the context of regular databases, a flexible query whose interpretation is based on fuzzy sets takes a set of regular relations as inputs and produces a result which is a weighted (type-1) fuzzy relation where the degree attached to each tuple depends on the satisfaction of fuzzy predicates involved in the query. Since a regular relation is a special case of a fuzzy relation (grades equal to 1), the compositionality principle is maintained. The extension of the three usual types of language (relational algebra, relational calculus, and SQL) has been addressed in Tahani (1977), Bosc et al. (1988), Wong and Leung (1990), Takahashi (1991), Lee and Kim (1993), and Bosc and Pivert (1995b). An extended relational algebra consists of the same operators as those usually employed (see Chapter 2), except that they are now defined over weighted (type-1) relations.

The Cartesian product \((r \times s)\) of two fuzzy relations \(r\) and \(s\) of respective universes \(X\) and \(Y\) is defined as:

\[ r \times s = \{ \mu/\langle u, v \rangle \mid u \in \text{support}(r) \land v \in \text{support}(s) \land \mu = \min(\mu_r(u), \mu_s(v)) \}. \]

If \(r\) and \(s\) are defined on the same universe \(X\), the set operations union (\(\cup\)), intersection (\(\cap\)), and difference (\(-\)) are defined as:

\[ r \cup s = \{ \mu/\langle u, v \rangle \mid (u \in \text{support}(r) \land u \notin \text{support}(s) \land \mu = \mu_r(u)) \cup \{ \mu/\langle u, v \rangle \mid (u \in \text{support}(s) \land u \notin \text{support}(r) \land \mu = \mu_s(u)) \} \cup \{ \mu/\langle u, v \rangle \mid (u \notin \text{support}(r) \land u \notin \text{support}(s) \land \mu = \max(\mu_r(u), \mu_s(u)) \}, \]

\[ r \cap s = \{ \mu/\langle u, v \rangle \mid (u \in \text{support}(r) \land u \in \text{support}(s) \land \mu = \min(\mu_r(u), \mu_s(u))) \}, \]

\[ r - s = \{ \mu/\langle u, v \rangle \mid u \in \text{support}(r) \land \mu = \min(\mu_r(u), \ 1 - \mu_s(u)) \}. \]

The algebraic operations, i.e., the selection \(\sigma(r, \phi)\) where \(\phi\) is a fuzzy predicate, the join \(r[A \theta B]\) where \(A\) is a subset of \(X\), \(B\) is a subset of \(Y\), \(\theta\) is a fuzzy comparator (a binary fuzzy relation between the \(A\)-field value in the tuple \(x\) of \(r\) and the \(B\)-field value in the tuple \(y\) of \(s\)), and the projection \(\pi_Y(r)\) where \(Y\) is a proper subset of \(X\), are defined as follows:

\[ \sigma(r, \phi) = \{ \mu/\langle u \rangle \mid u \in \text{support}(r) \land \mu = \min(\mu_r(u), \mu_\phi(u)) \}, \]

\[ r[A \theta B]s = \{ \mu/\langle u, v \rangle \mid u \in \text{support}(r) \land v \in \text{support}(s) \land \mu = \min(\mu_r(u), \mu_s(v), \mu_\theta(u.A, v.B)) \}, \]

\[ \pi_Y(r) = \{ \mu/\langle v \rangle \mid (\exists u \in \text{support}(r) \text{ s.t. } u.Y = v) \land \mu = \sup_{u \in \text{support}(r)} \mu_r(u) \}. \]
Table 5.1 Extensions of the relations Emp (left) and Dep (right)

<table>
<thead>
<tr>
<th>#e</th>
<th>name</th>
<th>salary</th>
<th>bonus</th>
<th>age</th>
<th>dep</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>Dupont</td>
<td>2,500</td>
<td>0.1</td>
<td>1,800</td>
<td>38</td>
</tr>
<tr>
<td>76</td>
<td>Martin</td>
<td>4,000</td>
<td>0.9</td>
<td>2,600</td>
<td>42</td>
</tr>
<tr>
<td>26</td>
<td>Durant</td>
<td>3,000</td>
<td>0.4</td>
<td>1,500</td>
<td>37</td>
</tr>
<tr>
<td>12</td>
<td>Dubois</td>
<td>2,500</td>
<td>0.1</td>
<td>900</td>
<td>39</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#d</th>
<th>budget</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>200</td>
<td>Toulouse</td>
</tr>
<tr>
<td>23</td>
<td>150</td>
<td>Paris</td>
</tr>
</tbody>
</table>

The equivalence: \( r[A \theta B]s = \sigma(r \times s, A \theta B) \) holds as in the usual case. Obviously, these operations may apply to regular relations to build fuzzy relations which are used as intermediate or final results.

**Example 5.1.** Consider the relations Emp (\( #e, name, salary, bonus, age, dep \)) and Dep (\( #d, budget, city \)) and the query: “find the departments (number) with a high budget where there is no employee less than 40 years old with a medium salary”. This query can be stated as the following algebraic expression:

\[
\pi(#d)(\sigma(\text{Dep}, \text{budget is high})) - \pi(\text{dep})(\sigma(\text{Emp}, \text{age} < 40 \text{ and salary is medium})).
\]

With the extensions given in Table 5.1 (where the values associated with salary (resp. budget) denote the grades for the predicate medium (resp. high)), the result of this expression is: \( \{1/17, 0.2/23\} - \{0.1/17, 0.4/23\} = \{0.9/17, 0.2/23\}. \)

Beyond these operations, which are canonical extensions of the regular ones, the division of relations requires more attention. It will be dealt with in detail in Chapter 7.

5.3 An Overview of a Basic Version of SQLf

5.3.1 Introduction

The first fuzzy set-based extension of the language SEQUEL (Chamberlin et al., 1976) was proposed in Bosc et al. (1988), which only allowed the expression of simple fuzzy queries (mainly projection–selection–join queries). The choice of SEQUEL was justified by the fact that it was, at that time, the most widely known relational database language. Following the arrival of standard SQL, a more complete extension, called SQLf (Bosc and Pivert, 1995b), was defined by studying some aspects
previously not dealt with, particularly those which concerned the extension of
nesting operators and also selection operators applying to sets of tuples,
e.g. quantified statements. The aim was to make SQLf as similar to SQL as
possible, particularly with regard to query equivalences (Bosc and Pivert,
1991). For each type of query, an investigation was made as to whether
the equivalences between several formulations, which are valid in a Boolean
context, remain when the relations derive from fuzzy restrictions, and when
the used connectives themselves become fuzzy. The aim of this section is
to present the main features of SQLf. First, we show the different forms
that the multi-relation block in SQLf can take, and we provide the semantics of the considered fuzzy operations. The matter of “complex queries”
is dealt with next by studying nesting and set-oriented operators. For each
operator, we recall that in SQL several equivalent formulations exist, and
we show which of these equivalences remain valid in an imprecise querying
context. Then, we study the case of queries implying relation partitioning,
in particular those that involve fuzzy quantifiers.

5.3.2 The multiple relation base block

In SQL, a projection–selection–join query can be expressed using a multi-
relation block, without nesting. Such a block is written as:

\[
\text{select } \langle \text{attributes} \rangle \text{ from } \langle \text{relations} \rangle \text{ where } \langle \text{condition} \rangle.
\]

Its role is to select the tuples belonging to the Cartesian product of the spe-
cific relations which satisfy the requirement, and to project them onto the
specified attributes. It is worth recalling that this projection is not exactly
the usual relational projection, since duplicates are authorized (a multiset
is obtained), unless otherwise specified (using the keyword \textit{distinct}).

In SQLf, the select block remains fundamentally the same. The differences are essentially twofold. The first difference is related to the use of
a parameter intended for the calibration of the result (either a number of
desired responses denoted by \( n \), or a qualitative threshold denoted by \( t \), or both). The second difference concerns the nature of the authorized condi-
tions, which are no longer solely Boolean. In SQLf, the new formulation of
the select block is as follows:

\[
\text{select } [n \mid t \mid n, t] \langle \text{attributes} \rangle
\]
\[
\text{from } \langle \text{relations} \rangle \text{ where } \langle \text{fuzzy condition} \rangle \quad (Q1)
\]
where \( \langle \text{relations} \rangle \) corresponds to a list of usual (crisp) relations, and \( \langle \text{fuzzy condition} \rangle \) may imply both fuzzy and Boolean basic conditions at the same time, linked by some connectives.

### 5.3.3 Subqueries

#### 5.3.3.1 Nesting with “in”

In SQL, the operator *in* is used for the membership operation. In SQLf, some imprecision may appear at different levels inside queries of that kind: on the one hand, one may want to know whether an element belongs to a fuzzy relation, and on the other hand, one may want to check for fuzzy membership (to either a usual or a fuzzy relation).

**Membership to a fuzzy relation.** In SQL, an example of a query involving the operator *in* is given by: “find the employees younger than 35 who work in a department whose budget is higher than $100,000”. It can be expressed as:

```sql
select #e from Emp where age < 35 and #dep in (select #d from Dep where budget > 100,000).
```

In SQLf, since a block delivers a fuzzy relation, it is natural to extend the operator *in* in order to express membership to a fuzzy set. Fuzzy set theory makes it possible to determine the extent to which an element belongs to a certain category, by using the membership function associated with the set under consideration. In the present context, one has to deal with fuzzy relations likely to involve duplicates, since these may result from a projection. Thus, we are led to define the operator *in* as follows:

\[
\mu_{\text{in}}(a, E) = \sup_{b \in \text{support}(E)}(\min(\mu_a(a, b), \mu_E(b)))
\] (5.1)

where \(a\) is the element and \(E\) the considered (multi-) set. This semantics preserves the equivalences valid in SQL. For example, the query: “find the young employees who work in a high-budget department” may be expressed as:

```sql
select #e from Emp where age is young and #dep in (select #d from Dep where budget is high),
```
and also, by means of a multi-relation block, as:

```sql
select #e from Emp, Dep where age is young and
Emp.#dep = Dep.#d and budget is high.
```

In this latter case, the grade of an employee $e$ in the result $res$ is given by:

$$
min(\mu_{young}(e.sal), sup_{t\in Dep}min(\mu_{high}(budget), \mu_{=}(t.#d, e.#dep))).
$$

(5.2)

In the former case, if $SQ$ stands for the inner block, we have:

$$
\mu_{res}(e) = min(\mu_{young}(e), min(e.#dep, SQ)),
$$

(5.3)

and if we replace $min(e.#dep, SQ)$ by its value according to Expression (5.1), one can see that both values obtained are the same. This means that, if $fc_1(r)$ (resp. $fc_2(s)$) denotes a fuzzy condition applying to relation $r$ (resp. $s$), it remains possible to use either:

```sql
select r.* from r, s where fc_1(r) and fc_2(s) and r.A = s.B,
```

or:

```sql
select * from r where fc_1 and A in (select B from s where fc_2).
```

Symmetrically, using Expression (5.1), a straightforward way to define $not in$ is:

$$
\mu_{not in}(a, E) = 1 - \mu_{in}(a, E).
$$

(5.4)

**Fuzzy membership.** In the previous case, the transition from a multi-block query to a nested query corresponded to the presence of an equijoin clause in the query. It is also worth studying the case where the join clause relies on a closeness operator ($\approx$, about). For this purpose, SQLf offers the ability to express queries checking the membership to a set (fuzzy or not), by evaluating the extent to which an element of the set exists that is close to the considered element. To this end, a new constructor is necessary which will be denoted by $in_f$. For example, the query: “find the employees who work in a department whose budget is about 1,000 times their own salary” can be expressed by the multi-block query:

```sql
select #e from Emp, Dep where
Emp.#dep = Dep.#d and Dep.budget \approx Emp.salary * 1,000,
```
Fuzzy Preference Queries to Relational Databases

and, using an in nesting (replacement of the equijoin clause on the department):

\[
\text{select } \#e \text{ from } \text{Emp where } \#\text{dep in}
\]

\[
(\text{select } \#d \text{ from } \text{Dep where } \text{budget } \approx \text{Emp.salary } \times 1,000),
\]

and also using the nesting operator \(\text{in}_{\ell}\) expressing the fuzzy membership of a set:

\[
\text{select } \#e \text{ from } \text{Emp where } \text{salary } \times 1000 \text{ in}_{f}
\]

\[
(\text{select } \text{budget from } \text{Dep where } \#d = \text{Emp.\#dep}).
\]

The resemblance between the considered element and those of the set must be taken into account when calculating the grade of membership. Therefore, in Expression (5.1), the equality degree is replaced by a closeness degree. The operator \(\text{in}_{f}\) is defined as:

\[
\mu_{\text{in}_{f}}(a, E) = \sup_{b \in \text{support}(E)}(\min(\mu_{\approx}(a, b), \mu_{E}(b))).
\] (5.5)

Note that operator \(\text{in}\) is a special case of \(\text{in}_{f}\) which corresponds to using the strict equality as a special case of the closeness operator \(\approx\). This operator may be defined for example as:

\[
\mu_{\approx}(x, y) = \begin{cases} 
1 & \text{if } |x - y| \leq \delta_1, \\
0 & \text{if } |x - y| \geq \delta_2, \\
\text{linear in between},
\end{cases}
\] (5.6)

where \(\delta_1\) and \(\delta_2\) (\(\delta_1 \leq \delta_2\)) are two constants which may depend both on the attribute domain and the user.

It is possible to show that the equivalence between:

\[
\text{select } r.* \text{ from } r, s \text{ where } f_{c_1}(r) \text{ and } f_{c_2}(s) \text{ and } r.A \approx s.B
\]

and

\[
\text{select } * \text{ from } r \text{ where } f_{c_1} \text{ and } A \text{ in}_{f} (\text{select } B \text{ from } s \text{ where } f_{c_2})
\]

is ensured by Expression (5.5). More precisely, this equivalence follows from the equality between:

\[
\sup_{y \in s} \min(\mu_{\approx}(x.A, y.B), \mu_{f_{c_2}}(y))
\]

and

\[
\sup_{b \in \text{support}(sf)} \min(\mu_{\approx}(x.A, b), \mu_{sf}(b)),
\]

where \(sf\) is the result of the inner select block.
Operator \( \text{not in}_f \) is defined in a way similar to \( \text{not in} \); see Expression (5.4). As illustrated by the previous examples, it allows for queries which would be expressed as equijoins in relational algebra. Similarly, an extended equijoin operator can be associated with \( \text{in}_f \). On the other hand, \( \text{not in}_f \) can be associated with a non-standard relational operator called antijoin (Kim, 1982):

\[
\text{antijoin}(r, s, \theta, A, B) = \{ u \in r | \forall v \in s, \neg (u.A \theta v.B) \}.
\]

A query such as:

\[
\text{select * from } r \text{ where } fc_1 \text{ and } A \text{ not in}_f (\text{select } B \text{ from } s \text{ where } fc_2)
\]

is expressed in relational algebra as:

\[
\text{antijoin}(\sigma(r, fc_1), \sigma(s, fc_2), \approx, A, B),
\]

where \( \sigma(r, fc) \) denotes the fuzzy selection of relation \( r \) by condition \( fc \).

The following example illustrates this kind of query: “find the young employees whose salary is close to none of the salaries of the employees having a large bonus”. In SQLf it can be expressed as:

\[
\text{select } \#e \text{ from } \text{Emp where age is young and salary not in}_f (\text{select } \text{salary from } \text{Emp where bonus is large}).
\]

### 5.3.3.2 Nesting with “exists”

A second type of nesting relies on the keyword \( \text{exists} \), and is used in SQL to test the non-emptiness of a set (relation delivered by a subquery). In order to generalize this operator, we consider the case where it uses a fuzzy set as an argument. The objective is to calculate the extent to which a fuzzy set is not empty. At least two interpretations are possible:

- quantitative, based on the cardinality of the fuzzy set,
- qualitative, based on the extent to which at least one element belongs to the set; here, one is looking for the largest value \( \alpha \) such that the \( \alpha \)-cut of the set is not empty; in other words, the largest grade of membership is obtained (the height of this set), i.e:

\[
\mu_{\text{exists}}(E) = \sup_{x \in \text{support}(E)} \mu_E(x).
\]

This second interpretation is preferred since, on the one hand, it does not require any knowledge of a membership function applicable to the
cardinality of the set, and on the other hand, it preserves the usual equivalences. In particular, one still has an equivalence between the expressions:

\[
\text{select } * \text{ from } r \text{ where } A \text{ in } (\text{select } B \text{ from } s)
\]

and

\[
\text{select } * \text{ from } r \text{ where exists } (\text{select } * \text{ from } s \text{ where } B \approx r.A).
\]

In both cases, if \(r'\) denotes the resulting relation, we have:

\[
\mu_{r'}(x) = \sup_{y \in s} \mu_{\approx}(y.B, x.A).
\]

The interpretation of \(\text{not exists}\) can be deduced canonically from that of \(\text{exists}\):

\[
\mu_{\text{not exists}}(E) = 1 - \mu_{\text{exists}}(E).
\] (5.8)

It should be mentioned that in SQLf, it is possible to express certain fuzzy equijoins using \(\text{exists}\) due to the equivalence between, on the one hand, this relational operator (equijoin) and the \(\text{in}\) (or \(\text{ini}\)) construct, and, on the other hand, \(\text{exists}\) and \(\text{in}\) (or \(\text{ini}\)) for these kinds of queries. For instance, the query: “find the employees who work in a department whose budget is about 1,000 times their own salary” can be expressed as:

\[
\text{select } #e \text{ from } \text{Emp where exists (select } * \text{ from } \text{Dep where } #d = \text{Emp.#dep and budget } \approx \text{Emp.salary} * 1,000).
\]

The operator \(\text{exists}\) can also be used for the expression of set difference, as we will see later.

5.3.3.3 Nesting with “all” and “any”

In SQL, quantified predicates are as follows: \(A \theta \{\text{all} \mid \text{any}\} \text{ SQ},\) where \(A\) is an attribute, \(\text{SQ}\) is a subquery and \(\theta\) is a comparator (\(=, \neq, <, \text{ etc}\)). In the context of SQLf, two kinds of extension are possible. The first consists of extending these predicates in order to permit the use of fuzzy relational operators (\(\approx, \text{slightly greater}, \text{ etc.}\)); the second aims at authorizing the use of fuzzy quantifiers (\(\text{around 3, almost all,} \ldots\)) instead of only \(\text{all}\) and \(\text{any}\). The first extension is presented here, while the second is discussed in Section 6.2.
Extension involving a fuzzy relational operator. An example of such a query is: “find the employees whose salary is much greater than that of any other employee in the same department”. The objective is to adopt an interpretation such that the usual SQL equivalences remain valid. For instance, such an equivalence holds in SQL between the statements:

\[
\text{select } * \text{ from } r \text{ where } A = \text{any} (\text{select } B \text{ from } s ...)
\]

and

\[
\text{select } * \text{ from } r \text{ where } A \text{ in} (\text{select } B \text{ from } s ...).
\]

More generally, the evaluation of “\(A \theta \text{ any } E\)” consists in checking whether there exists at least one value \(y\) from \(E\) such that \(A \theta y\) is true. In other words, there is an equivalence between the expressions:

\[
\text{select } * \text{ from } r \text{ where } A \theta \text{ any } (\text{select } B \text{ from } s \text{ where } fc)
\]

and

\[
\text{select } * \text{ from } r \text{ where exists } (\text{select } B \text{ from } s \text{ where } fc \text{ and } A \theta B),
\]

which leads us to interpret the generalized predicate \(\theta \text{ any}\) as:

\[
\mu_{\theta\text{any}}(a, E) = \sup_{b \in \text{support}(E)}\min(\mu_E(b), \mu_{\theta}(a, b)). \tag{5.9}
\]

Similarly, the evaluation of “\(A \theta \text{ all } E\)” comes down to assessing whether the statement “no element \(y\) from \(E\) is such that \(A \theta y\)” is false. We have an equivalence between the expressions:

\[
\text{select } * \text{ from } r \text{ where } A \theta \text{ all } (\text{select } B \text{ from } s \text{ where } fc)
\]

and

\[
\text{select } * \text{ from } r \text{ where not exists } (\text{select } B \text{ from } s \text{ where } fc \text{ and } \text{not } (A \theta B)).
\]

One thus deduces the following interpretation of \(\theta \text{ all}\):

\[
\mu_{\theta\text{all}}(a, E) = 1 - \sup_{b \in \text{support}(E)}\min(\mu_E(b), 1 - \mu_{\theta}(a, b)). \tag{5.10}
\]

For all as well as for \text{any}, the evaluation of the predicate consists of assessing the extent to which the result of the inner block satisfies the condition. The chosen interpretation is more qualitative than quantitative. It is worth mentioning that the extension suggested in Expressions (5.9) and (5.10) preserves the equivalence between “\(= \text{ any}\)” and \text{in}, which is a natural consequence of the equivalence between \text{in} and \text{exists}.
5.3.3.4 Nesting and scalar comparison

As mentioned above, SQLf queries can involve fuzzy comparisons between values. This kind of mechanism can also be used to extend predicates involving a comparison between an attribute value and the result of an aggregate function returned by a regular nested block (here we only consider aggregates defined on crisp sets). The general form is:

\[
\text{select } X \text{ from } r \text{ where } A \theta (\text{select } \text{agg}(B) \text{ from } s),
\]

where \( A \) and \( B \) are attributes defined on the same domain, \( \theta \) is a comparison operator, and \( \text{agg} \) denotes an aggregate function. The semantics of such a query is given by the formula:

\[
\mu_{\text{res}}(x) = \sup_{t \in r \text{ and } t.X = x} \mu_{\theta}(t.A, v),
\]

(5.11)

where \( \text{res} \) is the result of the query, and \( v \) is the value returned by the nested block. For example, “find the employees whose salary is much greater than the average of the salaries of the employees in the same department” can be expressed as:

\[
\text{select } #e \text{ from Emp E where } \text{salary} \gg (\text{select avg(salary) from Emp where } #d = E.#d).
\]

5.3.4 Set-oriented operators

In SQL, the only necessary set operator is the union; set difference and intersection can be expressed by means of other constructs. The union operator defines a relation resulting from the combination of the tuples of two other relations, after the clauses appearing in the select blocks have been applied. The extension of this operator to the case where the select blocks deliver fuzzy relations is straightforward. It is based on the usual definition of the union operator:

\[
\mu_{r \cup s}(x) = \max(\mu_r(x), \mu_s(x)).
\]

(5.12)

In the general case, the intersection can be expressed using a join or an exists nesting, and the difference can be expressed with a not exists nesting. It is therefore worth checking whether the fuzzy extensions previously proposed for joins and exists preserve these equivalences according to the usual semantics of the fuzzy intersection (resp. difference). Their interpretation
Simple Fuzzy Queries

is usually given by:

\[ \mu_{r \cap s}(x) = \min(\mu_r(x), \mu_s(x)), \quad (5.13) \]

\[ \mu_{r - s}(x) = \min(\mu_r(x), 1 - \mu_s(x)). \quad (5.14) \]

Consider the difference and the query:

\( \text{(select } A_1, \ldots, A_n \text{ from } r \text{ where } fc_1) \text{ minus } \text{(select } A_1, \ldots, A_n \text{ from } s \text{ where } fc_2), \)

where \( A_i \) is an attribute of \( r \) and \( s \). Let \( rf \) be the result of the first block, \( sf \) that of the second block, and \( res \) the global result. From Expression (5.14), we have:

\[ \mu_{res}(a_1, \ldots, a_n) = \min(\mu_{rf}(a_1, \ldots, a_n), 1 - \mu_{sf}(a_1, \ldots, a_n)), \]

where

\[ \mu_{rf}(a_1, \ldots, a_n) = \sup_{x \in r \land x.A_1 = a_1 \land \ldots \land x.A_n = a_n} \mu_{fc_1}(x) \]

and

\[ \mu_{sf}(a_1, \ldots, a_n) = \sup_{y \in s \land y.A_1 = a_1 \land \ldots \land y.A_n = a_n} \mu_{fc_2}(y). \]

The translation of the query using \textit{not exists} leads to:

\( \text{select } A_1, \ldots, A_n \text{ from } r \text{ where } fc_1 \text{ and not exists } \text{(select } * \text{ from } s \text{ where } fc_2 \text{ and } A_1 = r.A_1 \text{ and } \ldots \text{ and } A_n = r.A_n). \)

According to the semantics of \textit{not exists}, we obtain:

\[ \mu_{res}(a_1, \ldots, a_n) = \sup_{x \in r \land x.A_1 = a_1 \land \ldots \land x.A_n = a_n} \min(\mu_{fc_1}(x), 1 - \sup_{y \in s} \min(\mu_{fc_2}(y), \mu=(y.A_1, a_1), \ldots, \mu=(y.A_n, a_n))). \]

The equivalence of these two interpretations is straightforward. In the same way, it can be checked that the extensions of \textit{join} and \textit{exists} make it possible to express the intersection of fuzzy relations in accordance with Expression (5.13).

Intersection can also be translated with an “in” or “= any” nesting, whereas set difference is expressible with a “not in” or “≠ all”
nesting. For example, assuming the existence of a relation \( \text{Anc} \) of schema \((#e, #d, \text{years})\), the query: “find the young employees who have never worked for more than three years in any department” may be expressed by:

\[
\begin{align*}
(a) & \quad \text{select } #e \text{ from } \text{Emp} \text{ where age is young and not exists } \\
& \quad (\text{select } * \text{ from } \text{Anc} \text{ where } #e = \text{Emp}.#e \text{ and years } > 3) \\
(b) & \quad \text{select } #e \text{ from } \text{Emp} \text{ where age is young and } #e \text{ not in } \\
& \quad (\text{select } #e \text{ from } \text{Anc} \text{ where years } > 3) \\
(c) & \quad \text{select } #e \text{ from } \text{Emp} \text{ where age is young and } #e \neq \text{all } \\
& \quad (\text{select } #e \text{ from } \text{Anc} \text{ where years } > 3).
\end{align*}
\]

According to the preceding general result (the equivalence between difference and \( \text{not exists} \)), and due to the respective equivalences between these different nestings, one may assert that these formulations preserve the usual semantics of the set difference operator. A similar result can be obtained in the same way for the intersection.

### 5.3.5 Relation partitioning

In SQL, a relation may be partitioned into subrelations (groups) according to the values of one or several attributes (those specified in a \textit{group by} clause). By doing so, each partition groups tuples which have the same value on this (these) considered attribute(s). This functionality is retained in SQLf, and in this context, the \textit{having} clause can be used along with a fuzzy-set-oriented condition aimed at the selection of groups. In this subsection, we discuss the various possible conditions involving aggregate functions (\( \min \), \( \sum \), etc.). The case of more complex conditions involving fuzzy quantifiers is discussed in Section 6.2.

#### 5.3.5.1 Qualification of partitions by means of aggregate functions

Like numerous other relational languages, SQL offers the possibility of evaluating numerical operators upon a set of tuples. This is performed by means of operators, called aggregate functions, among which are: \( \sum \), \( \text{count} \), \( \text{avg} \), \( \min \), and \( \max \). In the following, we limit the scope of the discussion to the case where aggregates apply to a crisp (non-fuzzy) relation. A more general situation is discussed in Section 6.3. An example of an SQL query involving an aggregate for selecting groups is: “find the departments where the
average salary of the engineers is higher than $3,500”:

\[
\text{select } \#\text{dep from Emp where job = “Engineer”}
\text{group by } \#\text{dep having avg(salary)} \geq 3,500.
\]

In SQLf, the process is similar but there is the additional possibility of using the result of an aggregate as an argument for fuzzy criteria. For example, the query: “find the departments where the average salary of the engineers is about $3,500” is expressed in SQLf by:

\[
\text{select } \#\text{dep from Emp where job = “Engineer”}
\text{group by } \#\text{dep having avg(salary)} \approx 3,500.
\]

Some equivalences concerning queries which involve aggregates inside a having clause exist in SQL. For instance, the previous SQL query may also be expressed using a particular nesting in which the inner block returns a single value (by means of an aggregate) which is part of a simple comparison term:

\[
\text{select } \#\text{dep from Emp E where } 3500 \leq
\left( \text{select avg(salary) from Emp where } \#\text{dep} = E.\#\text{dep and job = “Engineer”} \right).
\]

This equivalence is valid since the condition \#\text{dep} = E.\#\text{dep} does the same as a group by. More generally, with a regular (non-fuzzy) relation, it can be shown that the following SQLf queries are equivalent:

\[
\text{select } A \text{ from } r \text{ where } bc \text{ group by } A
\text{ having } fc_1(\text{agg}_1(B_1)) \text{ connective } \cdots \text{ connective } fc_p(\text{agg}_p(B_p))
\]

and

\[
\text{select } A \text{ from } r_1 \text{ where } bc \text{ and }
\left( \text{select agg}_1(B_1) \text{ from } r \text{ where } A = r_1.A \text{ and } bc \right)
\text{ connective } \cdots \text{ connective }
\left( \text{select agg}_p(B_p) \text{ from } r \text{ where } A = r_1.A \text{ and } bc \right),
\]

where \text{agg}_i stands for an aggregate function, \text{bc} denotes a Boolean condition and \text{fc}_i represents a fuzzy predicate.
5.4 Interface for User-Defined Terms and Operators

Since the semantics of a fuzzy term relies on its membership function, any fuzzy querying system must provide the users with a convenient way to define the membership functions of the fuzzy terms (and quantifiers, if any) that they wish to include in a query.

In practice, the membership function associated with a fuzzy set $F$ is often chosen to be of a trapezoidal shape. $F$ is then expressed by the quadruplet $(A, B, a, b)$ where $\text{core}(F) = [A, B]$ and $\text{supp}(F) = [A - a, B + b]$; see Figure 5.1. If the function is a right (resp. left) shoulder, it can be encoded by $(A, k_2, a, 0)$ (resp. $(k_1, B, 0, b)$) where $[k_1, k_2]$ represents the domain of the attribute concerned.

In Kacprzyk and Zadrożny (1995) and Zadrożny and Kacprzyk (1996), the authors describe a fuzzy querying system built on top of the commercial DBMS Microsoft Access, where the definition of fuzzy predicates, fuzzy comparators and fuzzy quantifiers is performed through an appropriate graphical user interface. Goncalves and Tineo (2001a, b) introduce commands aimed at defining fuzzy predicates (create fuzzy predicate) and fuzzy comparators (create comparator) into SQLf.

For simple fuzzy queries — those without any fuzzy modifiers, quantifiers or comparators — another solution is to include the definitions of the fuzzy predicates in the query itself. Assuming that trapezoidal membership functions are used, when a fuzzy predicate concerns a numerical attribute one just has to specify its core (ideal values) and support (acceptable values):

- $\text{att} \leq u$ translates into: $(\text{ideal} : \text{att} \leq u, \text{acceptable} : \text{att} < u + v)$,

![Fig. 5.1 Trapezoidal membership function]
Simple Fuzzy Queries

- $att \approx u$ translates into:

  $(\text{ideal} : att \in [u-v, u+v], \text{acceptable} : att \in [u-v-w, u+v+w])$,

- $att \geq u$ translates into: $(\text{ideal} : att \geq u, \text{acceptable} : att > u-v)$.

However, when a fuzzy predicate concerns a categorical attribute, it has to be given in extension.

**Example 5.2.** Consider a relation $cars$ of schema $(\text{make, category, price, color, mileage})$. An example of a fuzzy query involving the definition of fuzzy terms is:

```sql
select * from cars
where (color is $C_1$) and (make is $C_2$) and
    (category is $C_3$) and (price is low$_P$) and (mileage is low$_M$)
with preferences
$C_1$: \{1/blue, 1/black, 0.8/green, 0.7/gray, 0.5/red, 0.3/orange\}
$C_2$: \{1/VW, 0.9/Audi, 0.7/BMW, 0.6/Seat, 0.6/Opel, 0.4/Ford\}
$C_3$: \{1/sedan, 0.8/roadster, 0.7/coupe, 0.4/SUV\}
low$_P$: (ideal: price $\leq$ 4,000, acceptable: price $\leq$ 6,000)
low$_M$: (ideal: mileage $\leq$ 15,000, acceptable: mileage $\leq$ 20,000).
```

In Bodenhofer and Kung (2004), the authors suggest the use of fuzzy orderings in order to define the semantics of fuzzy predicates defined on numerical domains. They consider three comparison operators, namely *is at least*, *is at most*, and *is within*, that they make tolerant by means of a clause *tolerate up to*. For instance, the fuzzy predicate "is at most $q$ tolerate up to $C" may be associated with a membership function $\mu$, based on the use of Łukasiewicz’s t-norm, defined as follows:

$$\mu(x) = L_C(x, q)$$

where

$$L_C(u, v) = \begin{cases} 
1 & \text{if } u \leq v \\
\max \left(1 - \frac{1}{C} \cdot |u - v|, 0 \right) & \text{otherwise.}
\end{cases}$$
An example of the queries that the authors consider is:

```
select * from cars
where make = "Ford"
and price is at most 20,000 tolerate up to 5,000
and mileage is at most 30,000 tolerate up to 10,000;
```

It is also possible to weight the different atomic conditions involved in a (generalized) conjunction, by means of a clause `weighted by`.

### 5.5 Contextual Queries

In this section, we deal with the situation where the user is not aware of the content of the database that he/she wants to query, and thus is unable to define (fuzzy or non-fuzzy) selection predicates referring explicitly to some constants from the domains. Despite this absence of knowledge, the user may want to retrieve the best answers with respect to some relative conditions. As an example, consider a user who wants to move to a different country and aims at finding the “best” cities to settle in according to the following preferences: population between 50,000 and 100,000 (hard constraint), low average price per square meter, low crime rate, medium average annual temperature. We assume that the user does not know the values taken by the attributes “average price per square meter”, “crime rate”, “average annual temperature” in the country considered, but still wants to find the best solutions relative to the existing possibilities. Note that in the system Preference SQL (Kießling and Köstler, 2002), one may express conditions of the form “attribute A is as high (resp. low) as possible” — although not “A is medium” — but again, since it is based on Pareto order, one can only “combine” such conditions so as to obtain a partial order. In other words, Pareto-based approaches such as Preference SQL do not permit the expression of trade-offs between a predicate of the form “A is low (resp. high)” and other predicates. This seems desirable in many cases, however.

The idea that we advocate in the following is to use the fuzzy-set-based framework and to define the fuzzy predicates `high`, `medium` and `low` in a relative way, using the minimal, average and maximal values of the attribute values present in the associated query-defined context (in the example above: the cities whose population is between 50,000 and 100,000). There is
a connection between this idea and the proposal by Tudorie et al. (2006) where a fuzzy predicate can be defined relative to the context created by another one. For example, in a query such as “find the inexpensive cars among the high-speed ones”, the authors suggest adapting the definition of the fuzzy term *inexpensive* by taking into account the price values associated with high-speed cars. However, the authors do not formalize the notion of a context.

Even though the approach by Agrawal et al. (2006) deals with preferences and contexts, it does not have much in common with the idea developed here since: (i) it deals with preferences over categorical attributes whereas we consider numerical ones; (ii) it handles contexts and preferences which are given explicitly (and statically) by the user in the form of a set of conditions “attribute = value”, whereas we deal with contexts which are specified dynamically by means of a specific clause in an extended SQL language, as well as with preferences which are expressed in an implicit and relative way; (iii) the approach proposed in Agrawal et al. (2006) does not combine contextual preferences with non-contextual ones, while the method presented here makes it possible to do so thanks to the commensurability assumption underlying the fuzzy-set-based model. As to the approaches of Holland and Kießling (2004), Stefanidis et al. (2007), and van Bunningen et al. (2006), they deal with a very different notion of context, namely the context in which the user is situated, which implies quite different issues.

In the following, we show how conditions of the form “A is low (resp. medium, high)” can be modeled in the framework of a fuzzy-set-based query language. The basic idea is to interpret these conditions relative to a given context specified in the user query. First, we clarify this latter notion.

**Definition 5.1.** A *query-defined context* is a referential of values returned by a (sub)query, on which a predicate can be defined in a relative manner.

For instance, if one has available a relation describing employees, one may define the predicate young (interpreted as “age is low”) in the context of the engineers’ ages, or the ages of those employees whose monthly salary is less than $2,500, and so on. Clearly, the relative meaning of young depends on the referential considered. First consider the simple case where a single context is given in the query.
5.5.1 Queries with one level of context

The syntax of the basic form of SQLf queries considered is as follows:

\[
\text{select } k \ X \ \text{from } \ r \ \text{where} \ \text{cond}_1 \ \text{and} \ \text{cfc} \\
\text{context} \ \{ \text{Boolean query} \mid \text{fuzzy query with a threshold } \alpha \}.
\]

In this query, \( k \) denotes a desired number of answers (the best ones), \( X \) is a set of attributes from relation \( r \), \( \text{cond}_1 \) is a selection condition which may involve Boolean predicates and explicit fuzzy predicates (i.e., predicates whose membership function is user-defined), and \( \text{cfc} \) is a contextual fuzzy condition. Notice that contextual fuzzy terms can be altered by means of fuzzy modifiers such as \textit{very}, \textit{relatively}, etc., so as to express a great variety of nuanced contextual predicates.

The new clause introduced by the keyword \textit{context} aims to define the referential of tuples that serves as a basis for constructing the fuzzy terms present in \( \text{cfc} \). Two possibilities are offered; one may either use:

- a Boolean subquery (not only a selection–projection–join query, but any kind),
- a fuzzy query involving a qualitative threshold \( \alpha \).

A constraint is that the result of the context clause must include attributes of the same domains as those concerned by the contextual predicates. In the case of a fuzzy context query, the context is made of the tuples whose membership degree to the result of the fuzzy query is at least equal to \( \alpha \). Hence, the context is always defined as a \textit{crisp set}.

\textbf{Example 5.3.} Consider a relation \( FC(\text{name}, \ \text{pop}, \ \text{crime}, \ \text{temp}, \ \text{univ}) \) describing some French cities. Attribute \text{crime} corresponds to the crime rate and attribute \text{temp} to the annual average temperature in a given city, while \text{pop} gives the number of inhabitants. Attribute \text{univ} indicates whether there is a university in the city. Here are some examples:

(1) find the 10 best cities whose population is between 50,000 and 100,000, where a university is located, and where the crime rate is low relative to the set of French cities in the same range of population.

\[
\text{select } 10 \ \text{name from } FC \\
\text{where } \text{pop between 50k and 100k and univ = “yes” and crime is low} \\
\text{context (select * from FC where pop between 50k and 100k).}
\]
Simple Fuzzy Queries

(2) find the 10 best cities whose population is large, where the annual average temperature is around 15 degrees, and where the crime rate is low relative to the set of French cities whose population fits the fuzzy predicate large at a degree at least equal to 0.7. Here, the fuzzy predicate large is assumed to be user-defined.

```
select 10 name from FC
where pop is large and temp is around 15 and crime is low
context (select 0.7 * from FC where pop is large).
```

As mentioned above, the idea is to define the membership functions of fuzzy predicates such as “A is low”, “A is medium” and “A is high” in terms of the minimum, average and maximum A-values in the result of the query which defines the context. For interpreting low, medium and high, we propose the following default definitions (see also Figure 5.2):

- **lowA** decreases linearly from 1 to 0 when A moves from min(A) to avg(A).
- **mediumA** is represented by an isosceles triangle centered in avg(A) and whose support is the interval \([\text{avg}(A) - \delta, \text{avg}(A) + \delta]\), where \(\delta = \min(\text{avg}(A) - \min(A), \max(A) - \text{avg}(A))\). This corresponds to interpreting medium as “close to the average”.
- **highA** increases linearly from 0 to 1 when A moves from avg(A) to max(A).

It is of course possible to argue about these definitions. One could choose, for instance, to say that full satisfaction is assumed around the minimum (for low) or the average (for medium) or the maximum (for high). One could also imagine defining a partition of more than three implicit fuzzy predicates, and so on. In order to make the system more user-friendly, it is
of course conceivable to let the user personalize these membership functions through an appropriate interface. Note that the implicit assumption made here is that the context includes enough data to represent a statistically significant subset of the data (so as to be able to compute significant values for \( \min \), \( \text{avg} \) and \( \max \)).

**Example 5.4.** Consider the (partial) extension of a relation \( FC \) describing French cities represented in Table 5.2, and the query:

\[
\text{select 10 name from } FC \\
\text{where (pop between 50k and 100k) and (crime is low) and (temp is medium)} \\
\text{context (select * from } FC \text{ where pop between 50k and 100k).}
\]

Suppose that the minimum (resp. average, resp. maximum) value for attribute \( \text{crime} \) over the set of cities in the specified population interval is 0.8 (resp. 4.1, resp. 11.3) and the minimum (resp. average, resp. maximum) value for attribute \( \text{temp} \) over the same set of cities is 8 (resp. 14, resp. 22). One can then define the fuzzy predicates \( \text{low}_{\text{crime}} \) and \( \text{medium}_{\text{temp}} \) as described above, and the evaluation of the query on the first tuple (Montauban) provides the degrees 0.21 (for “\( \text{crime is low} \)” and 0.75 (for “\( \text{temperature is medium} \)”)). If we assume that the conjunction is interpreted as a minimum, the first tuple receives an overall satisfaction degree of 0.21.

It is appropriate to mention the possibility of introducing a notion of default context so as to make the user’s task easier. Informally, the context default clause corresponds to a query involving the Boolean part of the where clause from the original user query (i.e., the where clause without its fuzzy conditions, in particular those involving contextual predicates).
For instance, the query from Example 5.4 could be expressed more simply as:

```sql
select 10 name from FC
where (pop between 50k and 100k) and (crime is low) and
(temp is medium)
context default.
```

### 5.5.2 Queries with several levels of context

We now extend the approach presented above, to consider several levels of context. An example is: “find the apartments with a medium price located in cities of less than 100,000 inhabitants with a low crime rate” where low is assumed to be defined with respect to the set of cities of less than 100,000 inhabitants, while medium is defined relative to the set of flats in the city where a given apartment is located. Consider the relations: `City (#id, name, pop, crime, temp)` and `Flat (#app, #city, category, price, surface)`, and the query: “find the 10 best apartments of category F5 which have a medium price and are located in a city whose population is between 50,000 and 100,000 and whose crime rate is low”. The term low — referring to the crime rate — is defined relative to the context of cities whose population is between 50,000 and 100,000, while the term medium — concerning the price — is defined in the context of the city where the considered apartment is located, for the category considered. In SQLf, this query can be expressed as:

```sql
select 10 #app from Flat f1
where price is medium and category = “F5” and #city in
(select #id from City
where pop between 50k and 100k and crime is low
context default)
context
(select * from Flat where #city = f1. #city and category = “F5”).
```

It is also conceivable to jointly use contextual fuzzy predicates defined on different contexts. For instance, consider a relation `Emp (id, name, age, ed-level, salary)` describing the employees of a given company and the query: “find every employee who has a medium education level (relative to the entire set of employees) and a high salary (relative to the set of employees around his/her age at a degree at least equal to 0.7)”. Expressing this query
in SQLf involves extending the context clause introduced above so as to deal with different referentials. A possible formulation is:

```sql
select id, name from Emp e
where (ed-level is medium) and (salary is high)
context
  (medium: default;
   high: (select 0.7 * from Emp where age around e.age)).
```

In this query, it is assumed that around is an explicit (i.e., user-defined) fuzzy comparator extending the equality.

Another extension consists of using the “top-k” mechanism in a nested manner. This allows restriction of the referential over which the contextual terms are defined to the most relevant elements only. For instance, in the preceding query about flats, one might choose to assess only the flats which are located in a city which is sufficiently satisfactory (e.g., among the 20 best) with respect to the criterion on crime. Yet another way of introducing nesting is to have a context clause which itself involves a context clause.

### 5.6 Evaluation of Simple Fuzzy Queries

In conventional DBMSs, the issue of query evaluation remains somewhat open since, given a query, the optimal evaluation plan cannot in general be determined. For fuzzy queries, the process becomes even more tricky, since access paths cannot be directly used. In this context, it appears useful to understand the connections which exist between properties tied to regular (Boolean) conditions and fuzzy ones, so that fuzzy query processing can be reduced to Boolean query processing (at least partly). An evaluation method, called derivation (Bosc and Pivert, 1993), exploiting such properties, is described in the following. The applicability of this method to the evaluation of different types of SQLf queries is discussed, as well as the integration of a derivation-based SQLf query interface on top of a regular relational DBMS.

#### 5.6.1 Derivation principle

The strategy presented here assumes that a threshold $\alpha$ is associated with an SQLf query in order to retrieve the $\alpha$-level cut of its answer set. The idea
is to use an existing database management system which will process regular Boolean queries. An SQL query is derived from an SQLf expression in order to retrieve a superset of the α-level cut of its answer set. Then, the fuzzy query can be processed on this superset, thus avoiding an exhaustive scan of the whole database. The principle is to express the α-level cut in terms of a query involving only regular operators and expressions. The principal problem is to distribute the α-level cut operation applying to a selection expression into its constituent elements.

5.6.1.1 Transformation of base predicates
Consider a subset of base predicates of the form: (i) “attribute is fuzzy term”, (ii) “attribute θ value”, or (iii) “attribute₁ θ attribute₂” where θ is a fuzzy comparison operator (e.g., age is young, quantity ≈ 150, qty₁ ≫ qty₂). The objective is to replace such predicates with Boolean expressions, given a threshold α. In the first and second cases, the α-level cut of the initial expression is of the form “attribute ∈ [v₁, v₂]” if the characteristic function of the fuzzy term or the comparator (θ) has a trapezoidal shape. In the third case, the expression obtained may be defined as a function of the difference between (or the quotient of) the two attribute values.

Example 5.5. Consider the SQLf expression:

\[
\text{select 0.7 #e, name from Emp where age is young.}
\]

If the membership function of young is that represented in Figure 5.3, one has:

\[
\mu_{\text{young}}(x) \geq 0.7 \iff \text{age} \leq 30.
\]
In other terms:
\[ DNC(\text{age is young}, \geq, 0.7) = [\text{age} \leq 30], \]
where \( DNC(\mu_{fc} \geq, \alpha) \) denotes the derived necessary condition obtained from the condition \( \mu_{fc}(s) \geq \alpha \) (in this example, as for any atomic predicate involving one attribute, the condition is necessary and sufficient, which is the ideal situation). The derived SQL query would then be:

\begin{verbatim}
select #e, name from Emp where age \leq 30.
\end{verbatim}

It should be noted that predicates involving several attributes (e.g., the predicate \( \text{slim} \) involving the two attributes \( \text{height} \) and \( \text{weight} \)) cannot be derived in general. Nevertheless, it is worth noting that a non-derivable predicate can always be replaced by \( \text{true} \) in the final derived query.

5.6.1.2 Single-block projection-selection-join queries

First, we present the derived expressions tied to the usual connectives (and/or):

\[
\mu_{P_1 \land \ldots \land P_n}(x) \geq \alpha \leftrightarrow \min(\mu_{P_1}(x), \ldots, \mu_{P_n}(x)) \geq \alpha
\]
\[
\leftrightarrow \mu_{P_1}(x) \geq \alpha \land \ldots \land \mu_{P_n}(x) \geq \alpha.
\]

Thus:

\[
DNC(P_1 \text{ and } \cdots \text{ and } P_n, \geq, \alpha) = DNC(P_1, \geq, \alpha) \text{ and } \ldots \text{ and } DNC(P_n, \geq, \alpha).
\]

Similarly:

\[
\mu_{P_1 \lor \ldots \lor P_n}(x) \geq \alpha \leftrightarrow \max(\mu_{P_1}(x), \ldots, \mu_{P_n}(x)) \geq \alpha
\]
\[
\leftrightarrow \mu_{P_1}(x) \geq \alpha \lor \cdots \lor \mu_{P_n}(x) \geq \alpha.
\]

Thus:

\[
DNC(P_1 \text{ or } \cdots \text{ or } P_n, \geq, \alpha) = DNC(P_1, \geq, \alpha) \text{ or } \cdots \text{ or } DNC(P_n, \geq, \alpha).
\]

We also get the rules (useful when the negation is used):

\[
DNC(P_1 \text{ and } \cdots \text{ and } P_n, <, \alpha) = DNC(P_1, <, \alpha) \text{ or } \cdots \text{ or } DNC(P_n, <, \alpha)
\]
and

\[ DNC(P_1 \lor \cdots \lor P_n, <, \alpha) = DNC(P_1, <, \alpha) \land \cdots \land DNC(P_n, <, \alpha). \]

**Example 5.6.** Consider the initial SQLf expression:

```
select 0.8 #e, name from Emp
where age is young and salary \gg bonus.
```

If the characteristic functions for *young* and \( \gg \) are those pictured in Figure 5.4, one obtains:

\[ DNC(\text{age is young and salary } \gg \text{bonus}, \geq, 0.8) = DNC(\text{age is young}, \geq, 0.8) \land DNC(\text{salary } \gg \text{bonus}, \geq, 0.8) = (\text{age } \leq 29) \land (\text{salary} - \text{bonus}) \geq 1,200). \]

The final SQL query is then:

```
select #e, name from Emp
where age \leq 29 and (salary - bonus) \geq 1,200.
```

The above example corresponds to a case where the derivation process is exact, i.e., the derived query delivers the exact \( \alpha \)-level cut of the initial fuzzy query. In some cases (e.g., predicates connected by mean operators), this can no longer be the case, and we only recover a superset of the desired \( \alpha \)-level cut. The operators belonging to the class of means, denoted by \( mc \),
have two general properties. The first is:

\[
\min(\mu_{P_1}(x), \ldots, \mu_{P_n}(x)) \leq mc(\mu_{P_1}, \ldots, \mu_{P_n})(x) \\
\leq \max(\mu_{P_1}(x), \ldots, \mu_{P_n}(x)).
\]

As a consequence, two “weak” rules hold for any mean:

\[
mc(\mu_{P_1}, \ldots, \mu_{P_n}) \geq \alpha \Rightarrow DNC(P_1, \geq, \alpha) \text{ or } \cdots \text{ or } DNC(P_n, \geq, \alpha),
\]

\[
mc(\mu_{P_1}, \ldots, \mu_{P_n}) < \alpha \Rightarrow DNC(P_1, <, \alpha) \text{ or } \cdots \text{ or } DNC(P_n, <, \alpha).
\]

The second property is monotonicity:

\[
\forall i, x_i \geq x_i' \Rightarrow \mu_{mc}(x_1, \ldots, x_n) \geq \mu_{mc}(x'_1, \ldots, x'_n).
\]

Using this, one can point out some more specific rules for different mean operators (see Bosc and Pivert, 1995a). For example, to derive a query involving an arithmetic mean \((am)\) connecting two fuzzy predicates \(P_1\) and \(P_2\), we would have to use the specific weak derivation rules:

\[
am(P_1, P_2) \geq \alpha \\
\Rightarrow DNC(P_1, \geq, max(2\alpha - 1, 0)) \text{ and } DNC(P_2, \geq, max(2\alpha - 1, 0)),
\]

\[
am(P_1, P_2) < \alpha \\
\Rightarrow DNC(P_1, <, max(2\alpha - 1, 0)) \text{ and } DNC(P_2, <, max(2\alpha - 1, 0)).
\]

We now consider the t-norm product \(\otimes\). The derivation of a Boolean condition associated with \((\otimes(P_1, P_2) \geq \alpha)\) is possible thanks to the monotonicity of the product, and we obtain:

\[
\otimes(P_1, P_2) \geq \alpha \Rightarrow DNC(P_1, \geq, \alpha) \text{ and } DNC(P_2, \geq, \alpha).
\]

However, the property of monotonicity does not make it possible to derive a condition from \((\otimes(P_1, P_2) < \alpha)\). In such a situation, one can use exclusion cases, for instance:

\[
(\mu_{P_1}(x) \geq \sqrt{\alpha} \land \mu_{P_2}(x) \geq \sqrt{\alpha}) \Rightarrow \mu_{\otimes(P_1, P_2)}(x) \geq \alpha,
\]

from which we obtain the necessary condition:

\[
\otimes(P_1, P_2) < \alpha \Rightarrow DNC(P_1, <, \sqrt{\alpha}) \text{ or } DNC(P_2, <, \sqrt{\alpha}).
\]
5.6.1.3 Nested queries

We now review the Boolean expressions corresponding to the derivation of the subqueries presented in Section 5.3.3.

Recall that in SQLf, the operator \( \text{in} \) allows for testing whether the value of an attribute in the current tuple belongs to the (fuzzy) set of values returned by a subquery. The predicate \( \text{a in (select } B \text{ from } S \text{ where fc) } \) is defined by:

\[
\mu_{\text{in}}(a, SQ) = \sup_{b \in SQ} \land b = a \mu_{SQ}(b),
\]  

(5.15)

where \( SQ \) denotes the subquery returning a fuzzy bag of \( B \)-values from relation \( S \). According to Expression (5.15), we have:

\[
\left[ a \text{ in (select } B \text{ from } S \text{ where fc)} \right] \geq \alpha 
\Leftrightarrow 
\sup_{s \in S} \land s.B = a \mu_{fc}(s) \geq \alpha 
\Leftrightarrow 
\exists s \in S \text{ such that } (s.B = a \land \mu_{fc}(s) \geq \alpha) \Leftrightarrow 
\]

a \text{ in (select } B \text{ from } S \text{ where } \text{DNC}(fc, \geq, \alpha)).

The final derivation step is strong if there exists a strong derivation rule for condition \( fc \); it is weak otherwise.

In case of a negation, we have:

\[
\left[ a \text{ not in (select } B \text{ from } S \text{ where fc)} \right] \geq \alpha 
\Leftrightarrow 
1 - \sup_{s \in S} \land s.B = a \mu_{fc}(s) \geq \alpha 
\Leftrightarrow 
\sup_{s \in S} \land s.B = a \mu_{fc}(s) \leq 1 - \alpha 
\Leftrightarrow 
\forall s \in S \text{ such that } s.B = a, \mu_{fc}(s) \leq 1 - \alpha.
\]

Finally, the derived expression is:

\( a \text{ not in (select } B \text{ from } S \text{ where } \text{DSC}(fc, >, 1 - \alpha)), \)

where \( \text{DSC}(fc, >, 1 - \alpha) \) denotes the derived sufficient condition associated with \( fc \). Because of the negation, one cannot use a necessary condition for the inner block. On the contrary, one needs a sufficient condition so as not to risk obtaining extra tuples from the inner block. Derived sufficient conditions can be obtained the same way as necessary conditions, using properties associated with the operators involved. For instance, one obtains
the two general rules for any mean operator \( mc \):

\[
DSC(P_1, \geq, \alpha) \text{ and } \cdots \text{ and } DSC(P_n, \geq, \alpha) \Rightarrow mc(P_1, \ldots, P_n) \geq \alpha,
\]

\[
DSC(P_1, <, \alpha) \text{ and } \cdots \text{ and } DSC(P_n, <, \alpha) \Rightarrow mc(P_1, \ldots, P_n) < \alpha,
\]

and based on the property of monotonicity, specific rules can be found. For instance, in the case of the arithmetic mean \( (am) \), we have:

\[
DSC(P_1, \geq, 2\alpha) \text{ or } \cdots \text{ or } DSC(P_n, \geq, 2\alpha) \Rightarrow am(P_1, \ldots, P_n) \geq \alpha
\]

\[
DSC(P_1, <, 2\alpha - 1) \text{ or } \cdots \text{ or } DSC(P_n, <, 2\alpha - 1) \Rightarrow am(P_1, \ldots, P_n) < \alpha.
\]

For similar reasons, the condition “exists (select \( B \) from \( S \) where \( fc \) \( \geq \alpha \)” can be transformed into:

\[
\text{exists (select } B \text{ from } S \text{ where } DNC(fc, \geq, \alpha))
\]

and its negation into:

\[
\text{not exists (select } B \text{ from } S \text{ where } DSC(fc, >, 1 - \alpha)).
\]

**Example 5.7.** To illustrate the transformation procedure, consider the query: “find the departments having a medium budget and no well-paid employees” written in SQLf:

\[
\text{select } 0.5 \# d \text{ from } \text{Dep where budget is medium and not exists (select * from Emp where salary is well-paid and } \# \text{dep = Dep.}\# d).}
\]

Consider the fuzzy terms *medium* and *well-paid*, represented in Figure 5.5. The 0.5-cut of the query can be calculated on the basis of the regular

![Fig. 5.5 Membership functions of medium (left) and well-paid (right)](image-url)
SQL expression:

```sql
select #d from Dep
where budget ≥ 2.2 and budget ≤ 2.8 and not exists
  (select #dep from Emp where salary > 5000 and #dep = Dep.#d).
```

5.6.2 Derivation-based processing of SQLf queries

5.6.2.1 General strategy

In this section, we study the possible architecture of a relational database system able to deal with fuzzy queries. There are two principal ways (and architectures) to process fuzzy queries, as described in Bosc and Pivert (2000):

- use a regular DBMS, and develop an additional layer playing the role of an interface, or
- build a completely new system including fuzzy query processing techniques (algorithms) in its kernel.

We focus on the first strategy which is strongly connected with the derivation method, since it implies a query transformation step: the basic principle is indeed to express the $\alpha$-cut of a fuzzy query in terms of a query involving only regular (Boolean) expressions. It is then possible to use an existing relational DBMS to process the (derived) regular SQL query. By doing so, one can expect to take advantage of the implementation mechanisms of the DBMS to reach acceptable performances.

However, the problem is a bit more complex than that, since the derived query provides a regular (crisp) relation whereas we want to obtain a fuzzy relation, i.e., a relation containing weighted tuples. The computation of the final membership degrees can be performed by the DBMS during the processing of the derived SQL query, if the system allows for the inclusion of external functions in the query. If this is not the case, then the degrees must be computed by means of an external procedural program. We will make this assumption in the following, for the sake of generality.

Fuzzy query processing mainly reduces to a transformation procedure sitting on top of an existing DBMS, according to the architecture depicted in Figure 5.6. The translation mechanism generates a procedural evaluation program written in a host language, including one or more SQL queries obtained by applying the derivation principle to the initial fuzzy query,
and it also determines the expressions that allow for the computation of the degrees. The resulting program processes the SQL queries, computes the membership degrees, and calibrates the answer if necessary.

5.6.2.2 Single-block selection–projection–join queries

The general form of a single-block PSJ query is:

\[
\text{select } * \text{ from } R_1, \ldots, R_n \text{ where } fc,
\]

where \( fc \) is a condition that may involve fuzzy selections and fuzzy joins. Denote by \( X \) the set of attributes of \( R_1, \ldots, R_n \). Below we show the external program, written in a host language, that can be used to evaluate such a query.

Algorithm 5.1.

\[
\begin{align*}
\text{declare } & c \text{ cursor for select } * \text{ from } R_1, \ldots, R_n \text{ where } DNC(fc, \geq, \alpha); \\
\text{open } & c; \\
\text{fetch } & c \text{ into } x; \\
\text{while } & \text{code}(c) \neq \text{active_set_empty} \text{ do}
\end{align*}
\]
Simple Fuzzy Queries

begin
    compute \( \mu \) for the current tuple \( x \);
    if \( \mu \geq \alpha \) then result := result + \{ (\mu/x) \};
    fetch c into x;
end;
close c;

Obviously, the test "\( \mu \geq \alpha \)" can be omitted in case of a strong derivation.

5.6.2.3 Nested queries

The evaluation method is based on the use of nested loops. In the case of a nested query, the computation of a final degree implies computation of a set of degrees generated by the inner block. The external program then involves several cursors, one for each level of nesting in the derived query. Since there is an equivalence between a query involving a [not] in nesting and a query involving a [not] exists nesting (Bosc and Pivert, 1995b), we only consider this latter case.

"Exists" nesting. The generic query format considered is:

\[
\begin{align*}
\text{select } * \text{ from } R \text{ where } fc_R \text{ and exists } \\
(\text{select } * \text{ from } S \text{ where } S.B = R.A \text{ and } fc_S).
\end{align*}
\]

(Q\text{ex})

The final degree of a tuple \( t \) of relation \( R \), i.e., its degree of membership to the resulting relation \( res \), is expressed by:

\[
\mu_{res}(t) = \min(\mu_{fc_R}(t), \mu_{\text{exists}}(SQ(t)))
\]

where \( \mu_{\text{exists}}(SQ(t)) = \sup_{t' \in S \land t'.B = t.A \land fc_S}(t') \).

Since one wants to compute the \( \alpha \)-cut of the initial query, one imposes the condition: \( \mu_{res}(t) \geq \alpha \), with the two following pruning conditions:

\[
\mu_{res}(t) \geq \alpha \Rightarrow \mu_{fc_R}(t) \geq \alpha \Rightarrow DNC(fc_R, \geq, \alpha)(t) \quad \text{(H1)}
\]

and:

\[
\mu_{\text{exists}}(SQ(t)) \geq \alpha
\]

\( \Leftrightarrow \sup_{t' \in S \land t'.B = t.A \land fc_S}(t') \geq \alpha \). \quad \text{(H2)}

If Expression (H1) does not not hold, then computation of \( \mu_{\text{exists}}(SQ(t)) \), as in Expression (H2), is not necessary. Equivalence (H2) holds because:

\[
\begin{align*}
(1) \quad & \mu_{\text{exists}}(SQ(t)) \geq \alpha \Leftrightarrow \sup_{t' \in S \land t'.B = t.A \land fc_S}(t') \geq \alpha \mu_{fc_S}(t') \geq \alpha,
\end{align*}
\]
the tuples \( t' \) that belong to the set
\[
\{ t' | t' \in S \land t'.B = t.A \land DNC(f_{c_S}, \geq, \alpha)(t') \}
\]
but not to
\[
\{ t' | t' \in S \land t'.B = t.A \land \mu_{fc_S}(t') \geq \alpha \}
\]
are such that \( \mu_{fc_S}(t') < \alpha \).

Thus, the computation of \( \mu_{\exists t_S}(SQ(t)) \) only implies the calculation of \( \mu_{fc_S}(t') \) for each tuple which belongs to the result of the query derived from the inner block.

The corresponding evaluation algorithm is as follows:

**Algorithm 5.2.**

```plaintext
declare c1 cursor for select * from R where DNC(f_{c_R}, \geq, \alpha);
declare c2 cursor for select * from S where S.B = a and DNC(f_{c_S}, \geq, \alpha);
open c1;
fetch c1 into t; a := t.A;
while code(c1) \neq active_set_empty do
begin
    \( \mu := f_{c_R}(t) \);
    if \( \mu \geq \alpha \) then /* useful in case of a weak derivation of \( f_{c_R} \) */
    begin
        open c2;
        fetch c2 into t'; /* t' = first tuple of the derived inner relation */
        \( \mu' := 0 \);
        while code(c2) \neq active_set_empty do
        begin
            \( \mu' := \max(\mu', f_{c_S}(t')) \); /* computation of the supremum */
            fetch c2 into t';
        end;
        close c2;
        \( \mu := \min(\mu, \mu') \); /* computation of the final degree */
        if \( \mu \geq \alpha \) then result := result + \{\langle \mu/t \rangle\};
    end;
fetch c1 into t; a := t.A;
end;
close c1
```
**Remark 5.1.** Another possible evaluation strategy is based on the equivalence (see Bose and Pivert, 1995b) between $Q_{ex}$ and the query:

$Q': \text{select } R.\ast \text{ from } R, S \text{ where } f_{cR} \text{ and } f_{cS} \text{ and } R.A = S.B.$

Then, one can use the evaluation method defined for single-block PSJ queries (see Section 5.6.2.2).

**“Not exists” nesting.** The generic form of the considered query is:

\[
\text{select } \ast \text{ from } R \text{ where } f_{cR} \text{ and not exists} \left( \text{select } \ast \text{ from } S \text{ where } S.B = R.A \text{ and } f_{cS} \right). \quad (Q_{nex})
\]

The final degree of a tuple $t$ from relation $R$, i.e., its degree of membership to the resulting relation $\text{res}$, is now expressed by:

\[
\mu_{\text{res}}(t) = \min(\mu_{f_{cR}}(t), 1 - \mu_{\exists}(SQ(t))).
\]

Pruning condition $H1$ is still valid, since $\mu_{\text{res}}(t) \geq \alpha \Rightarrow DNC(f_{cR}, \geq, \alpha)(t)$. However, we have:

\[
\mu_{\text{res}}(t) \geq \alpha \Rightarrow \mu_{\exists}(SQ(t)) \leq 1 - \alpha \Rightarrow \sup_{t' \in S \land t'.B = t.A} \mu_{f_{cS}}(s) \leq 1 - \alpha,
\]

thus we can use the following pruning condition $H3$:

\[
E(t) = \{t'| t' \in S \land t'.B = t.A \land DSC(f_{cS}, >, 1 - \alpha)(t')\} \neq \emptyset \Rightarrow \mu_{\text{res}}(t) < \alpha. \quad (H3)
\]

This condition can be used to detect a failure. However, if $E(t)$ is empty, one has to evaluate the condition: $\mu_{\exists}(SQ(r)) \leq 1 - \alpha$, which implies calculation of $\mu_{f_{cS}}(s)$ for every tuple $t'$ of $S$ such that $t'.B = t.A$. Nevertheless, the scan of $S$ can be stopped as soon as a tuple $t'$ is encountered such that $\mu_{f_{cS}}(t') > 1 - \alpha$ (we are then in a failure case). This test, which is useful in case of a weak derivation of $f_{cS}$, corresponds to pruning condition $H4$:

\[
\exists t' \in S \text{ s.t. } t'.B = t.A \land \mu_{f_{cS}}(t') > 1 - \alpha \Rightarrow \mu_{\text{res}}(t) < \alpha. \quad (H4)
\]
Finally, the evaluation algorithm is:

Algorithm 5.3.

```
declare c1 cursor for select * from R where DNC(fc_R, ≥, α);
declare c2 cursor for select * from S
    where S.B = :a and DSC(fc_S, >, 1 - α);
declare c3 cursor for select * from S where S.B = :a;
open c1;
fetch c1 into t; a := t.A;
while code(c1) ≠ active_set_empty do
begin
  µ := µfc_R(t);
  if µ ≥ α then /* useful in case of a weak derivation of fc_R */
  begin
    open c2;
    fetch c2 into t';
    if code(c2) = active_set_empty then /* condition H3 */
  begin
    open c3;
    fetch c3 into t';
    µ' := 0;
    failure := false;
    while code(c3) ≠ active_set_empty and not failure do
    begin
      if µfc_S(t') ≤ 1 - α then /* useful in case of a weak derivation of fc_S */
      begin
        µ' := max(µ', µfc_S(t')); /* computation of the supremum */
        fetch c3 into t';
        end
      else failure := true; /* condition H4 */
    end;
    close c3;
  end;
if not failure then
begin
  µ := min(µ, 1 - µ'); /* computation of the final degree */
  result := result + {⟨µ/t⟩};
end
```
Simple Fuzzy Queries

end;
close c2;
end;
fetch c1 into t; a := t.A;
end;
close c1;

It is important to note that pruning condition H3 is useful only if the evaluation of the associated derived query (cf. the declaration of cursor c2) does not entail the scanning of relation S exhaustively. Otherwise, this condition must be removed from the algorithm in order to avoid scanning relation S twice (then, only two cursors are necessary, i.e., c1 and c3, and pruning condition H4 becomes more important).

Remark 5.2. In the presence of a not exists nesting, an evaluation method based on rewriting cannot be applied, since there is no equivalence between such a nested query and a single-block query.

5.6.2.4 Efficiency of the derivation-based processing strategy

Single-block PSJ queries. Depending on the connectors used in the initial SQLf query, the Boolean query obtained may deliver the exact α-cut or a superset (in which case extra elements must be removed, which implies an additional test on the degree). The proportion of additional tuples (which may be seen as an index of efficiency) is examined in Bosc and Pivert (1995a). It appears that this ratio is in general much less than 20% when two fuzzy predicates are aggregated by a mean operator, for instance, but it can reach 70% when four predicates are combined.

It is important to note that the derivation principle can also be used when the user does not specify a threshold α in the query. The derivation method is then applied with the threshold condition α > 0, which allows the tuples which do not satisfy the query at all to be discarded. Of course, this is also true for nested queries.

Regarding this evaluation process, three types of single-block queries can be distinguished, according to the kind of derivation rules that can be used:

• exact derivation: queries involving predicates acting on single attributes and usual connectives (conjunction and disjunction interpreted respectively as min and max);
• weak derivation: queries involving mean operators; queries involving norms, co-norms other than \( \min \) and \( \max \); queries involving at least one predicate that cannot be exactly derived;
• queries for which no derivation rules apply: in this case, the derivation method is useless. Alternative methods (specific algorithms) must be applied (see Chapters 6, 7, and 8).

**Nested queries.** In case of a weak derivation of the outer block, one can prevent useless evaluations of the inner block, thanks to the calibration performed at the outer level.

If the nesting operator is not negated, one can apply the derivation principle to the inner block, and the calibration operation can also be distributed onto the result of the inner derived query. This is especially interesting in the case of several levels of nesting.

If the nesting operator is negated, derivation of the inner block is not sufficient to allow for the computation of the final degrees. The derived query can only be used as a pruning condition to detect a failure case. In this case, even if the derivation of the inner query is exact, the evaluation method will not, in general, be as efficient as in the previous case.

### 5.7 Conclusion

In this chapter, we have presented the basic features of SQLf, an extension of SQL which makes it possible to express fuzzy queries to a classical relational database. The principle underlying this extension is to authorize fuzzy operators wherever it makes sense, while preserving, as much as possible, the syntactic and semantic characteristics of SQL (in particular when it comes to query equivalences). Even though SQLf is not the only language that has been proposed for database fuzzy querying (other proposals such as FSQl (Galindo et al., 1998) and FQUERY for Access (Kacprzyk and Zadrozny, 1995) can be found in the literature), it constitutes the most complete fuzzy extension of SQL. It is also appropriate to mention that in SQLf, the satisfaction degrees are somewhat “transparent” to the user (who can only act on them through a global threshold), contrary to the case with FSQl, for instance, where the user may express conditions on the local degrees tied to atomic selection predicates, which, in our opinion, somewhat contradicts the philosophy of fuzzy sets and can make the semantics of a query difficult to grasp.
It is important to emphasize that the added value, in terms of expressivity, comes at a cost in terms of query elicitation: a flexible querying system implementing a language such as SQLf must include an appropriate interface through which the user can define fuzzy predicates and operators in a convenient way. We have seen that different philosophies can be considered for defining such interfaces.

An important fact, however, is that the price to pay in terms of query processing complexity remains limited (at least for the simple queries considered in this chapter), thanks in particular to a processing strategy (derivation) which takes advantage of the connections between properties tied to fuzzy conditions and Boolean ones, and makes it possible to exploit the capabilities of regular DBMSs for processing fuzzy queries.
Chapter 6

Fuzzy Queries Involving Quantified Statements or Aggregates

6.1 Introduction

In this chapter, we consider fuzzy queries of a more complex type than those studied in the previous chapter. More precisely, we focus on SQLf queries involving fuzzy quantified statements or aggregates applying to a fuzzy relation.

There are several ways to make use of fuzzy quantifiers in SQLf queries. First, they may appear inside a where clause in order to combine predicates in a less demanding way than can be expressed by a conjunction. An example is “find the employees who satisfy most of the properties among {to be young, to be well-paid, to have a high commission, to have a highly qualified job}”. Second, fuzzy quantifiers generalize the quantifiers all and any, which are the only ones authorized in SQL, so they can be used as extended nesting operators. Third, they may appear in a having clause in order to qualify groups based on their compliance with a gradual cardinality-based condition. An example is “find the departments where at least half of the employees earn more than $3,000 a month”. SQLf queries involving fuzzy quantified statements of the form “QX are A” (see Section 3.7) are studied in Section 6.2.

Note that a fuzzy quantified statement is just a special case of a condition matching an aggregate (namely count) to a fuzzy predicate. However, since much research work has been devoted to this particular case, and also because of its peculiarities, we believe fuzzy quantified statements deserve a specific section. As for other types of aggregates applying to a fuzzy
relation, the corresponding generic query is of the form:

\[
\text{select } X \text{ from } r \text{ where } \text{cond} \\
\text{group by } Y \text{ having } \{\text{agg}_1 \theta \text{agg}_2 | \text{agg is } \psi\},
\]

where \(X \subseteq Y\), \(\text{agg}_1\), \(\text{agg}_2\), and \(\text{agg}\) denote aggregates, and both \(\text{cond}\) and \(\psi\) are fuzzy conditions. In other words, it is possible in SQLf to select a group by means of a condition which compares two aggregates or matches an aggregate to a fuzzy predicate, even in the situation where the underlying relation is fuzzy (because of condition \(\text{cond}\)). An \(\alpha\)-cut-based interpretation of such queries is provided in Section 6.3.

### 6.2 Quantified Statements

#### 6.2.1 Introduction

In this section, we are interested in introducing fuzzy quantifiers (Zadeh, 1983; see also Section 3.7) into SQLf queries. Such quantifiers can be used to express an intermediary attitude between conjunction (“all of the criteria must be satisfied”) and disjunction (“at least one criterion must be satisfied”). They model linguistic expressions such as, “most of”, “about a third”, and are notably used to construct fuzzy predicates (with quantifications). For instance, the fuzzy quantifier “most of” can be used to define the quantified statement “most of the elements of set \(X\) satisfy criterion \(A\)”. There are two types of quantified statements: those of the form “\(Q X \text{ are } A\)” (“most of the elements of \(X\) satisfy \(A\)”) and those of the form “\(Q X B \text{ are } A\)” (“most of the elements of set \(X\) which satisfy \(B\) also satisfy \(A\)”). In the following, we only consider the first type, since the second one raises serious interpretation issues (in particular, in its way of dealing with situations where no — or almost no — element from \(X\) is \(B\)).

In this section, we show that the interpretation of fuzzy quantified statements can be based on Sugeno’s or Choquet’s integral. First, we present fuzzy quantifications and their connection with fuzzy integral theory. Then, we recall different propositions, respectively by Zadeh (1983), Prade (1990) and Yager (1988) for interpreting fuzzy quantifications. Since the use of the OWA operator in Yager (1988) is limited to the computation of increasing quantifications, we show how to extend its use to decreasing quantifications. We also show that the approaches advocated in Prade (1990) and in Yager (1988), when restricted to monotonous quantifiers, lead to the computation of fuzzy integrals.
6.2.2 Fuzzy Queries Involving Quantified Statements or Aggregates

6.2.2.1 Introduction

Fuzzy quantifiers can be increasing (proportional) (Yager, 1988), which means that if the criteria are all entirely satisfied, then the statement “QX are A” is entirely true, whereas if the criteria are all entirely unsatisfied, then the statement “QX are A” is entirely false. Moreover, the transition between those two extremes is continuous and monotonous. Therefore, when Q is increasing (e.g., “most”, “at least a half”), function \( \mu_Q \) is increasing. Similarly, decreasing quantifiers (e.g., “at most two”, “at most a half”) are defined by decreasing functions. Zadeh (1983) distinguishes between absolute and relative quantifiers. Absolute quantifiers refer to a number while relative ones refer to a proportion. An example of an increasing relative quantifier is “at least half”, while “at most three” illustrates the notion of a decreasing absolute quantifier.

An absolute quantifier is represented by a function \( \mu_Q \) from an integer range to [0, 1] whereas a relative quantifier is a mapping \( \mu_Q \) from [0, 1] to [0, 1]. In both cases, the value \( \mu_Q(j) \) is defined as the truth value of the statement “QX are A” when exactly \( j \) elements from X fully satisfy A (where it is assumed that A is fully unsatisfied for the other elements). Here, we deal with absolute quantifiers, but the results are still valid for relative quantifiers, it is just necessary to turn every \( \mu_Q(x) \) into \( \mu_Q(x/n) \) in every formula, \( n \) being the cardinality of the crisp set X. The characteristics of monotonous fuzzy quantifiers are given in Table 6.1. Figure 6.1 gives examples of monotonous decreasing and increasing quantifiers. Calculating the truth degree of the statement “QX are A” raises the problem of determining the cardinality of the set of elements from X which satisfy A. If A is a Boolean predicate, this cardinality is a precise integer (\( k \)), and then the truth value of “QX are A” is \( \mu_Q(k) \). If A is a fuzzy predicate, this cardinality cannot be established precisely, and computing the quantification corresponds to establishing the value of function \( \mu_Q \) for an imprecise argument.

<table>
<thead>
<tr>
<th>Increasing quantifier</th>
<th>Decreasing quantifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mu_Q(0) = 0 )</td>
<td>( \mu_Q(0) = 1 )</td>
</tr>
<tr>
<td>( \exists k \text{ such that } \mu_Q(k) = 1 )</td>
<td>( \exists k \text{ such that } \mu_Q(k) = 0 )</td>
</tr>
<tr>
<td>( \forall a, b, \text{ if } a &gt; b \text{ then } \mu_Q(a) \geq \mu_Q(b) )</td>
<td>( \forall a, b, \text{ if } a &gt; b \text{ then } \mu_Q(a) \leq \mu_Q(b) )</td>
</tr>
</tbody>
</table>
Example 6.1. Let $Q$ be the fuzzy quantifier “at least three” represented in Figure 6.1. Let $X$ be the crisp set made up of the five employees from Table 6.2, and let $A$ be the fuzzy predicate “well-paid” described by the fuzzy set from Figure 6.2. The satisfaction degrees related to the fuzzy predicate “well-paid” are given by the fuzzy set $\{0.4/e_1, 1/e_2, 0.8/e_3, 0.1/e_4, 0.9/e_5\}$. If we want to evaluate the quantification “at least three employees are well-paid”, we face the problem of determining the value $\mu_Q(N)$, $N$ being the number of well-paid employees. We can observe that the three largest satisfaction degrees are 1, 0.9 and 0.8, the others being 0.4 and 0.1. In this example, we can then consider that value $N$ is “close to 3”;
consequently, the evaluation of the statement "at least three employees are well-paid" should be close to 1 (since $\mu_Q(3)$ equals 1). This example illustrates the need to compute the value of $\mu_Q$ when the number of satisfied criteria is not perfectly known, i.e., when the set gathering the elements which satisfy $A$ is fuzzy.

6.2.2.2 Fuzzy integrals and increasing quantifiers

In this subsection, we show that, given a measure defined on a regular set, both the Sugeno and Choquet fuzzy integrals (see Section 3.5) are tools enabling calculation of the value of the measure when its argument is a fuzzy set. Thus, if an increasing quantifier $Q$ is defined as a measure, one has available a method for defining the truth value of the statement "$QX$ are $A$".

Let $\mathcal{P}(X)$ denote the power set of $X = \{x_1, x_2, \ldots, x_n\}$. Recall that a fuzzy measure (Sugeno, 1974) on a finite, discrete regular set $X$ is a function $g$ from $\mathcal{P}(X)$ to the unit interval which satisfies the following three axioms:

1. $g(X) = 1$,
2. $g(\emptyset) = 0$,
3. $\forall A, B$ such that $A \subseteq B$, $g(A) \leq g(B)$ (monotonicity).

Both the Sugeno (1974) and Choquet (see Murofushi and Sugeno, 1989) fuzzy integrals, respectively denoted by $S_g$ and $C_g$ in the following, define a characteristic of a function $h$ (from a universe $X$ to the unit interval) according to the measure $g$. Both are averaging operators over the values taken by $h$. Using the notation $h(x_i) = h_i$, and ordering the $h_i$'s in an increasing way ($h_1 \leq h_2 \leq \cdots \leq h_n$), we have:

$$C_g(h) = \sum_{i=1}^{n} h_i \times (g(F_i) - g(F_{i+1})),$$

$$S_g(h) = \sup_i \min(h_i, g(F_i)),$$

where $F_i = \{x_i, x_{i+1}, \ldots, x_n\}$ and $F_{n+1} = \emptyset$. Moreover, $S_g(h)$ is a median (Kandel and Byatt, 1978), i.e., it delivers the $n$th element of an ordered list of $2n - 1$ elements. Consequently:

$$S_g(h) = \sup_i \min(h_i, g(F_i)) = \text{med}(h_1, h_2, \ldots, h_n, g(F_2), g(F_3), \ldots, g(F_n)).$$

In the following, we assume that the function $h$ is a membership function of a fuzzy set denoted by $H$. Therefore a fuzzy integral can be viewed as a function defined on fuzzy sets.
Example 6.2. Consider a set $X$ of five elements and a measure $g$ such that $g(F) = |F|/5$, where $|F|$ denotes the usual cardinality of a crisp set $F$. If $h_1 = 0.1, h_2 = 0.4, h_3 = 0.8, h_4 = 0.9, h_5 = 1$, we obtain:

$$
C_g(h) = 0.1 \times (1 - 4/5) + 0.4 \times (4/5 - 3/5) + 0.8 \times (3/5 - 2/5) + 0.9 \times (2/5 - 1/5) + 1 \times (1/5 - 0) = 0.64,
$$

$$
S_g(h) = \text{sup}(\text{min}(0.1, 1), \text{min}(0.4, 4/5), \text{min}(0.8, 3/5), \text{min}(0.9, 2/5), \text{min}(1, 1/5)) = 0.6,
$$

$$
\text{med}(0.1, 1/5, 0.4, 2/5, 3/5, 0.8, 4/5, 0.9, 1) = 3/5 = 0.6.
$$

If $h$ and $h'$ are the membership functions of two fuzzy sets $H$ and $H'$, both fuzzy integrals $C_g(h)$ and $S_g(h)$ satisfy the following properties (see Sugeno, 1974) (replacing $I$ by $C$ or $S$):

- if $g$ is a fuzzy measure and $H$ is a regular set, then $I_g(h) = g(H)$,
- if $H$ and $H'$ are such that $H' \subseteq H(\forall x, h'(x) \leq h(x))$ then, for any fuzzy measure $g$: $I_g(h') \leq I_g(h)$.

These properties show that $g$ and $C_g$ (resp. $g$ and $S_g$) are identical when $H$ is a regular set and that $C_g$ (resp. $S_g$) complies with the property of monotonicity. We can therefore consider $C_g(h)$ and $S_g(h)$ as two possible values of $g$ for the fuzzy set $H$ (Grabisch et al., 1992).

The definition of an increasing quantifier $Q$ as a fuzzy measure $g$ requires the definition of an underlying universe $F$:

$$
g: \mathcal{P}(F) \rightarrow [0, 1] \text{ such that } g(E) = \mu_Q(|E|). \quad (6.1)
$$

Since we need $g(F) = 1$, this definition requires:

$$
\mu_Q(|F|) = 1. \quad (6.2)
$$

When one wants to evaluate “$QX$ are $A$”, the satisfaction of Expression 6.2 entails taking $F$ either as $X$ ($Q$ is increasing relative, or increasing absolute, and $\mu_Q(|X|) = 1$), or as $X$ plus dummy elements which do not satisfy $A$ at all (when $\mu_Q(|X|) < 1$). The dummy elements are added so that $\mu_Q(|F|)$ reaches 1, but of course they do not influence the evaluation of “$QX$ are $A$” (for any dummy element $z$, $\mu_A(z) = 0$). By doing so, the meaning of “$QX$ are $A$” is not altered.

Let $E_A$ denote the set of elements from $X$ satisfying predicate $A$. The set $E_A$ is a (possibly fuzzy) subset of $X$ and then of $F$. When $E_A$ is a crisp set — the satisfaction degrees equal 0 or 1 — the truth value of
Fuzzy Queries Involving Quantified Statements or Aggregates

“Q X are A” is \( g(E_A) = \mu_Q(|E_A|) \). In the general case, \( E_A \) is a fuzzy set and the evaluation of “Q X are A” requires the computation of \( g \) for the fuzzy set \( E_A \), given by \( C_g(E_A) \) or \( S_g(E_A) \). Denoting by \( k \) the cardinality of universe \( F \), these values are given by the following formulas, assuming that we order the membership degrees related to \( E_A \) increasingly:

\[
C_g(E_A) = \sum_{i=1}^{k} \mu_{E_A}(x_i) \times (g(F_i) - g(F_{i+1}))
\]

\[
S_g(E_A) = \sup_i \min(\mu_{E_A}(x_i), g(F_i)),
\]

where \( F_i = \{x_i, x_{i+1}, \ldots, x_k\} \), \( F_{k+1} = \emptyset \), and \( \mu_{E_A}(x_1) \leq \mu_{E_A}(x_2) \leq \cdots \leq \mu_{E_A}(x_k) \).

It is important to note that, if \( \mu_Q(n) \neq 1 \), \((k-n)\) dummy elements have been added to \( X \) in order to build \( F \).

**Example 6.3.** The quantifier “at least three”, illustrated in Figure 6.1, may be represented using the measure \( g \) such that:

\[
g(E) = \begin{cases} 
0 & \text{if } |E| \leq 1, \\
0.5 & \text{if } |E| = 2, \\
1 & \text{if } |E| \geq 3.
\end{cases}
\]

Consider the quantified statement “at least three employees are well-paid” and the data from Example 6.1. The cardinality of \( X \) equals 5 and \( \mu_Q(|X|) = 1 \), therefore \( F \) is taken as \( X \). After an appropriate ordering of the elements \( x_1 = e_4, x_2 = e_1, x_3 = e_3, x_4 = e_5, x_5 = e_2 \), the satisfaction degrees are the following:

\[
\mu_{E_A}(x_1) = 0.1, \mu_{E_A}(x_2) = 0.4, \mu_{E_A}(x_3) = 0.8, \mu_{E_A}(x_4) = 0.9, \mu_{E_A}(x_5) = 1.
\]

The truth value of the quantified statement “at least three employees are well-paid” is given either by \( C_g(E_A) \) or \( S_g(E_A) \) (with \( F_i = \{x_1, \ldots, x_5\} \) and \( F_6 = \emptyset \)):

\[
C_g(E_A) = \sum_{i=1}^{5} \mu_{E_A}(x_i) \times (g(F_i) - g(F_{i+1}))
\]

\[
= \sum_{i=1}^{5} \mu_{E_A}(x_i) \times (\mu_Q(5 - i + 1) - \mu_Q(5 - i))
\]
These results confirm our intuition, which says that they should be “close to 1”. We have considered the case where no dummy elements have been added, because the cardinality of \( X \) was 5 and \( \mu_Q(5) = 1 \). We now examine the case of the set \( X = \{e_1, e_2\} \), where \( e_1 \) and \( e_2 \) are taken from the previous example. The fuzzy set of well-paid employees now becomes \( \{0.4/e_1, 1/e_2\} \).

In this case, the cardinality of \( X \) is not sufficient to represent \( \mu_Q \) as a fuzzy measure \( g \), since \( g(|X|) = \mu_Q(2) \) is different from 1. This means that we need to add one dummy element \( e_3 \) that satisfies \( \mu_A(e_3) = 0 \). We can now take \( F = \{e_1, e_2, e_3\} \) as the referential. After an appropriate ordering of the elements (\( x_1 = e_3, x_2 = e_1, x_3 = e_2 \)), the satisfaction degrees are the following:

\[
\mu_{E_A}(x_1) = 0, \quad \mu_{E_A}(x_2) = 0.4, \quad \mu_{E_A}(x_3) = 1.
\]

The evaluation of the quantification is then given either by \( C_g(E_A) \) or \( S_g(E_A) \) (with \( F_i = \{x_i, \ldots, x_3\} \) and \( F_4 = \emptyset \)):

\[
C_g(E_A) = \sum_{i=1}^5 \mu_{E_A}(x_i) \times (g(F_i) - g(F_{i+1}))
\]

\[
= \sum_{i=1}^3 \mu_{E_A}(x_i) \times (\mu_Q(3 - i + 1) - \mu_Q(3 - i))
\]

\[
= (0 \times 0.5) + (0.4 \times 0.5) + (1 \times 0)
\]

\[
= 0.2,
\]

\[
S_g(E_A) = \sup_i \min(\mu_{E_A}(x_i), g(F_i))
\]

\[
= \sup_i \min(\mu_{E_A}(x_i), \mu_Q(|F_i|))
\]

\[
= \sup_i \min(\mu_{E_A}(x_i), \mu_Q(4 - i)) = 0.4.
\]

### 6.2.3 Interpretation of statements of the type “\( Q X \) are \( A \)”

We now present different proposals from the literature for interpreting quantified statements of the type “\( Q X \) are \( A \)”, and we show that these
approaches have strong connections with the Sugeno and Choquet fuzzy integrals when the quantifier is monotonous.

6.2.3.1 Interpretation according to Zadeh

Let $X$ be the usual (crisp) set $\{x_1, x_2, \ldots, x_n\}$. Zadeh (1983) defines the cardinality of the set of elements of $X$ which satisfy $A$, denoted by $\Sigma_{\text{count}}(A)$, as:

$$\Sigma_{\text{count}}(A) = \sum_{i=1}^{n} \mu_A(x_i).$$

The quantification “$Q X$ are $A$” is then evaluated by $\mu_Q(\Sigma_{\text{count}}(A))$. It can be seen, however, that a large number of elements with a small degree $\mu_A(x)$ has the same effect as a small number of elements with a high degree $\mu_A(x)$, due to the definition of $\Sigma_{\text{count}}$.

**Example 6.4.** Consider the following sets:

$$X_1 = \{0.9/x_1, 0.9/x_2, 0.9/x_3, 0.8/x_4, 0.8/x_5, 0.7/x_6, 0.6/x_7\},$$

$$X_2 = \{1/x_1, 1/x_2, 0.3/x_3, 0.2/x_4, 0.1/x_5, 0/x_6, 0/x_7\},$$

$$X_3 = \{1/x_1, 1/x_2, 1/x_3, 1/x_4, 1/x_5, 0.8/x_6, 0.3/x_7\}$$

and the quantifier “at least five” represented in Figure 6.3. The $\Sigma_{\text{count}}(A)$ values associated with the sets $X_1$, $X_2$, and $X_3$ are 5.6, 2.6, and 6.1 respectively. The associated values of the quantification are 1, 0.2, and 1 respectively.

![Fig. 6.3 The fuzzy quantifier “at least five”](image-url)
6.2.3.2 Interpretation according to Prade

In Prade (1990), it is assumed that the elements from set $X$ may be imprecise, but here we only consider the case where the elements of set $X$ are precisely known. The method proposed in Prade (1990) involves two steps:

1. computation of the fuzzy cardinality $\pi$ of the set $E_A$ (gathering the elements which satisfy $A$),
2. evaluation of a degree of matching between this fuzzy cardinality $\pi$ and the quantifier $Q$ viewed as a fuzzy predicate.

This matching degree corresponds to the truth value of the quantification. Since $E_A$ is a fuzzy set, its cardinality can be represented as a possibility distribution $\pi$ (Dubois and Prade, 1985a). This possibility distribution is built from the number $k$ of entirely satisfied criteria (possibly equal to 0) according to the following definition:

$$\forall i < k, \pi(i) = 0,$$
$$\pi(k) = 1,$$
$$\forall j > k, \pi(j) = \text{the } j\text{th largest value } \mu_A(x).$$

Note that the restriction of $\pi$ to values greater or equal to $k$ is decreasing.

The second step of the process performs a fuzzy pattern matching (Cayrol et al., 1980; Dubois et al., 1988) between the possibility distribution $\pi$ obtained in the first step and the quantifier viewed as a predicate. Consequently, it computes a pair of degrees: $\Pi$, the extent to which $\pi$ possibly satisfies $Q$, and $N$, the extent to which $\pi$ necessarily (certainly) satisfies $Q$, according to the formulas:

$$\Pi = \max_{u \geq k} \min(\mu_Q(u), \pi(u)),$$
$$N = \min_{u \geq k} \max(\mu_Q(u), 1 - \pi(u)).$$

We can conclude that the truth value of the statement “$QX$ are $A$” is situated in the interval $[N, \Pi]$.

**Example 6.5.** Consider the sets $X_1$, $X_2$, $X_3$, and the quantifier “at least five” from Example 6.4. Taking set $X_3$ first, we obtain the following possibility distribution for its imprecise cardinality:

$$\pi = \frac{1}{5} + \frac{0.8}{6} + \frac{0.3}{7}.$$
The formulas above yield \( \Pi = 1 \) and \( N = 1 \), and we can therefore say that \( X_3 \) satisfies “at least five elements are \( A \)” with a degree of 1. Similar computations for \( X_1 \) and \( X_2 \) yield the intervals \([0, 0.8] \) and \([0, 0.3]\) respectively.

We now show that when \( Q \) is increasing (resp. decreasing), the quantified statement “\( QX \) are \( A \)” only requires the computation of \( \pi \) (resp. \( N \)), which is a Sugeno integral.

If \( Q \) is increasing, we can write that \( N \) equals \( \mu_Q(k) \), \( k \) being the number of entirely satisfied criteria, since when \( i > j \geq k \), \( \max(\mu_Q(i), 1 - \pi(i)) \geq \max(\mu_Q(j), 1 - \pi(j)) \). Indeed, since \( Q \) is increasing,

\[
i > j \geq k \Rightarrow \mu_Q(i) \geq \mu_Q(j),
\]

and since \( \pi \) is decreasing,

\[
i > j \geq k \Rightarrow 1 - \pi(i) \geq 1 - \pi(j).
\]

Thus, \( N = \min_{k \leq i} \max(\mu_Q(i), 1 - \pi(i)) = \mu_Q(k) \). Hence, \( N \) is the truth value of the quantification if we consider only the entirely satisfied criteria. \( Q \) being an increasing quantifier, when the other criteria are taken into account, this truth value can only increase. The useful information is supplied by \( \Pi \), whose value takes into account all of the \( \mu_A(x) \)'s. It has been shown in Prade (1990) and Yager (1984, 1991b) that, when \( Q \) is increasing, \( \Pi \) is given by the Sugeno fuzzy integral \( S_g(E_A) \) (using the notation introduced in Section 6.2.2.2).

Similarly, when \( Q \) is decreasing, \( \Pi \) equals \( \mu_Q(k) \), where \( k \) is the number of entirely satisfied criteria. When all the criteria are taken into account, the only changes affect \( N \), which is still associated with a Sugeno integral, as follows:

\[
N = \min_{k \leq u} \max(\mu_Q(u), 1 - \pi(u)) \Rightarrow 1 - \max_{k \leq u} \min(1 - \mu_Q(u), \pi(u)).
\]

Defining \( g' \) as: \( \forall E, g'(E) = 1 - \mu_Q(|E|) \), we obtain:

\[
N = 1 - S_{g'}(E_A).
\]

6.2.3.3 Yager’s competitive type aggregation

The interpretation by decomposition described in Yager (1984) is limited to increasing quantifiers. The proposition “\( QX \) are \( A \)” is true if an ordinary
subset $C$ of $X$ satisfies the conditions $c_1$ and $c_2$ given as

- $c_1$: there are $Q$ elements in $C$,
- $c_2$: each element $x$ of $C$ satisfies $A$.

We emphasize that finding one such subset is sufficient, since the quantifier is increasing. The truth value of the proposition: “$QX$ are $A$” is then defined as:

$$
\mu(X) = \sup_{C \subseteq X} \min(\mu_{c_1}(C), \mu_{c_2}(C))
$$

with

$$
\mu_{c_1}(C) = \mu_Q(|C|)
$$

and

$$
\mu_{c_2}(C) = \inf_{x \in C} \mu_A(x).
$$

It has been shown in Yager (1984) that:

$$
\mu(X) = \sup_{1 \leq i \leq n} \min(\mu_Q(i), \mu_A(x_i)), \quad (6.3)
$$

where the elements of $X$ are ordered in such a way that $\mu_A(x_1) \geq \cdots \geq \mu_A(x_n)$. As can be seen, Expression (6.3) corresponds to a Sugeno integral (see Sections 3.5.3, 3.7, and 6.2.2.2).

### 6.2.3.4 Interpretation based on the OWA operator

Yager (1988) considers the case of an increasing monotonous quantifier, and proposes an ordered weighted averaging operator (OWA) to evaluate quantifications of the type “$QX$ are $A$”. We first present this approach, then we show how it can be extended in order to evaluate decreasing quantifications (Bosc et al., 1995). Finally, we show that this method boils down to using a Choquet fuzzy integral.

#### Case of an increasing quantifier

The OWA operator is defined in Yager (1988) as:

$$
OWA(x_1, \ldots, x_n; w_1, \ldots, w_n) = \sum_{i=1}^{n} w_i \times x_{k_i}
$$

where $x_{k_i}$ is the $i$th largest value among the $x_k$’s and $\sum_{i=1}^{n} w_i = 1$.

Let $n$ be the crisp cardinality of $X$. The truth value of the statement “$QX$ are $A$” is computed by an OWA of the $n$ values $\mu_A(x_i)$. The weights $w_i$
involved in the calculation of the OWA are given by \( w_i = \mu_Q(i) - \mu_Q(i - 1) \). A value \( w_i \) corresponds to the increase in satisfaction when one moves from a situation where exactly \( (i - 1) \) elements totally satisfy \( A \) to a situation where exactly \( i \) elements totally satisfy \( A \) (assuming that the other elements do not satisfy \( A \) at all). The aggregated value which is calculated is:

\[
\text{OWA}(\mu_A(x_1), \ldots, \mu_A(x_n); w_1, \ldots, w_n) = \sum_{i=1}^{n} w_i \times c_i
\]

where \( c_i \) is the \( i \)th largest value among the \( \mu_A(x_k) \)'s.

**Example 6.6.** Consider the sets \( X_1, X_2, \) and \( X_3 \), and the quantifier “at least five” from Example 6.4. We have:

\[
w_1 = 0, \quad w_2 = 0, \quad w_3 = 1/3, \quad w_4 = 1/3, \quad w_5 = 1/3, \quad w_6 = 0, \quad w_7 = 0.
\]

We evaluate the statement “at least five elements of \( X_1 \) are \( A \)” and we obtain the degree 0.83 (= 0.9 \times 1/3 + 0.8 \times 1/3 + 0.8 \times 1/3). In the same way, we obtain the degrees 0.2 for \( X_2 \) and 1 for \( X_3 \).

**Extension to a decreasing quantifier.** We now describe a method for computing the truth value of a statement “\( Q X \) are \( A \)” involving a decreasing quantifier \( Q \) by means of an OWA. We first give an introductory example which presents the interpretation of this type of quantification. Then, we point out the need to distinguish between statements involving a relative quantifier and those involving an absolute one. Finally, we show that computing the truth value of a statement “\( Q X \) are \( A \)” when \( Q \) is decreasing boils down to computing a value \((1 - \alpha)\), \( \alpha \) being the truth value (computed by an OWA) of the quantified statement “\( Q' X \) are \( \neg A \)”, where \( Q' \) is another representation — defined on the proportion of unsatisfied criteria — of quantifier \( Q \).

Consider the set \( X = \{x_1, x_2, x_3, x_4\} \). Consider also the relative decreasing monotonous quantifier \( Q_1 \) and the quantifier \( Q_1' \) defined by \( \mu_{Q_1}(x) = \mu_{Q_1}(1 - x) \) (see Figure 6.4). The objective is to compute the truth value of the relative decreasing quantification “\( Q_1 X \) are \( A \)”. If, among the four elements of \( X \), there is one which totally satisfies \( A \) while the three remaining elements do not satisfy \( A \) at all, then the quantification is satisfied with a degree equal to \( \mu_{Q_1}(1/4) \). Now, by definition, \( \mu_{Q_1}(1/4) = \mu_{Q_1'}(3/4) \), and \( 3/4 \) is the proportion of criteria which are not satisfied. By extension, evaluating “\( Q_1 X \) are \( A \)” boils down to evaluating the relative increasing quantification “\( Q_1' X \) are \( \neg A \)”. This can be done, for
instance, using the OWA-based approach described above, replacing the satisfaction degrees $\mu_A(x_i)$ by the refutation degrees $(1 - \mu_A(x_i))$. Note, however, that interpretations other than the OWA-based one could be used too.

The generalization of this method to the case of an absolute quantification leads to the consideration of the particular situation illustrated by the following example. Consider the absolute decreasing quantifier “no more than three” from Figure 6.5. Consider the quantifier $Q'_2$ — defined on the number of unsatisfied criteria — such that $\forall x$, $\mu_{Q'_2}(x) = \mu_{Q_2}(n - x)$, $n$ being the cardinality of $X$. If $n$ is greater than or equal to 5, then $Q'_2$ is increasing monotonous and it is possible to evaluate “$Q_2 X$ are $A$” by an OWA aggregating the refutation degrees. If $n$ is smaller than 5, then $Q'_2$ is not monotonous increasing — since $\mu_{Q'_2}(0) \neq 0$ — and it is therefore a special case. Note that this problem does not arise for relative quantifiers. This is why, in the following, we distinguish between relative decreasing and absolute decreasing monotonous quantifiers.
Decreasing relative quantifier. Let $Q_{rd}$ be a decreasing relative quantifier and $n$ the cardinality of $X$. One has:

$$\mu_{Q_{rd}}\left(\frac{i}{n}\right) = \mu_{Q_{rd}}\left(1 - \frac{i}{n}\right).$$

Consequently, one can express the truth value of the decreasing relative quantification as the OWA aggregation of the refutation degrees $(1 - \mu_A(x_i))$ with $w_i = \mu_{Q_{rd}}(1 - \frac{i}{n}) - \mu_{Q_{rd}}(1 - \frac{i+1}{n})$. Again, note that interpretations other than the OWA-based one (for instance, that relying on the competitive type aggregation presented in Section 6.2.3.3) could be used.

**Example 6.7.** Consider the decreasing relative quantifier “no more than a quarter” represented in Figure 6.6. Consider the following sets:

$$X_1 = \{0.6/x_1, 0.5/x_2, 0.4/x_3, 0.4/x_4, 0.3/x_5\},$$
$$X_2 = \{1/x_1, 0.1/x_2, 0.1/x_3, 0/x_4, 0/x_5\},$$
$$X_3 = \{0.9/x_1, 0/x_2, 0/x_3, 0/x_4, 0/x_5\}.$$

It is assumed that $X_1$ (resp. $X_2, X_3$) is a fuzzy subset which describes the elements of a crisp set $X = \{x_1, x_2, x_3, x_4, x_5\}$ which are $A_1$ (resp. $A_2, A_3$). Since the cardinality $n$ of $X$ equals 5, we get:

$$w_1 = \mu_{Q_{rd}}(4/5) - \mu_{Q_{rd}}(1) = 0, \quad w_2 = \mu_{Q_{rd}}(3/5) - \mu_{Q_{rd}}(4/5) = 0,$$
$$w_3 = \mu_{Q_{rd}}(2/5) - \mu_{Q_{rd}}(3/5) = 0.4, \quad w_4 = \mu_{Q_{rd}}(1/5) - \mu_{Q_{rd}}(2/5) = 0.6,$$
$$w_5 = \mu_{Q_{rd}}(0) - \mu_{Q_{rd}}(1/5) = 0.$$

The OWA-based aggregation of the refutation degrees yields the truth value $0.54 (= 0.4 \times 0.6 + 0.6 \times 0.5)$ (resp. $0.9, 1$) for the quantified statement “$Q_1 X$ are $A_1$” (resp. $A_2, A_3$).

![Fig. 6.6 The decreasing relative quantifier “no more than a quarter”](image)
Decreasing absolute quantifier. We consider a decreasing absolute quantifier $Q_{da}$ and we denote by $m$ the smallest integer such that $\mu_{Q_{da}}(m) = 0$. First, we consider the case where the cardinality of $X$ is smaller than $m$. A solution is to add $(m - n)$ dummy elements — which do not satisfy $A$ at all — to $X$. By doing so, one can evaluate the absolute decreasing quantification as the OWA-based aggregation of the $m$ degrees attached to the elements of $X$ (completed).

Now we denote by $n$ the cardinality ($\geq m$) of set $X$ (possibly supplemented by dummy elements). In a context where $\forall i, \mu_{A(x_i)} \in \{0, 1\}$, $\mu_{Q_{da}}(n - i)$ is the degree of satisfaction attached to the quantification when there are $(n - i)$ totally satisfied criteria or, reciprocally, when there are $i$ totally unsatisfied criteria. For the same reason, $\mu_{Q_{da}}(n - i + 1)$ is the degree of satisfaction in the case where there are $(i - 1)$ totally unsatisfied criteria. Thus, we can conclude that:

$$w_i = \mu_{Q_{da}}(n - i) - \mu_{Q_{da}}(n - i + 1)$$

is the increase in the satisfaction degree when one moves from a situation where $(i - 1)$ criteria are totally unsatisfied to a situation where $i$ criteria are totally unsatisfied. In the general context where $\forall i, \mu_{A(x_i)} \in [0, 1]$, the truth value of a decreasing absolute quantification is therefore the OWA-based aggregation of the refutation degrees $(1 - \mu_{A(x_i)})$, using the weights $w_i$ defined in the above expression. As in the case of decreasing relative quantifiers, approaches other than that based on OWA aggregation could be used to interpret the reformulated quantified statement.

**Example 6.8.** Consider the quantifier “no more than three” represented in Figure 6.5 (such that $m = 5$) and the following fuzzy sets:

- $X_1 = \{0.7/x_1, 0.8/x_2\}$
- $X_2 = \{1/x_1, 1/x_2, 1/x_3, 0.1/x_4\}$
- $X_3 = \{1/x_1, 1/x_2, 1/x_3, 0.7/x_4, 0.1/x_5\}$

We compute:

- $w_1 = \mu_{Q_2}(4) - \mu_{Q_2}(5) = 0.5$
- $w_2 = \mu_{Q_2}(3) - \mu_{Q_2}(4) = 0.5$
- $w_3 = \mu_{Q_2}(2) - \mu_{Q_2}(3) = 0$
- $w_4 = \mu_{Q_2}(1) - \mu_{Q_2}(2) = 0$
- $w_5 = \mu_{Q_2}(0) - \mu_{Q_2}(1) = 0$

The OWA-based aggregation of the refutation degrees yields the truth value 1 ($= 0.5 \times 1 + 0.5 \times 1$) (resp. 0.95 and 0.6) for $X_1$ (resp. $X_2$, $X_3$).
Fuzzy Queries Involving Quantified Statements or Aggregates

Relationship between an OWA-based quantifier and its negation. Consider an increasing (absolute or relative) quantifier $Q$ and the decreasing quantifier $\overline{Q}$ defined as: $\forall x, \mu_{\overline{Q}}(x) = 1 - \mu_{Q}(x)$. It is possible to establish that, if a set $X$ satisfies the quantification "$QX$ are $A$" with a degree $\alpha$, then it satisfies the quantification "$\overline{Q}X$ are $A$" with a degree $\beta = 1 - \alpha$.

Let $n$ be the cardinality of $X$ (considering the possible addition of dummy elements). The proof is based on the following points:

- the expression of both OWA-based computations use the same values $\mu_A(x_i)$ (recall that the dummy elements, if any, are such that $\mu_A(x_i) = 0$),
- we denote by $w_i$ the weights used for evaluating "$QX$ are $A$" and by $\overline{w}_i$ those used for evaluating "$\overline{Q}X$ are $A$". Then, $w_i = \overline{w}_{n-i+1}$. Thus:

$$\alpha = \sum_{i=1}^{n} w_i \times \mu_A(x_i) = \sum_{i=1}^{n} \overline{w}_{n-i+1} \times \mu_A(x_i)$$

$$= 1 - \sum_{i=1}^{n} \overline{w}_{n-i+1} + \sum_{i=1}^{n} \overline{w}_{n-i+1} \times \mu_A(x_i)$$

$$= 1 - \sum_{i=1}^{n} \overline{w}_{n-i+1} \times (1 - \mu_A(x_i)) = 1 - \beta.$$

Link with the Choquet fuzzy integral. In Section 6.2.2 we showed that, defining an increasing quantifier $Q$ by a fuzzy measure $g$, the Choquet integral $C_g(E_A)$ is a method to compute the truth value of the quantified statement "$QX$ are $A$". It is shown in Bosc et al. (1995) that if $Q$ is increasing or decreasing, then the OWA-based evaluation of the quantification corresponds to computation of the Choquet fuzzy integral $C_g(E_A)$.

6.2.3.5 Interpretation of quantifiers of the type “about $S$”

We now describe an interpretation of quantified statements involving a quantifier of the type “about $S$”, based on the decomposition of the quantifier into two monotonous ones.

Principle of the interpretation. We define a fuzzy quantifier $Q$ of the type “about $S$” in the following way:

- $\mu_Q(S) = 1$ and $\exists t$ such that $\forall x \in [S - t, S + t]$, $\mu_Q(x) = 1$ ($t$ may equal 0),
- $\forall \alpha$, $\mu_Q(S + \alpha) = \mu_Q(S - \alpha)$ (symmetry).
The property of symmetry is not essential but corresponds to the intuitive definition of “about $S$”. Such a quantifier can be absolute (if $S$ is a number) or relative (if $S$ is a proportion). The semantics of the quantifier suggests that, in the proximity of $S$ completely satisfied criteria — the others being completely unsatisfied — the quantified statement “$QX$ are $A$” is totally satisfied, and that the “damage” caused by an additional number $\alpha$ of satisfied criteria is the same as that caused by a lack of $\alpha$ satisfied criteria.

The quantified statement “about $S$ $X$ are $A$” is equivalent to the statement “at least $S$ $X$ are $A$ and at most $S$ $X$ are $A$”. Therefore, computing the truth value attached to “about $S$ $X$ are $A$” boils down to computing the conjunction of the truth values attached respectively to “at least $S$ $X$ are $A$” and to “at most $S$ $X$ are $A$”. The first step of the evaluation is thus to break down the quantifier “about $S$” into two quantifiers “at least $S$” and “at most $S$”. The second step is to calculate the respective truth values $\alpha$ and $\beta$ attached to the quantified statements “at least $S$ $X$ are $A$” and “at most $S$ $X$ are $A$”. Finally, the truth value associated with “about $S$ $X$ are $A$” is equal to $\min(\alpha, \beta)$.

Example 6.9. Consider the following sets

$$X_1 = \{0.9/x_1, 0.9/x_2, 0.7/x_3, 0.1/x_4\},$$
$$X_2 = \{1/x_1, 1/x_2, 1/x_3, 0.3/x_4\},$$
$$X_3 = \{1/x_1, 1/x_2, 0.7/x_3, 0/x_4\},$$
$$X_4 = \{0.8/x_1, 0.5/x_2, 0/x_3, 0/x_4\}$$

and compute the truth value attached to the statement “about three $X$s are $A$” using an OWA-based interpretation. Let $Q$ be the quantifier “about three” represented in Figure 6.7. From $Q$, we can derive the quantifiers $Q_1$.
and $Q_2$ represented in Figure 6.8, and evaluate the quantifications “at least three $X$s are $A$” and “at most three $X$s are $A$” for each set.

**Computation of “at least three $X$s are $A$”**. We have: $\mu_{Q_1}(0) = \mu_{Q_1}(1) = 0$, $\mu_{Q_1}(2) = 0.9$, and $\forall x \geq 3$, $\mu_{Q_1}(x) = 1$. Hence, $w_1 = 0$, $w_2 = 0.9$, $w_3 = 0.1$, and $\forall i > 3$, $w_i = 0$. The OWA-based evaluation of “at least three $X$s are $A$” yields degree $0.88 (= 0 \times 0.9 + 0.9 \times 0.9 + 0.1 \times 0.7)$ for $X_1$, 1 for $X_2$, 0.97 for $X_3$, and 0.45 for $X_4$.

**Computation of “at most three $X$s are $A$”**. We have: $m = 5$, $\forall x \leq 3$, $\mu_{Q_2}(x) = 1$, $\mu_{Q_2}(4) = 0.9$, and $\forall x \geq 5$, $\mu_{Q_2}(x) = 0$. Hence, $w_1 = 0.9$, $w_2 = 0.1$, and $\forall i \geq 3$, $w_i = 0$. The OWA-based evaluation of “at most three $X$s are $A$” yields degree 0.99 for $X_1$, 0.97 for $X_2$, 1 for $X_3$, and 1 for $X_4$.

**Computation of “about three $X$s are $A$”**. Finally, $X_1$ satisfies “about three $X$s are $A$” to the degree 0.88 ($= \min(0.88, 0.99)$), $X_2$ to the degree 0.97, $X_3$ to the degree 0.97, and $X_4$ to the degree 0.45.

### 6.2.4 Integration into SQLf

First, we consider that the fuzzy quantifier aggregates different satisfaction degrees related to a same tuple (horizontal quantification; see Kacprzyk and
Ziolkowski, 1986). An example of such a query is: “retrieve the employees satisfying most of the following predicates: well-paid, young, ..., important position”. Second, we consider that the fuzzy quantifier applies to a set \( X \) of tuples (vertical quantification). An example of such a query is: “retrieve the departments where most of the employees are well-paid”. Set \( X \) can be obtained by means of either a partitioning (\( \text{group by} \) clause in SQLf), or a block nesting.

### 6.2.4.1 Horizontal quantification

The format of an SQLf query involving horizontal quantification is:

\[
\text{select } * \text{ from } r \text{ where } Q(f_c_1, f_c_2, \ldots, f_c_n). \tag{E1}
\]

In this expression, \( r \) is a regular relation, \( Q \) is a fuzzy quantifier and \( \{f_c_1, \ldots, f_c_n\} \) is a set of \( n \) fuzzy conditions. This query retrieves the tuples satisfying \( Q \) conditions among \( \{f_c_1, f_c_2, \ldots, f_c_n\} \). Therefore, the satisfaction degree obtained by a tuple \( t \) of \( r \) (with respect to query E1) is the truth value of the statement “\( QX \) are \( A \)” where \( X = \{f_c_1, f_c_2, \ldots, f_c_n\} \) and \( A = \text{“to be satisfied by } t\} \) (\( \forall f_c_i, \mu_A(f_c_i) = \mu_{f_c_i}(t) \)).

Considering a relation \( \text{Emp} (#e, #dep, salary, position, age, address) \) describing employees of a company, the query: “retrieve the employees fulfilling at least two conditions among well-paid, high commission, young” is expressed by the following SQLf query (format E1):

\[
\text{select } #e \text{ from Emp where } \text{at least 2 (salary is well-paid, commission is high, age is young}).
\]

### 6.2.4.2 Nested queries (extension of any and all)

An interesting feature of SQL is that it allows the nesting of query blocks to an arbitrary depth. In particular, two select blocks can be connected by means of the existential (\( \exists \)) or the universal quantifier (\( \forall \)) using the keywords any or all (see Chapter 2). In this case, the internal block (subquery) is seen as a set, and a comparison between the value of the current tuple and the values of that set is performed. For instance, the query “find the employees whose salary is not smaller than that of any engineer” can be expressed:

\[
\text{select } #e \text{ from Emp where salary } \geq \text{ all (select salary from Emp where position = “engineer”).}
\]
The objective in SQLf is to authorize fuzzy quantifiers in addition to all and any as well as any fuzzy comparator (much greater, more or less equal, etc.). The general form of such SQLf queries is:

$$\text{select } * \text{ from } r \\
\text{where } att_1 \theta Q (\text{select distinct } att_2 \text{ from } s \text{ where } bc), \quad (E2)$$

where $\theta$ is a fuzzy comparator, $att_1$ (resp. $att_2$) is an attribute of relation $r$ (resp. $s$), $Q$ is a fuzzy quantifier, and $bc$ is a Boolean condition. To help define its semantics, we reformulate this query in the form of a fuzzy statement “$Q X$ are $A$”. Note that the quantifier $Q$ applies to the set made of values $x$ for $att_2$ appearing in tuples (belonging to $s$) which satisfy $bc$. The use of the keyword distinct in the inner block ensures that it delivers a set and not a multiset. The degree to which a tuple $t'$ of $r$ with value $u$ on $att_1$ belongs to the result is the truth value of the statement “$Q X$ are $A$”, where $X = \{t.att_2 \mid t \in s \land bc(t)\}$, and $A$ is the predicate $\theta(u, x)$ (i.e., $\mu_A(x) = \mu_\theta(u, x)$). The query “find the employees whose salary is greater than that of almost all the employees over 30 years old” can be written as:

$$\text{select } * \text{ from } \text{Emp where } \text{salary} > \text{almost all} \\
(\text{select distinct } \text{salary} \text{ from } \text{Emp where } \text{age} > 30).$$

6.2.4.3 Quantified statement in the having clause

We now consider the partitioning of relations and the use of vertical quantification. Just as in SQL, the partitioning is expressed in SQLf using the keyword group by. The SQLf expression which can be associated with this kind of query is as follows:

$$\text{select } att \text{ from } r \text{ where } bc_1 \\
\text{group by } att \text{ having } Q(bc_2) \text{ are } A, \quad (E3)$$

where $bc_1$ and $bc_2$ denote Boolean conditions. This query splits relation $r$ into subsets (partitions) according to the value of attribute $att$. The degree to which a value $x$ of attribute $att$ is an answer to query $E3$ is the truth value of the statement “$Q X$ are $A$”, where $X$ is the group made of those tuples satisfying conditions $bc_1$ and $bc_2$, and sharing the value $x$ for attribute $att$. 
An equivalent expression for query E3 is:

\[
\text{select } \text{att from } r \text{ where } bc_1 \text{ and } bc_2 \\
\text{group by att having } Q \text{ are } A. \quad (E4)
\]

The query “retrieve the departments where at least two employees over 30 years old are well-paid” may be expressed by the following SQLf queries:

\[
\text{select } \# \text{dep from } \text{Emp group by } \# \text{dep} \\
\text{having at least } 2 (\text{age } > 30) \text{ are } (\text{salary is well-paid}), \quad \text{(format E3)}
\]

\[
\text{select } \# \text{dep from } \text{Emp where } \text{age } > 30 \\
\text{group by } \# \text{dep having at least } 2 \text{ are } (\text{salary is well-paid}). \quad \text{(format E4)}
\]

### 6.2.5 Evaluation of SQLf queries involving quantified statements

In this section, we describe algorithms which make it possible to determine whether, for a given set \( X \), the satisfaction degree attached to the quantified statement “\( Q X \text{ are } A \)” is above a user-specified threshold \( \alpha \) (a case of qualitative calibration). Suppose that the number of elements in set \( X \) is known, but that \( X \) is assumed not to be sorted on the satisfaction degrees (its elements are accessed in a random order). The objective is to find conditions for stopping the calculus as soon as possible. Generally speaking, the loop that governs data access can stop in two situations:

- when it is certain that \( X \) cannot reach the desired level \( \alpha \),
- when it is certain that \( X \) will reach the desired level, but one does not need to know the precise value of the satisfaction degree attached to \( X \).

This reasoning is very similar to that in the design of “trial and error” or “branch and bound” algorithms, where some pruning conditions are used for trimming the search tree, i.e., the number of candidates to be examined.

#### 6.2.5.1 Zadeh’s interpretation

According to Zadeh (1983), the evaluation of the quantification “\( Q X \text{ are } A \)” for a set \( X = \{x_1, x_2, \ldots, x_n\} \) is

\[
\mu_Q \left( \sum_{i=1}^{n} \mu_A(x_i) \right).
\]
Using a threshold $\alpha$, the problem is thus to determine the sets $X$ which satisfy $\mu_Q(\sum_{x_i \in X} \mu_A(x_i)) \geq \alpha$. Depending on the type of quantifier $Q$ involved (increasing, decreasing, or increasing–decreasing), three cases may be considered (see Figures 6.9 and 6.10), which can be captured by the following common expression:

$$a \leq \mu_Q \left( \sum_{x_i \in X} \mu_A(x_i) \right) \leq b.$$ 

(6.4)

Note that values $a$ and $b$ can reach respectively 0 (for a decreasing quantifier) and 1 (increasing relative quantifier). Thus, we have to characterize the sets $X = \{x_1, x_2, \ldots, x_n\}$ such that:

$$a \leq \mu_Q \left( \sum_{x_i \in X} \mu_A(x_i) \right) \leq b.$$ 

(6.4)
Na"ıve algorithm. The straightforward algorithm works by performing an exhaustive scan of set \( X = \{x_1, x_2, \ldots, x_n\} \). We write \( S_k = \sum_{i=1}^{k} \mu_A(x_i) \). We aim to obtain and then test the value \( S_n \), which corresponds to the satisfaction degree attached to set \( X \) with respect to the quantification “\( Q X \) are \( A \)”. This “na"ıve” method is encoded by Algorithm 6.1.

Algorithm 6.1.

\[
S_0 := 0; \\
\text{for } i := 1 \text{ to } n \text{ do } S_i := S_{i-1} + \mu_A(x_i) \text{ enddo; }
\]

if \( S_n \in [a, b] \) then \( X \) is an answer else \( X \) is not qualified;

Improved algorithm. The idea is to modify the na"ıve algorithm using pruning conditions which enable us to decide whether or not the set will comply with the requirement in Expression (6.4). Letting \( k \in [0, n-1] \), we have \( S_n - S_k = \sum_{i=k+1}^{n} \mu_A(x_i) \). Now, \( \forall i, \mu_A(x_i) \in [0, 1] \), which implies that \( 0 \leq S_n - S_k \leq (n-k) \). We can thus write:

\[
\forall k \in [0, 1], S_k \leq S_n \leq S_k + (n-k).
\]

Therefore, two pruning conditions can be used to stop the algorithm at iteration \( k \):

- If \( S_k \in [a, b] \) and if \( S_k + (n-k) \in [a, b] \), then it is certain that \( S_n \in [a, b] \).
  In this case, \( X \) is certainly an answer.
- If \( S_k > b \) or \( S_k + (n-k) < a \), then it is certain that \( S_n \notin [a, b] \).
  In this case, we know for sure that \( X \) is not an answer.

This improved method is encoded by Algorithm 6.2.

Algorithm 6.2.

\[
S_0 := 0; \\
\text{for } i := 1 \text{ to } n \text{ do } S_k := S_{k-1} + \mu_A(x_k); \\
\quad \text{if } S_k > b \text{ or } S_k + (n-k) < a \text{ then exit(failure) } \\
\quad \text{else if } S_k \in [a, b] \text{ and } S_k + (n-k) \in [a, b] \text{ then exit(success); }
\]

enddo;

exit success: \( X \) is an answer (it satisfies “\( Q X \) are \( A \)” to a degree \( \geq \alpha \));
exit failure: \( X \) is not an answer;

The algorithm for evaluating a quantification involving a relative fuzzy quantifier would be the same, except that each expression \( \mu_A(x_k) \) and \( (n-k) \) would have to be divided by \( n \), i.e., the cardinality of set \( X \).
6.2.5.2 Interpretation based on Sugeno’s integral

Recall that the value of “Q X are A” is given by (see Section 6.2.2.2):
\[ S_g(E_A) = \text{med}(\mu_{E_A}(x_1), \ldots, \mu_{E_A}(x_k), \mu_Q(k-1), \mu_Q(k-2), \ldots, \mu_Q(1)) \]

In this formula, \( k \) denotes the cardinality of \( X' \) where \( X' \) is either \( X \) itself, or \( X \) completed with dummy elements. One thus has:

- if \( n < k \) then \( \forall j \in [1, k-n], \mu_{E_A}(x_j) = 0 \), \( n \) being the cardinality of \( X \),
- the other values \( \mu_{E_A}(x_j) \) are the satisfaction degrees of the elements of \( X \) with respect to \( A \).

Moreover, the values \( \mu_Q(1), \ldots, \mu_Q(k-1) \) are constant for a given quantifier \( Q \) and are computed once and for all at the beginning of the algorithm.

The algorithm consists mainly of a loop accessing the elements of \( X \). Consider the situation at the end of the \( i \)th iteration. At that point, \( (n-i) \) values are still missing in the computation of \( S_g(E_A) \). If we assume these missing values to be zeros and \( S_g(E_A) \) is already greater than \( \alpha \), due to the monotonicity of Sugeno’s integral we can conclude that whatever the missing values, the actual \( S_g(E_A) \) will be over \( \alpha \). Similarly, if we assume these values to be ones and \( S_g(E_A) \) is less than \( \alpha \), then we can conclude that whatever the degrees of the missing elements, \( S_g(E_A) \) will be less than \( \alpha \). These two remarks form the basis of the pruning conditions used in the algorithm.

The implementation requires two sorted arrays \( MI \) and \( MS \) of \( (2k-1) \) elements. At the \( i \)th iteration, \( MI \) contains the previously treated elements \( \mu_A(x) \), the values \( \mu_Q(1), \ldots, \mu_Q(k-1) \), and \( (n-i) \) zeros for the remaining elements (not yet accessed). \( MS \) contains the previously treated elements \( \mu_A(x) \), the values \( \mu_Q(1), \ldots, \mu_Q(k-1) \), and \( (n-i) \) values set to 1 (the elements which are still missing are assumed to have the maximal degree).

When moving from step \( i \) to step \( i+1 \), a new satisfaction degree \( \mu_A(x_i) \) for the current element \( x_i \) will replace a zero (resp. 1) in \( MI \) (resp. \( MS \)).

Let \( MI[k]_i \) (resp. \( MS[k]_i \)) be the value of the median element \( MI[k] \) (resp. \( MS[k] \)) at iteration \( i \). Then:

\[ \forall i, MI[k]_i \leq S_g(E_A) \leq MS[k]_i, \quad (6.5) \]
\[ MI[k]_n = MS[k]_n = S_g(E_A). \quad (6.6) \]

Expression (6.5) relies on the fact that \( S_g \) is monotonic, which implies that:

\[ S_g(y_1, y_2, \ldots, y_n) \geq S_g(y_{i_1}, y_{i_2}, \ldots, y_{i_j}, 0, \ldots, 0) \]
and

\[ S_g(y_1, y_2, \ldots, y_n) \leq S_g(y_{i_1}, y_{i_2}, \ldots, y_{i_j}, 1, \ldots, 1), \]

where \( \{i_1, i_2, \ldots, i_j\} \) is any subset of \( \{1, 2, \ldots, n\} \). Expression (6.6) is based on the fact that, at the end of the iterations \((i = n)\), the contents of MI and MS are equal and \( S_g(E_A) \) is their median element. The evaluation procedure is encoded by Algorithm 6.3.

**Algorithm 6.3.**

```
for i := 1 to k do begin MI[i] := 0; MS[i] := 0; end;
for i := k + 1 to 2k - 1 do MI[i] := µQ(i - k);
for i := 2k - n to 2k - 1 do MS[i] := 1;
for i := k - n + 1 to 2k - n - 1 do MS[i] := µQ(i - k + n);
for i := 1 to n do
  if MI[k] ≥ α then exit(success) else if MS[k] < α then exit(failure);
  ordered insertion of µA(x_i) into MI /* µA(x_i) replaces a zero */
  ordered insertion of µA(x_i) into MS /* µA(x_i) replaces a 1 */
endo;
exit success: X is an answer (it satisfies “QX are A” to a degree ≥ α);
exit failure: X is not an answer;
```

We now consider the use of a decreasing quantifier \( Q \). Recall that in this case, the value of “QX are A” is given by \( 1 - S_g'(E_A) \) where \( \forall E, g'(E) = 1 - µ_Q(|E|) \). We define the increasing quantifier \( Q' \) as:

\[ \forall i, \mu_{Q'}(i) = 1 - \mu_Q(i). \]

Measure \( g' \) is computed from \( µ_{Q'} \) using Expressions 6.1 and 6.2. To compute \( S_g'(E_A) \) in an efficient way, we introduce pruning conditions which take advantage of the fact that only the Xs sought for are such that the degree of “QX are A” is over a threshold \( α \). Since one has:

\[ \forall i, MI_i[k] \leq S_g'(E_A) \leq MS_i[k], \]

the property

\[ \forall i, 1 - MS_i[k] \leq 1 - S_g'(E_A) \leq 1 - MI_i[k] \]
Fuzzy Preference Queries To Relational Databases

Fuzzy Queries Involving Quantified Statements or Aggregates

holds and may be used to prune the search space. Algorithm 6.3 is modified the following way:

\[
\text{if } MI[k] \geq \alpha \text{ then exit(success)} \\
\text{if } MS[k] < \alpha \text{ then exit(failure)}
\]

is replaced by:

\[
\text{if } (1 - MS[k]) \geq \alpha \text{ then exit(success)} \\
\text{if } (1 - MI[k]) < \alpha \text{ then exit(failure))}.
\]

6.2.5.3 Interpretation based on Choquet’s integral

We now deal with the OWA-based evaluation of a quantified statement “\(QX\) are \(A\)” in the case where \(Q\) is a monotonic quantifier. First, we present a straightforward algorithm, then we show that the properties of the OWA operator can be used to design a more efficient algorithm.

Na"ive algorithm. The “na"ive” algorithm performs the evaluation of the OWA-based aggregation by scanning the entire set \(X\) considered. It is assumed that \(W\) is the vector that describes the weights of the OWA operator (recall that the \(W[i]\)’s are derived from \(\mu_Q\)). We denote by \(n\) the cardinality of \(X\) and the result of the aggregation by \(GV\). The procedure is encoded by Algorithm 6.4.

**Algorithm 6.4.**

for each \(x_i \in X\) do \(V[i] := \mu_A(x_i)\) enddo;
\(V' := \text{rank-order (} V)\);
compute vector \(W\);
\(GV := 0\);
for \(i := 1\) to \(n\) do \(GV := GV + V'[i] \times W[i]\) enddo;
if \(GV \geq \alpha\) then set \(X\) is qualified and its satisfaction degree equals \(GV\).

The straightforward algorithm in the case where \(Q\) is decreasing is the same except that the final degree is \(1 - GV\), where \(GV\) is the OWA-based evaluation of “\(QX\) are \(A\)” with \(\overline{Q}\) defined as \(\forall x, \mu_{\overline{Q}}(x) = 1 - \mu_Q(x)\).

**Some properties of the OWA operator.** The OWA operator has the following interesting properties (for any given vector of weights \(W\)):

\[
OWA(x_1, \ldots, x_n) \leq OWA(x_1, \ldots, x_i, 1, \ldots, 1), \quad (6.7)
\]

\[
OWA(x_1, \ldots, x_n) \geq OWA(x_1, \ldots, x_i, 0, \ldots, 0), \quad (6.8)
\]
These properties are related to the monotonicity of mean operators, and to the fact that $\forall i, x_i \in [0, 1]$. From these basic properties, one can derive conditions bearing on the $x_i$'s which are necessary for the satisfaction of the global condition:

$$OWA(x_1, \ldots, x_n) \geq \alpha.$$ 

From Expressions (6.7) and (6.8), we have:

$$OWA(x_1, \ldots, x_i, 1, \ldots, 1) < \alpha \Rightarrow OWA(x_1, \ldots, x_n) < \alpha, \quad (6.11)$$

$$OWA(x_1, \ldots, x_i, 0, \ldots, 0) \geq \alpha \Rightarrow OWA(x_1, \ldots, x_n) \geq \alpha. \quad (6.12)$$

From Expressions (6.9) and (6.10), we have:

$$1 - OWA(x_1, \ldots, x_i, 1, \ldots, 1) \geq \alpha \Rightarrow 1 - OWA(x_1, \ldots, x_n) \geq \alpha, \quad (6.13)$$

$$1 - OWA(x_1, \ldots, x_i, 0, \ldots, 0) < \alpha \Rightarrow 1 - OWA(x_1, \ldots, x_n) < \alpha. \quad (6.14)$$

These conditions can be used for computing partial evaluations of an OWA-based aggregation, as we see below.

**Improved algorithm.** We first consider the case where quantifier $Q$ is increasing. The algorithm relies on the properties in Expressions (6.11) and (6.12). We assume that we have already accessed $k$ elements of set $X$ (elements $x_1$ to $x_k$), and the values $v_i = \mu_A(x_i)$ for $i \in [1, k]$ are known. If we also assume that the $(n - k)$ missing values equal 1 and the result of the OWA-based evaluation remains smaller than $\alpha$, according to Expression (6.11), we can be certain that set $X$ will never reach the desired level $\alpha$. We compute the aggregation:

$$OWA(v_1, \ldots, v_k, 1, \ldots, 1) = \sum_{i=1}^{n-k} w_i + \sum_{i=1}^{k} w_{n-k+i} \times v_i,$$
Fuzzy Queries Involving Quantified Statements or Aggregates

where \( v_{j_i} \) is the \( i \)th largest value among the \( v_j \)'s. This computation only requires that the values \( V[1] \) to \( V[k] \) are sorted. In addition, the expression does not have to be recalculated from scratch when one moves from step \( k \) to step \( k + 1 \), since the first part can be computed incrementally. Once again, it is possible to express a condition that is likely to stop the outer loop before it is complete:

\[
\text{insert } \mu_A(x_k) \text{ into } V[i : k]; \\
\text{compute } A = \sum_{i=1}^{n-k} W[i] + \sum_{i=1}^{k} W[n - k + i] \times V[i]; \\
\text{if } A < \alpha \text{ then exit(failure)};
\]

When \( k = n \) (last element of set \( X \)), the value of \( A \) corresponds to the overall degree \( GV \) associated with set \( X \).

If one is only interested in knowing whether \( X \) qualifies or not (but does not care about its precise satisfaction degree), one can also use Expression (6.12). If \( k \) elements (\( x_1 \) to \( x_k \)) have already been accessed and the result of the OWA-based aggregation already exceeds \( \alpha \), assuming that the \( (n - k) \) missing values equal zero, then set \( X \) qualifies for sure. One has to compute the aggregation:

\[
\text{OWA}(v_1, \ldots, v_k, 0, \ldots, 0) = \sum_{i=1}^{k} w_i \times v_{k_i}.
\]

Here again, one only has to sort the values \( V[1] \) to \( V[k] \). The related pruning condition can be inserted into the algorithm in the following way:

\[
\text{insert } \mu_A(x_k) \text{ into } V[i : k]; \\
\text{compute } B = \sum_{i=1}^{k} W[i] \times V[i]; \\
\text{if } B \geq \alpha \text{ then exit(success)};
\]

Again, when \( k = n \), the value of \( B \) equals that of \( A \) and corresponds to the overall satisfaction degree \( GV \) associated with set \( X \).

The final evaluation procedure is given in Algorithm 6.5.

**Algorithm 6.5.**

compute vector \( W \);  
for each \( x_k \) in \( X \) do  
\[
\text{insert } \mu_A(x_k) \text{ into } V[1 : k]; \\
A = \sum_{i=1}^{n-k} W[i] + \sum_{i=1}^{k} W[n - k + i] \times V[i]; \\
\text{if } A < \alpha \text{ then exit(failure)};
\]


Fuzzy Preference Queries to Relational Databases

\[ B = \sum_{i=1}^{k} W[i] \times V[i]; \]

if \( B \geq \alpha \) then exit(success);
enddo;

\textbf{exit} success: \( X \) is an answer;
\textbf{exit} failure: \( X \) is not an answer;

When quantifier \( Q \) is decreasing, the properties in Expressions (6.13) and (6.14) can be used. The definitions of \( A \) and \( B \) become, respectively:

\[
A = 1 - \left( \sum_{i=1}^{n-k} W[i] + \sum_{i=1}^{k} W[n-k+i] \times V[i] \right)
\]

and

\[
B = 1 - \sum_{i=1}^{k} W[i] \times V[i].
\]

The previous algorithm must then be modified the following way:

if \( A < \alpha \) then exit(failure)
(\text{resp. if } B \geq \alpha \text{ then exit(success)})

is replaced by:

if \( A \geq \alpha \) then exit(success)
(\text{resp. if } B < \alpha \text{ then exit(failure)}).

6.3 Aggregates

6.3.1 Introduction

In SQLf, as in SQL, it is possible to consider aggregates (such as cardinality, sum, maximum or average) which are functions applying to a set of items. Aggregates can be integrated into SQLf queries as in the following example calling on the aggregate average (\texttt{avg}):\[
\text{select } \# \text{dep from Emp group by } \# \text{dep}
\text{having avg(salary) is high.}
\]

The underlying relation \texttt{Emp(#e, #dep, age, salary)} describes employees, and this query aims at retrieving departments where the average salary is high. A department is all the more satisfactory as it satisfies the fuzzy
condition: “the average salary of its employees is high”. In this example, the aggregate \( \text{avg} \) applies to a crisp set of salaries and, for each department, the average salary is computed and matched against the fuzzy condition \( \text{high} \). However, when the items to be aggregated are attached a satisfaction degree, the interpretation becomes more tricky since the aggregate applies to a fuzzy referential. This is the case of the query aimed at the retrieval of departments where “the average salary of young employees is high” which could be expressed in SQLf as:

\[
\text{select } \#\text{dep from Emp where age is young } \\
\text{group by } \#\text{dep having } \text{avg(salary)} \text{ is high.}
\]

This query is difficult to interpret, since the aggregate — here, the average — applies to the fuzzy set of salaries of young employees. The objective of this section is to describe a sound technique for interpreting (and computing) queries where an aggregate applies to a fuzzy set.

A condition in which the value of an aggregate is matched to a fuzzy predicate is written \( \text{agg}(A) \text{ is } C \) for short (e.g., \( \text{avg}(A) \text{ is high} \)), where \( A \) is the fuzzy set of salaries of young employees).

In the context of SQLf, a unique degree of satisfaction for \( \text{agg}(A) \text{ is } C \) (denoted by \( \text{truth}[\text{agg}(A) \text{ is } C] \)) is needed in order to be consistent with other predicates. In this section, we describe an interpretation of conditions of type \( \text{agg}(A) \text{ is } C \) in terms of a single degree (Bosc and Liétard, 2008) and we define SQLf structures devoted to such conditions. We do not deal with the case where the aggregate is \( \text{count} \), since this corresponds to interpreting a fuzzy quantified proposition, which has already been discussed in the preceding section.

### 6.3.2 The case of monotonic predicates and aggregates

This section presents a generalization of the Sugeno-integral-based approach that was presented in Section 6.2 in a context of fuzzy quantified propositions. The objective is to extend this approach so as to evaluate conditions of the form \( \text{agg}(A) \text{ is } C \) when the aggregate \( \text{agg} \) and the fuzzy condition \( C \) are monotonic in the same way (i.e., both increasing or decreasing). The idea is to consider that the more numerous the interpretations of set \( A \) (i.e., its \( \alpha \)-cuts) with a high satisfaction for the condition “the aggregate is \( C \)”, the higher the truth value for \( \text{agg}(A) \text{ is } C \). Examples of monotonic aggregates are \( \text{max}, \text{min}, \text{and count} \), but also \( \text{sum} \) if the values to be added are either all positive or all negative.
The evaluation relies on the set-oriented function \( m \) defined as follows (over a crisp set \( E \) onto which the aggregate applies):

\[
m(E) = \mu_C(\text{agg}(E)).
\]

Obviously, when \( A \) is a crisp set:

\[
\text{truth}[\text{agg}(A) \text{ is } C] = C(\text{agg}(A)) = m(A),
\]

and this function satisfies the following property of monotonicity (since the aggregate and \( C \) are assumed to be monotonic in the same way):

\[
E \subseteq F \Rightarrow m(E) \leq m(F).
\]

For two crisp sets \( E \) and \( F \), this property of monotonicity can be rewritten:

\[
(E \subseteq F) \Rightarrow \text{truth}[\text{agg}(E) \text{ is } C] \leq \text{truth}[\text{agg}(F) \text{ is } C].
\] (6.15)

This expression is the starting point for the evaluation of \( \text{truth}[\text{agg}(A) \text{ is } C] \) when \( A \) is fuzzy. In this case, the right-hand part of the implication can be read as “the higher \( \text{truth}[\text{agg}(E) \text{ is } C] \) is for a crisp set \( E \) included in \( A \), the higher \( \text{truth}[\text{agg}(A) \text{ is } C] \) should be”. This leads to the following definition (where \( \text{truth}[\text{agg}(E) \text{ is } C] \) is given by \( m(E) \)):

\[
\text{truth}[\text{agg}(A) \text{ is } C] = \max_{E \in S} \min_{x \in E} \mu_A(x), m(E))
\]

\[
= \max_{E \in S} \min_{x \in E} \mu_A(x), \mu_C(\text{agg}(E))),
\]

where \( S \) is the power set of the support of \( A \) restricted to sets onto which the aggregate applies. In this definition:

- \( \min_{x \in E} \mu_A(x) \) is the extent to which the crisp set \( E \) is included in \( A \),
- \( \mu_C(\text{agg}(E)) \) is the value of \( \text{truth}[\text{agg}(E) \text{ is } C] \),
- as before, the operators \( \max \) and \( \min \) are respectively a generalized existential quantifier and a generalized conjunction.

Because of the monotonicity property of function \( m \), \( \text{truth}[\text{agg}(A) \text{ is } C] \) can be rewritten:

\[
\text{truth}[\text{agg}(A) \text{ is } C] = \max_{\alpha_i \in D} \min(\alpha_i, m(A_{\alpha_i}))
\]

\[
= \max_{\alpha_i \in D} \min(\alpha_i, \mu_C(\text{agg}(A_{\alpha_i}))), \quad (6.16)
\]

where \( D \) is the set made of the actual different membership degrees in \( A \).

In the particular case where \( m \) is a fuzzy measure, Expression (6.16) is nothing but a Sugeno fuzzy integral.
Example 6.10. Consider the aggregate “maximum” and the statement “\(\text{max}(A) \text{ is high}\)” with the following fuzzy set \(A\):

\[
A = \{0.1/100, 0.3/105, 0.8/670, 1/120\}.
\]

We have:

\[
\text{truth}[\text{max}(A) \text{ is high}] = \text{max}(\min(0.1, \mu_{\text{high}}(670)), \min(0.3, \mu_{\text{high}}(670)), \min(0.8, \mu_{\text{high}}(670)), \min(1, \mu_{\text{high}}(120))).
\]

If we assume that \(\mu_{\text{high}}(670) = 1\) and \(\mu_{\text{high}}(120) = 0.2\), we obtain 0.8 as the truth value of “\(\text{max}(A) \text{ is high}\)”.

This approach defines \(\text{truth}[\text{agg}(A) \text{ is } C]\) as the highest level \(\beta\) for which each \(\alpha\)-level cut \(A_\alpha\), with \(\alpha \in [0, \beta]\), satisfies the expression “\(\text{agg}(A) \text{ is } C\)” at least to the degree \(\beta\) (Bosc and Liétard, 2005):

\[
\text{truth}[\text{agg}(A) \text{ is } C] = \max\{\beta \mid \forall \alpha \in [0, \beta], \mu_C(\text{agg}(A_\alpha)) \geq \beta\}.
\]

Referring to the previous example, any level \(\alpha\) in \([0, 0.8]\) is such that the \(\alpha\)-cut of \(A\) satisfies “the maximum is high” at least to the degree 0.8. When \(\text{truth}[\text{agg}(A) \text{ is } C]\) equals 1, each \(\alpha\)-cut of \(A\) (i.e., each interpretation of set \(A\)) is such that the value of the aggregate fully satisfies \(C\). As mentioned above, according to this view, the more numerous the interpretations of set \(A\) (i.e., its \(\alpha\)-cuts) with a high satisfaction for the condition “the aggregate is \(C\)”, the higher the truth value for “\(\text{agg}(A) \text{ is } C\)”. The number of interpretations of \(A\) is measured by the length of an interval of levels (interval of type \([0, \beta]\)) with the meaning: the larger the interval, the higher the number of interpretations.

Obviously, when the satisfaction of “the value of the aggregate is \(C\)” is high for a large interval of levels \(\alpha\), we obtain a high value for \(\text{truth}[\text{agg}(A) \text{ is } C]\) (for example 0.8 when: \(\forall \alpha \in [0, 0.8], \mu_C(\text{agg}(A_\alpha)) \geq 0.8\)). On the contrary, a low value for \(\text{truth}[\text{agg}(A) \text{ is } C]\) means that each \(\alpha\)-cut (or at least each \(\alpha\)-cut with a significant level) satisfies “the value of the aggregate is \(C\)” only poorly. This behavior is due to the monotonicity of both the aggregate and the predicate. These two points make it legitimate to consider \(\text{truth}[\text{agg}(A) \text{ is } C]\) as a suitable measure of the global satisfaction of the \(\alpha\)-cuts; they also provide an interpretation in terms of \(\alpha\)-cuts for \(\text{truth}[\text{agg}(A) \text{ is } C]\).
Although sound, this approach remains restricted to monotonic aggregates and predicates. In the next section, we propose a generalization in which the constraint of monotonicity is not imposed. This generalization maintains the meaning introduced in this section, namely: the more numerous the interpretations of set $A$ (i.e., its $\alpha$-cuts) with a high satisfaction for the constraint “the aggregate is $C$”, the higher the truth value for “$agg(A)$ is $C$”.

6.3.3 Dealing with the general case

In this section, a technique to compute $\text{truth}[agg(A) \text{ is } C]$ in the general case is introduced. First, it is shown in Section 6.3.3.1 that a straightforward adaptation of the interpretation introduced in Section 6.3.2 (Expression (6.16)) is inappropriate. Then, a new approach is introduced in Section 6.3.3.2 which is also based on the interpretation in terms of $\alpha$-cuts introduced in Section 6.3.2. Finally, Section 6.3.3.3 focuses on the relationship between “$agg(A)$ is $C$” and “$agg(A)$ is $\neg C$”, which differs from the usual case.

6.3.3.1 A straightforward interpretation

In the general case, the main drawback of using Expression (6.16) is that a low value for $\text{truth}[agg(A) \text{ is } C]$ does not necessarily mean that every $\alpha$-cut (or at least, every $\alpha$-cut with a significant level) poorly satisfies “the aggregate is $C$”. More precisely, we may have a low result for $\text{truth}[agg(A) \text{ is } C]$ when a unique $\alpha$-level cut with a low level poorly satisfies “the aggregate is $C$”, even when the others (including all significant $\alpha$-cuts) strongly satisfy “the aggregate is $C$”.

To overcome this problem, it is necessary to design other generalizations that discard $\alpha$-cuts with low level and poor satisfaction with respect to the condition. As a consequence, instead of considering an interval of type $[0, \beta]$, the idea is to consider any kind of interval. A first generalization is to look for an interval of levels $\alpha$ that is as large as possible, where $\mu_C(agg(A_\alpha))$ is as large as possible:

$$\text{truth}[agg(A) \text{ is } C] = \max \{ (\beta - \lambda) | \forall \alpha \in [\lambda, \beta], \mu_C(agg(A_\alpha)) \geq (\beta - \lambda) \}.$$

Obviously, the higher $\text{truth}[agg(A) \text{ is } C]$ becomes, the larger the interval of levels with a higher $\mu_C(agg(A_\alpha))$ should be. When $\text{truth}[agg(A) \text{ is } C]$ equals 1, it means that for each $\alpha$-cut (i.e., for each interpretation of set $A$),
the value of the aggregate fully satisfies predicate $C$. The main drawback of this approach appears when a high satisfaction for “the aggregate is $C$” is obtained for many $\alpha$-cuts but over small intervals (since only a few levels may have a low satisfaction). In this case, we would expect a rather high result for $\text{truth}[	ext{agg}(A) \text{ is } C]$ but this generalization leads to a low result, as illustrated in the next example.

**Example 6.11.** The condition “$\text{avg}(A) \text{ is around 600}$” is considered along with the following fuzzy set $A$:

$$A = \{0.38/1197, 0.39/3, 0.4/600, 1/600\}.$$  

If we assume that:

$$\mu_{\text{around 600}}(600) = 1 \quad \text{and} \quad \mu_{\text{around 600}}(401) = 0.2,$$

then we obtain the result represented in Table 6.3. According to this generalization:

$$\text{truth}[\text{avg}(A) \text{ is around 600}] = \max\{(0.38 - 0), (1 - 0.4)\} = 0.6.$$  

A higher value for “$\text{avg}(A) \text{ is around 600}$” is expected, since $\mu_{\text{around 600}}(\text{avg}(A_\alpha))$ differs from 1 only when $\alpha$ lies within $[0.38, 0.39]$.$\diamondsuit$

This drawback is mainly due to the fact that a unique interval of levels is considered in these generalizations. Another way is to take several intervals into consideration and to integrate the contribution of each of them. When referring to the previous example, the two intervals $[0, 0.38]$ and $[0.39, 1]$ could be taken into account. Such a generalization is introduced below.

### 6.3.3.2 The proposed interpretation

The interpretation proposed in this section defines $\text{truth}[	ext{agg}(A) \text{ is } C]$ as the highest $\beta$ which satisfies the gradual condition “for each level $\alpha$ in $[0, 1]$, $\mu_C(\text{agg}(A_\alpha))$ is at least equal to $\beta$”. We emphasize that “for each” is seen here as a fuzzy expression. So we need to maximize a conjunction expressing

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>0.38</th>
<th>0.39</th>
<th>0.4</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{avg}(A_\alpha)$</td>
<td>600</td>
<td>401</td>
<td>600</td>
<td>600</td>
</tr>
<tr>
<td>$\mu_{\text{around 600}}(\text{avg}(A_\alpha))$</td>
<td>1</td>
<td>0.2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6.3 The truth values of “$\text{avg}(A_\alpha) \text{ is around 600}$”
the idea that $\beta$ is high and “for each level $\alpha$ in $[0, 1]$, $\mu_C(\text{agg}(A_\alpha))$ is at least equal to $\beta$":

$$\text{truth}[\text{agg}(A) \text{ is } C] = \max_{\beta \in [0, 1]} \min(\beta, \text{each}(\beta)), \quad (6.17)$$

where each($\beta$) is a degree expressing that $\mu_C(\text{agg}(A_\alpha)) \geq \beta$ for each interpretation $A_\alpha$ of $A$ with $\alpha$ in $[0, 1]$.

Obviously, this definition depends on each($\beta$) and, as was pointed out above, a solution is to sum up the lengths of the intervals (of levels) where the threshold $\beta$ is reached:

$$\text{each}(\beta) = \sum_{[\alpha_i, \alpha_j] \text{ s.t. } \forall \alpha \in [\alpha_i, \alpha_j], \text{agg}(A_\alpha) \text{ is defined } \land \mu_C(\text{agg}(A_\alpha)) \geq \beta} (\alpha_j - \alpha_i). \quad (6.18)$$

The higher each($\beta$), the more numerous the levels $\alpha$ for which $\mu_C(\text{agg}(A_\alpha)) \geq \beta$. In particular, each($\beta$) equals 1 when for each level $\alpha$ in $[0, 1]$, $\mu_C(\text{agg}(A_\alpha))$ is larger than (or equal to) $\beta$.

A priori, the computation of Expression (6.17) needs to handle an infinity of values for $\beta$. However, it is possible to restrict computations to values for $\beta$ belonging to the set of “effective” $\mu_C(\text{agg}(A_\alpha))$ values (Bosc and Liétard, 2005) in the spirit of Expression (6.16):

$$\text{truth}[\text{agg}(A) \text{ is } C] = \max_{\beta \in D} \min(\beta, \text{each}(\beta)), \quad (6.19)$$

where $D = \{\beta \mid \exists \alpha \text{ such that } \beta = \mu_C(\text{agg}(A_\alpha))\}$.

**Example 6.12.** The statement “avg($A$) is high” is considered with the following fuzzy set $A$:

$$A = \{0.49/4050, 0.5/103, 0.5/14, 0.5/12, 0.5/25, 0.5/16, 0.5/20, 0.5/10, 0.51/700, 0.8/500, 1/600\}.$$

If we assume that:

$$\mu_{\text{high}}(200) = 0, \quad \mu_{\text{high}}(550) = 0.9 \quad \text{and} \quad \mu_{\text{high}}(600) = 1,$$

we obtain the values represented in Table 6.4. From Table 6.4 we have: $D = \{0, 0.9, 1\}$ which leads to: each(0) = 1, each(0.9) = 0.99 and each(1) = 0.21.
Fuzzy Queries Involving Quantified Statements or Aggregates

Table 6.4 The truth values of “avg($A_\alpha$) is high”

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>0.49</th>
<th>0.5</th>
<th>0.51</th>
<th>0.8</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg($A_\alpha$)</td>
<td>550</td>
<td>200</td>
<td>600</td>
<td>550</td>
<td>600</td>
</tr>
<tr>
<td>$\mu_{\text{high}}(\text{avg}(A_\alpha))$</td>
<td>0.9</td>
<td>0</td>
<td>1</td>
<td>0.9</td>
<td>1</td>
</tr>
</tbody>
</table>

According to Expression (6.19), we obtain:

$$\text{truth}[\text{avg}(A) \text{ is high}] = \max(\min(0, 1), \min(0.9, 0.99), \min(1, 0.21)) = 0.9.$$  

The statement “$\text{avg}(A) \text{ is high}” is rather true (to the degree 0.9), since most of the interpretations of fuzzy set $A$ strongly satisfy “the average is high”. Note that the gap between the levels 0.49 and 0.5 does not have a significant impact on the result, since it represents very few $\alpha$-cuts. In that respect, this behavior differs from that illustrated in Example 6.11.

The next proof shows that this approach is a generalization of that introduced in Section 6.3.2 for monotonic aggregates and predicates. In other words, when the aggregate and the predicate are monotonic in the same sense, Expression (6.19) boils down to Expression (6.16).

**Proof.** Recall that Expression (6.19) is:

$$\text{truth}[^{\text{agg}}(A) \text{ is } C] = \max_{\beta \in D} \min(\beta, \text{each}(\beta)),$$

where $D = \{\beta \mid \exists \alpha \text{ such that } \beta = \mu_C(\text{agg}(A_\alpha))\}$. This can be rewritten:

$$\text{truth}[^{\text{agg}}(A) \text{ is } C] = \max_{\alpha_i \in D} \min(\mu_C(\text{agg}(A_{\alpha_i})), \text{each}(\mu_C(\text{agg}(A_{\alpha_i}))),$$

where $D$ is the set made up of the different membership degrees in $A$. When the aggregate and $C$ are monotonic in the same sense, $\text{each}(\mu_C(\text{agg}(A_{\alpha_i})))$ can be replaced by $\alpha_i$:

$$\text{truth}[\text{agg}(A) \text{ is } C] = \max_{\alpha_i \in D} \min(\alpha_i, \mu_C(\text{agg}(A_{\alpha_i}))),$$

and we recover Expression (6.16).

6.3.3.3 Algebraic properties related to negation

This subsection is devoted to the property valid in the crisp case stating that “$^{\text{agg}}(A) \text{ is } C$” is the negation of “$^{\text{agg}}(A) \text{ is } \neg C$” and vice versa. We start with the case where the aggregate is defined for the empty set, or the fuzzy set $A$ is normalized (in this latter case, the aggregate is always applicable since the fuzzy set $A$ is not at all empty). This leads to the
following Proposition 6.1 (the interested reader can find its proof in Bosc and Liétard, 2006).

**Proposition 6.1.** When the aggregate $agg$ is defined for the empty set, or when the fuzzy set $A$, is normalized, one has:

$$\text{truth}[agg(A) \text{ is } C] = 1 - \text{truth}[agg(A) \text{ is } \neg C].$$

In particular, this property holds for the statement “$\text{card}(A) \text{ is high}$”, as well as for the statement “$\text{max}(A) \text{ is high}$”, provided that fuzzy set $A$ is normalized, since the aggregate maximum is not defined on the empty set.

In the general case (i.e., both $A$ and $C$ are fuzzy), we show that this property does not hold when $A$ is not normalized and the aggregate is not defined for the empty set. In this situation, fuzzy set $A$ is more or less empty ($A$ being not normalized) which implies that the aggregate more or less does not apply (the aggregate being undefined for the empty set). As a consequence, the following property holds (a proof of which can be found in Bosc and Liétard, 2008).

**Proposition 6.2.** When the aggregate $agg$ is not defined for the empty set and $A$ is a non-normalized fuzzy set, one has:

$$\text{truth}[agg(A) \text{ is } C] \leq \sup_{x \in \text{support}(A)} \mu_A(x).$$

In other words, the truth value of “$agg(A) \text{ is } C$” is bounded by the height of the fuzzy set $A$, which indeed reflects that the aggregate is somewhat inapplicable (fuzzy set $A$ being more or less empty). As illustrated in the next example, due to this property, “$agg(A) \text{ is } C$” cannot be the negation of “$agg(A) \text{ is } \neg C$”, i.e., their truth values are not complementary.

**Example 6.13.** Consider the fuzzy set $A$:

$$A = \{0.1/100, 0.1/10, 0.1/20000, 0.2/300\},$$

and the statement “$\text{max}(A) \text{ is high}$”. The fuzzy set $A$ is close to the empty set, and maximum does not apply to the empty set. Consequently, $\text{truth}[\text{max}(A) \text{ is high}]$ and $\text{truth}[\text{max}(A) \text{ is not high}]$ cannot be expected to be high, and more precisely, neither of them can be over 0.2 (due to Proposition 6.2). This result agrees with intuition, and as a consequence, we cannot have $\text{truth}[\text{max}(A) \text{ is not high}] = 1 - \text{truth}[\text{max}(A) \text{ is high}]$. This example shows that the property stating that “$agg(A) \text{ is } C$” is the negation of “$agg(A) \text{ is } \neg C$” does not hold in the particular
case where \( A \) is not normalized and the aggregate is not defined for the empty set.

### 6.3.4 SQLf queries involving aggregates

We now describe how the SQLf language can be extended by authorizing conditions which involve aggregates computed on fuzzy sets. The approach to evaluating such conditions advocated in the previous subsection is convenient in SQLf, since it delivers a unique degree of truth. In addition, no particular assumption is made as to its applicability, e.g., in terms of monotonicity.

It is worth mentioning that queries aimed at delivering the value of an aggregate as a final result are outside of the scope of this approach. This is the case, for instance, with the query: “for each department, retrieve the average salaries of young employees”, which does not reduce to evaluating a condition of the type “\( \text{agg}(A) \) is \( C \)”.

Throughout this section, examples are based on the relation Emp(\( \#e \), salary, age, nationality, children, distance, \( \#\text{dep} \)) which describes employees and whose key is \( \#e \). For a given employee, the value of the attribute distance is the distance between his or her home and workplace, the value of the attribute children is the number of children he/she has. We first describe the use of aggregates in queries involving a partitioning of relations, then we deal with aggregates in the context of nested queries.

#### 6.3.4.1 Aggregates in the having clause

In the basic version of SQLf (see Chapter 5), the partitioning is limited to a crisp relation (the application of aggregates being limited to crisp sets). Since the proposition introduced in the previous subsection allows the application of an aggregate to a fuzzy set, it becomes possible to partition a fuzzy relation and to deal with fuzzy subsets. As a consequence, a fuzzy predicate may appear in the where clause, and a more general pattern is now available:

```sql
select (calibration) (set\_1 of attributes) from (relations)
where (fuzzy predicate) group by (set\_2 of attributes)
having \text{agg}(att) \text{ is } C.
```

As in SQL, (set\_1 of attributes) must be a subset of (set\_2 of attributes) for the query to make sense. The result is obtained by means of a procedure
which, from a conceptual point of view, consists of the following five-step process:

**Step 1:** The Cartesian product of the relations mentioned in the `from` clause is performed and the rows fulfilling the fuzzy condition mentioned in the `where` clause are retained (a fuzzy set of rows is obtained).

**Step 2:** Rows are gathered according to their values in \(\langle \text{set}_2 \text{ of attributes} \rangle\). This step delivers different fuzzy sets of rows (fuzzy relations) which are denoted by \(p\) hereafter.

**Step 3:** For each fuzzy relation \(p\) resulting from the partitioning, the aggregate \(agg\) is applied to its (fuzzy) projection \(p[\langle \text{set}_1 \text{ of attributes} \rangle]\) onto the attribute \(att\). This projection is defined by:

\[
\forall x \in \text{support}(p)[\langle \text{set}_1 \text{ of attributes} \rangle], \mu_{p[\langle \text{set}_1 \text{ of attributes} \rangle]}(x) = \max_{t \in \text{support}(p) | t.att = x} \mu_p(t).
\]

Each fuzzy relation \(p\) is then associated with the degree \(\text{truth}[\text{agg}(p[\langle \text{set}_1 \text{ of attributes} \rangle])]\) which expresses its satisfaction with respect to the set condition mentioned in the `having` clause.

**Step 4:** The fuzzy set \(res\) of answers is delivered. The support of \(res\) is made of values of \(\langle \text{set}_1 \text{ of attributes} \rangle\) appearing in the fuzzy subsets \(p\): \n
\[
\text{support}(res) = \bigcup_p \text{support}(p)[\langle \text{set}_1 \text{ of attributes} \rangle].
\]

For a given fuzzy relation \(p\), \((\langle \text{support}(p) \rangle)[\langle \text{set}_1 \text{ of attributes} \rangle])\) is a singleton (the value of \(\langle \text{set}_1 \text{ of attributes} \rangle\) being unique for each fuzzy set \(p\), since \(\langle \text{set}_1 \text{ of attributes} \rangle\) is included in the set of attributes mentioned in the `group by` clause). Each element \(a\) in \(\text{support}(res)\) is an answer to the query to a degree given by:

\[
\mu_{res}(a) = \max_{p | a = \text{support}(p)[\langle \text{set}_1 \text{ of attributes} \rangle]} \text{truth}[\text{agg}(p[\langle \text{set}_1 \text{ of attributes} \rangle])]\] is \(C\).\]

This last expression can be simplified in the particular case where \(\langle \text{set}_1 \text{ of attributes} \rangle = \langle \text{set}_2 \text{ of attributes} \rangle\). In this case, for a given value \(a\), there exists a unique fuzzy relation \(p\) where the value of \(\langle \text{set}_1 \text{ of attributes} \rangle\) equals \(a\). In this case, we have:

\[
\mu_{res}(a) = \text{truth}[\text{agg}(p[\langle \text{set}_1 \text{ of attributes} \rangle])]\] is \(C\),\]

where \(p\) is the fuzzy relation where \(\text{support}(p)[\langle \text{set}_1 \text{ of attributes} \rangle] = \{a\}\).

**Step 5:** A quantitative or a qualitative calibration is performed depending on \(\langle \text{calibration} \rangle\). If no value appears in \(\langle \text{calibration} \rangle\), the entire fuzzy set of answers is delivered to the user.
Example 6.14. The query: “retrieve the departments where the average salary of young employees is high” involves an aggregate function (the average) which applies to each fuzzy set made of young employees from the same department. This query is expressed in SQLf as follows:

```
select #dep from Emp where age is young
group by #dep
having avg(salary) is high.
```

Relation Emp is partitioned into fuzzy subsets, we consider the subset \( p \) associated with department 100, as given in Table 6.5. This fuzzy relation is associated with the degree \( \text{truth}[\text{avg}(A) \text{ is high}] \) where \( A \) is the result of the projection of \( p \) onto the attribute salary:

\[
A = \{0.2/51000, 0.8/3000, 0.9/1000, 1/8000, 1/12000\}.
\]

If it is assumed that \( \mu_{\text{high}}(15000) = 1, \mu_{\text{high}}(6000) = 0.3, \mu_{\text{high}}(7000) = 0.5, \) and \( \mu_{\text{high}}(10000) = 1, \) the results associated with each \( \alpha \)-cut are those given in Table 6.6. From this table, we obtain the set of values \( D = \{0.3, 0.5, 1\} \), which leads to each(0.3) = 1, each(0.5) = 0.4, each(1) = 0.3. Finally, we obtain:

\[
\text{truth}[\text{avg}(A) \text{ is high}] = \max(\min(0.3, 1), \min(0.5, 0.4), \min(1, 0.3)) = 0.4.
\]

From Table 6.6, it can be checked that few \( \alpha \)-cuts strongly satisfy “to have a high average salary”. Consequently, the interpretation delivers a rather low result (degree 0.4) for the condition “the average salary of young employees is high”.

<table>
<thead>
<tr>
<th>#e</th>
<th>salary</th>
<th>age</th>
<th>nationality</th>
<th>children</th>
<th>distance</th>
<th>#dep</th>
<th>young</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>3,000</td>
<td>34</td>
<td>French</td>
<td>3</td>
<td>20</td>
<td>100</td>
<td>0.8</td>
</tr>
<tr>
<td>21</td>
<td>12,000</td>
<td>18</td>
<td>French</td>
<td>1</td>
<td>190</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>8,000</td>
<td>24</td>
<td>French</td>
<td>0</td>
<td>200</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>42</td>
<td>51,000</td>
<td>40</td>
<td>German</td>
<td>2</td>
<td>80</td>
<td>100</td>
<td>0.2</td>
</tr>
<tr>
<td>46</td>
<td>1,000</td>
<td>30</td>
<td>French</td>
<td>1</td>
<td>110</td>
<td>100</td>
<td>0.9</td>
</tr>
</tbody>
</table>

| \( \alpha \) | 0.2 | 0.8 | 0.9 | 1    |
| avg(\( A_\alpha \)) | 15,000 | 6,000 | 7,000 | 10,000 |
| \( \mu_{\text{high}}(\text{avg}(A_\alpha)) \) | 1 | 0.3 | 0.5 | 1 |
We now imagine that, in Table 6.5, several employees have the same salary. This is a particular case where several occurrences of the same value are aggregated. As a consequence, the aggregate applies to a fuzzy multiset instead of a fuzzy set, and the interpretation of $\text{truth}[\text{agg}(A) \text{ is } C]$ is slightly changed (since $A$ is a fuzzy multiset instead of a fuzzy set). However, as shown in the next example, the approach still works, the only change being in the nature of the $\alpha$-cuts, which become crisp multisets. As to partitioning, it is necessary to revise the projection so as to keep duplicate values (i.e., the projection delivers a fuzzy multiset).

**Example 6.15.** The query: “retrieve the departments where the average salary of young employees is around 2,500” is considered:

\[
\text{select } \#\text{dep from Emp where age is young group by } \#\text{dep having avg(salary) is around 2500.}
\]

We consider the subset $p$ of tuples associated with department 100 where three employees (13, 29, 32) have the same salary (3,000) (see Table 6.7). Fuzzy relation $p$ (i.e., department 100) is associated with the degree $\text{truth}[\text{avg}(A) \text{ is around 2500}]$, where $A$ is the fuzzy projection of $p$ onto the attribute salary. Since several employees have the same salary of 3,000 (employees 13, 29, 32) this salary is attached to several degrees of satisfaction with respect to the condition to be young. As a consequence, in this particular case, the projection delivers a fuzzy multiset of salaries (duplicate salary values are retained):

\[
A = \{0.2/51000, 0.8/3000, 0.9/1000, 1/3000, 1/3000\}.
\]

Assuming that:

\[
\mu_{\text{around 2500}}(2500) = 1, \quad \mu_{\text{around 2500}}(2333) = 0.8, \\
\mu_{\text{around 2500}}(3000) = 0.2, \quad \mu_{\text{around 2500}}(12200) = 0,
\]

<table>
<thead>
<tr>
<th>#e</th>
<th>salary</th>
<th>age</th>
<th>nationality</th>
<th>children</th>
<th>distance</th>
<th>#dep</th>
<th>young</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>3,000</td>
<td>34</td>
<td>French</td>
<td>3</td>
<td>200</td>
<td>100</td>
<td>0.8</td>
</tr>
<tr>
<td>29</td>
<td>3,000</td>
<td>18</td>
<td>German</td>
<td>1</td>
<td>100</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>32</td>
<td>3,000</td>
<td>24</td>
<td>Italian</td>
<td>0</td>
<td>220</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>51,000</td>
<td>40</td>
<td>German</td>
<td>2</td>
<td>380</td>
<td>100</td>
<td>0.2</td>
</tr>
<tr>
<td>47</td>
<td>1,000</td>
<td>30</td>
<td>French</td>
<td>1</td>
<td>10</td>
<td>100</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Table 6.7 Young employees of department 100
the results associated with each α-cut (which is a multiset) are those represented in Table 6.8. From this table, we have the set of values \( D = \{0.2, 0.8, 1\} \) which leads to each(0.2) = 0.8, each(0.8) = 0.7, each(1) = 0.6. Finally, we obtain:

\[
\text{truth[avg}(A)\text{ is around 2500]} = \max(\min(0.2, 0.8), \min(0.8, 0.7), \min(1, 0.6)) = 0.7.
\]

This result confirms our intuition, since most of the α-cuts strongly satisfy “to have an average around 2,500”.

6.3.4.2 Nested queries

As in SQL, it is possible in SQLf to build predicates calling on subqueries and to compare the value of an attribute with the value of an aggregate delivered by a subquery. The simplest case is when a unique value is returned by the subquery; in SQLf such a condition is written:

\[
\text{⟨attribute⟩} \theta_f \text{⟨select agg(att) from ⟨relations⟩ where ⟨Boolean condition⟩⟩},
\]

where \( \theta_f \) is a usual comparator in \{>, <, =, ≠, ≥, ≤\} or a fuzzy one (e.g., much larger than). A restriction in the first version of SQLf made it impossible to consider a fuzzy predicate in the where clause of the inner selection block, since the application of aggregates was restricted to crisp sets. Thanks to the approach described above, a more general pattern can be used:

\[
\text{⟨attribute⟩} \theta_f (\text{select agg(att) from ⟨relations⟩ where ⟨fuzzy condition⟩}).
\]

This condition states that the value of the attribute is in relation \( \theta_f \) with the aggregate returned by the inner selection block. The approach described above makes it possible to evaluate such a condition without evaluating the inner selection block, as is done in the regular case. The aggregate applies to the projection \( r[att] \), where \( r \) is the fuzzy relation made of rows of the Cartesian product of ⟨relations⟩ satisfying ⟨fuzzy condition⟩. The
truth value of this condition is \( \text{truth}[\text{agg}(r[\text{att}]) \text{ is } C] \), where \( C \) means “to be in relation \( \theta_i^{-1} \) with the value of (attribute)”.

**Example 6.16.** The query: “retrieve the employees whose salary is around the average salary of their young colleagues” is expressed in SQLf by:

```sqlf
select #e from Emp E1
where salary ≈ (select avg(salary) from Emp E2
               where E1.#dep = E2.#dep and E2.age is young and
               E1.#e ≠ E2.#e).
```

This query involves two copies of relation \( \text{Emp} \) (namely \( E1 \) and \( E2 \)) and the fuzzy comparator \( \approx \), which stands for “around” and can be defined, e.g., as:

\[
\mu_\approx(x, y) = 1 - \min\left(\frac{|x - y|}{2000}, 1\right)
\]

for any pair of real values \((x, y)\).

Consider employee 23 of department 100 (see Table 6.5) whose salary is 8,000. The truth value of the condition:

\[
salary ≈ (select \text{avg}(salary) \text{ from Emp E2 where } E1.#dep = E2.#dep \text{ and E2.age is young and E1.#e } \neq E2.#e),
\]

is given by \( \text{truth}[\text{avg}(A) \text{ is } C] \), where \( A \) is the fuzzy set of salaries tied to his/her young colleagues given in Table 6.9, and condition \( C \) is defined by:

\[
\mu_C(y) = \mu_\approx(8000, y) = 1 - \min\left(\frac{|8000 - y|}{2000}, 1\right).
\]

The results obtained for each \( \alpha \)-cut are reported in Table 6.10. From this table, we see that the set of values to be considered in the computation of

<table>
<thead>
<tr>
<th>salary</th>
<th>young</th>
</tr>
</thead>
<tbody>
<tr>
<td>3,000</td>
<td>0.8</td>
</tr>
<tr>
<td>12,000</td>
<td>1</td>
</tr>
<tr>
<td>51,000</td>
<td>0.2</td>
</tr>
<tr>
<td>1,000</td>
<td>0.9</td>
</tr>
</tbody>
</table>
the final degree is $D = \{0.25\}$. As $\text{each}(0.25) = 0.1$, we obtain:

$$\text{truth}[\text{avg}(A) \text{ is } C] = \min(0.25, 0.1) = 0.1.$$ 

The satisfaction degrees attached to the $\alpha$-cuts are very low and, as a consequence, a low result is obtained for the degree of satisfaction of the considered query.

The keyword all (resp. any) can be used when the inner selection block returns several values. The condition is then:

$$\langle \text{attribute} \rangle \theta_i \text{ all (resp. any) (select } \text{agg}(\text{att}) \text{ from } \text{relations} \rangle$$

$$\text{where } (\text{fuzzy condition})$$

$$\text{group by } \langle \text{set of attributes} \rangle).$$

For a given value $v$ of $\langle \text{attribute} \rangle$, the evaluation of a nested condition involving all or any is equivalent to the following three-step process:

**Step 1:** The partition defined in the nested query is performed (each fuzzy relation thereby obtained is denoted by $p$ hereafter).

**Step 2:** The truth value of the comparison is computed for each fuzzy relation $p$. Each fuzzy relation $p$ is associated with the degree $\text{truth}[\text{agg}(p[\text{att}]) \text{ is } C] = \theta_i^{-1} - \text{ of } v (\mu_C(x) = \mu_{\theta_i}(v, x)).$

**Step 3:** The overall degree of satisfaction is computed. When the keyword all (resp. any) is used, this degree expresses the fact that all (resp. at least one) fuzzy relations satisfy the comparison. It is defined as the smallest (resp. largest) value $\text{truth}[\text{agg}(p[\text{att}]) \text{ is } C]$. As before, the process to evaluate this condition does not entail the computation of the value returned by the inner selection block. More details can be found in Bosc and Liétard (2008).
6.3.5 Evaluation of SQLf queries involving aggregates

6.3.5.1 Queries involving aggregates over fuzzy sets

We now deal with the evaluation of SQLf queries involving conditions of the type “\(\text{truth}(\text{agg}(A)) \text{ is } C\)”. We first present a generic algorithm suited to the processing of such queries, then we discuss its different variations corresponding to the various ways in which such conditions can be used.

A generic algorithm. In the general case, Expression (6.19) is used to compute \(\text{truth}(\text{agg}(A)) \text{ is } C\) whose definition is:

\[
\text{truth}(\text{agg}(A) \text{ is } C) = \max_{\beta \in D} \min(\beta, \text{each}(\beta)),
\]

where \(D = \{\beta | \exists \alpha \text{ such that } \beta = \mu_C(\text{agg}(A_\alpha))\}\) and:

\[
\text{each}(\beta) = \sum_{\alpha_i, \alpha_j \text{ s.t. } \forall \alpha \in [\alpha_i, \alpha_j], \text{agg}(A_\alpha) \text{ is defined}}(\alpha_j - \alpha_i).
\]

The algorithm which computes \(\text{truth}(\text{agg}(A)) \text{ is } C\) needs to access the different values \(\beta\) (in an outer loop) and to compute \(\text{each}(\beta)\) (in an inner loop) for each of them. It is assumed, first, that the \(n\) levels of \(\alpha\)-cuts are increasingly ranked:

\[
\alpha_0 = 0 < \alpha_1 < \alpha_2 < \cdots < \alpha_n
\]

and stored in the array \(A\): \(\forall i \in [0, n], A[i] = \alpha_i\).

A second assumption concerns the values \(\mu_C(\text{agg}(A_\alpha))\), which are pre-computed and stored in the array \(B\):

\[
\forall i \in \{1, 2, \ldots, n\}, B[i] = \mu_C(\text{agg}(A_\alpha)).
\]

The final algorithm is based on an outer loop which scans \(B\) in order to retrieve the largest value \(\min(B[i], \text{each}(B[i]))\). Obviously:

\[
\text{each}(B[i]) = \sum_{j \text{ such that } B[j] \geq B[i]}(A[j] - A[j - 1]).
\]

The different values \(\text{each}(B[i])\) are dynamically computed during the processing in an inner loop, and stored in the variable \(E\). We obtain Algorithm 6.6:

Algorithm 6.6.

\[
\begin{align*}
\text{res} & := 0; \\
\text{for } i & := 1 \text{ to } n \text{ do} \\
E & := 0;
\end{align*}
\]
for $j := 1$ to $n$ do
enddo;
res := max(res, min(B[i], E));
enddo;

It is shown in the following how the previous algorithm can be used to evaluate SQLf queries calling on aggregates. The two cases of Section 6.3.4 (aggregates in the having clause, nesting) are dealt with in turn. We point out that the generation of the inputs to the algorithm (the two arrays $A$ and $B$) can generally be performed in a reasonable time.

**Aggregates in the having clause.** In this case, the query is written (see Section 6.3.4.1):

\[
\textbf{select} \ (\text{calibration}) \ (\text{set}_1 \ of \ attributes) \ \textbf{from} \ (\text{relations}) \\
\textbf{where} \ (\text{fuzzy predicate}) \\
\textbf{group by} \ (\text{set}_2 \ of \ attributes) \\
\textbf{having} \ \text{agg(att)} \ is \ C
\]

where ($\text{set}_1 \ of \ attributes$) is a subset of ($\text{set}_2 \ of \ attributes$). In this query, the fuzzy set of tuples satisfying the condition mentioned in the where clause is partitioned into several fuzzy subsets.

It is necessary to compute the degree of satisfaction with respect to the set condition mentioned in the having clause of each fuzzy subset $p$ resulting from the partitioning. This degree is given by $\text{truth}(\text{agg}(p[\text{att}]))$ is $C$ and can be obtained using Algorithm 6.6, which implies that arrays $A$ and $B$ (which represent the inputs) must be appropriately initialized.

The values $\alpha$ in array $A$ represent the different membership degrees appearing in the fuzzy set $p$ resulting from the partitioning. It is assumed that they are increasingly ranked, i.e.:

\[\alpha_0 = 0 < \alpha_1 < \alpha_2 < \cdots < \alpha_n\]

and:

\[\forall i \in [0, n], \ A[i] = \alpha_i.\]

Array $B$ contains the values $\mu_C(\text{agg}(p[\text{att}]_{\alpha_i}))$:

\[\forall i \in \{1, 2, \ldots, n\}, \ B[i] = \mu_C(\text{agg}(p[\text{att}]_{\alpha_i})).\]
These two arrays can be computed by scanning the fuzzy set $p$ resulting from the partitioning.

In terms of processing time, the main cost of the evaluation comes from the partitioning, since it is necessary to access each tuple of the considered relation. If it is assumed that each fuzzy set resulting from the partitioning can be stored in main memory (this can be done using an index on the attributes mentioned in the *group by* clause, or else the partitioning can be entirely made in main memory), the additional cost conferred by the proposed algorithm remains quite reasonable.

**Example 6.17.** Consider the query: “retrieve the departments where the average salary of young employees is high” from Example 6.14, which can be expressed in SQLf as:

```sql
select #dep from Emp where age is young
  group by #dep
having avg(salary) is high.
```

As described in Example 6.14, relation `Emp` is partitioned into fuzzy subsets (made of young employees), each of them containing the employees working in the same department (a fuzzy set $p$ describes the young employees of a given department). Relation `Emp` is partitioned into fuzzy subsets, and we consider the subset $p$ associated with department 100 as given in Table 6.11. This fuzzy relation is associated with the degree $\text{truth}[\text{avg}(A) \text{ is high}]$, where $A$ is the fuzzy projection of $p$ onto the attribute `salary`:

$$A = \{0.2/51000, 0.8/3000, 0.9/1000, 1/8000, 1/12000\}.$$  

If it is assumed that:

$$\mu_{\text{high}}(15000) = 1, \quad \mu_{\text{high}}(6000) = 0.3,$$

$$\mu_{\text{high}}(7000) = 0.5, \quad \mu_{\text{high}}(10000) = 1,$$

<table>
<thead>
<tr>
<th>#e</th>
<th>salary</th>
<th>age</th>
<th>nationality</th>
<th>children</th>
<th>distance</th>
<th>#dep</th>
<th>young</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>3,000</td>
<td>34</td>
<td>French</td>
<td>3</td>
<td>20</td>
<td>100</td>
<td>0.8</td>
</tr>
<tr>
<td>21</td>
<td>12,000</td>
<td>18</td>
<td>French</td>
<td>1</td>
<td>190</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>8,000</td>
<td>24</td>
<td>French</td>
<td>0</td>
<td>200</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>42</td>
<td>51,000</td>
<td>40</td>
<td>German</td>
<td>2</td>
<td>80</td>
<td>100</td>
<td>0.2</td>
</tr>
<tr>
<td>46</td>
<td>1,000</td>
<td>30</td>
<td>French</td>
<td>1</td>
<td>110</td>
<td>100</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Table 6.11  Young employees of department 100
then the results associated with each $\alpha$-cut are those given in Table 6.12. Array $A$ contains the different levels of $\alpha$-cuts:


while array $B$ contains the different satisfactions of the $\alpha$-cuts of $A$ with respect to the condition “average is high” ($B[i] = \mu_{\text{C}(\text{agg}(A\alpha))}$ where $\alpha_i = A[i]$):


It is then possible to apply the algorithm for the computation of Expression 6.19. Table 6.13 shows the evolution of computation (the indicated values for $E$ and $\text{res}$ are those obtained at the end of the iteration). At the end of the processing, the final result (in variable $\text{res}$) is 0.4, which, again, is in accordance with that obtained in Example 6.14.

<table>
<thead>
<tr>
<th># iteration</th>
<th>$i$</th>
<th>$j$</th>
<th>$E$</th>
<th>$\text{res}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0.2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0.2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
<td>0.2</td>
<td>0.3</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>0.8</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>0.9</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>1</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>1</td>
<td>0.2</td>
<td>0.3</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>0.2</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>0.3</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>0.4</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>4</td>
<td>1</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>0.2</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>3</td>
<td>0.2</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>0.3</td>
<td>0.4</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.12 The truth values of “avg($A\alpha$) is high”

<table>
<thead>
<tr>
<th>$\alpha$</th>
<th>0.2</th>
<th>0.8</th>
<th>0.9</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{avg}(A\alpha)$</td>
<td>15,000</td>
<td>6,000</td>
<td>7,000</td>
<td>10,000</td>
</tr>
<tr>
<td>$\mu_{\text{high}}(\text{avg}(A\alpha))$</td>
<td>1</td>
<td>0.3</td>
<td>0.5</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 6.13 Evolution of the computation of truth[avg($A\alpha$) is high]
Nested queries. When the aggregate appears in the select clause of a subquery, the following fuzzy predicate is defined (see Section 6.3.4.2):

\[
\langle \text{attribute} \rangle \theta_f \left( \text{select } \text{agg} (\text{att}) \text{ from } \langle \text{relations} \rangle \text{ where } \langle \text{fuzzy condition} \rangle \right),
\]

where \( \theta_f \) is a usual comparator in \{ >, <, =, \neq, \geq, \leq \} or a fuzzy relational operator (e.g., much larger than). The truth value of this condition is \( \text{truth}\left(\text{agg}(r[\text{att}])\right) = C \) where \( C \) means “to be in relation \( \theta_f^{-1} \) with the value of \( \langle \text{attribute} \rangle \)” and \( r \) is the fuzzy relation made of the rows of \( \langle \text{relations} \rangle \) satisfying \( \langle \text{fuzzy condition} \rangle \).

From a processing point of view, the where clause of the subquery must be evaluated first, since it produces a fuzzy relation \( r \). The projection of \( r \) onto \( \text{att} \) is then performed and \( \text{truth}\left(\text{agg}(R[\text{att}])\right) = C \) can be computed using Algorithm 6.6. The first step is then to build the two arrays \( A \) and \( B \) by scanning the fuzzy relation \( r[\text{att}] \). The values in array \( A \) are the different membership degrees in \( r[\text{att}] \) (satisfaction degrees of tuples from \( \langle \text{relations} \rangle \) with respect to \( \langle \text{fuzzy condition} \rangle \)). Array \( B \) is such that:

\[
B[i] = \mu_C(\text{agg}(r[\text{att}][\alpha_i])),
\]

where \( \alpha_i = A[i] \).

Example 6.18. We consider the query “retrieve the employees whose salary is around the average salary of their young colleagues” from Example 6.16:

\[
\begin{align*}
\text{select } & \#e \text{ from } \text{Emp } E1 \\
& \text{where } \text{salary } \approx \\
& \text{(select } \text{avg}(\text{salary}) \text{ from } \text{Emp } E2 \\
& \text{where } E1.\#\text{dep } = E2.\#\text{dep} \text{ and } E2.\text{age } \text{is young and } \text{E1.}\#e \neq E2.\#e). \\
\end{align*}
\]

This query involves two copies of relation \( \text{Emp} \) (namely \( E1 \) and \( E2 \)) and the fuzzy comparator \( \approx \), which stands for around and can be defined as:

\[
\mu_\approx(x, y) = 1 - \min \left( \frac{|x - y|}{2000}, 1 \right) \text{ for any pair of real values } (x, y).
\]

We consider employee 23 of department 100 (see Table 6.14) whose salary is 8,000. The truth value of the condition:

\[
\text{salary } \approx \text{(select } \text{avg}(\text{salary}) \text{ from } \text{Emp } E2 \text{ where } E1.\#\text{dep } = E2.\#\text{dep} \text{ and } E2.\text{age } \text{is young and } E1.\#e \neq E2.\#e) \\
\]

\]

\]

\]

\]

\]

\]
Fuzzy Queries Involving Quantified Statements or Aggregates

Table 6.14 Young employees of department 100

<table>
<thead>
<tr>
<th>#</th>
<th>salary</th>
<th>age</th>
<th>nationality</th>
<th>children</th>
<th>distance</th>
<th># dep</th>
<th>young</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>3,000</td>
<td>34</td>
<td>French</td>
<td>3</td>
<td>20</td>
<td>100</td>
<td>0.8</td>
</tr>
<tr>
<td>21</td>
<td>12,000</td>
<td>18</td>
<td>French</td>
<td>1</td>
<td>190</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>8,000</td>
<td>24</td>
<td>French</td>
<td>0</td>
<td>200</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>42</td>
<td>51,000</td>
<td>40</td>
<td>German</td>
<td>2</td>
<td>80</td>
<td>100</td>
<td>0.2</td>
</tr>
<tr>
<td>46</td>
<td>1,000</td>
<td>30</td>
<td>French</td>
<td>1</td>
<td>110</td>
<td>100</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Table 6.15 Salaries of the young colleagues of employee 23

<table>
<thead>
<tr>
<th>salary</th>
<th>young</th>
</tr>
</thead>
<tbody>
<tr>
<td>3,000</td>
<td>0.8</td>
</tr>
<tr>
<td>12,000</td>
<td>1</td>
</tr>
<tr>
<td>51,000</td>
<td>0.2</td>
</tr>
<tr>
<td>1,000</td>
<td>0.9</td>
</tr>
</tbody>
</table>

is given by $\text{truth[avg}(A)\text{ is } C]$, where $A$ is the fuzzy set of salaries tied to the employee’s young colleagues (see Table 6.15).

Because the inner selection block is evaluated first, the relation represented in Table 6.15 is stored in main memory. It is scanned to fill in arrays $A$ and $B$. Array $A$ contains the different levels of $\alpha$-cuts:


while array $B$ contains the satisfaction degrees associated with the different $\alpha$-cuts of $r[\text{salary}]$ with respect to the condition “$\approx 8000$” ($B[i] = \mu_C(\text{avg}(r[\text{att}]_{\alpha_i}))$, where $\alpha_i = A[i]$ and $\mu_C(y) = \mu_{\approx}(8000, y) = 1 - \min\left(\frac{|8000 - y|}{2000}, 1\right)$):


As mentioned above, when the keyword all (resp. any) is used, the condition becomes:

$$(\text{attribute}) \theta_f \text{ all (resp. any) (select agg(att) from (relations) where (fuzzy condition) group by (set of attributes))}.$$
following three steps:

1. The partition mentioned in the inner selection block is built. Each fuzzy set $p$ resulting from the partitioning is assumed to be either in main memory or accessible via an index.

2. The truth value of the comparison is performed for each fuzzy relation $p$. The algorithm applies successively to each fuzzy relation $p$ in order to compute $\text{truth}[\text{agg}(p[att])]$ is $C$ where $C$ means “to be in relation $\theta_{\gamma}^{-1}$ with $v$” ($\mu_{C}(x) = \mu_{\theta_{\gamma}}(v, x)$). Array $A$ contains the different levels of satisfaction appearing in $p[att]$, while array $B$ is such that $B[i] = \mu_{C}(\text{agg}(p[att]_{\alpha_{i}}))$ where $\alpha_{i} = A[i]$.

3. The overall degree of satisfaction is computed. It is defined as the smallest (resp. largest) $\text{truth}[\text{agg}(p[att])]$ is $C$ value.

### 6.4 Conclusion

In this chapter, we have discussed the expression, interpretation and evaluation of SQLf queries involving quantified statements or aggregates.

In the case of quantified statements, we have shown that such a construct can be used in SQLf queries for:

1. aggregating atomic fuzzy predicates inside a where clause,
2. expressing a condition at the level of a group of tuples inside a having clause,
3. generalizing the nesting operators all and any.

Two interpretations of fuzzy quantified propositions have been pointed out, one based on Sugeno’s integral, the other on Choquet’s. We have limited the scope of the study to quantified statements of the type “$Q X$ are $A$”, since the more general form, namely “$Q X B$ are $A$” raises serious interpretation problems. In particular, the aforementioned fuzzy-integral-based interpretations do not satisfy the following property: “If $Q$ is an increasing quantifier, and if all the degrees $\mu_{B}(x)$ tend toward 0, then the truth value of the statement “$Q X B$ are $A$” should tend toward 0”. For the form “$Q X$ are $A$”, it has been shown that query processing algorithms can take advantage of some properties associated with Sugeno’s and Choquet’s fuzzy integrals in order to allow for early termination.

Conditions involving aggregates applied to fuzzy relations can be seen as a generalization of fuzzy quantified statements, in the sense that count is just a special case of aggregate (which has the nice property of being
monotonic). The problems introduced by this generalization have been pointed out, and an interpretation based on an extension of Sugeno’s fuzzy integral has been described. It must be noted, however, that this approach makes it possible to evaluate conditions comparing an aggregate to another aggregate or to a constant, or matching it against a fuzzy predicate, but does not provide a way of computing the value of an aggregate over a fuzzy set (which is a rather tricky issue; see Dubois and Prade, 1990). The integration of this latter functionality into SQLf is still a matter for future research.
Chapter 7

Division and Antidivision of Fuzzy Relations

7.1 Introduction

It turns out that the division and antidivision operators, presented in Sections 2.2.2 and 2.3.4, are rich from a semantic point of view, as they convey the idea of association between concepts of a given domain. For instance, consider a database containing the two relations *curriculum* (*c*) and *profile* (*p*) with respective schemas \( C(\text{cand}, \text{skill}, \text{level}) \) and \( P(\text{skill}, \text{importance}) \). Tuple \( \langle c, s, l \rangle \) of *curriculum* states that candidate \( c \) possesses skill \( s \) with level \( l \), while tuple \( \langle s, i \rangle \) of *profile* means that for the considered position, skill \( s \) has importance \( i \). The query to look for candidates with at least level 2 for all the skills of importance 3 can be written by means of the division:

\[ \text{div(c-n2, p-i3, \{skill\}, \{skill\})} \]

where \( c\text{-n2} = \text{project(select(c, level \geq 2), \{emp, skill\})} \), and
\[ p\text{-i3} = \text{select(p, importance = 3)} \].

Now, assume that a user would like to retrieve the candidates who master all the “highly important” skills of the profile with a level “close to 5”. A solution is to translate the two terms “highly important” and “close to 5”, e.g., as “level greater than 8” and “importance in the interval \([4, 6]\)”, but it should be noted that a small change in these choices may have a significant impact on the result delivered. A more gradual behavior might help, by introducing preferences in the query such that:

- “highly important” and “close to 5” are fuzzy predicates,
- a division operating on fuzzy relations is made available.
Such a division of fuzzy relations is presented in Section 7.2. Similarly, imagine the case where the divisor relation \((p-i3)\) is made of 20 elements \((s_1, \ldots, s_{20})\) and that the dividend relation \((c-n2)\) contains the tuples (pairs):

\[
\{(e_1, s_1), (e_1, s_6), (e_1, s_{20}), (e_2, s_1), (e_2, s_2), \ldots, \\
(e_2, s_{19}), (e_3, s_1), (e_3, s_2), \ldots, (e_3, s_{18})\}.
\]

Neither \(e_1\), nor \(e_2\), nor \(e_3\), is satisfactory for the initial query. However, it seems legitimate to think that if \(e_1\) is definitely unsatisfactory, \(e_2\) and \(e_3\) are almost satisfactory (with an advantage for \(e_2\)) as they are associated with 19 and 18 of the required skills, respectively. The ability to discriminate between these three candidates may be desirable, and it is a matter of tolerance to exceptions. Such a mechanism is provided through the tolerant division discussed in Section 7.3.

A quite different extension of division can also be envisaged in order to deal with divisors where preferences are not assigned to tuples individually but to subsets of tuples. For example, suppose a user is interested in candidates who possess the skills \(\{s_2, s_4, s_9\}\) and if possible \(\{s_1, s_6\}\) and if possible \(\{s_{15}\}\). In such a context, the ordering of candidates takes into account the fact that the sets of skills are hierarchically ordered, which will allow for production of a discriminated set of answers, as is the case in the two previous extensions. The notion of stratified division is dealt with in Section 7.4, while stratified antidivision queries and stratified queries mixing a division and an antidivision constitute the subject of Section 7.5.

### 7.2 Division of Fuzzy Relations

#### 7.2.1 Principles

The division of fuzzy relations (as well as the extensions presented in the forthcoming sections) raises the question of rationality or legitimacy. We consider that the resulting operator deserves to be called a division if and only if its result may be characterized in terms of a quotient, i.e., the largest relation according to some inclusion criterion. Although some researchers (Cubero et al., 1994; Dubois and Prade, 1996; Mouaddib, 1993; Yager, 1991a) have proposed extended divisions, the above point is most often absent, and we concentrate on approaches defining “sound” extensions.
The extension of division to fuzzy relations is based on an adaptation of the expression defining the original division (see Expression (2.2)):

\[ \text{div}(r, s, A, B) = \{ x \mid \text{project}(s, B) \subseteq \Gamma_r^{-1}(x) \} , \]

where:

- the regular inclusion is replaced by a fuzzy one (i.e., a degree of inclusion in the spirit of Section 3.4),
- the two operands are fuzzy relations instead of regular ones.

If the dividend \( r \) and the divisor \( s \) have the respective schemas \( R(A, X) \) and \( S(B) \), this yields:

\[ \forall x \in \text{project}(\text{supp}(r, X)), \mu_{\text{div}(r, s, A, B)}(x) = \text{deg}(s \subseteq \Gamma_r^{-1}(x)) , \]

with \( \mu_{\Gamma_r^{-1}(x)}(a) = \mu_r(a, x) \), \( \text{project}(\text{supp}(r, X)) \) representing the domain of \( X \) restricted to those values appearing in the dividend (relation \( r \)). Several types of degrees of inclusion exist (see for example Section 3.4.2) and they may be used in the above formula, provided that the result obtained is a quotient. To assess the fact that the result \( t \) of the extended division is a quotient entails an adaptation of the double characterization given in Section 2.2.2 for the regular division, which yields:

\[ \forall x, (x \in \text{project}(\text{supp}(r, X))) \Rightarrow (\text{cartprod}(s, \mu_t(x)/\langle x \rangle)) \subseteq r, \quad (7.1) \]

\[ \forall x, (x \in \text{project}(\text{supp}(r, X))) \Rightarrow (\forall d_1 > \mu_t(x), \text{cartprod}(s, d_1/\langle x \rangle) \not\subseteq r). \quad (7.2) \]

It is then necessary to specify the Cartesian product as well as the inclusion used in these two expressions. It is of interest to consider a generalized version of the Cartesian product (with respect to that introduced in Section 3.3.3) whose general format is:

\[ \mu_{\text{cartprod}(r, s)}(uv) = \text{cnj}(\mu_r(u), \mu_s(v)) , \]

where \( \text{cnj} \) denotes an extended conjunction operator. As for the inclusion, it is based on the crisp inclusion of fuzzy sets (Expression (3.6)) as this property is satisfied or not. In Bosc et al. (2007b,d), it is shown that the extension based on a logical view of the inclusion, i.e.:
returns a resulting relation \( t \) which is a quotient provided that:

- if an R-implication \((\Rightarrow_R)\) is used in Expression (7.3), the Cartesian product used in Expressions (7.1) and (7.2) is based on the generative norm of \( \Rightarrow_R \) (e.g., the minimum for Gödel’s implication, the product for Goguen’s implication),
- if an S-implication \((\Rightarrow_S)\) is used in Expression (7.3), the Cartesian product used in Expressions (7.1) and (7.2) is based on the noncommutative conjunction associated with \( \Rightarrow_S \) (see Section 3.4.1).

**Example 7.1.** Consider the fuzzy relations:

\[
r = \{0.7/\langle a_1, x \rangle, 0.4/\langle a_2, x \rangle, 1/\langle a_3, x \rangle, 1/\langle a_1, y \rangle, 0.6/\langle a_2, y \rangle, 0.2/\langle a_3, y \rangle\}
\]

and

\[
s = \{1/a_1, 0.5/a_2, 0.3/a_3\},
\]

whose respective schemas are \( R(A, X) \) and \( S(B) \). The result \( t \) of the division of \( r \) by \( s \) according to Expression (7.3) is successively computed for three implications. With Gödel’s implication, the result yields:

\[
\mu_t(x) = \min(1 \Rightarrow_{Gd} 0.7, 0.5 \Rightarrow_{Gd} 0.4, 0.3 \Rightarrow_{Gd} 1) = 0.4,
\]

\[
\mu_t(y) = \min(1 \Rightarrow_{Gd} 1, 0.5 \Rightarrow_{Gd} 0.6, 0.3 \Rightarrow_{Gd} 0.2) = 0.2.
\]

When performing the Cartesian product of \( s \) and \( t \) with the norm minimum, one obtains the relation:

\[
\{0.4/\langle a_1, x \rangle, 0.4/\langle a_2, x \rangle, 0.3/\langle a_3, x \rangle, 0.2/\langle a_1, y \rangle, 0.2/\langle a_2, y \rangle, 0.2/\langle a_3, y \rangle\},
\]

which is strictly included in \( r \); thus Expression (7.1) holds. It is easy to check that Expression (7.2) holds as well, because of the presence of the tuples \( \langle a_2, x \rangle \) and \( \langle a_3, y \rangle \) whose grades equal those they have in \( r \).

Similarly, with the Kleene–Dienes implication, the result \( t \) of the division of \( r \) by \( s \) is such that:

\[
\mu_t(x) = \min(1 \Rightarrow_{KD} 0.7, 0.5 \Rightarrow_{KD} 0.4, 0.3 \Rightarrow_{KD} 1) = 0.5,
\]

\[
\mu_t(y) = \min(1 \Rightarrow_{KD} 1, 0.5 \Rightarrow_{KD} 0.6, 0.3 \Rightarrow_{KD} 0.2) = 0.6.
\]

When performing the Cartesian product of \( s \) and \( t \) with the appropriate noncommutative conjunction \( ncc_1(a, b) = 0 \) if \( a + b < 1 \), \( b \) otherwise, one obtains: \( \{0.5/\langle a_1, x \rangle, 0.6/\langle a_1, y \rangle, 0.6/\langle a_2, y \rangle\} \), which is strictly included in \( r \).
Relation $t$ is maximal, because if 0.5 (resp. 0.6) is increased to 0.5+ (resp. 0.6+), the value $\text{ncc}1(0.5, 0.5^+)$ (resp. $\text{ncc}1(0.5, 0.6^+)$) leads to assignment of the degree 0.5+ (resp. 0.6+) to $\langle a_2, x \rangle$ (resp. $\langle a_2, y \rangle$) which is over 0.4 (resp. 0.6), and Expression (7.2) holds as well.

Last, we use Lukasiewicz's implication. The result of the division is:

$$t = \{0.7/x, 0.9/y\}.$$  

When performing the Cartesian product of $s$ and $t$ with the associated norm $\tau(a, b) = \max(a + b - 1, 0)$, one obtains:

$$\{0.7/\langle a_1, x \rangle, 0.2/\langle a_2, x \rangle, 0.9/\langle a_1, y \rangle, 0.4/\langle a_2, y \rangle, 0.2/\langle a_3, y \rangle\},$$

which is included in $r$. In addition, it can easily be checked that $t$ is maximal and once again, the characterization Expressions (7.1) and (7.2) hold.

As mentioned in Section 3.4.2, the use of an S-implication also accounts for a difference-based inclusion degree using Expression (3.11). As to the cardinality-based inclusion (see Expression (3.9)) for the division of fuzzy relations, it is shown in Bosc et al. (2007d) that no conjunction (neither norm nor noncommutative conjunction) can be found that characterizes the result of the division as a maximal relation. Indeed, it turns out that:

- the smallest Cartesian product of the divisor and the smallest result of a division may lead to a relation which is not included in the dividend,
- the largest result of a division may not be maximal.

### 7.2.2 On the choice of implication

The division of fuzzy relations constitutes an enrichment of database query languages which, in addition, turns out to be sound from a theoretical point of view, provided that R-implications or S-implications are used. To illustrate the impact of the choice of a given type of fuzzy implications, we will consider the application context with the two relations *curriculum* and *profile* previously introduced. We consider queries of the form: “find the candidates with a high level in all the skills of the considered (fuzzy) profile”. Depending on whether an R-implication or an S-implication is chosen, the meaning of this query will vary significantly. With an R-implication, the levels of the skills in the profile play the role of thresholds (according to the meaning of R-implications). The profile is seen as a prototype, i.e., the description of an employee with the minimal levels of skills required to be completely satisfactory. Any actual candidate is penalized as soon as he/she
has (at least) one skill with a level below the objective appearing in the prototype. The choice of different R-implications impacts the result when the threshold is not reached. When an S-implication is used, the profile accounts for the importance of each skill, or alternatively, the extent to which a skill is considered more or less critical. As a consequence, any non-completely important skill will confer a guaranteed degree of satisfaction, whatever the level attained by the candidate.

**Example 7.2.** Consider the relations:

\[
\begin{align*}
\text{curriculum} & = \{1/(\text{John}, s_1), 0.6/(\text{John}, s_2), 0.4/(\text{John}, s_3), \\
& \quad 0.8/(\text{Peter}, s_1), 0.6/(\text{Peter}, s_2)\}, \\
\text{profile} & = \{1/s_1, 0.8/s_2, 0.2/s_3\}.
\end{align*}
\]

The division of *curriculum* by *profile* using Gödel’s (resp. Goguen’s) implication returns \{0.6/John\} (resp. \{0.75/John\}), whereas the use of the Kleene–Dienes (resp. the Reichenbach) implication leads to \{0.6/John, 0.8/Peter\} (resp. \{0.68/John, 0.8/Peter\}). In this example, a candidate who is eliminated with an R-implication (here Peter does not possess skill \(s_3\) at all and he is discarded using either of the two considered R-implications), might be the best one with an S-implication (Peter is only somewhat penalized, since competence \(s_3\) is not considered highly important in this context).

### 7.2.3 Primitivity of the extended division operator

We saw in Section 2.2.2 that regular division is a non-primitive operator which can be rewritten as an algebraic expression calling on difference, projection and Cartesian product operators; see Expression (2.3). One may wonder about the primitivity (or not) of the extended division defined by Expression (7.3). This question was investigated in Bosc (1998) and the main result achieved is the following. The division of fuzzy relations initially expressed using Expression (7.3), i.e., according to a logical view of the inclusion based on R-implications or S-implications, can be reformulated as:

\[
div(r, s, A, B) = \\
differ\left(\text{project}\left(\text{supp}(r), X\right), \quad (7.4) \\
\text{project}\left(\text{differ}(\text{supp}(r), \text{cartprod}(\text{project}(r, X), s)), X\right)\right).
\]
Division and Antidivision of Fuzzy Relations

provided that:

- the Cartesian product is performed with the norm minimum,
- the conjunction intervening in the difference is the noncommutative conjunction associated with the implication when an R-implication is used for the division, or the norm associated with the implication if the division is done with an S-implication.

It should be noted that relation $r$ must be replaced by its support in its first two occurrences (in the previous formula), which is intended to define the referential of $X$-values of interest.

**Example 7.3.** Consider the relations:

$r = \{1/\langle a_1, x \rangle, 0.5/\langle a_2, x \rangle, 0.8/\langle a_1, y \rangle\}$, $s = \{1/a_1, 0.6/a_2\},$

whose respective schemas are $R(A, X)$ and $S(B)$. Using the definition based on an implication, as in Expression (7.3), we obtain:

$\text{div}(r, s, A, B) = \{0.5/x\}$ for Gödel’s implication,

$\text{div}(r, s, A, B) = \{0.5/x, 0.4/y\}$ for the Kleene–Dienes implication,

$\text{div}(r, s, A, B) = \{0.9/x, 0.4/y\}$ for Łukasiewicz’s implication.

Now, we compute the result of the algebraic expression of division (Expression (7.4)) for the set difference operators corresponding to the conjunctions associated with these three implications. The components that do not depend on the difference are:

$\text{project}(\text{supp}(r)) = \{x, y\},$

$\text{cartprod}(\text{project}(\text{supp}(r)), s) = \{1/\langle a_1, x \rangle, 0.6/\langle a_2, x \rangle, 1/\langle a_1, y \rangle, 0.6/\langle a_2, y \rangle\}.$

With the noncommutative conjunction $\text{nccl}(a, b) = 0$ if $(a + b) \leq 1$, $b$ otherwise, associated with Gödel’s implication, the inner difference yields $\{0.5/\langle a_2, x \rangle, 1/\langle a_1, y \rangle\}$ as $\text{nccl}(1, 0) = 0$, $\text{nccl}(0.6, 0.5) = 0.5$, $\text{nccl}(1, 0.2) = 0.2$, $\text{nccl}(0.6, 1) = 1$, and the final result is $\{0.5/x\}$.

With the norm $\text{T}(a, b) = \text{min}(a, b)$ associated with the Kleene–Dienes implication, the inner difference returns $\{1/\langle a_1, x \rangle, 0.5/\langle a_2, x \rangle, 0.6/\langle a_2, y \rangle\}$ as $\text{min}(1, 0) = 0$, $\text{min}(0.6, 0.5) = 0.5$, $\text{min}(1, 0.2) = 0.2$, $\text{min}(0.6, 1) = 0.6$, and the final result is $\{0.5/x, 0.4/y\}$.

Finally, with the norm $\text{T}(a, b) = \text{max}(a + b - 1, 0)$ associated with Łukasiewicz’s implication, the inner difference delivers $\{0.1/\langle a_2, x \rangle,$
0.2/(a_1, y), 0.6/(a_2, y)}, and the final result is \{(0.9/x, 0.4/y}\}. Observe that the two methods of computation lead to the same results for the three implications considered.

7.2.4 Expressing extended division in SQLf

From a practical point of view, it is necessary to express division queries in a user-oriented language such as SQLf (see Section 2.3). Recall that there are two ways of expressing a division in SQL: one based on relation partitioning and a transformation of set inclusion into a cardinality constraint, and another one calling on nested predicates (see Section 2.3.4). It turns out (see Bosc et al. (2007b,d) for details) that none of these approaches works for the division of fuzzy relations. The adopted formulation is based on the introduction of a set containment operator, in the spirit of that initially present in SEQUEL2 (Chamberlin et al., 1976). This operator, generically denoted by “F contains E”, delivers the degree of inclusion of E in F. Thus, the division of r(A, X) by s(B, Y) is written as:

\[
\text{select } X \text{ from } r \text{ where } \text{fuzzy-condition-1} \\
\text{group by } X \\
\text{having set}(A) \text{ contains}[\text{implication}] \\
(\text{select } B \text{ from } s \text{ where } \text{fuzzy-condition-2}).
\]

The degree of inclusion is computed using Expression (3.10) with the implication whose name is given as a parameter (“KD” for Kleene–Dienes, “Gd” for Gödel, “Lu” for Łukasiewicz, and so on).

Example 7.4. Consider the query to look for candidates who master all the “highly important” skills of the profile with a level “close to 5”, addressed to the database containing the two relations curriculum and profile introduced previously. Using Reichenbach’s implication, this query is expressed as:

\[
\text{select } \text{cand from curriculum where level is close_to_5} \\
\text{group by cand} \\
\text{having set(skill) contains[Rb]} \\
(\text{select skill from profile where importance is high}),
\]

where the fuzzy terms “close_to_5” and “high” are user-defined predicates.
7.3 Tolerant Division

As mentioned in the introduction to this chapter, defining tolerant divisions may be appealing when it is necessary to discriminate between situations where division represents too high a demand with the risk of returning no answer. Two approaches to the design of tolerant divisions are reviewed below.

7.3.1 Exception-based tolerant division

The first approach to extension relies on replacement of the inclusion at the basis of the division of regular or fuzzy relations (Expressions (2.2) and (7.3)) by a tolerant inclusion in the sense of Section 3.4.2.5. Of course, the semantics of such a division depends on the choice of a quantitative or a qualitative tolerant inclusion.

7.3.1.1 Quantitative tolerant division

Quantitative tolerant division makes use of the quantifier “almost all” which is a softened version of the universal quantifier (∀) from which the grades:

• \( \mu_{\text{almost all}}(0) = w_n = 0 \),

• \( \mu_{\text{almost all}}(1) = w_0 = 1 \),

• \( \mu_{\text{almost all}}(1 - \frac{i}{n}) = w_i \) for \( 1 < i < n \)

are issued, and where \( w_i \) expresses the degree of satisfaction when \( i \) out of the \( n \) elements of the referential are ignored. Algebraically, the tolerant division of \( r(A, X) \) by \( s(B) \) can be written as:

\[
\mu_{\text{qt-tol-div}}(r, s, A, B(x)) = \max_{k \in [1, n]} \min(\alpha_k, w_{k-1}) = \min_{k \in [1, n]} \max(\alpha_k, w_k), \quad (7.5)
\]

where \( \alpha_k \) is the \( k \)th smallest implication degree \( (\mu_s(a) \Rightarrow \mu_r(a, x)) \) and \( w_k = \mu_{\text{almost all}}(1 - \frac{k}{n}) \) assuming that the support of relation \( s \) involves \( n \) elements. Bosc et al. (2007b) showed that the result of this tolerant division with any R- or S-implication used in the calculus of the \( \alpha_j \)'s in Expression (7.5) is a quotient, provided that:

• an appropriate tolerant Boolean inclusion is used in the characterization formulas,
either the divisor is a normalized relation (i.e., at least one element has the maximal degree 1), or else the relaxed quantifier “almost all” is non-fuzzy.

In SQLf, the expression of this division is similar to that given previously:

\[
\text{select } X \text{ from } r \text{ where fuzzy-condition-1} \\
\text{group by } X \\
\text{having set}(A) \text{ qt-tol-contains}[\text{implication, almost all}] \\
(\text{select } B \text{ from } s \text{ where fuzzy-condition-2}).
\]

The set comparison operator becomes “qt-tol-contains”, whose arguments are the implication used and the user-defined softened quantifier “almost all”.

### 7.3.1.2 Qualitative tolerant division

If qualitative tolerant inclusion is used, the division obtained allows for low-intensity exceptions, i.e., some compensation takes place when the situation is close to full inclusion (see Section 3.4.2.5). The technical aspects of such inclusions are dealt with in Section 3.4.2.5; here, we simply give the algebraic expression of the tolerant division of \( r \) by \( s \) according to the implication used. This expression is a direct adaptation of Expressions (3.13) and (3.14), both calling on two parameter values \( \alpha \) and \( \beta \) in the unit interval.

For an R-implication, we have:

\[
\mu_{\text{qlR-tol-div}}(r, s, A, x)(x) = \min_{x \in X} \mu_s(a) \Rightarrow_R (\mu_r(a, x)) + \delta,
\]

where

\[
\delta = \begin{cases} 
0 & \text{if } \mu_s(a) - \mu_r(a, x) \geq \beta, \beta \in [0, 1], \\
\mu_s(a) - \mu_r(a, x) & \text{if } \mu_s(a) - \mu_r(a, x) \leq \alpha, \alpha \in [0, 1] \text{ and } \alpha < \beta,
\end{cases}
\]

It is easy to see that it is possible to characterize the resulting relation \( t \) of such a tolerant division as a quotient, provided that the characterization accounts for the tolerance introduced in the division as follows:

\[
\forall x, (x \in \text{project}(\text{supp}(r), X)) \Rightarrow (\text{cartprod}(s, \mu_t(x)/\langle x \rangle)) \subseteq r', (7.6)
\]

\[
\forall x, (x \in \text{project}(\text{supp}(r), X)) \Rightarrow \\
(\forall d_1 > \mu_t(x), \text{cartprod}(s, d_1/\langle x \rangle) \not\subseteq r'), (7.7)
\]
Division and Antidivision of Fuzzy Relations

with the Cartesian product performed with the generative norm of the R-implication considered for the division and \( r' \) such that \( \mu_{r'}(u) = \mu_r(u) + \delta \).

**Example 7.5.** Consider the relations \( s = \{1/a, 0.1/b, 0.6/c\} \) and \( r = \{0.7/(a, x), 0.4/(c, x)\} \). Element \( x \) does not belong to the result of the non-tolerant division of \( r \) by \( s \) using Gödel’s implication. However, with \( \alpha = 0.1, \beta = 0.3 \), qualitative exception-tolerant division assigns \( x \) the degree 0.45. The Cartesian product of \( s \) and 0.45/\( x \) delivers \( \{0.45/(a, x), 0.1/(b, x), 0.45/(c, x)\} \), which is included in \( r' = \{0.7/(a, x), 0.1/(b, x), 0.45/(c, x)\} \). The degree 0.45 cannot be upgraded, as it would lead to a tuple \((c, x)\) in the Cartesian product with a degree strictly greater than that it has in \( r' \) (0.45), and we can conclude that the result obtained is maximal.

For an S-implication, qualitative tolerant division can be written as:

\[
\mu_{qls-tol-div}(r, s, A, B)(x) = \min_x \in X (\mu_s(a) - \delta_1 \Rightarrow_\delta (\mu_r(a, x) + \delta_2)),
\]

where

\[
\begin{align*}
\delta_1 &= \begin{cases} 
\mu_s(a) & \text{if } \mu_s(a) \leq \alpha, \\
0 & \text{if } \mu_s(a) \geq \beta, \\
\left(\frac{\alpha}{\beta - \alpha}\right) \ast (\beta - \mu_s(a)) & \text{in between,}
\end{cases} \\
\delta_2 &= \begin{cases} 
1 - \mu_r(a, x) & \text{if } \mu_r(a, x) \geq 1 - \alpha, \\
0 & \text{if } \mu_r(a, x) \leq 1 - \beta, \\
\left(\frac{\alpha}{\beta - \alpha}\right) \ast (\mu_r(a, x) + \beta - 1) & \text{in between.}
\end{cases}
\end{align*}
\]

The result delivered in this case can also be characterized as a quotient thanks to:

\[
\forall x, (x \in \text{proj}(\text{supp}(r), X)) \Rightarrow (\text{cartprod}(s', \{\mu_t(x)/x\})) \subseteq r',
\]

\[
\forall x, (x \in \text{proj}(\text{supp}(r), X)) \Rightarrow (\forall d_1 > \mu_t(x), \text{cartprod}(s', \{d_1/x\}) \not\subseteq r'),
\]

where the Cartesian product is performed with the noncommutative conjunction associated with the S-implication chosen in the tolerant division. Relations \( r' \) and \( s' \) are such that \( \mu_{r'}(u) = \mu_r(u) + \delta_2, \mu_{s'}(u) = \mu_s(u) - \delta_1 \).

**Example 7.6.** Consider the relations \( s = \{0.7/a, 0.9/b, 0.18/d\} \) and \( r = \{1/(a, x), 1/(b, x), 0.2/(c, x), 0.85/(d, x)\} \). Using the
Kleene–Dienes implication, non-tolerant division assigns \( x \) the grade 0.75. Using qualitative tolerant inclusion along with the thresholds \( \alpha = 0.2 \), \( \beta = 0.45 \), one obtains the degree 0.91 for \( x \). The (modified) relations \( r' \) and \( s' \) are respectively \( s' = \{0.7/a, 0.9/b, 0.1/c, 0.3/d\} \) and \( r' = \{1/(a, x), 1/(b, x), 0.2/(c, x), 0.91/(d, x)\} \). The Cartesian product of \( s' \) and 0.91/x yields \( \{0.91/(a, x), 0.91/(b, x), 0.91/(d, x)\} \), which is included in \( r' \). If 0.91 is increased, tuple \( (d, x) \) in the Cartesian product receives a degree strictly greater than 0.91, and the inclusion no longer holds. Here again, the result obtained by qualitative tolerant division is maximal.

In SQLf, the expression of qualitative tolerant division does not significantly differ from that of the quantitative tolerant form. It is written as:

\[
\text{select } X \text{ from } r \text{ where } \text{fuzzy-condition-1} \\
\text{group by } X \\
\text{having set}(A) \text{ ql-tol-contains}[\text{implication}, \alpha, \beta] \\
(\text{select } B \text{ from } s \text{ where } \text{fuzzy-condition-2}).
\]

7.3.2 Resemblance-based tolerant division

We now consider a tolerant division based on the use of a resemblance relation. In order to recover a superset of the result of the original division, the principle is to replace the dividend relation \( r \) by an enlarged relation \( r' \), obtained by composing \( r \) with a resemblance relation saying that values of the domain common to the dividend and divisor relations \( (\text{dom}(A) = \text{dom}(B)) \) may be somewhat close. In the Boolean context, an equivalence relation (reflexive, symmetric, and transitive) would be used for transforming the dividend. In the fuzzy framework, a similarity relation, where transitivity is replaced by:

\[
\forall x, y, z \in X, \text{sim}(x, y) \geq \sup_z \min(\text{sim}(x, z), \text{sim}(z, y)),
\]

or a proximity relation if transitivity is skipped, can be used. Whatever the type of resemblance relation used (denoted later by \( \text{rsb} \)), the dividend \( r \) is dilated, in the sense that the elements of the referential that are somewhat close to an element initially present in \( r \) are added to \( r \). One considers that an element \( (a, x) \) that is missing in the dividend, can be replaced by another \( (b, x) \) which is present and such that \( b \) is close to (resembles) \( a \). The mechanism suggested here produces resemblance-based tolerant division.
If $r$ and $s$ are two fuzzy relations of respective schemas $R(A, X)$ and $S(B)$, the resemblance-based tolerant division of $r$ by $s$ is defined as follows:

$$\mu_{rsb-tol-div}(r, s, A, B)(x) = \min_{a \in \text{supp}(s)} \mu_s(a) \Rightarrow f \mu_{dil}(r)(a, x),$$

(7.8)

where $dil(r)$ is a dilated variant of $r$ obtained using the resemblance relation $rsb$, i.e.:

$$\mu_{dil(r)}(a, x) = \sup_{b \in \text{dom}(A)} \top(\mu_r(b, x), \mu_{rsb}(a, b)),$$

where $\top$ denotes a triangular norm. It is worth noting that, if a value initially absent can be added to the dividend $r$ (with a given degree of membership), the degree of a value initially present can also be increased. The approximate division of $r(A, X)$ by $s(B)$ looks for the $X$-values associated in $r$ with all the elements of the divisor or their neighbors (or substitutes). The less demanding the norm used, the larger the final dividend obtained, and the greater the result of the tolerant division.

By its very nature, resemblance-based tolerant division returns a quotient. Indeed, one may observe that this division reduces to a non-tolerant division where the dividend is not the initial one, but the one stemming from the use of the resemblance relation. The result $t$ obtained can be characterized as a quotient thanks to Expressions (7.6) and (7.7), where the relation called $r'$ becomes $dil(r)$.

**Example 7.7.** We consider:

- the proximity relation defined over colors $\text{simcol} = \{1/(\text{red, red}), 0.7/(\text{red, carmine}), 0.2/(\text{red, vermillion}), 1/(\text{carmine, carmine}), 1/(\text{vermillion, vermillion}), 0.8/(\text{vermillion, orange}), 1/(\text{orange, orange})\}$,
- the dividend relation $r = \{1/(\text{carmine, } p_1), 0.6/(\text{red, } p_1), 1/(\text{red, } p_2), 0.6/(\text{vermillion, } p_1), 0.4/(\text{orange, } p_2)\}$,
- the divisor relation $s = \{1/(\text{carmine, 0.3/orange})\}$.

It is assumed that the tuple $d/(c, p)$ of $r$ means that color $c$ is present in painting $p$ with the level of significance $d$. The two tuples of $s$ represent reference colors which are of interest to the user. Non-tolerant division of $r$ by $s$ using Gödel’s implication delivers an empty result. The dilation of $r$ with the norm minimum (the largest one) yields:

$$dil(r) = \{1/(\text{carmine, } p_1), 0.7/(\text{red, } p_1), 0.6/(\text{vermillion, } p_1), 0.6/(\text{orange, } p_1), 1/(\text{red, } p_2), 0.7/(\text{carmine, } p_2), 0.4/(\text{orange, } p_2), 0.4/(\text{vermillion, } p_2)\},$$
and resemblance-based tolerant division of \( r \) by \( s \) with Gödel’s implication in Expression (7.8) returns the relation \( t = \{1/p_1, 0.7/p_2\} \). The Cartesian product of \( s \) and \( \{1/p_1\} \) using the conjunction \( \top(a, b) = \min(a, b) \), which is the generative norm of Gödel’s implication, is: \( \{1/(\text{carmine}, p_1), 0.3/(\text{orange}, p_1)\} \), which is included in \( \text{dil}(r) \). Moreover, since 1 is the maximal grade, the grade assigned to \( p_1 \) cannot be increased. Similarly, the product of \( s \) and \( \{0.7/p_2\} \) is \( \{0.7/(\text{carmine}, p_2), 0.3/(\text{orange}, p_2)\} \), which is also included in \( \text{dil}(r) \) and is maximal.

7.4 Stratified Division

7.4.1 Introduction

The key idea behind the stratified division is to split the divisor into subsets organized hierarchically. By so doing, ordinal preferences are attached to the divisor, which is called a “stratified divisor”. The general approach is to consider that the degree of satisfaction assigned to an element \( x \) of the dividend depends on the number of subsets \( S_i \) of the divisor with which it is connected. Three quite natural meanings can be distinguished, depending on the role allotted to the divisor seen as a hierarchical set:

- a direct extension of the division in a conjunctive way, where the first layer of the divisor is mandatory and the following layers are considered only desirable: find the elements \( x \) connected with \( S_1 \) and if possible ... and if possible \( S_n \).
- a disjunctive view where \( x \) is all the more satisfactory as it is connected with all the values of a highly preferred (sub)set of the divisor: find the elements \( x \) connected with \( S_1 \) or else ... or else \( S_n \),
- an intermediate approach where \( x \) is all the more highly ranked as it is connected with numerous and preferred (sub)sets of the divisor: find the elements \( x \) connected with \( S_1 \) and-or ... and-or \( S_n \).

These three types of queries can be expressed in an SQL-like style, where the dividend may be any intermediate relation and the stratified divisor is either explicitly given by the user or results from a series of subqueries. Here also, we use a syntax based on a containment operator:

```sql
select top k X from r [where condition]
group by X
having set(A)
    contains \{v_{1,1}, ..., v_{1,j_1}\} connective ... connective \{v_{n,1}, ..., v_{n,j_n}\},
```
Division and Antidivision of Fuzzy Relations

where “connective” stands for either “and if possible”, or “or else”, or “and-or”. Such a statement induces an ordering over the divisor, namely $(S_1 = \{v_{1,1}, \ldots, v_{1,j_1}\}) \succ \cdots \succ (S_n = \{v_{n,1}, \ldots, v_{n,j_n}\})$, where $a \succ b$ denotes preference of $a$ over $b$. Associated with this preference relation is an ordinal scale $L$ with labels $l_i$ (such that $l_1 > \cdots > l_n > l_{n+1}$) which is used to assign levels of satisfaction to elements pertaining to the result of a stratified division ($l_1$ corresponds to the highest satisfaction and $l_{n+1}$ expresses rejection; they are the counterparts of 1 and 0 in the unit interval). Clearly, the framework considered is purely ordinal, and the user just has to specify a stratified divisor.

Example 7.8. First, consider the case of a user looking for wineshops offering Saint Emilion Grand Cru, Pomerol, and Margaux, and if possible Gewurztraminer Vendanges Tardives and Chablis Premier Cru, and if possible Pommard and Chambertin, assuming that a database involving the relation $\text{wineshops}(\text{shop-name}, \text{wine})$ is available. This query can be written as:

```sql
select top 6 shop-name from wineshops
  group by shop-name
  having set (wine) contains {Saint Emilion Grand Cru, Pomerol, Margaux} and if possible {Gewurztraminer Vendanges Tardives, Chablis Premier Cru} and if possible {Pommard, Chambertin},
```

and induces the scale $L = l_1 > l_2 > l_3 > l_4$.

We next move to the context of medical diagnosis, and we illustrate the use of subqueries to build the stratified divisor. Consider relation $\text{disease}(\text{name}, \text{symptom}, \text{frequency})$ which describes the symptoms associated with some diseases as well as the frequency with which a given symptom appears for a given disease, and relation $\text{patient}(\text{p-name}, \text{symptom})$ which gives the symptoms shown by some patients. The following stratified division query looks for the patients who have all of the 100% frequent symptoms of influenza, and if possible all of the symptoms whose frequency is above 80%, and if possible all of the symptoms whose frequency is above 50%:

```sql
select top 10 p-name from patient
  group by p-name
  having set (symptom) contains
```
Fuzzy Preference Queries to Relational Databases

\[(\text{select} \ \text{symptom} \ \text{from} \ \text{disease} \ \text{where} \ \text{name} = \text{“flu”} \ \text{and} \ \text{frequency} = 100)\]

and if possible

\[(\text{select} \ \text{symptom} \ \text{from} \ \text{disease} \ \text{where} \ \text{name} = \text{“flu”} \ \text{and} \ \text{frequency} \ \text{between} \ 80 \ \text{and} \ 99)\]

and if possible

\[(\text{select} \ \text{symptom} \ \text{from} \ \text{disease} \ \text{where} \ \text{name} = \text{“flu”} \ \text{and} \ \text{frequency} \ \text{between} \ 50 \ \text{and} \ 79).\]

Here also, the scale is \(L = l_1 > l_2 > l_3 > l_4\) since the divisor is composed of three strata.

7.4.2 The queries

The three types of queries, called conjunctive (CJ), disjunctive (DJ), and full-discrimination-based (FD), are the following:

- find the best \(k\) elements associated with \(S_1\) and if possible \(\ldots\) and if possible \(S_n\) (CJ),
- find the best \(k\) elements associated with \(S_1\) or else \(\ldots\) or else \(S_n\) (DJ),
- find the best \(k\) elements associated with \(S_1\) and-or \(\ldots\) and-or \(S_n\) (FD),

and it is assumed that the schema of relation \(r\) is \((A, X)\).

7.4.2.1 Conjunctive queries (CJ)

In CJ queries, to be qualified an element \(x\) must be connected with all the elements having the maximal importance (\(S_1\)). In this respect, this type of stratified query is strongly related to bipolar queries, a topic developed in Chapter 8. In addition, as soon as it ceases to be connected with all the elements of a set \(S_i\), its association with the values in any set \(S_{k+p}\) does not matter for its final ranking. An element \(x\) is all the more preferred as it is associated with all the values of the succession of sets \(S_1\) to \(S_i\) where \(i\) is large (if possible \(n\) for “perfection”). In other words, \(x\) is preferred to \(y\) if \(x\) is associated with all the values of the sets \(S_1\) to \(S_n\) and \(y\) is associated with a shorter list of sets. More formally, we denote:

\[I(x) = \{i \mid S_i \not\subseteq \Gamma_r^{-1}(x)\}\]

and if \(I(x) \neq \emptyset\), \(\text{imin}(x) = \min(I(x))\),
Division and Antidivision of Fuzzy Relations

where, as before, $\Gamma_r^{-1}(x)$ stands for the set of $A$-values associated with $x$ in relation $r$, i.e.:

$$\Gamma_r^{-1}(x) = \{a | (a, x) \in r\}.$$

The grade of satisfaction $sat(x)$ obtained by $x$ is expressed thanks to the scale $L$ stemming from the stratified divisor as follows:

$$sat(x) = \begin{cases} 
    l_1 & \text{if } I(x) = \emptyset, \\
    l_{n+2-i_{\min}(x)} & \text{otherwise}.
\end{cases} \quad (7.9)$$

By doing this, the satisfaction is seen to be a composition of the results of the division of the dividend with each of the layers of the divisor.

An alternative way of modeling CJ queries stems from the extension of Expression (7.3) by:

- dealing with the preferences applying to the divisor,
- using a symbolic/ordinal implication.

Indeed, we may use an augmented relational framework where each tuple of a relation $rel$ is assigned a (symbolic) level of preference taken from the scale $L$, denoted by $pref_{rel}(t)$, and any tuple can be written $pref_{rel}(t)/t$. Since the dividend relation is not concerned with explicit preferences, its tuples are assigned the maximal level $l_1$, while the tuples which are absent are (virtually) assigned the lowest level $l_{n+1}$. The level of preference attached to a tuple of the divisor is directly induced from the place of the corresponding element in the hierarchy provided by the user. The implication can be chosen from among the fuzzy implications (Fodor and Yager, 2000) provided that:

- it is compatible with an ordinal context,
- it conveys the semantics of importance associated with the layered divisor.

It turns out that the implication defined as:

$$s_1 \Rightarrow_o s_2 = \max(\text{rev}(s_1), s_2), \quad (7.10)$$

where $s_1, s_2$ are two symbols of an ordinal scale and $\text{rev}$ denotes order reversal, meets the above requirements. With the scale $L = l_1 > \cdots > l_{n+1}$, one has: $\forall i \in [1, n], \text{rev}(l_i) = l_{n+2-i}$. Thus equipped, if $V$ denotes the values
of the divisor, the stratified division is defined as follows:

\[
sat(x) = \min_{v \in V} \text{pref}_s(v) \Rightarrow_o \text{pref}_s(v, x).
\]

(7.11)

As a consequence, if \( x \) is associated with all the values of the entire divisor, the maximal level of preference \( l_1 \) is obtained, and as soon as an association \( <v, x> \) is missing in the dividend, the level of preference of \( x \) decreases all the more as \( v \) is highly preferred.

**Example 7.9.** Consider the divisor: \( \{a, b\} \succ c \succ \{d, e\} \) and the dividend relation:

\[
r = \{(a, x_1), (b, x_1), (c, x_1), (d, x_1), (a, x_2), (b, x_2),
\]
\[
(a, x_3), (b, x_3), (d, x_3), (e, x_3), (c, x_4), (b, x_5), (d, x_5)\}.
\]

One has: \( n = 3 \), \( I(x_1) = \{3\} \), \( imin(x_1) = 3 \) and, according to Expression (7.9), \( sat(x_1) = l_2 \); similarly, \( sat(x_2) = sat(x_3) = l_3 \), \( sat(x_4) = sat(x_5) = l_4 \), and the final result is: \( x_1 \succ \{x_2, x_3\} \).

Now, referring to Expression (7.11), one has:

\[
sat(x_1) = \min({\text{pref}}_s(a) \Rightarrow_o \text{pref}_s(a, x_1), \text{pref}_s(b) \Rightarrow_o \text{pref}_s(b, x_1),
\]
\[
\text{pref}_s(c) \Rightarrow_o \text{pref}_s(c, x_1), \text{pref}_s(d) \Rightarrow_o \text{pref}_s(d, x_1), \text{pref}_s(e) \Rightarrow_o \text{pref}_s(e, x_1)) = \min(l_1 \Rightarrow_o l_1, l_1 \Rightarrow_o l_1, l_2 \Rightarrow_o l_1, l_2 \Rightarrow_o l_3 \Rightarrow_o l_3 \Rightarrow_o l_4)
\]
\[
= \min(l_1, l_1, l_1, l_1, l_2) = l_2.
\]

Similarly, one has:

\[
sat(x_2) = \min(l_1, l_1, l_3, l_2, l_2) = l_3, sat(x_3) = \min(l_1, l_1, l_3, l_1, l_1) = l_3,
\]
\[
sat(x_4) = \min(l_4, l_4, l_3, l_2, l_1) = l_4, sat(x_5) = \min(l_4, l_4, l_3, l_1, l_2) = l_4.
\]

As expected, these results coincide with those obtained previously with Expression (7.9).\( \diamond \)

### 7.4.2.2 Disjunctive queries (DJ)

While CJ queries have a conjunctive behavior, DJ queries are intentionally disjunctive instead, and \( S_1 \) is no longer a mandatory subset. Here, the order of the subsets is taken according to the user’s preferences, so that element \( x \) is all the more preferred as it is connected with all the values of \( S_k \) and \( k \) is small (ideally 1 for “perfection”). In this case again, the associations with the subsets of higher index \( > k \), and then lower importance, do not play any role in the discrimination strategy. In other words, \( x \) is preferred
Division and Antidivision of Fuzzy Relations

If $x$ is associated with all the values of the set $S_k$ (and no $S_j$ with $j < k$) and $y$ is associated with $S_p$ (and no $S_m$ with $m < p$) and $p > k$. We denote:

$$I'(x) = \{i \mid S_i \subseteq \Gamma^{-1}_r(x)\}$$

The grade of satisfaction attached to $x$ is expressed as:

$$\text{sat}(x) = \begin{cases} l_{n+1} & \text{if } I'(x) = \emptyset, \\ l_{\text{ipmin}(x)} & \text{otherwise} \end{cases}$$

(7.12)

The satisfaction is still a combination of the results of the division of the dividend with each of the layers of the divisor.

**Example 7.10.** Consider the divisor: \{a, b\} \succ c \succ \{d, e\} and the dividend:

$$r = \{(a, x_1), (d, x_1), (c, x_1), (c, x_2), (d, x_2), (e, x_2), (a, x_3), (b, x_3), (c, x_3), (d, x_3), (e, x_3), (c, x_4), (b, x_5)\}.$$  

One has: $n = 3$, $I'(x_2) = \{2, 3\}$, $\text{ipmin}(x_2) = 2$, and $\text{sat}(x_2) = l_2$; similarly, $\text{sat}(x_1) = l_3$, $\text{sat}(x_3) = l_1$, $\text{sat}(x_4) = l_2$, $\text{sat}(x_5) = l_4$, and then the final result is: $x_3 \succ \{x_2, x_4\} \succ x_1$.

**7.4.2.3 Full discrimination-based queries (FD)**

Queries of type FD are designed to counter the common inability of CJ and DJ queries to distinguish between elements which are equally ranked because additional associations are not taken into account (e.g., $x_2$ and $x_4$ in the above example). Therefore, the principle when interpreting FD queries is to consider all the layers for which a complete association occurs. An element is all the more preferred as it is associated with a set $S_i$ highly preferred; this same point of view applies when breaking ties. In this case, the grade of satisfaction for $x$ may be expressed by using a vector $V(x)$ of dimension $n$, where $V(x)[i] = 1$ if $x$ is associated with all the values of $S_i$, 0 otherwise. Ordering the elements boils down to comparing such vectors according to the lexicographic order ($\succ \text{lex}$):

$$x \succ y \iff V(x) \succ \text{lex} V(y) \iff$$

$$\exists k \in [1, n] \text{ such that } \forall j < k, V(x)[j] = V(y)[j] \text{ and } V(x)[k] > V(y)[k].$$

In this view, the scale $L$ is not used directly even if the order of the elements of the vectors reflects it, in the sense that, if $i < j$, $V(x)[i]$ is more important than $V(x)[j]$ as $l_i > l_j$. It is, however, possible to use a scale to perform the
comparison of elements in the context of FD queries, although this scale is
not the initial one and is much larger ($2^n$ levels instead of $(n + 1)$ in the
original one). Consider the function which maps a vector $V$ onto an integer
score $s$ as follows:

$$sat(x) = \sum_{i=1,\ldots,n} V(x)[i] * 2^{n-i}. \quad (7.13)$$

It is straightforward to prove that the preference of $x$ over $y$ as defined
above is equivalent to $sat(x) > sat(y)$. In addition, from a calculus point
of view, dealing with such scores is easier than comparing vectors.

**Example 7.11.** Consider the divisor: \{a, b\} ≽ c ≽ \{d, e\} and the dividend:

$$r = \{(a, x_1), (d, x_1), (e, x_1), (c, x_2), (d, x_2), (e, x_2), (a, x_3), (b, x_3), (d, x_3), (c, x_4), (b, x_5)\}.$$  

One has:

$V(x_1) = (0, 0, 1), V(x_2) = (0, 1, 1), V(x_3) = (1, 0, 0), V(x_4) = (0, 1, 0), V(x_5) = (0, 0, 0).$

According to Expression (7.13):

$sat(x_1) = 1, sat(x_2) = 3, sat(x_3) = 4, sat(x_4) = 2, sat(x_5) = 0,$

and then the final result is: $x_3 ≽ x_2 ≽ x_4 ≽ x_1.$

**Relationship with the lexicographic order.** As has been indicated,
FD queries potentially use all the layers of the divisor in order to discrim-
inate between the elements returned by a query; they are also founded on
the lexicographic order. It turns out that the other two types of queries
can also be thought of in this way. Interpreting any CJ or DJ query can
be done through a vector accounting for the association with the values of
each complete stratum of the divisor, as is done for FD queries. For CJ
queries:

$sat(x) = l_{n-k+2}$ where $k$ is the smallest index such that $V(x)[k] = 0$
($k = n + 1$ if none).

In other words, the comparison of $x$ and $y$ can be based on a modified
vector $V'$ obtained from $V$ by propagating the first 0 to the right, which
Division and Antidivision of Fuzzy Relations

yields the equivalence:

\((\text{sat}(x) > \text{sat}(y)) \Leftrightarrow (V'(x) \succ_{\text{lex}} V'(y)).\)

Similarly, for DJ queries:

\(\text{sat}(x) = l_k\) where \(k\) is the smallest index such that \(V(x)[k] = 1\) \((k = n + 1\) if none).

Here also, the comparison of \(x\) and \(y\) can be based on a modified vector \(V''\) obtained from \(V\) by propagating the first 1 to the right, which yields:

\((\text{sat}(x) > \text{sat}(y)) \Leftrightarrow (V''(x) \succ_{\text{lex}} V''(y)).\)

**Example 7.12.** Consider again the data of Example 7.11, in particular the vectors \(V\) obtained there. From them, we obtain the modified vectors:

\[ V'(x_1) = (0, 0, 0), V'(x_2) = (0, 0, 0), V'(x_3) = (1, 0, 0), \]
\[ V'(x_4) = (0, 0, 0), V'(x_5) = (0, 0, 0), \]

which leads us to keep only \(x_3\) as the result of the conjunctive stratified query. Similarly, the modified vectors \(V''\) are:

\[ V''(x_1) = (0, 0, 1), V''(x_2) = (0, 1, 1), V''(x_3) = (1, 1, 1), \]
\[ V''(x_4) = (0, 1, 1), V''(x_5) = (0, 0, 0), \]

and the result of the disjunctive stratified query is: \(x_3 \succ \{x_2, x_4\} \succ x_1\). These two results are exactly those obtained with Expressions (7.9), (7.11) and (7.12).

**7.4.3 Quotient property of the result delivered**

We now show that the result delivered by the three types of queries is a quotient. We specify a characterization in the spirit of Expressions (7.1) and (7.2) for each type of stratified query, which says that the result obtained is a maximal relation (with respect to the satisfaction levels \((l_i's)\) assigned to its elements). For CJ queries, if it is assumed that \(x\) is assigned the grade \(l_i\) \((i \in [1, n])\) the following property holds:

\[ \forall k \in [1, n - i + 1], S_k \times \{x\} \subseteq r, \quad (7.14) \]
\[ \text{if } i > 1, S_{n-i+2} \times \{x\} \nsubseteq r. \quad (7.15) \]
In addition, any value $x$ which is not (at all) in the result (grade of satisfaction $l_{n+1}$) is such that: $S_1 \times \{x\} \not\subseteq r$, which expresses the fact that its grade cannot even be increased from $l_{n+1}$ to $l_n$.

**Example 7.13.** Consider the data of Example 7.9 where $n = 3$ and $sat(x_1) = l_2$. One has: $S_1 \times \{x_1\} = \{(a, x_1), (b, x_1)\} \subseteq r$, and $S_2 \times \{x_1\} = \{(c, x_1)\} \subseteq r$, whereas $S_3 \times \{x_1\} = \{(d, x_1), (e, x_1)\} \not\subseteq r$.

The same situation occurs for $x_2$.

Similarly, for $x_3$ one has: $S_1 \times \{x_3\} = \{(a, x_3), (b, x_3)\} \subseteq r$, whereas $S_2 \times \{x_3\} = \{(c, x_3)\} \not\subseteq r$.

Finally, for $i = 4$ and $5$: $S_1 \times \{x_i\} = \{(a, x_i), (b, x_i)\} \not\subseteq r$.

This illustrates the validity of Expressions (7.14) and (7.15).

**Example 7.14.** Consider the data of Example 7.10 where $n = 3$. One has: $sat(x_3) = l_1$ and $S_3 \times \{x_3\} = \{(a, x_3), (b, x_3)\} \subseteq r$, whereas both $S_2 \times \{x_3\} = \{(c, x_3)\} \not\subseteq r$ and $S_1 \times \{x_3\} = \{(a, x_3), (b, x_3)\} \not\subseteq r$.

Similarly, for $i = 2$ and $4$, $sat(x_i) = l_2$, and $S_2 \times \{x_i\} = \{(c, x_i)\} \subseteq r$ and $S_1 \times \{x_i\} = \{(a, x_i), (b, x_i)\} \not\subseteq r$.

For $x_3$, $sat(x_3) = l_1$ and $S_3 \times \{x_3\} = \{(a, x_3), (b, x_3)\} \subseteq r$.

Finally, $sat(x_3) = l_4$ and $S_4 \times \{x_3\} = \emptyset \subseteq r$ on the one hand, and $S_3 \times \{x_3\} = \{(d, x_3), (e, x_3)\} \not\subseteq r$, $S_2 \times \{x_3\} = \{(c, x_3)\} \not\subseteq r$, $S_1 \times \{x_3\} = \{(a, x_3), (b, x_3)\} \not\subseteq r$ on the other hand.

It turns out that Expressions (7.16) and (7.17) hold.

**Example 7.15.**

Turning to FD queries, recall that the grade of satisfaction of $x$ is simply expressed as a function of the values of the vector $V$ stating whether $x$ is connected or not with all the values of the different layers of the divisor...
Division and Antidivision of Fuzzy Relations

(see Expression (7.13)). The following property holds:

\[ \forall i \in [1, n] \text{ such that } V(x)[i] = 1, \quad S_i \times \{x\} \subseteq r, \]  
(7.18)

\[ \forall i \in [1, n] \text{ such that } V(x)[i] = 0, \quad S_i \times \{x\} \not\subseteq r, \]  
(7.19)

which means that if the grade \( sat(x) \) is increased, some inclusion constraint(s) of the type seen in Expression (7.18) will be violated.

**Example 7.15.** Consider again Example 7.11. \( V(x_1) = (0, 0, 1) \) and \( S_1 \times \{x_1\} = \{\langle a, x_1 \rangle, \langle b, x_1 \rangle\} \not\subseteq r \), \( S_2 \times \{x_1\} = \{\langle c, x_1 \rangle\} \subseteq r \), and \( S_3 \times \{x_1\} = \{\langle d, x_1 \rangle, \langle e, x_1 \rangle\} \subseteq r \).

\[ V(x_2) = (0, 1, 1) \text{ and } S_1 \times \{x_2\} = \{\langle a, x_2 \rangle, \langle b, x_2 \rangle\} \not\subseteq r, \quad S_2 \times \{x_2\} = \{\langle c, x_2 \rangle\} \not\subseteq r, \quad S_3 \times \{x_2\} = \{\langle d, x_2 \rangle, \langle e, x_2 \rangle\} \subseteq r. \]

\[ V(x_3) = (1, 0, 0) \text{ and } S_1 \times \{x_3\} = \{\langle a, x_3 \rangle, \langle b, x_3 \rangle\} \subseteq r, \quad S_2 \times \{x_3\} = \{\langle c, x_3 \rangle, \langle e, x_3 \rangle\} \not\subseteq r. \]

\[ V(x_4) = (0, 1, 0) \text{ and } S_1 \times \{x_4\} = \{\langle a, x_4 \rangle, \langle b, x_4 \rangle\} \not\subseteq r, \quad S_2 \times \{x_4\} = \{\langle c, x_4 \rangle\} \not\subseteq r, \quad S_3 \times \{x_4\} = \{\langle d, x_4 \rangle, \langle e, x_4 \rangle\} \subseteq r. \]

\[ V(x_5) = (0, 0, 0) \text{ and } S_1 \times \{x_5\} = \{\langle a, x_5 \rangle, \langle b, x_5 \rangle\} \not\subseteq r, \quad S_2 \times \{x_5\} = \{\langle c, x_5 \rangle\} \not\subseteq r, \quad S_3 \times \{x_5\} = \{\langle d, x_5 \rangle, \langle e, x_5 \rangle\} \subseteq r. \]

This is an illustration of the fact that Expressions (7.18) and (7.19) hold.

### 7.5 Queries Mixing Division and Antidivision

#### 7.5.1 Motivation

Most of what has been said about extending the division operator also holds for the antidivision operator, i.e., it is possible to define the antidivision of fuzzy relations, a tolerant antidivision and a stratified antidivision. Moreover, in each case the result obtained can be characterized as an anti-quotient (see Section 2.2.2). For instance, consider the case of a consumer who wants food products (e.g., noodles or vegetable oil) without certain additive substances. In the presence of the relation \( Products(p-name, add-s) \) describing which additives \( add-s \) are involved in products, a possible antidivision query is to search for products from which some prioritized...
substances are absent:

```sql
select top 6 p-name from Products group by p-name
having set(add-s) contains-none {'AS27', 'BT12', 'C3'}
and if possible none {'AS5', 'D2'} and if possible none {'D8'}.
```

The keyword “contains-none” is the counterpart of “contains” in a stratified division query. \( F \text{ contains-none } E \) returns true if and only if \( \forall x \in X, x \in E \Rightarrow x \notin F \). As with stratified division queries, an ordinal scale stems from the hierarchical divisor specified by the user.

Now consider a user who would like to retrieve elements both connected with more or less necessary values and not connected with some more or less rejected values. Such queries constitute a new family of queries of interest which have both a stratified division component and an antidivision component. Although this new family is applicable to disjunctive and/or full discrimination stratified queries, we only deal with conjunctive queries in the rest of this section.

### 7.5.2 Mixed stratified queries

Here, we start with a stratified division or antidivision of a relation \( r \) whose schema is \( (A, X) \) by a relation \( s \) defined over \( B \) with \( A \) and \( B \) compatible (sets of) attributes. The scale \( L \) derived from the stratified divisor is used to attach a preference level to each value \( a \) of the divisor \( s \), denoted by \( \text{pref}_s(a) \). Since the dividend relation \( r \) is not concerned with preferences, its tuples are assigned the maximal level \( l_1 \) (if necessary, tuples absent from it receive the lowest level \( l_n+1 \)). Letting \( V \) denote the values of the divisor \( s \), the definition of stratified division and antidivision is as follows:

\[
\text{pref}_{\text{strat-div}}(r, s, A, B)(x) = \min_{v \in V} \text{pref}_s(v) \Rightarrow \text{pref}_r(v, x)) = \min_{v \in V} \max(\text{rev(pref}_s(v)), \text{pref}_r(v, x))
\]

\[
\text{pref}_{\text{strat-antidiv}}(r, s, A, B)(x) = \min_{v \in V} \text{pref}_s(v) \Rightarrow \text{rev(pref}_r(v, x)) = \min_{v \in V} \max(\text{rev(pref}_s(v)), \text{rev(pref}_r(v, x))).
\]

As a consequence, if \( x \) is associated with all (resp. none) of the values of the entire divisor, the maximal level of preference \( l_1 \) is obtained, and as soon as an association \( (v, x) \) is missing (resp. found) the level of preference of \( x \) decreases all the more as \( v \) is highly preferred (resp. unwanted).
A starting view of mixed stratified queries would be to consider them as made of two components, one “positive” expressing a stratified division, another “negative” issued from an antidivision, according to the following pattern:

\[
\text{select top } k \ X \ \text{from } r \ \text{where condition} \\
\text{group by } X \\
\text{having set}(A) \ \text{contains} \\
\{v_{1,1}, \ldots, v_{1,j_1}\} \ \text{and if possible} \ldots \ \text{and if possible} \{v_{n,1}, \ldots, v_{n,j_n}\} \\
\text{and contains-none} \\
\{w_{1,1}, \ldots, w_{1,k_1}\} \ \text{and if possible none} \ldots \ \text{and if possible none} \\
\{w_{p,1}, \ldots, w_{p,k_p}\}.
\]

This means that the query refers to two scales: \(L_1 = l_1 > \cdots > l_{n+1}\) for the positive part, and \(L_2 = l'_1 > \cdots > l'_{p+1}\) for the negative part. The overall satisfaction of a given \(x\) would then require the combination of two symbols (one from each scale), which raises a serious problem.

To overcome this difficulty, a solution is to build mixed queries in such a way that only a single scale ever comes into play. Each level of the scale used in a query will be assigned a set of desired values (contributing the positive part) and a set of unwanted values (the negative part), one of them being possibly empty. A mixed stratified query is then expressed according to the following pattern:

\[
\text{select top } k \ X \ \text{from } r \ \text{where condition} \\
\text{group by } X \\
\text{having set}(A) \ \text{contains} [\text{pos: } \{v_{1,1}, \ldots, v_{1,j_1}\}, \text{neg: } \{w_{1,1}, \ldots, w_{1,k_1}\}] \\
\text{and if possible} \ldots \ \text{and if possible} \\
[\text{pos: } \{v_{n,1}, \ldots, v_{n,j_n}\}, \text{neg: } \{w_{n,1}, \ldots, w_{n,k_n}\}]
\]

where “pos: \(S\)” (resp. “neg: \(S\)”) denotes the positive (resp. negative) part associated with a set of desired (resp. unwanted) values at a given level of importance (layer), with which \(x\) must be (resp. not be) associated. In addition, note that it is possible to have “pos: \{\}\” or “neg: \{\}\”, i.e. an empty positive or negative set, but not both, at each layer. Moreover, the absence of an element of the positive part or the presence of an element of the negative part leads to the non-satisfaction of the associated layer. Obviously, the scale behind this query is \(L = l_1 > \cdots > l_{n+1}\).
The above type of query can be interpreted in a straightforward manner as follows. To be somewhat satisfactory, element $x$:

- must be associated with all the values $\{v_{1,1}, \ldots, v_{1,j}\}$ and none of the values $\{w_{1,1}, \ldots, w_{1,k}\}$,
- receives a level of satisfaction ($\text{pref}$) all the larger as it satisfies the association with all the values $\{v_{2,1}, \ldots, v_{2,k_2}\}, \ldots, \{v_{j,1}, \ldots, v_{j,k_j}\}$ and none of the values of $\{w_{2,1}, \ldots, w_{2,p_2}\}, \ldots, \{w_{j,1}, \ldots, w_{j,p_j}\}$, with $j$ taking a high value ($j = n$ for the maximal level $l_1$).

In other words, $x$ is preferred to $y$ if $x$ is connected with $\{v_{1,1}, \ldots, v_{1,k_1}\}, \ldots, \{v_{1,1}, \ldots, v_{1,k}\}$ and none of $\{w_{1,1}, \ldots, w_{1,p_1}\}, \ldots, \{w_{1,1}, \ldots, w_{1,p}\}$, while $y$ is associated with $\{v_{1,1}, \ldots, v_{1,k_1}\}, \ldots, \{v_{j,1}, \ldots, v_{j,k_j}\}$ and none of $\{w_{1,1}, \ldots, w_{1,p_2}\}, \ldots, \{w_{j,1}, \ldots, w_{j,p_j}\}$ and $i > j$. Formally, we denote by $s = \{V_1, \ldots, V_n\}$ the different layers of the divisor where each $V_i$ is made of a positive part $P_i$ and a negative part $N_i$. Alternatively, $s$ can be written as $(P, N)$, its positive and negative parts. The mixed stratified division is defined as:

$$
\text{pref}_{\text{mix-strat-div}}(r, s, A, B)(x) = \min_{i \in [1, n]} \min_{\{v\}, \text{pref}_s(v)} \Rightarrow_{A} \text{pref}_r(v, x),
$$

$$
\min_{w \in N_i, \text{pref}_s(w)} \Rightarrow_{A} \text{rev}(\text{pref}_r(w, x))
$$

$$
= \min_{i \in [1, n]} \min_{\{v\}, \text{pref}_s(v)} \max(\text{rev}(l_i), \text{pref}_r(v, x)),
$$

$$
\min_{w \in N_i, \text{max}(\text{rev}(l_i), \text{rev}(\text{pref}_r(w, x))))
$$

$$
= \min(\min_{v \in P} \max(\text{rev}(\text{pref}_s(v)), \text{pref}_s(v, x)),
$$

$$
\min_{w \in N} \max(\text{rev}(\text{pref}_s(w)), \text{rev}(\text{pref}_s(w, x)))).
$$

By construction, the result delivered by the above operation is a quotient, in the sense that it is a maximal relation.

**Example 7.16.** The following query in the context of non-weighted document retrieval illustrates the idea of mixed queries:

```sql
SELECT top 4 title FROM books WHERE year > 2007 GROUP BY title
HAVING SET(terms) CONTAINS [pos: {t1, t2, t3}, neg: {t4, t5}]
AND IF POSSIBLE [pos: {t6}, neg: {}] AND IF POSSIBLE [pos: {}, neg: {t7}].
```

Here, the user wants books about $t_1$, $t_2$, and $t_3$, but neither $t_4$ nor $t_5$; in addition, dealing with $t_6$ is appreciated and to a lesser extent not dealing
Division and Antidivision of Fuzzy Relations

with \( t_7 \). In the presence of the following documents:

- \( d_1 \) indexed by \( t_1, t_2, t_6, t_{12}, t_{23} \),
- \( d_2 \) indexed by \( t_1, t_2, t_4, t_5, t_{10}, t_{14} \),
- \( d_3 \) indexed by \( t_1, t_2, t_3, t_4, t_{13}, t_{20}, t_{21} \),
- \( d_4 \) indexed by \( t_1, t_2, t_3, t_7, t_{11} \),
- \( d_5 \) indexed by \( t_1, t_2, t_3, t_6, t_7, t_{15} \),
- \( d_6 \) indexed by \( t_1, t_2, t_3, t_6, t_8, t_9 \)

the degrees of satisfaction (\( \text{pref} \)) assigned to these documents are:

\[
\text{pref}(d_1) = \min(\min(l_1, l_1, l_1, l_1), \min(l_1, l_1, l_1)) = l_4,
\]

\[
\text{pref}(d_2) = \min(\min(l_1, l_1, l_1, l_1), \min(l_4, l_4, l_1)) = l_4,
\]

\[
\text{pref}(d_3) = \min(\min(l_1, l_1, l_1, l_3), \min(l_4, l_1, l_1)) = l_4,
\]

\[
\text{pref}(d_4) = \min(\min(l_1, l_1, l_1, l_3), \min(l_1, l_1, l_2)) = l_3,
\]

\[
\text{pref}(d_5) = \min(\min(l_1, l_1, l_1, l_1), \min(l_1, l_1, l_2)) = l_2,
\]

\[
\text{pref}(d_6) = \min(\min(l_1, l_1, l_1, l_1), \min(l_1, l_1, l_1)) = l_1,
\]

which leads to the final result: \( d_6 \gg d_5 \gg d_4 \), since the first three documents are totally unsatisfactory (term \( t_3 \) is missing in documents \( d_1 \) and \( d_2 \), whereas document \( d_3 \) contains the definitely rejected term \( t_4 \)).

7.6 Evaluation of Division Queries

We first present strategies for processing the division over fuzzy relations. Subsequently, the evaluation of tolerant and stratified division queries is examined.

7.6.1 Processing the division of fuzzy relations

In Bosc et al. (1999b), three methods for the processing of regular (i.e., non-tolerant) division of fuzzy relations have been devised and compared:

- translation of the original division query into an SQL query involving a comparison of cardinalities, and a user-defined function to calculate the satisfaction degrees,
- translation of the original division query into an SQL query involving an inclusion, and a user-defined function to calculate the satisfaction degrees,
- compilation of the original division query into a processing algorithm encoded in a procedural language such as Pro*C or PL/SQL.
It appears that the third technique is by far the most efficient. For example, some experimental measures we have made, using the Oracle DBMS, show that for a dividend relation made of 60,000 tuples and a divisor made of 1,000 tuples, the processing time is 62 s for the cardinality-based approach, 36 s for the inclusion-based approach, and only 18 s for the approach based on a compiled algorithm. Another interesting result is that a similar division query addressed to crisp (instead of fuzzy) relations takes 15 s instead of 18 s to be processed. The additional cost introduced by the fuzziness of the relations under study is thus only around 20%. Hereafter, we describe the principle of the algorithm implementing the division of relation $r$ of schema $R(A, X, \mu)$ by relation $s$ of schema $S(B, \mu)$, where $\mu$ denotes in both cases the membership degree associated with each tuple. Recall that the evaluation of the division is based on Expression (7.3), i.e.:

$$\forall x \in \text{project}(\text{supp}(r, X)), \mu_{\text{div}}(r, s, A, B)(x) = \min_s(\mu_s(a) \Rightarrow \mu_r(a, x)).$$

The idea is to use two nested loops. The first loop scans the different $X$-values present in relation $r$. For a given $x$, the inner loop scans the $B$-values $b$ in the divisor, checks by means of a selection query whether $\langle b, x \rangle$ is in the dividend (and if so, with which degree), and updates the satisfaction degree associated with $x$ in the result. In the algorithm, “impl” ($\Rightarrow$ in the above formula) denotes the fuzzy implication underlying the division (recall that either R-implications or S-implications are possible, which makes the division a parameterized operator).

If $r$ (resp. $s$) is the fuzzy dividend (resp. divisor) relation and $\lambda$ is the user-specified threshold (if the user does not specify one, $\geq \lambda$ can be replaced by $> 0$), we obtain Algorithm 7.1 as follows:

Algorithm 7.1.

define $cursX = \text{select distinct } X \text{ from } r$;
define $cursB = \text{select } \mu, B \text{ from } s$;
$\text{div} \leftarrow \emptyset$;
for all $x$ in $cursX$ do
begin
val_imp $\leftarrow 1$;
for all $\langle \mu_1, b \rangle$ in $cursB$ and val_imp $> 0$ do
select $\mu$ into $\mu_2$ from $r$ where $X = x$ and $A = b$;
— it is assumed that if $\langle b, x \rangle$ is not in $r$, $\mu$ takes the value 0 —
val_imp $\leftarrow \min(\text{val_imp}, \text{impl(}\mu_1, \mu_2\))$;
end;
if val_imp $\geq \lambda$ then $\text{div} \leftarrow \text{div} \cup \{\langle\text{val_imp}/x\rangle\}$ endif;
Division and Antidivision of Fuzzy Relations

end;
rank-order(div) according to the satisfaction degrees \( \text{val} \text{imp} \).

7.6.2 Processing the tolerant divisions of fuzzy relations

The method associated with Algorithm 7.1 serves as the basis for implementing the various tolerant divisions.

7.6.2.1 Quantitative exception-tolerant division

In order to deal with this type of tolerant division, this algorithm has to go through a few modifications. First, it is no longer possible to calculate \( \text{val} \text{imp} \) “incrementally” since the implication degrees attached to a given \( X \)-value \( x \) must be ranked in order to apply the weights \( w_i \) (see Section 7.3.1) appropriately. Consequently, it is necessary to store the different implication degrees attached to a given \( x \). As for the weights, they depend on the cardinality of the divisor and on the fuzzy quantifier “almost all”. Therefore, they can be computed before entering the outer loop of the previous algorithm. Considering that \( \text{val} \text{imp} \) can no longer be computed incrementally, it is not possible to stop the inner loop as soon as one obtains a zero implication degree (for a certain value \( b \) from \( s \)). However, it is still possible in some cases to stop that loop before scanning \( s \) entirely. We denote by \( k \) the number of \( w_i \)'s greater than zero. As soon as more than \( k \) implication values equal to zero have been found (for a given \( x \)), one can conclude that \( x \) does not belong to the result. In the case where the user specifies a threshold \( \lambda \) (\( \lambda > 0 \)), this pruning criterion can be refined in the following way. Let \( k' \) denote the number of \( w_i \)'s greater than or equal to \( \lambda \). As soon as more than \( k' \) implication degrees lower than \( \lambda \) are found (for a given \( x \)), one can conclude that \( x \) is not an answer. Compared to the reference algorithm suited to the classical division of fuzzy relations, the additional cost is related to: i) the necessity of scanning relation \( s \) once in order to compute its cardinality and then the weights \( w_i \), ii) the fact that the inner loop cannot be stopped as often. The pruning condition is not as favorable in the tolerant case, but this does not change the nature of the worst case that can be encountered (and thus the maximal complexity of the algorithm).

7.6.2.2 Qualitative exception-tolerant division

Here, the impact on Algorithm 7.1 is more limited. One just has to replace the calls \( \text{impl}(\mu_1, \mu_2) \) with \( \text{relaxed_impl}(\mu_1, \mu_2) \) which takes the thresholds \( \alpha \) and \( \beta \) (see Section 7.3.1) into account. The extra cost is thus practically
negligible, since the number of accesses to the relevant relations remains the same.

7.6.2.3 Resemblance-based tolerant division

Let $\text{Prox}(A_1, A_2, \mu)$ denote the proximity relation used (it is assumed that this relation is stored in the database). The basic idea is to retrieve, for a given $x$ of the dividend $r$ and a given $b$ of the divisor $s$, the best substitute $b'$ for $b$ (in the sense of $\text{Prox}$), i.e., the value $b'$ such that:

$$\min(\mu_{\text{Prox}}(b', b), \mu_r(b', x)) = \sup_{h \in \text{domain}(B)} \min(\mu_{\text{Prox}}(h, b), \mu_r(h, x)).$$

The processing algorithm becomes:

**Algorithm 7.2.**

1. define $\text{cursX} = \text{select distinct } X \text{ from } r$;
2. define $\text{cursB} = \text{select } \mu, B \text{ from } s$;
3. $\text{div} \leftarrow \emptyset$;
4. for all $x$ in $\text{cursX}$ do
   1. $\text{val}_\text{imp} \leftarrow 1$;
   2. for all $\langle \mu_1, b \rangle$ in $\text{cursB}$ and $\text{val}_\text{imp} > 0$ do
      1. $\mu_2 \leftarrow$ highest degree attached to $\langle x, b' \rangle$ in $r$ where $b'$ is a possible substitute for $b$;
      2. $\text{val}_\text{imp} \leftarrow \min(\text{val}_\text{imp}, \text{impl}(\mu_1, \mu_2))$
   enddo;
5. if $\text{val}_\text{imp} \geq \lambda$ then $\text{div} \leftarrow \text{div} \cup \langle \text{val}_\text{imp}/x \rangle$ endif;
6. enddo;
7. rank-order($\text{div}$) according to the satisfaction degrees $\text{val}_\text{imp}$.

The computation of the highest degree attached to $\langle x, b' \rangle$ in $r$ where $b'$ is a possible substitute for $b$ is performed the following way:

**Algorithm 7.3.**

1. define $\text{cursP} = \text{select } \mu, A_1 \text{ from } \text{Prox} \text{ where } A_2 = b \text{ and } \mu > 0$;
2. $d \leftarrow 0$;
3. for all $\langle \mu, a_1 \rangle$ in $\text{cursP}$ do
   1. select $\mu'$ into $\mu'$ from $r$ where $X = x$ and $A = a_1$;
      1. it is assumed that if $\langle a_1, x \rangle$ is not in $r$, $\mu$ takes the value zero —
      2. $d \leftarrow \max(\min(\mu, \mu'), d)$;
   enddo;
4. return $d$. 
As an improvement, the definition of $cursP$ can be replaced by the following sequence:

```sql
select mu into µ from r where X = x and A = b;
define cursP = select µ, A1 from Prox where A2 = b and µ > µ;
```

Indeed, for a given $b$, only the elements which are close to $b$ at a degree higher than the degree of membership of $(x, b)$ to $r$ can possibly upgrade the final satisfaction degree attached to $x$.

The extra cost is due to the fact that, for each $(x, b)$, one has to scan a subset of the possible substitutes to $b$. Let $k_b$ denote the cardinality of that subset. One performs $k_b$ selections on $r$ instead of one in the inner loop (plus the $k_b$ accesses to $Prox$ associated with $b$). The extra cost is thus equal to:

$$n \times m \times \left( 2 \sum_{b \in \text{project}(s, B)} k_b - 1 \right)$$

where $n$ (resp. $m$) denotes the cardinality of $r$ (resp. $s$). This extra cost is not negligible and can even be seriously penalizing if the number of substitutes for each element of the divisor is high.

### 7.6.3 Processing the conjunctive stratified division

We now tackle implementation issues related to stratified divisions. We focus on CJ queries which give a good flavor of what can be done in this domain. The objective is to suggest some algorithms which are suited to a reasonably efficient evaluation of such queries, and to assess the extra cost with respect to queries involving no preferences.

#### 7.6.3.1 Principle of the algorithms

Three algorithms implementing Expression (7.11) are described in turn. The first algorithm, called SSD, is based on a sequential scan of the dividend relation $r$. The idea is to access the tuples from the dividend “in gusts”, i.e., by series of tuples which share the same $X$-attribute value (in the spirit of what is performed by a “group by” clause). Moreover, inside a cluster the tuples $(x, a)$ are ordered increasingly on $A$. This is performed by the query:

```sql
select * from r order by X, A.
```
Thanks to a table which gives, for each value \((val-A)\) of the divisor, the layer to which it belongs \((str-A)\), one can update the number of values from each layer which are associated with the current element \(x\), while scanning the result of the query above. At the end of a group of tuples, the layers are checked in decreasing order of their importance. The process stops as soon as the current element \(x\) is no longer associated with all of the values from a layer \(S_i\). Three cases can occur:

- \(x\) is associated with all of the values from all the layers of the divisor and it gets the preference level \(l_1\),
- the stop occurs while checking layer \(S_i\) whose importance is not maximal \((i > 1)\) and \(x\) is assigned the preference level \(rev(l_i) = l_{n+2-i}\),
- the stop occurs while checking layer \(S_1\) and \(x\) is assigned the level \(l_{n+1}\), meaning that it is rejected.

In the second algorithm, called AGD, data accesses are guided by the divisor relation \(s\). Thus, instead of scanning the dividend exhaustively and then checking the layers satisfied by a given \(x\) by means of the aforementioned table, the \(X\)-values are first retrieved from the dividend, and for each such \(x\), the associations with the different layers are checked by means of an SQL query involving the aggregate \(count\). Again, a layer is checked only if the layers of higher importance had all of their values associated with \(x\). The first step is to retrieve the distinct values of attribute \(X\) present in \(r\) by means of the query:

\[
\text{select distinct } X \text{ from } r.
\]

Then, for each value \(x\) returned, the \(A\)-values from \(S_1\) which are associated with \(x\) in \(r\) are counted, by means of the query:

\[
\text{select count(*) from } r \text{ where } X = x \text{ and } A \text{ in (select } A \text{ from } S_1)\text{.}
\]

If the value returned equals the cardinality of \(S_1\), layer \(S_2\) is checked by means of a similar query, and so on. The loop stops as soon as a missing association with the current layer is detected. The preference level assigned to \(x\) is computed according to the same principle as in the previous algorithm.

The last strategy, called SRD, relies on a series of regular division queries. It consists of two steps:

- process as many regular division queries as there are layers in the divisor,
- merge the different results and compute the final preference degrees.
This algorithm has the following general shape:

(1) for each layer $S_i$ of the divisor, process a division query which retrieves the $x$'s which are associated in $r$ with all of the values from $S_i$. The layers are examined in decreasing order of importance, and an element $x$ is checked only if it belongs to the result of the query related to the previous layer.

(2) the results $T_1, \ldots, T_n$ of the previous division queries are merged by taking them in decreasing order of the corresponding layers. An element $x$ which belongs to $T_i$ (the result of layer $S_i$) but not to $T_{i+1}$ is assigned the preference level $l_{n-i+1}$ (assuming an empty table $T_{n+1}$). In the queries of this step, the algorithm used relies on an outer join.

7.6.3.2 Experiments and results

The main objective of the experimentation is to assess the additional processing cost related to the handling of preferences, and to compare the performances of the algorithms presented above. The experimentation was performed with the DBMS Oracle™ Enterprise Edition Release 8.0.4.0.0 running on an Alpha server 4000 bi-processor with 1.5 GB memory. A generic stratified division query was run on dividend relations of 300, 3,000 and 30,000 tuples, and a divisor including five layers made of respectively 3, 2, 1, 2 and 2 values. The query taken as a reference was the analogous division query without preferences, where the divisor is made of the sole first layer of the divisor. By doing this, we were able to assess the extra cost related only to the “preference part” of the query, i.e., to the presence of the non-mandatory layers. The reference division query was evaluated using three methods:

- sequential scan of the dividend (i.e., algorithm SSD without preferences, denoted by REF1),
- access guided by the divisor (i.e., algorithm AGD without preferences, denoted by REF2),
- algorithm REF3 based on a query involving a “group by” clause and a counting, as in the first step of algorithm SRD.

Moreover:

- we used synthetic data generated in such a way that the selectivity of each value $b$ from the divisor relative to any $x$ from the dividend was equal to 75% in the case of a division query (for a given value $b$ from the
Table 7.1 Results from the division experiments of Section 7.6.3.2

<table>
<thead>
<tr>
<th>Size of the dividend</th>
<th>300</th>
<th>3,000</th>
<th>30,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>REF1</td>
<td>15.8</td>
<td>144.6</td>
<td>1451</td>
</tr>
<tr>
<td>REF2</td>
<td>49.7</td>
<td>570</td>
<td>15536</td>
</tr>
<tr>
<td>REF3</td>
<td>11.4</td>
<td>40.5</td>
<td>361.9</td>
</tr>
<tr>
<td>SSD</td>
<td>99</td>
<td>1011</td>
<td>10451</td>
</tr>
<tr>
<td>AGD</td>
<td>84</td>
<td>1035</td>
<td>29927</td>
</tr>
<tr>
<td>SRD</td>
<td>89</td>
<td>332</td>
<td>2923</td>
</tr>
<tr>
<td>Number of answers</td>
<td>15</td>
<td>172</td>
<td>1693</td>
</tr>
</tbody>
</table>

divisor and a given $x$ from the dividend, tuple $\langle x, b \rangle$ had three chances out of four to be present in the dividend,

- each algorithm was run eight times so as to avoid any bias introduced by the load of the machine,
- the time unit equals $1/60$ second.

The results obtained for the division are reported in Table 7.1. It can be seen that:

- among the reference methods for non-stratified operations, by far the most efficient is REF3. This is because it is based on a single query involving a “group by” clause, which is very efficiently optimized by the system,
- the processing time of the algorithms based on a sequential scan of the dividend (i.e., SSD and REF1) vary linearly with respect to the size of the dividend, in contrast to those from the second family (REF2 and AGD); in the case of algorithm SRD (implemented with an outer join), its complexity shows some linearity as soon as the size of the dividend is above a certain threshold (which means that there is a fixed cost attached to it, which depends on the number of layers of the divisor),
- algorithm SSD becomes better than AGD as soon as the size of the dividend is over 1,000 tuples. But the best algorithm is SRD (implemented with an outer join), which outperforms all the others as soon as the dividend contains more than 300 tuples. It is worth noting that the ratio between SRD and REF3 is almost constant (around eight), which is due to the fact that SRD performs one query of type REF3 per layer (of which there are five in this case), plus the combination of the intermediate results.
In summary, it appears that algorithm SRD based on an outer join is the most efficient, except for very small sizes of dividend where AGD is slightly better. However, the extra cost of SRD with respect to the most efficient reference algorithm, namely REF3, is still significant (the multiplicative factor is around eight). What all these measures show is somewhat predictable: the best way to process a division (stratified or not) is to express it by means of a single query that can be efficiently handled by the optimizer of the system, and not by external programs which usually introduce a significant overhead. Consequently, if stratified division functionality were to be integrated into a commercial DBMS, it is clear that it would have to be handled by the optimizer at an internal level, and processed as one query, according to the format given in Section 7.4.2.1 and in such a way that the evaluation of a given \( x \) is done in one step.

7.7 Conclusion

This chapter has been devoted to the division operation, which turns out to be rich from a semantic point of view. The original operator of the relational algebra is intended to retrieve those values of a binary dividend relation that are connected with at least all the elements of another relation called divisor. Its extension in order to introduce preferences (and therefore graduality) has been considered according to various points of view:

- the operands become fuzzy (graded) relations and, depending on the type of fuzzy implication used to generalize the non-fuzzy operator, the division operator obtained conveys different meanings,
- some tolerance is introduced into the checking of the connection between elements of the divisor and dividend relations; here also, diverse semantics are available depending on whether the tolerance is of a quantitative or qualitative nature, or is a consequence of some similarity between the values of the domain serving to pair dividend and divisor relations,
- the divisor is stratified, i.e., it is composed of diverse subsets corresponding to values more or less required/desired; in this context, queries calling on both a positive and negative divisor, i.e., an ordered list of desired and unwanted elements, have been proposed, thus mixing division and antidivision operations.

We have focused on “elementary” extensions; one might think of composing some of them, for instance both softening the universal quantifier and having a stratified divisor.
Two central questions have been dealt with for the diverse approaches to extension: i) the quotient property of the delivered result, which makes it legitimate to call the extended operator a division, and ii) the rewriting of the extended operator in terms of primitive operations as is the case for the original division.

Incidentally, note the relationship between the division of fuzzy relations and quantified statements of the form “\(QXB\) are \(A\)”, where \(Q\) is a quantifier, \(X\) is a regular set of elements, and \(A, B\) are fuzzy predicates. Indeed, to find the extent to which “each store has ordered almost all cheap products in a moderate quantity” corresponds to evaluating the proposition “almost all cheap products are ordered in a moderate quantity”.

Finally, it is worth mentioning that the query/document matching process in information retrieval is based on the inclusion operation, as is the case for the division (see Expressions (2.1) and (2.2)). In Bosc et al. (2009b), the authors investigate the use of a graded inclusion (see Section 3.4.2.3) in textual information retrieval. In this context, documents and queries are represented by fuzzy sets, which are matched using a tolerant inclusion based on fuzzy implications and triangular norms. Some insights on the practical use of this graded inclusion in information retrieval are provided. Experiments show that, with appropriate settings (fuzzy operators and weighting schemes), it is possible for the fuzzy-set-based model to mimic classical systems and to provide results close to those returned by state-of-the-art systems.
Chapter 8

Bipolar Fuzzy Queries

8.1 Introduction

A complementary concept to flexible preferences is that of bipolarity, and especially its application to queries in the context of databases and information systems. Bipolarity refers to the propensity of the human mind to reason and make decisions on the basis of positive and negative effects (Dubois and Prade, 2002, 2008b). Positive statements express what is possible, satisfactory, permitted, desired, or considered as being acceptable. Negative statements, however, express what is impossible, rejected, or forbidden (De Tré et al., 2010). Negative preferences correspond to constraints, since they specify which values or objects have to be rejected (i.e., those that do not satisfy the constraints), while positive preferences correspond to wishes, as they specify which objects are more desirable than others (i.e., satisfy user wishes) without rejecting those that do not meet the wishes.

Several types of bipolarity have been described (Dubois and Prade, 2008a). In this chapter, asymmetric (also called heterogeneous) bipolarity is considered, where positive and negative poles refer to potentially different notions (attributes). More precisely, we deal with queries made up of two poles, one meant as a constraint — denoted by $C$ — and the other acting as a wish — denoted by $W$ — where a pair $(C, W)$ is interpreted as: “$C$ and if possible $W$”. In the situation considered later on, the two components of a query, although they can be assessed on the same scale (true/false or the unit interval), are not of the same nature, and it is convenient to specify how any pair of elements (tuples or objects) are compared depending on their scores with respect to the constraint and the wish. Recall, however, that in a complex constraint (resp. wish), the fuzzy-set-based approach requires the elementary preferences to be commensurable.
A commonly made choice (see in particular Dubois and Prade, 2008a) for interpreting a bipolar condition consists of discriminating between two objects \( x \) and \( y \) using first the constraint, and then, if needed (i.e., if \( x \) and \( y \) are not distinguishable on the constraint), using the wish. In this chapter, this point of view is chosen and a lexicographic order is used. If \((C(x), W(x))\) and \((C(y), W(y))\) denote the scores of \( x \) and \( y \) with respect to the constraint \( C \) and the wish \( W \), one has:

\[
 x \succ y \Leftrightarrow (C(x) > C(y)) \text{ or } (C(x) = C(y) \text{ and } W(x) > W(y))
\]

(8.1)

where \( x \succ y \) means that \( x \) is preferred to \( y \). A consequence is that an object which is beaten on the constraint cannot win even if it is much better on the wish.

In this chapter, we describe an extension of relational algebra (Bosc et al., 2010a) which provides a querying framework capable of handling bipolar fuzzy queries and relations. The set of operators defined in the following generalizes the “fuzzy relational algebra” that has been previously proposed for handling non-bipolar fuzzy queries and relations (see, e.g., Bosc et al., 1999a and Chapter 5), which is itself a generalization of classical relational algebra.

8.2 Preliminaries

8.2.1 About bipolarity

In the following, we denote by \( \mu(u) \) (resp. \( \eta(u) \)) the degree reflecting the extent to which an element \( u \) satisfies a constraint \( C \) (resp. a wish \( W \)). When \( C \) and \( W \) concern the same attribute, two consistency conditions may be considered (Dubois and Prade, 2008a):

- Strong consistency, as for twofold fuzzy sets (Dubois and Prade, 1987):

\[
 \sup_u \min(\eta(u), 1 - \mu(u)) = 0,
\]

(8.2)

which states that the support of the wish must be included in the core of the constraint. Then the wish can only discriminate between the tuples which have degree 1 for the constraint. As noted in Dubois and Prade (2008a), the pair \((W, C)\) under the condition in Expression (8.2) is a twofold fuzzy set, since \( \text{support}(W) \subseteq \text{core}(C) \).
Bipolar Fuzzy Queries

- Weak consistency in the spirit of intuitionistic fuzzy sets (Atanassov, 1986):

\[ \forall u, \eta(u) \leq \mu(u) \]  

which states that the wish must be included in the constraint (in the sense of Zadeh). The wish can then be used to discriminate between these tuples that somewhat satisfy the constraint. As noted in Dubois and Prade (2008a), the pair \((C, W)\) under the condition in Expression (8.3) is an intuitionistic fuzzy set in the sense of Atanassov (1986).

When \(C\) and \(W\) concern different attributes, the weak consistency of the wish with the constraint can be recovered by replacing \((C, W)\) with \((C, C \land W)\), as suggested in Dubois and Prade (2008a). In the following, we will interpret the conjunction (in \(C \land W\)) by means of the t-norm product so as not to lose any discrimination power (which would be the case if \(\min\) were chosen).

From now on only weak consistency is enforced. We define a bipolar relation (in the database sense) as a relation where each tuple \(t\) is associated with two degrees \(\mu(t)\) and \(\eta(t)\) in the unit interval, expressing the extent to which the tuple satisfies the constraint (resp. wish) that has been used to produce the relation. A tuple will be denoted by \((\mu, \eta)/t\). In base relations, \(\mu(t) = \eta(t) = 1, \forall t\). It is assumed that tuples such that \(\mu = 0\) do not appear in the relation (they do not belong to it at all). We denote by \(r\) a bipolar base relation and by \((C, W)\) the constraint/wish pair that is applied to \(r\) in order to build a bipolar fuzzy relation \(r'\). One has:

\[ r' = \{(\mu, \eta)/t|\mu = \mu_C(t) \land \eta = \eta_W(t)\}. \]  

When no constraint is expressed, Dubois and Prade (2008a) suggest to use \(C = \text{true}\). Note that this case is unlikely to be very useful in practice, since it corresponds to the situation where all the tuples from the queried relation are considered equally acceptable. The wish just allows a “bonus” to be given to some items; for example, “I prefer red cars, but if there are none, any car will do”. Reciprocally, when no wish is expressed (the case of a non-bipolar — fuzzy or not — condition), we use \(W = \text{false}\). In this case, the condition acts as a regular selection expression, i.e., it discards the items which do not satisfy the constraint at all (and ranks the others, if the constraint is fuzzy).
We introduce the operators $l_{\text{min}}$ and $l_{\text{max}}$, which play a major role in defining the conjunction (and intersection) and the disjunction (and union):

$$l_{\text{min}}((\mu, \eta), (\mu', \eta')) = \begin{cases} (\mu, \eta) & \text{if } \mu < \mu' \text{ or } (\mu = \mu' \text{ and } \eta < \eta') \\ (\mu', \eta') & \text{otherwise.} \end{cases}$$

$$l_{\text{max}}((\mu, \eta), (\mu', \eta')) = \begin{cases} (\mu, \eta) & \text{if } \mu > \mu' \text{ or } (\mu = \mu' \text{ and } \eta > \eta') \\ (\mu', \eta') & \text{otherwise.} \end{cases}$$

It is straightforward to prove that $l_{\text{min}}$ (resp. $l_{\text{max}}$) is commutative, associative, idempotent, monotonic, has $(1, 1)$ (resp. $(0, 0)$) as a neutral element, and $(0, 0)$ (resp. $(1, 1)$) as an absorbing element. These properties make it legitimate to use $l_{\text{min}}$ and $l_{\text{max}}$ as conjunction and disjunction operators respectively (in the spirit of triangular norms and co-norms).

Note that Expression (8.4) can also be written:

$$r' = \{(\mu, \eta)/t | \mu = l_{\text{min}}((\mu_{C}(t), \eta_{W}(t)), (\mu_{r}(t), \eta_{r}(t)))\}, \quad (8.5)$$

since $r$ is a base relation (where $\mu_{r}(t) = \eta_{r}(t) = 1$, $\forall t$).

8.3 Extended Algebraic Operators

In this section, we review the operators from classical relational algebra, and we give an extended version of them in the framework of bipolar fuzzy relations. For each operator, the starting point is its usual definition; we generalize that by stating how the pairs of degrees attached to every tuple of a bipolar relation must be taken into account. In the following, we use as an example the case of a user who wants to buy a used car. We consider the relation $Car$ of schema ($\#id$, make, model, mileage, years, price, color).

8.3.1 Intersection

The straightforward definition of intersection is:

$$\text{inter}(r, s) = \{((\mu'', \eta''), t) | (\mu, \eta)/t \in r \wedge (\mu', \eta')/t \in s \wedge (\mu'', \eta'') = l_{\text{min}}((\mu, \eta), (\mu', \eta'))\}. \quad (8.6)$$

Example 8.1. The query “find the cars that meet both John’s and Peter’s needs/preferences” may be expressed as $\text{inter}(Car_{J}, Car_{P})$, where $Car_{J} = \text{select}(Car, (C_{1}, W_{1}))$, $Car_{P} = \text{select}(Car, (C_{2}, W_{2}))$, and $(C_{1}, W_{1})$ (resp. $(C_{2}, W_{2})$) represents John’s (resp. Peter’s) bipolar requirement.
Bipolar Fuzzy Queries

Table 8.1 An instance of relation Car_J

<table>
<thead>
<tr>
<th>#id</th>
<th>make</th>
<th>model</th>
<th>mileage</th>
<th>years</th>
<th>price</th>
<th>(\mu)</th>
<th>(\eta)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ford</td>
<td>Mondeo</td>
<td>35,000</td>
<td>2008</td>
<td>22,000</td>
<td>0.7</td>
<td>0.2</td>
</tr>
<tr>
<td>2</td>
<td>Seat</td>
<td>Altea</td>
<td>50,000</td>
<td>2007</td>
<td>17,000</td>
<td>0.6</td>
<td>0.4</td>
</tr>
<tr>
<td>3</td>
<td>VW</td>
<td>Golf</td>
<td>39,000</td>
<td>2008</td>
<td>21,000</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>4</td>
<td>Toyota</td>
<td>Corolla</td>
<td>60,000</td>
<td>2006</td>
<td>18,000</td>
<td>0.9</td>
<td>0.4</td>
</tr>
<tr>
<td>5</td>
<td>VW</td>
<td>Passat</td>
<td>75,000</td>
<td>2004</td>
<td>16,000</td>
<td>0.8</td>
<td>0.1</td>
</tr>
<tr>
<td>6</td>
<td>Opel</td>
<td>Zafira</td>
<td>58,000</td>
<td>2007</td>
<td>15,000</td>
<td>1</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Table 8.2 An instance of relation Car_P

<table>
<thead>
<tr>
<th>#id</th>
<th>make</th>
<th>model</th>
<th>mileage</th>
<th>years</th>
<th>price</th>
<th>(\mu)</th>
<th>(\eta)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Seat</td>
<td>Altea</td>
<td>50,000</td>
<td>2007</td>
<td>17,000</td>
<td>0.9</td>
<td>0.3</td>
</tr>
<tr>
<td>4</td>
<td>Toyota</td>
<td>Corolla</td>
<td>60,000</td>
<td>2006</td>
<td>18,000</td>
<td>0.7</td>
<td>0.7</td>
</tr>
<tr>
<td>5</td>
<td>VW</td>
<td>Passat</td>
<td>75,000</td>
<td>2004</td>
<td>16,000</td>
<td>0.8</td>
<td>0.6</td>
</tr>
<tr>
<td>7</td>
<td>Renault</td>
<td>Laguna</td>
<td>18,000</td>
<td>2008</td>
<td>25,000</td>
<td>0.3</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Table 8.3 The result of \(\text{inter}(\text{Car}_J, \text{Car}_P)\)

<table>
<thead>
<tr>
<th>#id</th>
<th>make</th>
<th>model</th>
<th>mileage</th>
<th>years</th>
<th>price</th>
<th>(\mu)</th>
<th>(\eta)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Seat</td>
<td>Altea</td>
<td>50,000</td>
<td>2007</td>
<td>17,000</td>
<td>0.6</td>
<td>0.4</td>
</tr>
<tr>
<td>4</td>
<td>Toyota</td>
<td>Corolla</td>
<td>60,000</td>
<td>2006</td>
<td>18,000</td>
<td>0.7</td>
<td>0.7</td>
</tr>
<tr>
<td>5</td>
<td>VW</td>
<td>Passat</td>
<td>75,000</td>
<td>2004</td>
<td>16,000</td>
<td>0.8</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Note: the selection operator on bipolar fuzzy relations is formally defined in Section 8.3.6. With the extensions of \(\text{Car}_J\) and \(\text{Car}_P\) given in Tables 8.1 and 8.2 respectively, one obtains the result represented in Table 8.3.

8.3.2 Union

The definition of the union of fuzzy bipolar relations is:

\[
\text{union}(r, s) = \\
\{(\mu'', \eta'')/t | \exists (\mu, \eta)/t \in r \land \exists (\mu', \eta')/t \in s \land \\
(\mu'', \eta'') = \text{tmax}(\mu, \eta, (\mu', \eta'))\}\ 
\]  

(8.7)

Note: the selection operator on bipolar fuzzy relations is formally defined in Section 8.3.6. With the extensions of \(\text{Car}_J\) and \(\text{Car}_P\) given in Tables 8.1 and 8.2 respectively, one obtains the result represented in Table 8.3.\(\diamond\)
Example 8.2. The query “find the cars that meet John’s or Peter’s needs/preferences” may be expressed as $\text{union}(\text{Car}_J, \text{Car}_P)$. With the extensions of Car$_J$ and Car$_P$ given in Tables 8.1 and 8.2 respectively, one obtains the result represented in Table 8.4.

8.3.3 Cartesian product

The straightforward definition of the Cartesian product of bipolar fuzzy relations is:

$$ \text{cartprod}(r, s) = \{(\mu'', \eta'')/t', \mu', \eta')/t \in r \land (\mu'', \eta'')/t' \in s \land (\mu'', \eta'') = \text{lmin}((\mu, \eta), (\mu', \eta'))\}, $$ (8.8)

where $\oplus$ denotes concatenation.

8.3.4 Negation

Even though it does not appear very useful in practice to negate a bipolar condition involved in a selection clause, the negation operator plays a crucial role in defining the set difference, as we will see later. Several ways of defining the negation of a bipolar condition can be envisaged a priori. We denote $(C', W') = \neg(C, W)$. We consider three requirements for the negation of bipolar fuzzy conditions. Operator $\neg$ must be:

- order-reversing (R1):

  $$ \forall t_1, t_2, (\mu_C(t_1), \eta_W(t_1)) <_{\text{lex}} (\mu_C(t_2), \eta_W(t_2)) $$
  $$ \Leftrightarrow (\mu_{C'}(t_2), \eta_{W'}(t_2)) <_{\text{lex}} (\mu_{C'}(t_1), \eta_{W'}(t_1)), $$

  where $<_{\text{lex}}$ denotes lexicographic order,

- consistency-preserving (R2): $\forall t, \eta_W(t) \leq \mu_{C'}(t)$,

- involutive (R3): $\neg(\neg(C, W)) = (C, W)$. 

Table 8.4 Result of union $(\text{Car}_J, \text{Car}_P)$

<table>
<thead>
<tr>
<th>#</th>
<th>make</th>
<th>model</th>
<th>mileage</th>
<th>years</th>
<th>price</th>
<th>$\mu$</th>
<th>$\eta$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ford</td>
<td>Mondeo</td>
<td>35,000</td>
<td>2008</td>
<td>22,000</td>
<td>0.7</td>
<td>0.2</td>
</tr>
<tr>
<td>2</td>
<td>Seat</td>
<td>Altea</td>
<td>50,000</td>
<td>2007</td>
<td>17,000</td>
<td>0.9</td>
<td>0.3</td>
</tr>
<tr>
<td>3</td>
<td>VW</td>
<td>Golf</td>
<td>39,000</td>
<td>2008</td>
<td>21,000</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>4</td>
<td>Toyota</td>
<td>Corolla</td>
<td>60,000</td>
<td>2006</td>
<td>18,000</td>
<td>0.9</td>
<td>0.4</td>
</tr>
<tr>
<td>5</td>
<td>VW</td>
<td>Passat</td>
<td>75,000</td>
<td>2004</td>
<td>16,000</td>
<td>0.8</td>
<td>0.6</td>
</tr>
<tr>
<td>6</td>
<td>Opel</td>
<td>Zafira</td>
<td>58,000</td>
<td>2007</td>
<td>15,000</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>7</td>
<td>Renault</td>
<td>Laguna</td>
<td>18,000</td>
<td>2008</td>
<td>25,000</td>
<td>0.3</td>
<td>0.1</td>
</tr>
</tbody>
</table>
8.3.4.1 Approach inspired by twofold fuzzy sets

The definition of the negation proposed by Dubois and Prade (1987) in the framework of twofold fuzzy sets is:

\[ \neg(C, W) = (\neg W, \neg C), \quad (8.9) \]

where \( \neg \) is a standard fuzzy negation. However, using it as such in the context of bipolar fuzzy conditions is debatable for two reasons:

- first, this definition does not preserve the equivalence between a bipolar condition involving an empty wish and a non-bipolar condition, which induces the constraint \( \neg(C, \text{false}) = (\neg C, \text{false}) \). Indeed, with Expression (8.9), one has: \( \neg(C, \text{false}) = (\text{true}, \neg C) \). A solution is to represent a bipolar condition involving an empty wish as \((C, C)\) instead of \((C, \text{false})\) — note that this choice preserves weak consistency. Then \( \neg(C, C) = (\neg C, \neg C) \), and the aforementioned equivalence holds.
- second, this definition assumes that the negation of a constraint (resp. a wish) is a wish (resp. a constraint), whereas this is not so obvious in the case where \( C \) and \( W \) concern different attributes. Indeed, in general, the wish makes sense (i.e., is relevant) only when the constraint is satisfied. A typical example is: “I want a Volkswagen and if possible a black one”. With the definition above, the bipolar condition becomes “I do not want a black car, and if possible not a Volkswagen”, which does not appear to be a very intuitive negation of the initial statement.

In fact, when \( C \) and \( W \) concern different attributes, the solution is to replace \((C, W)\) by \((C, W \land C)\) — as proposed in Dubois and Prade (2008a) — to enforce the consistency condition. Then, adapting Expression (8.9), we obtain:

\[ \neg(C, W) = (\neg(C \land W), \neg C), \quad (8.10) \]

and the second problem mentioned above disappears. The negation of “I want a Volkswagen and if possible a black one” is now “a black Volkswagen is out of the question, and if possible I would like a car which is not a Volkswagen”.

Unfortunately, this negation is not order-reversing.

8.3.4.2 Counter-example

Consider the bipolar condition \( Q = (C, W) \) — where \( C \) and \( W \) are assumed to concern the same attribute — and a relation \( r \) containing the tuples \( t_1 \)
and \(t_2\) such that:
\[
\mu_C(t_1) = 0.8, \eta_W(t_1) = 0, \\
\mu_C(t_2) = 0.6, \eta_W(t_2) = 0.6.
\]

We have:
\[
\mu_C'(t_1) = 1, \eta_W'(t_1) = 0.2, \\
\mu_C'(t_2) = 0.4, \eta_W'(t_2) = 0.4,
\]
and we have both
\[
(\mu_C(t_1), \eta_W(t_1)) >_{\text{lex}} (\mu_C(t_2), \eta_W(t_2))
\]
and
\[
(\mu_C'(t_1), \eta_W'(t_1)) >_{\text{lex}} (\mu_C'(t_2), \eta_W'(t_2)).
\]

Note that viewing a bipolar requirement \((C,W)\) as an intuitionistic fuzzy set \((-C,W)\) (see Section 8.2.1) leads to the same problem. Indeed, since the standard negation of an intuitionistic fuzzy set \((E,F)\) is defined as \((F,E)\), one obtains the bipolar requirement \((-W,-C)\) — corresponding to the intuitionistic fuzzy set \((W,-C)\) — as the negation of \((C,W)\), i.e., the same as that obtained with the twofold fuzzy set view.

### 8.3.4.3 From “and if possible” to “or else”

Another approach that can be thought of has its starting point in the idea that, since \((C,W)\) has a conjunctive nature (“C and if possible W”), its negation should be of a disjunctive nature and could be expressed as “W’ or else C’”. A straightforward definition of \((W',C')\) is:
\[
W' = \neg C \quad \text{and} \quad C' = \neg W.
\]
The initial consistency condition then leads to \(\forall x, \eta_W'(x) \geq \mu_C'(x)\), and when \(C\) and \(W\) concern different attributes, \(C' = \neg W\) must be replaced by \(C' = \neg (C \land W)\). However, this view raises important problems of compositionality, since it leads to mixing in the same relation — resulting from a compound bipolar condition — of pairs of degrees with different semantics, as illustrated in the following example.

**Example 8.3.** Consider the compound bipolar condition \(Q = (C_1,W_1) \land \neg(C_2,W_2)\), where \(C_i\) and \(W_i\) are assumed to concern the same attribute,
Bipolar Fuzzy Queries

and a relation \( r \) containing the tuples \( t_1 \) and \( t_2 \) such that:

\[
\begin{align*}
\mu_{C_1}(t_1) &= 0.8, \eta_{W_1}(t_1) = 0.6, \\
\mu_{C_1}(t_2) &= 0.4, \eta_{W_1}(t_2) = 0.3, \\
\mu_{C_2}(t_1) &= 0.2, \eta_{W_2}(t_1) = 0, \\
\mu_{C_2}(t_2) &= 0.7, \eta_{W_2}(t_2) = 0.1.
\end{align*}
\]

We have:

\[
\begin{align*}
(\mu_Q(t_1), \eta_Q(t_1)) &= l\min((0.8, 0.6), (0.8, 1)) = (0.8, 0.6), \\
(\mu_Q(t_2), \eta_Q(t_2)) &= l\min((0.4, 0.3), (0.3, 0.9)) = (0.3, 0.9).
\end{align*}
\]

In one case \((t_1)\), the pair of degrees has the semantics “and if possible”, whereas in the other \((t_2)\) it has the meaning “or else”, which makes the result very difficult to interpret.

\[\square\]

8.3.4.4 An approach based on product and division

We start from the following definition:

\[
\neg(C,W) = (C',W') \text{ with: } \mu_{C'} = 1 - \mu_C \text{ and } \\
\eta_{W'} = \begin{cases} 
(1 - \mu_C) \cdot \left(1 - \frac{\eta_W}{\mu_C}\right) & \text{if } \mu_C \neq 0, \\
1 & \text{otherwise.}
\end{cases} \tag{8.11}
\]

This definition trivially satisfies properties (R1) and (R2), but requirement (R3), which expresses the involutivity property, is not perfectly satisfied, as illustrated by the example below.

**Example 8.4.** We use the notations: \(\neg(C,W) = (C',W')\) and \(\neg(C',W') = (C'',W'')\). Table 8.5 illustrates that, in most cases, the involutivity property holds, i.e., \((C,W) = (C'',W'')\). In fact, the only case where it does not hold is when \(\mu_C = 1\) and \(\eta_W \neq 1\) (illustrated by the row in bold in the table). Indeed \((1,b \neq 1)\) becomes \((0,0)\) through negation, which itself becomes \((1,1)\).

\[\square\]

A solution to this problem of imperfect involutivity is to tolerate the presence of pairs of the type \((0,b > 0)\) in a bipolar fuzzy relation — which represents a slight exception to the consistency requirement — considering that, when the constraint is entirely false, one does not care about the value
Fuzzy Preference Queries to Relational Databases

Table 8.5 Imperfect involutivity of the negation

<table>
<thead>
<tr>
<th>t</th>
<th>µ_C</th>
<th>η_W</th>
<th>µ_C'</th>
<th>η_W'</th>
<th>µ_C''</th>
<th>η_W''</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>t2</td>
<td>1</td>
<td>0.6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>t3</td>
<td>0.8</td>
<td>0.8</td>
<td>0.2</td>
<td>0</td>
<td>0.8</td>
<td>0.8</td>
</tr>
<tr>
<td>t4</td>
<td>0.8</td>
<td>0.5</td>
<td>0.2</td>
<td>0.075</td>
<td>0.8</td>
<td>0.5</td>
</tr>
<tr>
<td>t5</td>
<td>0.8</td>
<td>0.2</td>
<td>0.2</td>
<td>0.15</td>
<td>0.8</td>
<td>0.2</td>
</tr>
<tr>
<td>t6</td>
<td>0.8</td>
<td>0</td>
<td>0.2</td>
<td>0.2</td>
<td>0.8</td>
<td>0</td>
</tr>
<tr>
<td>t7</td>
<td>0.7</td>
<td>0.7</td>
<td>0.3</td>
<td>0</td>
<td>0.7</td>
<td>0.7</td>
</tr>
<tr>
<td>t8</td>
<td>0.6</td>
<td>0.6</td>
<td>0.4</td>
<td>0</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>t9</td>
<td>0.6</td>
<td>0.1</td>
<td>0.4</td>
<td>0.33</td>
<td>0.6</td>
<td>0.1</td>
</tr>
<tr>
<td>t10</td>
<td>0</td>
<td>0</td>
<td>0.4</td>
<td>0.4</td>
<td>0</td>
<td>0.6</td>
</tr>
<tr>
<td>t11</td>
<td>0.2</td>
<td>0.2</td>
<td>0.8</td>
<td>0</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>t12</td>
<td>0.1</td>
<td>0.1</td>
<td>0.9</td>
<td>0</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>t13</td>
<td>0.1</td>
<td>0</td>
<td>0.9</td>
<td>0.9</td>
<td>0.1</td>
<td>0</td>
</tr>
<tr>
<td>t14</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

However, while building a bipolar relation by means of a pair \((C, W)\) that concerns different attributes, one must then:

- not replace \((C, W)\) by \((C, C \land W)\) when \(\mu_C = 0\),
- also store the elements whose pair of degrees is of the type \((0, b \neq 0)\) in the bipolar relation,

so as not to disrupt the computation of set difference. In fact, instead of replacing \((C, W)\) by \((C, C \land W)\), one must replace \((C, W)\) by:

\[(C, W \land (C \lor (1 \Rightarrow_{RG} \neg C)))\],

where \(\Rightarrow_{RG}\) denotes the Rescher–Gaines implication.

However, instead of tolerating exceptions to the consistency requirement, one may instead choose to accept the slight twist to the involutivity property, inasmuch as:

- involutivity is not so crucial in the context considered, i.e. that of database querying,
the exception concerns a very specific situation, defining set difference with Expression (8.11) instead of (8.12) amounts to assigning the bipolar grade $(0, 0)$ instead of $(0, b > 0)$ to some elements in the result of the difference; but these elements are considered totally rejected anyway since their degree with respect to the constraint is zero.

**Example 8.5.** Table 8.6 illustrates that, with the definition in Expression (8.12), the involutivity property holds in all cases.

**Proof of involutivity.**

- Case where $\mu_C = 1$: $(\mu_C, \eta_W) = (1, b)$.
  
  $\mu_C' = 1 - 1 = 0, \eta_W' = (1 - \eta_W) = (1 - b)$
  
  $\mu_C'' = 1 - 0 = 1, \eta_W'' = 1 - \eta_W = b$
  
  Finally: $(\mu_C'', \eta_W'') = (1, b) = (\mu_C, \eta_W)$.

- Case where $\mu_C = 0$: $(\mu_C, \eta_W) = (0, b)$.
  
  $\mu_C' = 1 - 0 = 1, \eta_W' = (1 - \eta_W) = (1 - b)$
  
  $\mu_C'' = 1 - 1 = 0, \eta_W'' = 1 - \eta_W = b$
  
  Finally: $(\mu_C'', \eta_W'') = (0, b) = (\mu_C, \eta_W)$.

- General case: $(\mu_C, \eta_W) = (a \in [0, 1], b)$.
  
  $\mu_C' = 1 - a, \eta_W' = (1 - a) \times (1 - \frac{1}{a})$
  
  $\mu_C'' = 1 - (1 - a) = a,$
  
  $\eta_W'' = a \times \left(1 - \frac{(1 - a) \times (1 - \frac{1}{a})}{1 - a}\right) = a \times (1 - (1 - \frac{b}{a})) = b$
  
  Finally: $(\mu_C'', \eta_W'') = (a, b) = (\mu_C, \eta_W)$. 

}\end{verbatim}
It is straightforward to prove that with the definitions in Expressions (8.11) and (8.12), the De Morgan laws hold (using $l_{min}$ and $l_{max}$ for the conjunction and the disjunction respectively).

8.3.4.5 Negation based on antonymy

Another view of the "negation" is based on the use of antonyms (Trillas and Riera, 1981; Ovchinnikov, 1981; Trillas et al., 2008). For instance, it might make sense to say that the negation of "I want a car whose price is at most $20,000 and if possible at most $15,000" is "I want a car whose price is over $20,000 and if possible over $25,000"; or, in the case of different attributes for the constraint and the wish, the negation of: "I want to find a person who is well-paid and if possible young" could be "I want to find a person who is not well-paid and if possible old". In the first example, "$>25,000" is the condition obtained from "$\leq15,000" by symmetry with respect to "$\leq20,000". Formally, this corresponds to the definition:

$$\neg(C,W) = (\neg C, W') \quad \text{with} \quad W' = \text{ant}(W,C).$$  \hspace{1cm} (8.13)

In the second case, the fuzzy term old is assumed to be an antonym of young, and in Expression (8.13), $W' = \text{ant}(W,C)$ may be replaced by $W' = \text{ant}(W)$, since here the definition of the antonym of W does not depend on C. As usual, when C and W concern different attributes, $(\neg C, W')$ must be understood as $(\neg C, \neg C \land W')$.

Several ways for defining antonyms are possible, but as noted in Trillas et al. (2008), the inequality $\mu_{\text{ant}(P)} \leq \mu_{\neg P}$ should be respected, and shows $\neg P$ as a limit case for any possible antonym of P.

Expression (8.13) preserves consistency and is involutive (provided that $\text{ant}(\text{ant}(W,C), \neg C) = W$), but is not order-reversing in general. Note also that the antonym of a predicate P does not necessarily exist. This is the case in particular when P concerns a categorical attribute, for instance "color is red".

8.3.5 Difference

8.3.5.1 A logical view of difference

We first recall the definition of difference in the context of classical relational algebra:

$$\text{differ}(r, s) = \{ t \mid t \in r \land t \notin s \}. \hspace{1cm} (8.14)$$
Bipolar Fuzzy Queries

The most commonly used definition of the difference between two unipolar fuzzy relations \( r \) and \( s \) is:

\[
\mu_{\text{differ}(r,s)}(t) = \min(\mu_r(t), \mu_s(t)) = \min(\mu_r(t), 1 - \mu_s(t)).
\]  
(8.15)

In the case of a bipolar fuzzy relation, one may also consider a definition of difference based on negation. The straightforward negation-based definition of the difference of bipolar fuzzy relations is then as follows:

\[
(\mu_{\text{differ}(r,s)}(x), \eta_{\text{differ}(r,s)}(x)) = \min((\mu_r(x), \eta_r(x)), (\mu_s(x), \eta_s(x))).
\]  
(8.16)

An example of a relational algebraic query involving a difference between two bipolar fuzzy relations is given below.

**Example 8.6.** The query “find the cars that meet John’s needs/preferences but not Peter’s” may be expressed as:

\[
differ(Car_J, Car_P),
\]

where \( Car_J \) and \( Car_P \) represent two bipolar fuzzy relations resulting from two bipolar selection queries on \( Car \) issued respectively by John and Peter, as in Example 8.1. With the extensions of \( Car_J \) and \( Car_P \) from Tables 8.1 and 8.2, and using Expression (8.11) to interpret the negation, one obtains the result shown in Table 8.7.

8.3.5.2 A non-logical view of the difference

Another way to define difference is to take a set-oriented point of view. One of the corresponding usual definitions is: \( E - F = \) the smallest set that must be added to \( E \cap F \) so as to obtain \( E \). This point of view leads to

<table>
<thead>
<tr>
<th>#</th>
<th>id</th>
<th>make</th>
<th>model</th>
<th>mileage</th>
<th>years</th>
<th>price</th>
<th>( \mu )</th>
<th>( \eta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ford</td>
<td>Mondeo</td>
<td>35,000</td>
<td>2008</td>
<td>22,000</td>
<td>0.7</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Seat</td>
<td>Altea</td>
<td>50,000</td>
<td>2007</td>
<td>17,000</td>
<td>0.1</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>VW</td>
<td>Golf</td>
<td>39,000</td>
<td>2008</td>
<td>21,000</td>
<td>0.8</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Toyota</td>
<td>Corolla</td>
<td>60,000</td>
<td>2006</td>
<td>18,000</td>
<td>0.3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>VW</td>
<td>Passat</td>
<td>75,000</td>
<td>2004</td>
<td>16,000</td>
<td>0.2</td>
<td>0.05</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Opel</td>
<td>Zafira</td>
<td>58,000</td>
<td>2007</td>
<td>15,000</td>
<td>1</td>
<td>0.5</td>
<td></td>
</tr>
</tbody>
</table>
the definition:

\[ E - F = \cap \{ G | G \cup (E \cap F) = E \}, \]  

(8.17)

which implies:

\[
(\mu_{E - F}(x), \eta_{E - F}(x)) = \begin{cases} 
(0, 0) & \text{if } (\mu_E(x), \eta_E(x)) \leq_{\text{lex}} (\mu_F(x), \eta_F(x)), \\
(\mu_E(x), \eta_E(x)) & \text{otherwise}, 
\end{cases}
\]

where \( \leq_{\text{lex}} \) denotes lexicographic order. However, this definition is not very satisfactory from a semantic viewpoint. Consider an element \( x \) such that \( (\mu_E(x), \eta_E(x)) = (0.9, 0.7) \) and \( (\mu_F(x), \eta_F(x)) = (0.8, 0.6) \). With the definition above, one obtains:

\[
(\mu_{E - F}(x), \eta_{E - F}(x)) = (0.9, 0.7),
\]

which means that \( x \) clearly belongs to \( E - F \) whereas it also clearly belongs to \( F \).

Another solution is to extend the notion of bounded difference \( \ominus \) defined by Zadeh (1975a) for fuzzy sets:

\[
\mu_{E \ominus F}(x) = \max(0, \mu_E(x) - \mu_F(x)).
\]

In the case of bipolar fuzzy relations, one could use:

\[
(\mu_{E - F}(x), \eta_{E - F}(x)) = (\mu_{E \ominus F}(x), \eta_{E \ominus F}(x)).
\]  

(8.18)

Consider again an element \( x \) such that \( (\mu_E(x), \eta_E(x)) = (0.9, 0.7) \) and \( (\mu_F(x), \eta_F(x)) = (0.8, 0.6) \). With the latter definition, one obtains:

\[
(\mu_{E - F}(x), \eta_{E - F}(x)) = (0.1, 0.1).
\]

Expression (8.17) satisfies 15 out of the 17 main properties of regular set difference (Suppes, 1972). The only two which do not hold are:

- \( (E - F = E) \Leftrightarrow (E \cap F = \emptyset) \),
- \( (E - F) = (E \cup F) \Leftrightarrow (F = \emptyset) \),

but this is already the case with regular fuzzy sets.

On the other hand, with Expression (8.18), the following seven properties are lost:

- \( (E - F) - H = (E - F) \cap (E - H) \),
- \( E - (F - H) = (E - F) \cup (E \cap H) \),
- \( (E - F) \cup H = (E \cup H) - (F - H) \),
- \( (E \cap F) - H = E \cap (F - H) \).
Bipolar Fuzzy Queries

- \((E \cap F) - H = (E - H) \cap (F - H)\),
- \((E \cup F) - H = (E - H) \cup (F - H)\),
- \(E - (F \cap H) = (E - F) \cup (E - H)\).

The basic reason why these properties no longer hold in the fuzzy bipolar case is that the bounded difference involves an arithmetic difference between the degrees, whereas the union and intersection operators, which are based on the lexicographic order, do not modify the original degrees.

8.3.5.3 Counter-example (first property).
Consider the bipolar fuzzy relations \(E = \{(0.7, 0.1)/x\}, F = \{(0.3, 0.3)/x\}, \) and \(H = \{(0.2, 0)/x\}.\) From Expression (8.18), we have:
\[
\left(\mu_{(E - F) - H}, \eta_{(E - F) - H}\right)/x = (0.2, 0)/x
\]
whereas:
\[
\left(\mu_{(E - F) \cap (E - H)}, \eta_{(E - F) \cap (E - H)}\right)/x = (0.4, 0)/x.\)

8.3.5.4 Conclusion
It appears that the definition of the difference of bipolar fuzzy relations relying on Expressions (8.16) and (8.12) looks the most satisfactory, since this definition does not have any major semantic drawbacks.

8.3.6 Selection
The straightforward definition of selection is:
\[
select(r, (C, W)) = \{(\mu', \eta')/t| (\mu, \eta)/t \in r \land
\]
\[
(\mu', \eta') = l\min((\mu, \eta), (\mu_C(t), \eta_W(t)))\}, \quad (8.19)
\]
where \(\mu_C(t)\) (resp. \(\eta_W(t)\)) denotes the satisfaction degree of \(t\) with respect to \(C\) (resp. \(W\)).

Example 8.7. The query “find the cars among those preferred by Paul (in the sense of the bipolar requirement that was used to build the relation \(Car_{Paul}\)) which are less than $20,000 and if possible have a mileage less than 60,000” may be expressed as:
\[
select(Car_{Paul}, (price\ is\ lt.\ 20k,\ mileage\ is\ lt.\ 60k)).
\]
Fuzzy Preference Queries to Relational Databases

Table 8.8 An instance of relation Car

<table>
<thead>
<tr>
<th>#</th>
<th>id</th>
<th>make</th>
<th>model</th>
<th>mileage</th>
<th>years</th>
<th>price</th>
<th>µ</th>
<th>η</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ford</td>
<td>Mondeo</td>
<td>35,000</td>
<td>2008</td>
<td>22,000</td>
<td>0.7</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Seat</td>
<td>Altea</td>
<td>65,000</td>
<td>2007</td>
<td>21,000</td>
<td>0.9</td>
<td>0.3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>VW</td>
<td>Passat</td>
<td>66,000</td>
<td>2008</td>
<td>23,000</td>
<td>0.8</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Toyota</td>
<td>Corolla</td>
<td>60,000</td>
<td>2006</td>
<td>26,000</td>
<td>0.9</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>VW</td>
<td>Passat</td>
<td>75,000</td>
<td>2004</td>
<td>16,000</td>
<td>0.8</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Seat</td>
<td>Altea</td>
<td>58,000</td>
<td>2007</td>
<td>15,000</td>
<td>1</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Toyota</td>
<td>Corolla</td>
<td>18,000</td>
<td>2008</td>
<td>25,000</td>
<td>0.3</td>
<td>0.1</td>
<td></td>
</tr>
</tbody>
</table>

Table 8.9 Result of the bipolar selection on the relation Car

<table>
<thead>
<tr>
<th>#</th>
<th>make</th>
<th>model</th>
<th>mileage</th>
<th>years</th>
<th>price</th>
<th>µ</th>
<th>η</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ford</td>
<td>Mondeo</td>
<td>35,000</td>
<td>2008</td>
<td>22,000</td>
<td>0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>2</td>
<td>Seat</td>
<td>Altea</td>
<td>65,000</td>
<td>2007</td>
<td>21,000</td>
<td>0.8</td>
<td>0.4</td>
</tr>
<tr>
<td>3</td>
<td>VW</td>
<td>Passat</td>
<td>66,000</td>
<td>2008</td>
<td>23,000</td>
<td>0.4</td>
<td>0.16</td>
</tr>
<tr>
<td>5</td>
<td>VW</td>
<td>Passat</td>
<td>75,000</td>
<td>2004</td>
<td>16,000</td>
<td>0.8</td>
<td>0.6</td>
</tr>
<tr>
<td>6</td>
<td>Seat</td>
<td>Altea</td>
<td>58,000</td>
<td>2007</td>
<td>15,000</td>
<td>1</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Assume that the fuzzy predicates $lt_{20k}$ and $lt_{60k}$ are defined as follows:

$$
\mu_{lt_{20k}}(x) = \begin{cases} 
1 & \text{if } x \leq 20,000, \\
0 & \text{if } x > 25,000, \\
\frac{25,000 - x}{5000} & \text{otherwise}; 
\end{cases}
$$

$$
\mu_{lt_{60k}}(x) = \begin{cases} 
1 & \text{if } x \leq 60,000, \\
0 & \text{if } x > 70,000, \\
\frac{70,000 - x}{10,000} & \text{otherwise}. 
\end{cases}
$$

Using the data from Table 8.8, the result of the selection is represented in Table 8.9 (recall that the t-norm product is used for interpreting the conjunction when $W$ has to be replaced by $C \land W$ in order to preserve weak consistency).

In Expression (8.19), $C$ and/or $W$ can be a complex condition expressing an aggregation (e.g., involving conjunctions and/or disjunctions) of atomic predicates. However, it is important to note that, in general:

- $(C_1 \land C_2, W_1 \land W_2) \neq (C_1, W_1) \land (C_2, W_2)$,
- $(C_1 \lor C_2, W_1 \lor W_2) \neq (C_1, W_1) \lor (C_2, W_2)$. 

8.3.6.1 Counter-example

Consider a tuple \( t \) such that \( \mu_1(t) = 0.8, \eta_1(t) = 0.3, \mu_2(t) = 0.7, \) and \( \eta_2(t) = 0.4. \) Interpreting \( \land \) as \( \min \) and \( \lor \) as \( \max \) in the left-hand part of the previous inequalities, we have:

\[
(\min(0.8, 0.7), \min(0.3, 0.4)) = (0.7, 0.3) \quad \text{whereas} \\
\text{lmin}((0.8, 0.3), (0.7, 0.4)) = (0.7, 0.4), \\
(\max(0.8, 0.7), \max(0.3, 0.4)) = (0.8, 0.4) \quad \text{whereas} \\
\text{lmax}((0.8, 0.3), (0.7, 0.4)) = (0.8, 0.3). \]

Therefore, the user must employ the appropriate formulation when expressing his/her query. One may suppose that a formulation of the type \( (C_1, W_1) \land \cdots \land (C_n, W_n) \) — where the \( C_i \)'s and \( W_i \)'s are atomic conditions — will be chosen when each atomic bipolar condition involved concerns a specific attribute, as in: “find the cars which cost less than $20,000 \( (C_1) \) and if possible less than $15,000 \( (W_1) \), and which are less than 5 years old \( (C_2) \) and if possible less than 3 years old \( (W_2) \)”. However, in a bipolar condition \( (C, W) \) where \( C \) and/or \( W \) are composite, the set of attributes involved in the constraint and the wish respectively will be disjoint in general. An example is: “find the German cars whose price is less than $20,000, and if possible recent and powerful”.

8.3.7 Projection

We start from the usual definition of projection, i.e.:

\[
\text{project}(r, X) = \{x | \exists t \in r \text{ such that } t.X = x\}.
\]

In the case of bipolar fuzzy relations, the existential quantifier is naturally interpreted by means of operator \( \text{lmax} \), and so we obtain the definition:

\[
\text{project}(r, X) = \{(\mu, \eta)/x | \exists t \in r \text{ s.t. } t.X = x \land \\
(\mu, \eta) = \text{lmax}_{t \in r \text{ s.t. } t.X = x}(\mu(t), \eta(t))\}.
\] (8.20)

Example 8.8. The query “find the pairs (make, model) corresponding to the cars that meet Paul’s needs/preferences” may be expressed as:

\[
\text{project}(\text{select}(\text{Car}, (C_1, W_1)), \{\text{make, model}\}),
\]

where \( (C_1, W_1) \) represents Paul’s bipolar requirement. Assume that \( \text{Car}_{\text{Paul}} = \text{select}(\text{Car}, (C_1, W_1)) \) is the relation represented in Table 8.8. The result of the projection is represented in Table 8.10."
### 8.3.8 Join

The straightforward definition of the join operation is:

\[
\text{join}(r, s, (C(X, Y), W(Z, T))) = \\
\left\{ (\mu'', \eta'') / t \otimes t' | (\mu, \eta) / t \in r \wedge (\mu', \eta') / t' \in s \wedge \\
(\mu'', \eta'') = \text{lmin}(\text{lmin}((\mu, \eta), (\mu', \eta')), \\
(\mu_C(t.X, t'.Y), \eta_W(t.Z, t'.T))) \right\},
\]

where \( X, Y, Z, \) and \( T \) denote sets of attributes, and \( X \) and \( Y \) (resp. \( Z \) and \( T \)) are compatible. Condition \( C(X, Y) \) (resp. \( W(Z, T) \)) is of the form \( X \theta_1 Y \) (resp. \( Z \theta_2 T \)), where \( \theta_1 \) and \( \theta_2 \) are (fuzzy or not) comparators (\( =, <, \leq, >, \geq, \neq, \approx, < <, \) etc.). As usual, in the case of a Boolean equijoin, only one occurrence of the join attribute is kept (which implies the addition of a final projection to the definition above). For instance, if \( C(X, Y) \) is written as “\( X = Y \)” , only \( X \) is kept — the same applies for \( W \). The above definition straightforwardly preserves the usual equivalence between a join and a Cartesian product followed by a selection:

\[
\text{join}(r, s, (C(X, Y), W(Z, T))) = \\
\text{select(cartprod}(r, s), (C(X, Y), W(Z, T))).
\]

**Example 8.9.** Assume the existence of two relations \( \text{Dealer}_1 \) and \( \text{Dealer}_2 \), with the same schema as \( \text{Car} \). An example of a bipolar join query is “find the cars (\( \text{make}, \text{model} \)) which are sold both by \( \text{Dealer}_1 \) and \( \text{Dealer}_2 \) preferably in the same range of price”. This query can be expressed as:

\[
\text{join}(\text{Dealer}_1, \text{Dealer}_2, (\text{make}_1 = \text{make}_2 \text{ and} \\
\text{model}_1 = \text{model}_2, \text{price}_1 \approx \text{price}_2)).
\]
8.3.9 Division

8.3.9.1 Reminder about “classical” division

We assume that the dividend relation \( r \) has the schema \((A, X)\), while that of the divisor relation \( s \) is \((B)\), where \( A \) and \( B \) are compatible sets of attributes. The division of relation \( r \) by relation \( s \) is classically defined (Section 2.2.2) as:

\[
\text{div}(r, s, A, B) = \{ x \in r[X] | \forall a, a \in s \Rightarrow (x, a) \in r \},
\]

where \( r[X] \) denotes the projection of \( r \) over \( X \). In other words, an element \( x \) belongs to the result of the division of \( r \) by \( s \) if and only if it is associated in \( r \) with at least all the values \( a \) appearing in \( s \).

Example 8.10. Consider a database involving the two relations order \((o)\) and product \((p)\) with respective schemas \(O(np, store, qty)\) and \(P(np, price)\). Tuples \(\langle n, s, q \rangle\) of \(o\) and \(\langle n, pr \rangle\) of \(p\) state that product \(n\) has been ordered from store \(s\) in quantity \(q\) and that its price is \(pr\). Retrieving the stores from which all the products priced under $127 have been ordered in a quantity greater than 35, can be expressed thanks to a division as:

\[
\text{div}(o_{g35}, p_{u127}, np, np),
\]

where relation \(o_{g35}\) corresponds to pairs \(\langle n, s \rangle\) such that product \(n\) has been ordered from store \(s\) in a quantity over 35, and relation \(p_{u127}\) gathers products whose price is under $127. From the extensions of relations \(o_{g35}\) and \(p_{u127}\) given as:

\[
o_{g35} = \{ \langle p_{15}, s_{32} \rangle, \langle p_{12}, s_{32} \rangle, \langle p_{34}, s_{32} \rangle, \langle p_{26}, s_{32} \rangle, \langle p_{12}, s_{7} \rangle, \langle p_{26}, s_{7} \rangle, \langle p_{15}, s_{19} \rangle, \langle p_{12}, s_{19} \rangle, \langle p_{26}, s_{19} \rangle \},
\]

\[
p_{u127} = \{ \langle p_{15} \rangle, \langle p_{12} \rangle, \langle p_{26} \rangle \},
\]

the result of the division is \(\{ \langle s_{32} \rangle, \langle s_{19} \rangle \}\).

8.3.9.2 Reminder about the division of fuzzy relations

When the relations involved are fuzzy (i.e., contain graded tuples), Expression (8.22) becomes (see Chapter 7):

\[
\mu_{\text{div}(r, s, A, B)}(x) = \min_a \mu_s(a) \Rightarrow_f \mu_r(x, a) \quad (8.23)
\]

where \(\Rightarrow_f\) denotes a fuzzy implication (Dubois et al., 2000).

Example 8.11. Consider again the relations from Example 8.10 and two fuzzy conditions “quantity is around 30” and “price is around $100” applied
to \(O\) and \(P\) respectively, leading to two fuzzy relations \(o_{a30}\) and \(p_{a100}\) whose extensions are given as:

\[
\begin{align*}
o_{a30} &= \{0.4/\langle p_{11}, s_{32}\rangle, 1/\langle p_{17}, s_{32}\rangle, 0.7/\langle p_{29}, s_{32}\rangle, 0.8/\langle p_{11}, s_{7}\rangle, \\
                      &\quad 0.6/\langle p_{29}, s_{7}\rangle, 1/\langle p_{11}, s_{19}\rangle, 0.3/\langle p_{17}, s_{19}\rangle, 0.9/\langle p_{42}, s_{19}\rangle\}, \\
p_{a100} &= \{1/\langle p_{11}\rangle, 0.8/\langle p_{17}\rangle, 0.6/\langle p_{29}\rangle\}.
\end{align*}
\]

The division of \(o_{a30}\) by \(p_{a100}\) based on \(Luk\)’s implication \((p \Rightarrow q = 1 \text{ if } q \geq p, 1 - p + q \text{ otherwise})\) returns \(\{0.6/\langle s_{32}\rangle, 0.2/\langle s_{7}\rangle, 0.4/\langle s_{19}\rangle\}\).

### 8.3.9.3 Bipolar division of bipolar fuzzy relations

When two bipolar fuzzy relations \(r\) and \(s\) (assumed to be consistent) come into play, Expression (8.23) can be used to compute four degrees:

\[
\begin{align*}
\mu(x) &= \min_{a} \mu_{s}(a) \Rightarrow_{L} \mu_{r}(x, a) \quad (8.24) \\
\eta(x) &= \min_{a} \eta_{s}(a) \Rightarrow_{L} \eta_{r}(x, a) \quad (8.25) \\
\mu'(x) &= \min_{a} \mu_{s}(a) \Rightarrow_{L} \mu_{r}(x, a) \quad (8.26) \\
\eta'(x) &= \min_{a} \eta_{s}(a) \Rightarrow_{L} \eta_{r}(x, a) \quad (8.27)
\end{align*}
\]

Because of the monotonicity of fuzzy implications, and the consistency condition between the constraint and the wish used to generate each bipolar relation, one has:

\[
\forall x, \eta(x) \leq \mu'(x) \leq \mu(x)
\]

and

\[
\forall x, \eta(x) \leq \eta'(x) \leq \mu(x).
\]

In order for the result also to be a consistent bipolar relation, the only possible choices are \((\mu, \eta), (\mu, \mu'), (\mu', \eta), (\mu', \eta')\), and \((\eta', \eta)\). It seems natural to choose \((\mu, \eta)\) for the following reasons:

- it captures the two extreme cases: \(\mu(x)\) expresses the extent to which every "somewhat ideal" element of the divisor is associated in a somewhat necessary fashion with \(x\) (the lax view, corresponding to the constraint aimed at retrieving the acceptable elements), whereas \(\eta(x)\) is the extent to which every "somewhat necessary" element of the divisor is associated...
Bipolar Fuzzy Queries

in a somewhat ideal fashion with \( x \) (the drastic view, corresponding to
the wish aimed at retrieving the ideal elements);

- it keeps its discrimination power intact in the particular cases where \( r \)
  and/or \( s \) is a non-bipolar relation; this is not true with the other pairs
  (when \( r \) is unipolar, one has \( \mu(x) = \eta'(x) \) and \( \eta(x) = \mu'(x) \),
  and when \( s \) is unipolar, one has: \( \mu(x) = \mu'(x) \) and \( \eta(x) = \eta'(x) \)).

Note that Expressions (8.24) and (8.25) can serve as the basis for the defini-
tion of a (graded) bipolar fuzzy-implication-based inclusion between bipolar
fuzzy sets.

We denote by \( birel \) the operation, not part of the query language itself,
that builds a bipolar fuzzy relation from two consistent (in the sense of
weak consistency) regular fuzzy relations:

\[
birel(r, s) = \{(\mu, \eta)/t \in \text{support}(r) \land \mu_r(t) = \mu \land
\]
\[
t \in \text{support}(s) \land \mu_s(t) = \eta\}
\]
\[
\bigcup \{(\mu, 0)/t \in r \land \mu_r(t) = \mu \land t \notin \text{support}(s)\}
\]
where \( \mu_r(t) \) (resp. \( \mu_s(t) \)) denotes the membership degree associated with
tuple \( t \) in the regular fuzzy relation \( r \) (resp. \( s \)).
The definition of the division
of bipolar fuzzy relations can be rewritten as:

\[
div(r, s, A, B) = birel(div(r_C, s_W, A, B), div(r_W, s_C, A, B)),
\]
\[(8.29)\]
where \( rel_C \) (resp. \( rel_W \)) denotes the fuzzy relation obtained by keeping
only the degrees \( \mu \) (resp. \( \eta \)) attached to the tuples from the bipolar fuzzy
relation \( rel \).

**Example 8.12.** Consider relations \( r \) and \( s \) from Table 8.11. Using
Łukasiewicz’s implication, and denoting by \( res \) the bipolar relation resulting

<table>
<thead>
<tr>
<th>( X )</th>
<th>( A )</th>
<th>( \mu )</th>
<th>( \eta )</th>
<th>( B )</th>
<th>( \mu' )</th>
<th>( \eta' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 )</td>
<td>( a_1 )</td>
<td>0.2</td>
<td>0</td>
<td>( a_1 )</td>
<td>0.7</td>
<td>0.4</td>
</tr>
<tr>
<td>( x_1 )</td>
<td>( a_2 )</td>
<td>0.6</td>
<td>0.3</td>
<td>( a_2 )</td>
<td>1</td>
<td>0.6</td>
</tr>
<tr>
<td>( x_1 )</td>
<td>( a_3 )</td>
<td>1</td>
<td>0.6</td>
<td>( a_3 )</td>
<td>0.3</td>
<td>0</td>
</tr>
</tbody>
</table>
from the division, one obtains:

\[\mu_{\text{res}}(x_1) = \min(0.4 \Rightarrow \text{Lu} 0.2, 0.6 \Rightarrow \text{Lu} 0.6, 0 \Rightarrow \text{Lu} 1) = \min(0.8, 1, 1) = 0.8,\]

\[\eta_{\text{res}}(x_1) = \min(0.7 \Rightarrow \text{Lu} 0.1, 0.3 \Rightarrow \text{Lu} 0.6) = \min(0.3, 0.3, 1) = 0.3,\]

\[\mu_{\text{res}}(x_2) = \min(0.4 \Rightarrow \text{Lu} 0.6, 1.0 \Rightarrow \text{Lu} 0.1) = \min(0.6, 0.6, 1, 0, 0.6, 1) = 0.6,\]

\[\eta_{\text{res}}(x_2) = \min(0.7 \Rightarrow \text{Lu} 0.1, 1.0 \Rightarrow \text{Lu} 0.1) = \min(0.3, 0.1, 0.7) = 0.3.\]

The result is the bipolar relation: \([\{0.8, 0.3\}/\langle x_1 \rangle, \{0.6, 0.3\}/\langle x_2 \rangle]\), and we have \(\langle x_1 \rangle \succ \langle x_2 \rangle\) according to lexicographic order.

In practice, it is likely that either the divisor or the dividend will be bipolar but not both, since the meaning of the query then becomes rather complex; Expressions (8.24)–(8.25) nevertheless make it possible to deal with the general case where both divisor and dividend are bipolar.

**Example 8.13.** Consider again the relations \(O\) and \(P\) from Example 8.12. Assume that a bipolar relation \(O_\text{b}\) is built from \(O\) using the twofold condition: \((C_O: \text{qty is higher than 10}, W_O: \text{qty is higher than 15}), where higher than 10 and higher than 15 are assumed to be fuzzy predicates. An example of a division query involving a bipolar dividend is: “find the stores which have ordered at least 10 occurrences of each product, and if possible at least 15 occurrences of each product”.

Now assume that a bipolar relation \(P_\text{b}\) is built from \(P\) using the twofold condition: \((C_P: \text{price is less than 200}, W_P: \text{price is less than 150}), where less than 200 and less than 150 are fuzzy predicates. An example of a division query involving a bipolar divisor is: “from which all of the products priced at less than $200 and if possible all of those priced at less than $150 have been ordered”.\]

8.3.9.4 **About the quotient property**

As mentioned in Section 2.2.2, the justification of the term “division” assigned to this operation relies on the fact that a property similar to that of the quotient of integers holds. Indeed, the resulting relation \(\text{res}\) obtained with Expression (8.22) has the double characteristic of a quotient:

\[\forall t \in \text{res}, \text{cartprod}(s, \{t\}) \subseteq r,\]  \hspace{1cm} (8.30)

\[\forall t \notin \text{res}, \text{cartprod}(s, \{t\}) \nsubseteq r.\]  \hspace{1cm} (8.31)
Expressions (8.30) and (8.31) state that the relation \( res \) resulting from the division is the largest relation whose Cartesian product with the divisor returns a result smaller than or equal to the dividend (according to set inclusion).

Considering the definition given in the previous section, processing a division \( \text{div}(r, s, A, B) \) of two bipolar fuzzy relations \( r \) and \( s \) comes down to processing two divisions of non-bipolar fuzzy relations, namely \( \text{div}(r_C, s_W, A, B) \) and \( \text{div}(r_W, s_C, A, B) \) — see Expression (8.29). Since it has been proven that the result of the division of two fuzzy relations is a quotient, provided that the conjunction operator used for the Cartesian product is appropriately chosen (Bosc et al., 2007b), the result of a division of bipolar fuzzy relations can be characterized as a twofold quotient. However, when the implication used is an S-implication, the quotient property can only be guaranteed if the divisor is normalized. In the context of bipolar relations, this means that both \( s_C \) and \( s_W \) have to contain a tuple whose associated degree equals 1.

**Example 8.14.** Consider the relations from Example 8.12. Relation \( r \) is equivalent to \( \text{birel}(r_C, r_W) \) where

\[
\begin{align*}
r_C &= \{0.2/\langle x_1, a_1 \rangle, 0.6/\langle x_1, a_2 \rangle, 1/\langle x_1, a_3 \rangle, 1/\langle x_2, a_2 \rangle, 0.1/\langle x_2, a_3 \rangle\}, \\
r_W &= \{0.3/\langle x_1, a_2 \rangle, 0.6/\langle x_1, a_3 \rangle, 1/\langle x_2, a_2 \rangle\},
\end{align*}
\]

and relation \( s \) is equivalent to \( \text{birel}(s_C, s_W) \) where

\[
\begin{align*}
s_C &= \{0.7/\langle a_1 \rangle, 1/\langle a_2 \rangle, 0.3/\langle a_3 \rangle\}, \\
s_W &= \{0.4/\langle a_1 \rangle, 0.6/\langle a_2 \rangle\}.
\end{align*}
\]

The result of \( \text{div}(r_C, s_W, A, B) \) using Łukasiewicz’s implication is:

\[
\text{res}_C = \{0.8/\langle x_1 \rangle, 0.6/\langle x_2 \rangle\},
\]

whereas the result of \( \text{div}(r_W, s_C, A, B) \) is:

\[
\text{res}_W = \{0.3/\langle x_1 \rangle, 0.3/\langle x_2 \rangle\},
\]

and \( \text{birel}(\text{res}_C, \text{res}_W) \) is equivalent to the result obtained in Example 8.12. It is useful to check that \( \text{res}_C \) is a quotient. Interpreting the Cartesian product by means of Łukasiewicz’s t-norm (\( \bot_{Lu}(a, b) = \max(a + b - 1, 0) \)),
one obtains:
\[ \text{cartprod}(s_W, \{0.8/\langle x_1 \rangle\}) = \{0.2/\langle x_1, a_1 \rangle, 0.4/\langle x_1, a_2 \rangle\} \subseteq r_C \quad \text{and} \]
\[ \text{cartprod}(s_W, \{0.6/\langle x_2 \rangle\}) = \{0.2/\langle x_2, a_2 \rangle\} \subseteq r_C. \]

On the other hand, if the degree attached to \( \langle x_1 \rangle \) in \( r_C \) is upgraded to 0.8\(^{+}\) (i.e., to a value slightly greater than 0.8), one has:
\[ \text{cartprod}(s_W, \{0.8^{+}/\langle x_1 \rangle\}) = \{0.2^{+}/\langle x_1, a_1 \rangle, 0.4^{+}/\langle x_1, a_2 \rangle\} \not\subseteq r_C. \]

Similarly, if the degree attached to \( \langle x_2 \rangle \) in \( r_C \) is upgraded to 0.6\(^{+}\), one obtains:
\[ \text{cartprod}(s_W, \{0.6^{+}/\langle x_2 \rangle\}) = \{0^{+}/\langle x_2, a_1 \rangle, 0.2^{+}/\langle x_2, a_2 \rangle\} \not\subseteq r_C. \]

In the same way, it can be checked that \( r_{sW} \) is also a quotient.\( \diamond \)

8.3.9.5 About the non-primitivity of the operator

As mentioned in Section 2.2.2, when considering regular (non-fuzzy) relations, the division operator can also be defined in terms of other relational operators (which shows its non-primitivity). The division comes down to discarding from \( r[X] \) all \( x \)'s from \( r[X] \) such that \( \exists a \in s, \langle x, a \rangle \notin r \). It is expressed as:

\[
\text{div}(r, s, A, B) = \text{differ} (\text{project}(r, X), \\
\text{project} (\text{differ} (\text{cartprod}\text{project}(r, X), \text{project}(s, B)), r), X). \tag{8.32}
\]

In this formula, the expression \( \text{differ}(\text{cartprod}\text{project}(r, X), s, r) \) determines the tuples that associate those values of attributes \( X \) and \( B \) that are missing in \( r \). The \( X \)-values present in this set must then be discarded from the final result, as is done by the outermost difference.

Bosc \textit{et al.} (1997) showed that the division of fuzzy relations is also a non-primitive operator, and can be rewritten in a similar way to Expression (8.32)

\[
\text{div}(r, s, A, B) = \text{sproject}(r, X), \\
\text{project}(\text{gdiffer} (\text{cartprod}\text{project}(r, X), \text{project}(s, B)), r), X), \tag{8.33}
\]

where the operation denoted by \( \text{sproject}(r, X) \) returns the support of the projection of relation \( r \) on the set of attributes \( X \), and \( \text{gdiffer} \) is a difference based on the triangular norm or the noncommutative conjunction underlying the implication present in Expression (8.23). Since \( \text{div}(r, s, A, B) \) can
be rewritten as the combination of two divisions of regular fuzzy relations — as in Expression (8.29) — one has:

\[
\text{div}(r, s, A, B) = \\
\text{birel}(\text{differ}(s\text{project}(r_C, X), \\
\text{project}(\text{gdiffer}(\text{cartprod}(s\text{project}(r_C, X), \\
\text{project}(s_W, B)), r_C), X)), \\
\text{differ}(s\text{project}(r_W, X), \\
\text{project}(\text{gdiffer}(\text{cartprod}(s_W, X), \\
\text{project}(s_C, B)), r_W), X))).
\]

**Example 8.15.** Consider the relations from Examples 8.12 and 8.13, and check that

\[
s\text{project}(r_C, X) = \{\langle x_1 \rangle, \langle x_2 \rangle\}
\]

and

\[
\text{cartprod}(s\text{project}(r_C, X), \text{project}(s_W, B)) = \\
\{0.4/\langle x_1, a_1 \rangle, 0.6/\langle x_1, a_2 \rangle, 0.4/\langle x_2, a_1 \rangle, 0.6/\langle x_2, a_2 \rangle\}.
\]

Using

\[
\mu_{\text{gdiffer}(E,F)}(x) = \max(\mu_E(x) + (1 - \mu_F(x)) - 1, 0) \\
= \max(\mu_E(x) - \mu_F(x), 0),
\]

one obtains:

\[
\text{gdiffer}(\text{cartprod}(s\text{project}(r_C, X), \text{project}(s_W, B)), r_C) = \\
\{0.2/\langle x_1, a_1 \rangle, 0.4/\langle x_2, a_1 \rangle\},
\]

\[
\text{differ}(s\text{project}(r_C, X), \text{project}(\text{gdiffer}(\text{cartprod}(s\text{project}(r_C, X), \\
\text{project}(s_W, B)), r_C), X)) = \\
\{1/\langle x_1 \rangle, 1/\langle x_2 \rangle\} \setminus \{0.2/\langle x_1 \rangle, 0.4/\langle x_2 \rangle\} = \{0.8/\langle x_1 \rangle, 0.6/\langle x_2 \rangle\}.
\]
which is indeed equal to res\(_C\). In the same way, it can be checked that

\[
res_W = differ(sproject(r_W, X),
project(gdiffer(cartprod(sproject(r_W, X),
project(s_C, B)), r_W), X)).
\]

### 8.4 Implementation Aspects

In terms of data representation, a regular DBMS can be used to handle bipolar queries, since the only modification with respect to the classical case concerns the schemas of those relations that must include two additional attributes for storing the degrees \(\mu\) and \(\eta\). In this section, the changes implied by the presence of bipolarity are discussed in terms of query processing for the different extended operators.

**Intersection and union.** These operations have the same data complexity as usual. It is necessary to do a nested scan of the relations involved, which can be optimized if indexes exist. With respect to an intersection (resp. union) of unipolar fuzzy relations, the only difference concerns the use of \(l_{\text{min}}\) (resp. \(l_{\text{max}}\)) instead of \(\text{min}\) (resp. \(\text{max}\)) — see Expressions (8.6) and (8.7).

**Cartesian product.** The situation is the same as for intersection and union: a nested scan of the relations involved is necessary. Again, \(l_{\text{min}}\) is used to compute the final twofold degree attached to a tuple of the result — see Expression (8.8).

**Difference.** For this operator too, the processing algorithm has a similar structure as in the classical case. The computation of \(r - s\) implies a nested scan of the relations involved. Each tuple \(t\) in \(r\) is checked to see whether it is present in \(s\) or not. If \(t\) is in \(s\), Expressions (8.16) and (8.12) are used to compute the final twofold degree attached to a tuple of the result. If \(t\) is not present in \(s\), then it is in the result with the twofold degree \((\mu_r(t), \eta_r(t))\).

**Selection.** In the absence of any index of the attribute(s) concerned by the constraint \(C\), the processing of a selection is similar as in the usual case: the relation concerned is sequentially scanned, and both conditions \(C\) and \(W\) are evaluated. If an index on the attribute(s) involved in \(C\) exists, one can take advantage of the derivation method proposed in Bosc and Pivert.
Bipolar Fuzzy Queries

(2000) for efficiently accessing the tuples $t$ which belong to the support of $C$ (i.e., those such that $\mu_C(t) > 0$). For each such tuple, $W$ is then evaluated in order to compute the final twofold degree. In every case, the (linear) data complexity is the same as in the case of unipolar fuzzy queries.

**Projection.** Here too, the same type of algorithm as usual (e.g., based on a sort of the relation concerned) must be used. The only change concerns the elimination of duplicate, which entails computation of a twofold degree by means of $l_{max}$ (instead of $\max$ in the unipolar fuzzy case).

**Join.** If there are no indexes on the join attributes, a nested scan of the relations involved is necessary. The only difference with respect to the unipolar case is that the join condition is twofold; this does not imply any overhead in terms of data complexity, which remains in $\theta(|r| \cdot |s|)$ (where $r$ and $s$ denote the relations to be joined).

If the join condition in the constraint part $C$ of the join condition is Boolean, this is evaluated first, following which the wish part of the join condition is computed by means of a (possibly fuzzy) selection.

On the other hand, if the join condition in the constraint part $C$ of the join condition is fuzzy but concerns attributes on which indexes are available, the aforementioned derivation principle — described in Bosc and Pivert (2000) — can be used to evaluate it. The wish part $W$ of the join condition is then evaluated by means of a selection on the result of the previous step.

In every case, the data complexity is as usual (between $|r| + |s|$ in the best case, when indexes are available, and $\min(|r| \cdot (|s| + 1), |s| \cdot (|r| + 1))$ in the worst case).

**Division.** Here, it is necessary to process two divisions of unipolar fuzzy relations (Expression (8.29)), which means that the practical complexity is multiplied by two, but the class of data complexity stays the same. See Bosc et al. (1999b) and Section 7.6 for more details on the processing of a division of unipolar fuzzy relations.

To sum up, one can be reasonably optimistic about the data complexity of bipolar queries based on the extended relational algebra described here, and therefore about the performances of a “bipolar DBMS”. Indeed, introducing bipolarity into relations and queries does not modify the complexity class of any algebraic operator (all of them remain in PTIME, and a naïve evaluation simply leads to a doubling of the cost with respect to an equivalent non-bipolar query).
8.5 Conclusion

In this chapter, we have defined an extension of relational algebra suited to the handling of bipolar fuzzy relations and conditions. This corresponds to a type of extension of database query languages which differs in nature from those considered in the other chapters of this book. Here, the goal was not to introduce some more sophisticated fuzzy conditions, but rather to take into account the fact that the human mind often makes decisions on the basis of both positive and negative information. Negative preferences correspond to constraints, since they specify those values or objects which have to be rejected (i.e., those that do not satisfy the constraints), while positive preferences correspond to wishes, as they specify those objects which are more desirable than others. The framework described in this chapter makes it possible for a user to express twofold requirements made of a (possibly complex) constraint and a (possibly complex) wish. The satisfaction of the constraint and of the wish can be used to order the tuples of the result by means of lexicographic order (with priority given to the constraint).

To the best of our knowledge, this is the first complete algebraic framework aimed at handling bipolar fuzzy relations and conditions. We should mention, however, that a first step in that direction was made in Liétard et al. (2009) where the selection, projection and join operations were briefly tackled (but only a non-bipolar version of the join was considered, and the set-oriented operators were not formally defined). Zadrozny and Kacprzyk (2006) consider a specific interpretation of bipolar fuzzy selection queries. Instead of using the lexicographic order to combine constraints and wishes, they aim at generalizing the behavior of the winnow operator proposed by Chomicki (2003) in a framework of preference queries based on a dominance relation. Matthé and De Tré (2009) also deal with the interpretation of bipolar fuzzy selection queries. They define so-called satisfaction and dissatisfaction degrees following an approach closely related to Atanassov’s intuitionistic fuzzy sets (the only difference with respect to Atanassov’s approach is that the consistency condition is dropped). The authors discuss the operators needed for evaluating selection queries involving the standard logical operators conjunction, disjunction and negation, but do not attempt to define any bipolar relational algebra.

We do not claim that the approach described in this chapter is the only one possible for handling bipolarity in a database context, but it does provide a consistent framework (contrary to any approach based on twofold fuzzy sets or intuitionistic fuzzy sets, because of the problem related to the
negation in particular). If one wants to handle two separate degrees — which corresponds to a strict view of bipolarity — and obtain a total ordering of the results, lexicographic order looks like a rather straightforward choice, with a clear interpretation.

The bipolar fuzzy extension of relational algebra defined in this chapter can obviously be used as the basis for definition of a bipolar fuzzy SQL-like language, e.g., a bipolar extension of SQLf. However, it is still a matter for future research to investigate how those constructs that are specific to SQLf — for instance nesting operators, the relation partitioning mechanism, and group-oriented conditions — could be generalized in order to i) take bipolar fuzzy relations as inputs, ii) be endowed with bipolar behavior.
This page intentionally left blank
Chapter 9

Fuzzy Group By

9.1 Introduction

As noted in Silva et al. (2009), grouping capabilities have been extensively studied and implemented in data management systems. The standard group by operator has relatively good execution time and scalability properties. However, while its semantics are simple, it is also limited because it is based only on equality, i.e., all the tuples in a group have exactly the same values of the grouping attributes. In this chapter, we show how this core database operation can be extended by defining a “fuzzy grouping” mechanism. We do the following:

- introduce the fuzzy group by (FGB) operator, which extends standard group by to allow the formation of groups based on predefined fuzzy partitions of the attribute domains rather than equality of data,
- show how this mechanism makes it possible to perform some kind of data summarization “on demand”,
- point out the relevance of the FGB operator to fuzzy association rule mining.

More precisely, the following section introduces the definition of the FGB operator and shows how fuzzy summaries can be obtained by means of appropriate aggregates applied to the fuzzy groups that are produced by this operator. Thereafter, we discuss the different forms that the complementary having clause can take. A special focus is put on the way FGB can be used for mining fuzzy association rules whose head or body are bound to a specific fuzzy value. Implementation aspects are tackled next, and the chapter ends with a discussion about related work.
9.2 An Extended Group By Clause

In SQLf as in SQL, a group by clause builds a partition based on the (atomic) values of the attributes specified in this clause (see Section 2.3.3). Then, “group by A” leads to a partition where every group is associated with an A-value present in the relation. The idea we advocate here is to extend this mechanism so as to build a partition based on intervals or fuzzy sets of values.

9.2.1 Use of a crisp partition

The generic form of such a query is:

\[
\begin{align*}
\text{select} & \text{ label}(A) \left[ , \text{ aggregate}, \ldots \right] \text{ from } r \left[ \text{ where } \psi \right] \\
\text{group by} & \text{ label}(A) \\
\text{using} & \text{ part}(A) = \{L_1,\ldots,L_n\},
\end{align*}
\]

where \( \text{part}(A) \) is a partition defined on the domain of \( A \), \( \text{label}(A) \) denotes any label \( L_i \) from \( \text{part}(A) \), and \( \psi \) is a (fuzzy or crisp) condition.

Example 9.1. Let \( \text{Emp} \) be a relation of schema \((#e, e-name, position, age, w-dep, salary)\). Assume that we want to retrieve the average salary for each age class (twenties, thirties, etc.). In SQL, these have to be as many queries as there are age classes. However, it is possible to imagine an expression of the type:

\[
\begin{align*}
\text{select} & \text{ label}(age), \text{ avg}(salary) \text{ from } \text{Emp} \\
\text{group by} & \text{ label}(age) \\
\text{using} & \text{ part}(age) = \{[20, 29], [30, 39], [40, 49], [50, 59]\}.
\end{align*}
\]

With the data from Table 9.1, the result is:

\[
\{([20, 29], 2650), ([30, 39], 3200), ([40, 49], 4500), \\
([50, 59], 6100), ([60, 69], 3700) \}.
\]

In the case where \( \psi \) is a fuzzy condition, the only type of aggregate which can appear in the select clause is \text{count}, because of the difficulty of defining other aggregates on fuzzy sets (see Chapter 6). Indeed, the existing approaches dealing with the interpretation of aggregates in the general case (Dubois and Prade, 1990) cannot be used in the framework of SQLf, since they deliver a fuzzy set of possible evaluations (only allowing the lower and upper bounds of this fuzzy set of numbers to be computed). In SQLf, a
Fuzzy Group By

Table 9.1 Extension of relation Emp

<table>
<thead>
<tr>
<th>#e</th>
<th>e-name</th>
<th>position</th>
<th>age</th>
<th>w-dep</th>
<th>sal(k$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>Smith</td>
<td>engineer</td>
<td>51</td>
<td>3</td>
<td>65</td>
</tr>
<tr>
<td>76</td>
<td>Martin</td>
<td>engineer</td>
<td>40</td>
<td>5</td>
<td>45</td>
</tr>
<tr>
<td>26</td>
<td>Jones</td>
<td>secretary</td>
<td>24</td>
<td>3</td>
<td>19</td>
</tr>
<tr>
<td>12</td>
<td>Green</td>
<td>technician</td>
<td>39</td>
<td>3</td>
<td>32</td>
</tr>
<tr>
<td>19</td>
<td>Duncan</td>
<td>clerk</td>
<td>28</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>8</td>
<td>Brown</td>
<td>manager</td>
<td>54</td>
<td>1</td>
<td>57</td>
</tr>
<tr>
<td>31</td>
<td>Harris</td>
<td>technician</td>
<td>29</td>
<td>5</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>Davis</td>
<td>janitor</td>
<td>61</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>44</td>
<td>Howard</td>
<td>manager</td>
<td>22</td>
<td>3</td>
<td>45</td>
</tr>
<tr>
<td>23</td>
<td>Lewis</td>
<td>engineer</td>
<td>62</td>
<td>1</td>
<td>59</td>
</tr>
</tbody>
</table>

A single degree of satisfaction attached to a condition such as \(\text{agg}(A)\) is \(C\) — where \(\text{agg}\) denotes an aggregate and \(C\) a fuzzy condition — is needed in order to maintain compositionality.

For a given \(L_i\) from \(\text{part}(A)\), \(\text{count}(L_i)\) is computed as follows:

\[
\text{count}(L_i) = \sum_{t \in r \land t.A \in L_i} \mu_{\psi}(t).
\]

A variant of \(\text{count}\), denoted \(\text{count-rel}\), can also be introduced to compute the average membership degree inside a group. It is defined as:

\[
\text{count-rel}(L_i) = \frac{\sum_{t \in r \land t.A \in L_i} \mu_{\psi}(t)}{|\{t \in r \mid t.A \in L_i\}|}.
\]

Example 9.2. Consider the query:

\[
\text{select label(age), count, count-rel from Emp where salary is medium group by label(age)}
\]

\[
\text{using part(age) = \{[20, 29], [30, 39], [40, 49], [50, 59]\}.}
\]

With the fuzzy term \(\text{medium}\) defined as in Figure 9.1, and the data from Table 9.1, one obtains:

\[
\{(\langle 20, 29 \rangle, 1, 0.25), \langle 30, 39 \rangle, 0.7, 0.7), \langle 40, 49 \rangle, 1, 1),
\langle 50, 59 \rangle, 0.8, 0.4),\langle 60, 69 \rangle, 0.6, 0.3\}\).
\]

9.2.2 Use of a fuzzy partition

The extension to the fuzzy partition case — which allows vague classes to be taken into account, and permits more robustness by making query results less sensitive to the boundaries of the classes — is rather straightforward.
It is simply necessary to have available fuzzy partitions defined on the attribute domains. However, it is no longer possible to use any type of aggregate in the select clause: one is limited to using count even when the condition in the where clause is Boolean (since the groups themselves are fuzzy). In the following, we assume that Ruspini partitions (Ruspini, 1969) are used, i.e.,

$$\forall x \in X, \sum_{L_i \in P} \mu_{L_i}(x) = 1,$$

where $P$ denotes a partition defined on domain $X$ (see Figure 9.1).

**Example 9.3.** Consider the partition from Figure 9.1 (left) denoted by $\text{part}(\text{age})$, and the query “find, for each fuzzy age class, the number of employees who earn more than $30,000$”. It can be expressed as:

```sql
select label(\text{age}), count from Emp where salary > 30k group by label(\text{age}) using part(\text{age}) = \{\text{young}, \text{middle-aged}, \text{old}\}.
```

With the data from Table 9.2, the result is: \{\langle\text{young}, 0.8\rangle, \langle\text{middle-aged}, 3.7\rangle, \langle\text{old}, 1.5\rangle\} where each number returned corresponds to a $\Sigma$-count.

We now have the following definitions:

$$\text{count}(L_i) = \sum_{t \in r} \top(\mu_\psi(t), \mu_{L_i}(t.A)), \quad (9.1)$$

$$\text{count-rel}(L_i) = \frac{\sum_{t \in r} \top(\mu_\psi(t), \mu_{L_i}(t.A))}{\sum_{t \in r} \mu_{L_i}(t.A)}. \quad (9.2)$$

The fuzzy group by clause makes it possible to compute fuzzy summaries “on demand”, in contrast with the approach proposed in Saint-Paul et al. (2005) which first builds a summary of the whole database, then uses it to...
answer queries. An example of a query aimed at providing a fuzzy summary is given below.

**Example 9.4.** Consider the query: “what are the ages of the employees whose salary is medium?”. This can be expressed as:

```sql
select label(age), count from Emp
where salary is medium group by label(age)
using part(age) = {young, middle-aged, old}.
```

With the data from Table 9.2, and using $\top = \min$, one obtains:

\[
\{(\text{young}, 0.8), (\text{middle-aged}, 2.5), (\text{old}, 1.0)\}\.
\]

**Example 9.5.** Now consider the query: “what proportion of employees from each age class has a salary which is medium?”. This can be expressed as:

```sql
select label(age), count-rel from Emp
where salary is medium group by label(age)
using part(age) = {young, middle-aged, old}.
```

With the data from Table 9.2, and using $\top = \min$, one obtains:

\[
\{(\text{young}, 0.47), (\text{middle-aged}, 0.43), (\text{old}, 0.4)\}\.
\]

### 9.3 Having Clause

The different forms of a *having* clause that can be used as a complement to a *group by* clause are described below through the use of a few examples.
9.3.1 Inclusion constraint

An example which involves a Boolean condition in the having clause is: “find every age class such that at least 30% of the employees from that class have a high salary”:

```sql
select label(age) from Emp E1
group by label(age)
having count ≥
   (select count * 0.3 from Emp
    where salary is high and age is E1.label(age))
using part(age) = {young, middle-aged, old}.
```

Another example, which involves a fuzzy having clause, is: “find the extent to which all the employees of a given age class have a high salary”:

```sql
select label(age) from Emp E1
group by label(age)
having (select #e from Emp where salary is high) contains set(#e)
using part(age) = {young, middle-aged, old}.
```

The evaluation of such a query rests on a graded inclusion (see Section 3.4.2). It constitutes the prototype expression for fuzzy association rule mining, and will be detailed in Section 9.4.

9.3.2 Aggregate₁ θ aggregate₂

Even though a limitation exists as to the aggregates which can be computed on a fuzzy set — as already mentioned — it is still possible to evaluate conditions which compare two aggregates (Bosc et al., 2003b). The idea is to start with a definition that is valid for crisp sets, and then to extend it to fuzzy sets. In the case where $A$ and $B$ are crisp, it is possible to express the meaning of the statement $\text{agg}_1(A) \leq \text{agg}_2(B)$ using an implication according to the formula:

\[ \forall x, \text{[agg}_1(A) \geq x] \Rightarrow \text{[agg}_2(B) \geq x], \tag{9.3} \]

where $x$ is used to scan the definition domain of $\text{agg}_1$ and $\text{agg}_2$.

When $A$ and $B$ are both fuzzy sets, the expression $\text{agg}_1(A) \geq x$ (resp. $\text{agg}_2(B) \geq x$) is more or less satisfied. Its degree of truth $t(\text{agg}_1(A) \geq x)$ (resp. $t(\text{agg}_2(B) \geq x)$) can be obtained by considering
Fuzzy Group By

the different $\alpha$-level cuts of predicate $A$:

$$t(\text{agg}(A) \geq x) = \max_{\alpha \in [0, 1]} \min(\alpha, \mu_{\geq x}(\text{agg}(A_\alpha))).$$

Since $\geq x$ is a Boolean predicate, its truth value is either 0 or 1, and we have:

$$t(\text{agg}(A) \geq x) = \max_{\alpha \in [0, 1]} \text{such that agg}(A_\alpha) \geq x \alpha.$$  \hfill (9.3)

If the universal quantifier in Expression (9.3) is interpreted as a generalized conjunction, the satisfaction degree of $\text{agg}_1(A) \leq \text{agg}_2(B)$ is given by:

$$\min_{x \in D} t(\text{agg}_1(A) \geq x) \Rightarrow_f t(\text{agg}_2(B) \geq x),$$

where $\Rightarrow_f$ stand for a fuzzy implication (Fodor and Yager, 2000) and $D$ is the definition domain of $\text{agg}_1$ and $\text{agg}_2$. More detail can be found in Bosc et al. (2003b).

An example of such a query is: “find the age classes such that the maximum of the technicians’ salaries is greater than the minimum of the engineers’ salaries”. It can be expressed as:

```sql
select label(age) from Emp E1 where job = “technician”
group by label(age)
having max(salary) >
(select min(salary) from Emp
 where age is E1.label(age) and job = “engineer”)
using part(age) = \{young, middle-aged, old\}.
```

In such a query, the where clause could also involve a fuzzy condition.

9.3.3 Aggregate is $\psi$

It is also possible to compute the extent to which an aggregate satisfies a fuzzy condition $\psi$, by means of the approach proposed in Bosc and Liétard (2008) and presented in Section 6.3.2. For instance, when both the aggregate and the predicate $\psi$ are increasing, one may start from the following definition of the statement $\text{agg}(A) \text{ is } \psi$ when $A$ is a crisp set and $\psi$ is a Boolean condition:

$$\text{agg}(A) \text{ is } \psi \iff \exists n \text{ such that } \psi(n) \text{ and } \text{agg}(A) \geq n.$$  \hfill (9.5)
When \( A \) is a fuzzy set and \( \psi \) is a fuzzy condition, the preceding expression may be generalized into:

\[
t(\text{agg}(A) \text{ is } \psi) = \max_{\alpha \in [0, 1]} \min(\alpha, \mu_{\psi}(\text{agg}(A))).
\] (9.6)

The case where \( \text{agg} \) is not monotonous is more tricky, but can still be dealt with; see Section 6.3.3 and Bosc and Liétard (2008).

An example of a query involving a condition of the form \( \text{agg}(A) \text{ is } \psi \) inside a \textit{having} clause is: “find the age classes where the average salary of engineers is high”. It can be expressed as:

```sql
select label(age) from Emp where job = 'engineer'
group by label(age)
having avg(salary) is high
using part(age) = {young, middle-aged, old}.
```

Here again, the \textit{where} clause may also involve a fuzzy condition.

### 9.4 Application to Association Rule Mining

We now explain how the fuzzy \textit{group by} clause can be used for evaluating, in a simple way, fuzzy association rules of the type \((A \text{ is } L_i \rightarrow B \text{ is } L_j')\), where \( L_i \) and \( L_j' \) are two fuzzy labels defined respectively on the domains of \textit{age} and \textit{salary}. As mentioned in Bosc and Pivert (2001), at least two fuzzy extensions of association rules may be considered: those based on (fuzzy or scalar) cardinalities, and those based on fuzzy implications (see also Hüllermeier (2001) for the latter category). In what follows, we do not deal with fuzzy-cardinality-based rules, since they may be somewhat difficult to interpret for an end-user (and also because they cannot be represented and handled easily in a purely relational DBMS).

The approach based on a scalar cardinality (\(\Sigma\)-count) rests on a straightforward extension of the usual definition of confidence. Consider a fuzzy association rule of the type \((A \text{ is } L_i \rightarrow B \text{ is } L_j')\). In this approach, the validity (confidence) of the rule is defined as:

\[
\frac{\sum_{t \in r} \tau(\mu_{L_i}(t.A), \mu_{L_j'}(t.B))}{\sum_{t \in r} \mu_{L_i}(t.A)}.
\]

In the fuzzy-implication-based approach, the rule \((A \text{ is } L_i \rightarrow B \text{ is } L_j')\) expresses a constraint on the \( B \)-value for each tuple in the relation. The association rule, also denoted by \((A, L_i) \rightarrow (B, L_j')\), then means: “for every
tuple \( t \), the more \( t.A \) is \( L_i \), the more \( t.B \) is \( L_i' \), i.e.:

\[
(A, L_i) \rightarrow (B, L_i') \Leftrightarrow \forall t \in r, \mu_{L_i}(t.A) \Rightarrow_{f} \mu_{L_i'}(t.B),
\]

where \( \Rightarrow_{f} \) denotes a fuzzy R-implication. In this case, the confidence of the rule is equal to:

\[
\min_{t \in r} \mu_{L_i}(t.A) \Rightarrow_{f} \mu_{L_i'}(t.B).
\]

### 9.4.1 Rules of the type \( A \) is \( L_i \) \( \rightarrow \) \( B \) is \( L' \)

#### 9.4.1.1 Computation of the support

First, we introduce a variant of \( \text{count} \) named \( \text{count-g} \) whose general definition is:

\[
\text{count-g}(L_i) = \frac{\sum_{t \in r} \top(\mu_{\psi}(t), \mu_{L_i}(t.A))}{|r|},
\]

where \( \psi \) denotes the fuzzy condition which appears in the \( \text{where} \) clause of the query. Expression (9.7) computes the proportion of tuples of \( r \) which are \( \psi \land L_i \). Consider the generic SQLf query:

\[
\text{select label}(A), \text{count-g} \text{ from } r \text{ where } B \text{ is } L' \text{ group by label}(A) \text{ using part}(A) = \{ L_1, \ldots, L_n \}.
\]

This allows for computation of the support of every fuzzy association rule of the type “\( A \) is \( L_i \) \( \rightarrow \) \( B \) is \( L' \)” for a given \( L' \).

#### 9.4.1.2 Computation of the confidence

Consider the generic SQLf query:

\[
\text{select label}(A) \text{ from } r \text{ group by label}(A) \text{ having } (\text{select } K \text{ from } r \text{ where } B \text{ is } L') \text{ contains set}(K) \text{ using part}(A) = \{ L_1, \ldots, L_n \},
\]

where \( K \) denotes the primary key of the relation.

**Scalar cardinality.** Using an approach based on scalar cardinality, the confidence of the rule corresponds to the *cardinality-based* degree of inclusion of the fuzzy set:

\[
E(L_i) = (\text{select } \# e \text{ from } r \text{ where } A \text{ is } L_i)
\]
in the fuzzy set:

\[ F = \left( \text{select } \#e \text{ from } r \text{ where } B \text{ is } L^\prime \right), \]

i.e., to the degree \( \mu \) produced by the evaluation of the \textit{having} clause:

\[ \mu = \frac{\sum_{t \in r} \top \left( \mu_{L^\prime}(t.B), \mu_{L_i}(t.A) \right)}{\sum_{t \in r} \mu_{L_i}(t.A)}. \]

**R-implication.** Using an approach based on a fuzzy R-implication, the confidence of the rule corresponds to the \textit{implication-based} degree of inclusion of \( E(L_i) \) in \( F \) — see Chapter 3, Expression (3.10) — i.e., to the degree \( \mu \) produced by the evaluation of the \textit{having} clause when \textit{contains} is replaced by \textit{contains-f}:

\[ \mu = \min_{t \in r} \mu_{L_i}(t.A) \Rightarrow \mu_{L^\prime}(t.B). \]

For instance, with Lukasiewicz’s implication (\textit{contains-Lu}), we have:

\[ \mu = \min_{t \in r} \min(1, 1 - \mu_{L_i}(t.A) + \mu_{L^\prime}(t.B)). \]

**Example 9.6.** Consider the set of rules of the form:

\( age \text{ is } L_i \rightarrow salary \text{ is } \text{medium}, \)

where \( L_i \) belongs to \( p(age) \). These rules can be evaluated by means of the following two queries (the first query computes the supports, the second the associated confidence degrees):

\[ \text{select label}(age), \text{count-g from Emp where salary is medium group by label}(age) \]

\text{using part}(age) = \{young, middle-aged, old\},

which, using the data from Table 9.1, the partitions from Figure 9.1, and \( \top = \min \), returns:

\( \{0.08/\langle \text{young} \rangle, 0.25/\langle \text{middle-aged} \rangle, 0.1/\langle \text{old} \rangle\} \).

The second query:

\[ \text{select label}(age) \text{ from Emp group by label}(age) \]

\text{having (select } \#e \text{ from Emp where salary is medium contains set(\#e) using part}(age) = \{young, middle-aged, old\} \]

returns:

\( \{0.47/\langle \text{young} \rangle, 0.43/\langle \text{middle-aged} \rangle, 0.4/\langle \text{old} \rangle\} \).
If \textit{contains} were replaced by \textit{contains-Lu} in the second query, we would obtain:

\[
\{0.4/(\text{young}), 0.1/(\text{middle-aged}), 0/(\text{old})\}.\circ
\]

### 9.4.2 Rules of the type \(A\) is \(L\) \(\rightarrow\) \(B\) is \(L'_i\)

Now consider mining fuzzy association rules of the type “\(A\) is \(L\) \(\rightarrow\) \(B\) is \(L'_i\)” for a given \(L\). Again, two SQLf queries are necessary:

\begin{verbatim}
select label(B), count-g from r where A is L
  group by label(B)
  using part(B) = \{L'_1, \ldots, L'_n\}
\end{verbatim}

for computing the support of the rules considered, and:

\begin{verbatim}
select label(B) from r group by label(B)
  having set(K) contains (select K from r where A is L)
  using part(B) = \{L'_1, \ldots, L'_n\}
\end{verbatim}

for computing their confidence values.

**Example 9.7.** Consider the set of rules of the form:

\[\text{age is young} \rightarrow \text{salary is } L'_i,\]

where \(L'_i\) belongs to \(p(\text{salary})\). These rules can be evaluated by means of the following two queries (the first query computes the supports, the second the associated confidence degrees):

\begin{verbatim}
select label(salary), count-g from Emp where age is young
  group by label(salary)
  using part(salary) = \{low, medium, high\}
\end{verbatim}

and

\begin{verbatim}
select label(salary) from Emp group by label(salary)
  having set(#e) contains (select #e from Emp where age is young)
  using part(salary) = \{low, medium, high\}.
\end{verbatim}

With the data from Table 9.1 and the partitions from Figure 9.1, the first query returns:

\[
\{0.09/(\text{low}), 0.08/(\text{medium}), 0/(\text{high})\}.
\]
The second query — which corresponds to the scalar cardinality approach since it involves the operator \textit{contains} — returns the result:
\[
\{0.53/\langle\text{low}\rangle, 0.47/\langle\text{medium}\rangle, 0/\langle\text{high}\rangle\}.
\]
If \textit{contains} were replaced by \textit{contains-Lu} in the second query, we would have:
\[
\{0.2/\langle\text{low}\rangle, 0.4/\langle\text{medium}\rangle, 0.2/\langle\text{high}\rangle\}.
\]

\section{9.5 Evaluation of a Fuzzy Group By}

The complexity of the evaluation of an FGB clause is very similar to that of a regular \textit{group by} clause. Two cases may be distinguished:

- if the attribute appearing in the FGB clause — denote it by \(A\) — is indexed, it is possible to directly access the tuples which belong to a given fuzzy class \(L_i\); they are the tuples \(t\) such that \(t.A \in \text{support}(L_i)\). Recall that the support of a fuzzy label is expressed as an interval, and can be straightforwardly determined from the membership function associated with that label. The degree to which tuple \(t\) belongs to class \(L_i\) is equal to \(\mu_{L_i}(t.A)\).

- otherwise, as usual, one may sort the relation on attribute \(A\), and compare the \(A\)-value of each tuple with the (overlapping) segments which correspond to the supports of the different fuzzy labels in the partition of \(\text{domain}(A)\) in order to build the fuzzy groups. Here too, of course, \(\mu_{L_i}(t.A)\) must be computed for each tuple \(t\).

As can be seen, the cost of the evaluation of an FGB clause should be more or less equivalent to that of a regular \textit{group by} clause, since the only additional cost is that related to the computation of the membership degrees.

\section{9.6 Related Work}

\subsection{9.6.1 Extended group by}

Zhang and Huang (2007) propose some SQL constructs to make clustering facilities available from SQL in the context of spatial data. Basically, these constructs act as wrappers of conventional clustering algorithms, but no further integration with database systems is studied. Li \textit{et al.} (2007) extend the \textit{group by} operator to approximately cluster all the tuples in a predefined
number of clusters. Their framework makes use of conventional clustering algorithms, e.g., K-means, and employs summaries and bitmap indexes to integrate clustering and ranking into database systems. Silva et al. (2009) introduce a similarity group by operator in order to group objects with similar values. Our approach differs from Li et al. (2007) and Silva et al. (2009) in that:

- we focus on fuzzy grouping based on vague concepts, not on similarity-based grouping;
- we do not aim at “discovering” the clusters, since in our approach the groups are explicitly specified in the query (by means of a fuzzy partition), which incidentally gives them a well-identified meaning;
- Li et al. (2007) and Silva et al. (2009) do not consider a general fuzzy querying framework such as SQLf (where the fuzzy group by construct is just one piece of the puzzle) but only extend a particular feature of SQL.

9.6.2 Fuzzy OLAP

A few research works, e.g., Delgado et al. (2007), Kaya and Alhajj (2008), have been devoted to the introduction of fuzziness into OLAP systems. These approaches have some characteristics in common with that presented here (use of fuzzy partitions, fuzzy association rule mining) but they do not rely on a general purpose database querying language such as SQLf. They instead extend operators such as roll-up and drill-down, or devise specific rule mining algorithms.

9.6.3 Fuzzy database summarization techniques

Developed by Rasmussen and Yager (1997), SummarySQL is a fuzzy query language which can evaluate the truth degree of a summary guessed by the user. A summary expresses knowledge about the database in a statement of the form “$Q$ objects in $DB$ are $S$” or “$Q$ $R$ objects in $DB$ are $S$” where $DB$ stands for the database, $Q$ is a linguistic quantifier, and $R$ and $S$ are linguistic terms. The expression is evaluated for each tuple and the associated truth values are later used to obtain a truth value for the summary. The statements considered by the authors are in a sense more general than the fuzzy association rules that we deal with here, since they involve fuzzy quantifiers. However, our approach can easily be extended to capture such statements by relaxing the operator contains that appears in the having clause, using, e.g., one of the approaches described in Bosc
and Pivert (2008). When it comes to mining fuzzy statements, the main
difference lies in the fact that Rasmussen and Yager (1997) do not propose
any SQL construct to evaluate these statements “in a batch” as we do via
the FGB clause: the statements have to be checked one by one and no fuzzy
partitioning of the domains is used.

Saint-Paul et al. (2005) propose an approach to the production of lin-
guistic summaries structured in a hierarchy, i.e., a summarization tree where
the tuples from the database are rewritten using the linguistic variables
involved in fuzzy partitions of the attribute domains. The main difference
with our approach is that Saint-Paul et al. (2005) view summarization as an
independent process, which is not performed by means of SQL queries but
by a specific algorithm. As mentioned before, the FGB operator enables
summaries “on demand”, without having to summarize the whole database.

9.6.4 Mining association rules with SQL

The use of SQL queries for mining association rules has been advocated by
several authors; see, e.g., Meo et al. (1998), Clear et al. (1999), Thomas
and Sarawagi (1998), Yoshizawa et al. (2000), Imielinski and Virmani
(1999), Rajamani et al. (1999), Pereira et al. (2003). However, none of these
approaches considers an extended group by mechanism, and none consid-
ers fuzzy association rules either. To the best of our knowledge, the only
approach which uses a fuzzy extension of SQL for mining fuzzy association
rules (or “gradual functional dependencies”, as the authors call them) is
that of Rasmussen and Yager (1999), which relies on SummarySQL already
discussed above.

9.7 Conclusion

In this chapter, we have introduced a fuzzy group by (FGB) operator, and
have described how it could be integrated into the SQLf language. This
operator relies on user-specified fuzzy partitions of attribute domains, and
its main goal is to generate meaningful and useful groupings that go far
beyond what is captured by the regular group by operator. We have shown
how this construct makes it possible to generate fuzzy summaries “on
demand”, as well as to mine fuzzy association rules in a practical way. It is
of course important to make sure that queries involving a fuzzy grouping
have execution times comparable to those involving a classical group by. One
can be reasonably optimistic about this issue, given the results presented in Silva et al. (2009) about a similarity-based group by (SGB), which show that the overhead in this case is no more than 25%. FGB should be even more efficient than SGB since (i) the clusters are predefined, (ii) the use of fuzzy partitions still makes it possible to employ evaluation techniques based on sorts and/or indexes.

Among possible extensions of this work, the following are noteworthy:

- An investigation into the way measures other than support and confidence for assessing fuzzy association rules, see for example Dubois et al. (2006), could be taken into account.
- A variant of the approach described here, aimed at expert users, where fuzzy cardinalities would be computed instead of scalar ones. In the context of fuzzy association rule mining, this implies computation of the validity of the rules by means of a technique such as that described in Bosc et al. (2001).
- An extension of the format of the rules to be mined, for instance through a relaxation of the universal quantifier based on one of the approaches proposed in Bosc and Pivert (2008) (see also Section 3.4.2.5).
Chapter 10

Empty and Plethoric Answers

10.1 Introduction

When compared to Boolean queries, fuzzy queries reduce the risk of obtaining an empty set of answers, since the use of a finer discrimination scale — [0, 1] instead of {0, 1} — increases the chance of an element being considered (somewhat) satisfactory. Nevertheless, the situation may occur where none of the elements of the database satisfies the query even to a low degree. As noted by Motro (1986), there are two possible explanations for the emptiness of a set of answers: either the query has some “semantic flaw” (for example it includes a false presupposition), or the reason lies only in the current state of the database. In the case of a semantic flaw, an explanation may help the user reformulate his/her query. This generally involves having available some additional knowledge about the structure and the content of the database, for instance under the form of integrity constraints. On the other hand, if an empty set of answers is due only to the absence of any satisfactory element from the current database, a solution consists in retrieving data which are as suitable as possible with respect to the information requirement expressed by the initial query. In this chapter, we only consider this latter situation, and we present different techniques that can be used for relaxing conjunctive fuzzy queries.

The “plethoric answer problem” (PAP) — which may be seen as the companion to the “empty answer problem” (EAP) — occurs when the amount of returned data is too large to be manageable. In such cases, users have to go through this large set of answers to examine them and keep only the most relevant ones, which is a tedious and time-consuming task. The PAP has been addressed at some length by the information systems community, and two main approaches have been proposed. The first approach
Fuzzy Preference Queries to Relational Databases

aims at summarizing the result either by rewriting the answers using a coarser vocabulary for describing the attribute domains (see, e.g., Ughetto et al., 2008), or by keeping only the most typical ones (see, e.g., Hua et al., 2009). The second approach aims at strengthening the initial query in order to make it more selective (either by adding new conditions or by making some of the initial conditions more drastic). However, not much work can be found in the literature about the PAP in a context of fuzzy queries.

This chapter is organized as follows. Section 10.2 deals with the empty answer problem in a context of conjunctive fuzzy queries, and describes two approaches that may be used to overcome it. The first approach is based on the use of a modifier which softens one or several fuzzy predicates from the query, whereas the second takes advantage of a repository of previously submitted queries, and aims at replacing the failing query with a semantically close one, known to be non-failing. Section 10.3 deals with the plethoric answer problem and also describes two approaches aimed at dealing with it. In the first approach, a strengthening modifier is used to make the fuzzy predicates of the query more selective, whereas the second approach expands a fuzzy query $Q$ with new predicates selected from among those that are semantically correlated with $Q$.

10.2 Empty Answer Problem

As mentioned above, when the set of answers to a query is empty, rather than returning an empty set, a cooperative system may:

- exploit some domain knowledge in order to explain why no answer was found, thus helping the user reformulate his/her query,
- exploit some domain knowledge so as to automatically construct a modified (relaxed) query.

Between these two extreme possibilities, different ways of interacting with the user may be imagined. In the following, we mainly consider the second strategy. We first describe an approach that uses a weakening mechanism in order to make one or more predicates of the initial fuzzy query more tolerant.

10.2.1 Query relaxation

Query relaxation consists of modifying a query in order to find a less restrictive variant. Different methods for relaxing the constraints involved
Empty and Plethoric Answers

in a query have been defined in the literature (see, for instance, Cuppens and Demolombe, 1988; Guyomard and Siroux, 1989; Motro, 1990; Chu and Chen, 1992; Gaasterland et al., 1992b). Similar work has been carried out in the framework of information retrieval, starting with Salton et al. (1983). Most of these approaches rely on a “semantic distance” which makes it possible to determine a modified query that is expected to be as close as possible to the initial one, semantically speaking.

The general mechanism for weakening a query \( Q \) may be described in the following way: one must replace \( Q \) by its “best” modification \( Q' \) in such a way that:

1. \( Q' \) is semantically close to \( Q \) in the sense that the elements returned by \( Q' \) are still somewhat relevant with respect to the user’s need expressed by \( Q \);
2. the set of answers to \( Q' \) is non-empty.

In the Boolean case, two main approaches can be used to relax a conjunctive Boolean query when it fails to produce any answer:

- relaxation by generalization: transforming query conditions into more general ones (Motro, 1986);
- relaxation by removal: it aims at deleting some parts of the query in order to obtain a subquery that is less constraining (Godfrey, 1997).

In the case of conjunctive fuzzy queries, one may either relax some of the atomic conditions, or replace the conjunction by a less drastic operator (such as a mean or a quantifier of the form “almost all”). In the following, we only consider the first strategy, i.e., that based on predicate weakening, which has the advantage of preserving the conjunctive nature of the query.

### 10.2.2 Relaxation by predicate weakening

#### 10.2.2.1 Principle

Let \( Q \) be a fuzzy query to a given relational database, and let \( \Sigma^\alpha_Q \) (resp. \( \Sigma^0_Q \)) be the set of answers to \( Q \) whose satisfaction degree is at least equal to \( \alpha \) (resp. is non-zero).

In a fuzzy querying framework, the simplest way to relax a query consists of lowering the qualitative threshold present in the query, if any. For instance, a query involving a threshold equal to 0.8 may be relaxed...
successfully by reducing this threshold to 0.6. In the following, we do not consider this trivial case and we assume that an empty set of answers means that $\Sigma_Q^0$ is empty.

Relaxing a fuzzy query $Q$ involves weakening the fuzzy constraints contained in $Q$ in order to make $Q$ less selective. Such a modification can be achieved by applying a basic transformation to all or some of the predicates of the query. Consider a transformation $T$ applied to a predicate $P$ ($T^\uparrow(P)$ representing the modified predicate). $T$ is said to be weakening if it satisfies the following properties:

- **RC1:** $T^\uparrow$ does not decrease the membership degree for any element of the domain, i.e., $\forall u \in \text{domain}(A), \mu_{T^\uparrow(P)}(u) \geq \mu_P(u)$, where $A$ denotes the attribute concerned by $P$;
- **RC2:** $T^\uparrow$ must extend the support $\text{supp}(P)$ of predicate $P$, i.e.,
  \[
  \text{supp}(P) = \{ u | \mu_P(u) > 0 \} \subset \text{supp}(T^\uparrow(P)) = \{ u | \mu_{T^\uparrow(P)}(u) > 0 \};
  \]
- **RC3:** $T^\uparrow$ preserves the specificity of predicate $P$ (in order not to alter its semantics too much), i.e.,
  \[
  \text{core}(P) = \{ u | \mu_P(u) = 1 \} = \text{core}(T^\uparrow(P)) = \{ u | \mu_{T^\uparrow(P)}(u) = 1 \}.
  \]

Most of the usual modifiers (see, e.g., Kerre and de Cock, 1999) do not fulfill these three properties. However, in Bouchon-Meunier (1988), a parameterized modifier called $\nu$-rather is introduced, whose behavior is described in Figure 10.1, and which is well suited to the problem considered here, since it does comply with the three requirements above. An approach to the relaxation of fuzzy conjunctive queries based on $\nu$-rather is described in Andreasen and Pivert (1994, 1995). In the following, we present an alternative approach based on the use of a tolerance relation defined on

\[Fig. 10.1 \text{ Behavior of } \nu\text{-rather}\]
the attribute domains and the notion of *dilation*. The advantage of this approach over that based on $\nu$-rather lies in its more intuitive semantics, as discussed in Bosc *et al.* (2007a).

10.2.2.2 *Dilation mechanism*

**Definition 10.1.** A proximity relation (or a tolerance relation) is a fuzzy relation $E$ on a domain $X$, such that $\forall x, y \in X$,

- $\mu_E(x, x) = 1$ (reflexivity),
- $\mu_E(x, y) = \mu_E(y, x)$ (symmetry).

The quantity $\mu_E(x, y)$ can be viewed as a grade of approximate equality of $x$ with $y$. In a universe $X$ which is a subset of the real line, an absolute proximity relation can be conveniently modeled by a fuzzy relation $E$ of the form:

$$\mu_E(x, y) = \mu_Z(x - y),$$

which only depends on the value of the difference $x - y$. The parameter $Z$, called a tolerance indicator, is a fuzzy set on the real line centered in 0, such that:

1. $\mu_Z(r) = \mu_Z(-r)$. This property ensures the symmetry of the proximity relation $E$ (i.e., $\mu_E(x, y) = \mu_E(y, x)$),
2. $\mu_Z(0) = 1$, which states that $x$ is approximately equal to itself to a degree 1,
3. the support $\text{supp}(Z) = \{r | \mu_Z(r) > 0\}$ is bounded and is of the form $[-\Omega, \Omega]$, where $\Omega$ is a positive real number,
4. in terms of the trapezoidal membership function (TMF), $Z$ can be expressed by the quadruple $(-z, z, \delta, \delta)$ with $\Omega = z + \delta$ and $[-z, z]$ the core of $Z$ (i.e., $\{r, \mu_Z(r) = 1\}$).

It is important to emphasize that this kind of proximity relation evaluates the extent to which the amount $x - y$ is close to 0. The closer $x$ is to $y$, the closer $x - y$ and 0 become. Classical (or crisp) equality is recovered for $Z = 0$, defined as $\mu_0(x - y) = 1$ if $x = y$ and $\mu_0(x - y) = 0$ otherwise. Other interesting properties of the parameterized relation $E$ are given in Dubois *et al.* (2001). In what follows, we shall write $E[Z]$ to denote the proximity relation $E$ parameterized by $Z$. 
Fuzzy Preference Queries to Relational Databases

Dilation. Consider a fuzzy set $F$ on the numeric universe $U$ and an absolute proximity $E(Z)$, where $Z$ is a tolerance indicator. Dilating the fuzzy set $F$ by $Z$ will provide a fuzzy set $F^Z$ defined by

$$\mu_{F^Z}(r) = \sup_s \min(\mu_{E[Z]}(s,r), \mu_F(s))$$

$$= \sup_s \min(\mu_Z(r-s), \mu_F(s)), \text{ since } Z = -Z \quad (10.1)$$

$$= \mu_{F \oplus Z}(r), \text{ observing that } s + (r - s) = r.$$ 

Hence, $F^Z = F \oplus Z$, where $\oplus$ is the addition operation extended to fuzzy sets (Dubois and Prade, 1988). $F^Z$ gathers the elements of $F$ and the elements outside $F$ which are somewhat close to an element in $F$.

**Lemma 10.1.** Given a fuzzy set $F$ and a proximity relation $E[Z]$, for $F^Z = F \oplus Z$ we have $F \subseteq F^Z$.

In other words, the fuzzy set $F^Z$ is less restrictive than $F$, but still semantically close to $F$. Thus, $F^Z$ can be viewed as a relaxed variant of $F$. In terms of the TMF, if $F = (A, B, a, b)$ and $Z = (-z, z, \delta, \delta)$ then $F^Z = (A - z, B + z, a + \delta, b + \delta)$; see Figure 10.2. Recall that if $F = (A, B, a, b)$, then $\text{supp}(F) = [A - a, B + b]$ and $\text{core}(F) = [A, B]$. The scalar $a$ (resp. $b$) represents the left-hand (resp. right-hand) spread of $F$.

We emphasize that some practical applications require only one constituent part of a fuzzy set $F$ (either the core or the support) to be affected when applying a dilation operation. We denote by core dilation (resp. support dilation) the dilating transformation that affects only the core (resp. support) of $F$. The following proposition shows how to obtain such results.

![Fig. 10.2 Dilation operation](image-url)
Empty and Plethoric Answers

Proposition 10.1. Let \( F \) be a fuzzy set and \( E[Z] \) a proximity relation. We have:

- **Core dilation** is obtained using the family of tolerance indicators of the form \( Z = (-z, z, 0, 0) \).
- **Support dilation** is obtained using the family of tolerance indicators of the form \( Z = (0, 0, \delta, \delta) \).

Making use of this proposition, if \( F = (A, B, a, b) \) the core dilation (resp. support dilation) leads to \( F^Z = (A - z, B + z, a, b) \) (resp. \( F^Z = (A, B, a + \delta, b + \delta) \)).

10.2.2.3 Relaxing an atomic criterion

Let \( P \) be a fuzzy predicate and \( E[Z] \) an absolute proximity relation parameterized by a tolerance indicator \( Z \) of the form \( (0, 0, \delta, \delta) \) (i.e., such that only the support of \( P \) should be changed). Dilating \( P \) allows it to be transformed into an enlarged fuzzy predicate \( P' \), defined as follows:

\[
P' = T^\uparrow(P) = P \oplus Z.
\]

Clearly, the modified predicate \( P' \) contains the elements of \( P \) and the elements outside \( P \) which are somewhat close to an element in \( P \). Moreover, we can easily check that the desirable properties RC1–RC3 (Section 10.2.2.1) are satisfied by \( T^\uparrow \). Note that a transformation of the same nature was also proposed in the tolerant fuzzy pattern matching setting by Dubois and Prade (1995).

**Principle of the approach.** Let \( Q = P \) be an atomic query. If the set of answers to \( Q \) is empty, then \( Q \) is transformed into \( Q_1 = T^\uparrow(P) = P^Z \). This relaxation mechanism can be repeated \( n \) times until the answer to the modified query \( Q_n = T^\uparrow(n)(P) = P^{n.Z} = P \oplus n.Z \) is not empty. In practice, as pointed out in Bosc et al. (2007a), the only difficulty when applying this technique concerns its semantic limits (i.e., the maximum number of relaxation steps such that the final modified query \( Q_n \) is not too far, semantically speaking, from the original one). Indeed, there is no intrinsic criterion attached to this transformation which would stop the iterative process when the answer still remains empty (except of course the impossibility of going beyond the limits of the attribute domain concerned).
Controlling the relaxation. To enable some control over the relaxation process, one solution is to ask the user to specify, along with her/his query, a fuzzy set $F_P$ of more or less non-authorized values in the related domain. See Figure 10.3. The new query to be considered is now written as $Q'_i = Q_i \cap (F_P)^c$, where $(F_P)^c$ denotes the complement of $F_P$. The satisfaction degree of an element $u$ then becomes $\min(\mu_{Q_i}(u), 1 - \mu_{F_P}(u))$.

Thus the incremental relaxation process will now stop when the answer to $Q_i$ is not empty ($\Sigma_{Q_i}^0 \neq \emptyset$), or when the core of the complement of the fuzzy set associated with $Q_i$ is included in the core of $F_P$ ($\min(\mu_{Q_i}(u), 1 - \mu_{F_P}(u)) = 0$).

Relaxation algorithm. This incremental relaxation technique is encoded by Algorithm 10.1 (where $\Sigma_{Q_i}^0$ stands for the set of answers to $Q_i$ and $Q_i^c$ for the complement of the fuzzy set associated with $Q_i$).

Algorithm 10.1. Incremental relaxation of an atomic query.

let $Q := P$;
let $\delta$ be an absolute tolerance value; /* $Z = (0, 0, \delta, \delta)$ */
i := 0; $Q_i := Q$;
$Q'_i := Q_i \cap (F_P)^c$;
compute $\Sigma_{Q_i}^{Q'_i}$;
while ($\Sigma_{Q_i}^{Q'_i} = \emptyset$ and $(\text{core}(Q'_i) \not\subset \text{core}(F_P))$) do
begin
i := i + 1;
$Q_i := T^{(i)}(P) := P \oplus i.Z$;
$Q'_i := Q_i \cap (F_P)^c$;
compute $\Sigma_{Q_i}^{Q'_i}$;
end;
if $\Sigma_{Q_i}^{Q'_i} \neq \emptyset$ then return $\Sigma_{Q_i}^{Q'_i}$ endif.
**Particular case.** It is important to emphasize that, for some kinds of fuzzy predicates to be relaxed, the property of symmetry of the tolerance indicator $Z$ is not required. Consider, for instance, the predicate $P = (0, 25, 0, 10)$ representing the concept *young*. Weakening $P$ boils down to increasing the width of its support $\text{supp}(P)$. This has to be done on the right side of the TMF of $P$. The appropriate family of tolerance indicators will then be of the form $Z = (0, 0, 0, \delta)$, which leads to $T^\uparrow(P) = (0, 25, 0, 10+\delta)$.

10.2.2.4  **Relaxing a conjunctive query**

Among the fuzzy queries involving several predicates, the only problematic class is that of conjunctive queries. Indeed, an empty set of answers returned by a disjunctive query means that every predicate from the query has an empty support with respect to the actual database content. Therefore, it makes sense to relax all of the predicates from the query. The situation is the same when the connector, instead of being a disjunction, is an operator which is not zero-absorbing (for instance a mean). In the case of conjunctive queries, however, it is enough to have one predicate whose support is empty for the overall set of answers to be empty.

A conjunctive fuzzy query $Q$ is of the form $P_1 \land \cdots \land P_k$, where the symbol $\land$ stands for the connector *and*, interpreted by the $\text{min}$ operator, and $P_i$ is a fuzzy predicate.

It is worthy of note that $Q$ can be modified by means of a transformation (in order to perform a relaxation or intensification) in two distinct ways:

- a **global query modification** which consists of uniformly applying a transformation $T_j$ to each predicate $P_j$. Given a set of transformations $\{T_1, \ldots, T_k\}$ and a conjunctive query $Q = P_1 \land \cdots \land P_k$, the set of modified queries related to $Q$ resulting from applying $\{T_1, \ldots, T_k\}$ is

  $$T_1^{(i_1)}(P_1) \land \cdots \land T_k^{(i_k)}(P_k),$$

  where $i > 0$ and $T_j^{(i)}$ means that the transformation $T_j$ is applied $i$ times.

- a **local query modification** which affects only some predicates (or sub-queries). Given a set of transformations $\{T_1, \ldots, T_k\}$ and a conjunctive query $Q = P_1 \land \cdots \land P_k$, the set of modified queries related to $Q$ which result from applying $\{T_1, \ldots, T_k\}$ is

  $$T_1^{(i_1)}(P_1) \land \cdots \land T_k^{(i_k)}(P_k),$$

  where $\forall h, i_h \geq 0, \exists h, i_h > 0, T_j^{(i_h)}$ means that the transformation $T_j$ is applied $i_h$ times to $P_j$, and $T_j^{0}(P_j) = P_j$. 

Given a query \( Q = P_1 \land \cdots \land P_k \), if a subquery \( Q' \) of \( Q \) fails, then obviously the query \( Q \) itself fails.

**Definition 10.2.** Let \( Q = P_1 \land \cdots \land P_k \) be a failing query, \( Q' \), a failing subquery of \( Q \), is minimal if and only if no subquery of \( Q' \) fails.

In general, a failing query can have one or several minimal failing subqueries (MFSs for short). Identifying MFSs has been considered by several authors (Corella et al., 1984; Gaasterland et al., 1992a; Godfrey, 1997; Kaplan, 1982) as a means of providing cooperative answers to failing Boolean database queries. In particular, an efficient algorithm is proposed in Godfrey (1997) to find an MFS of a query of \( k \) conjuncts. This algorithm, which proceeds depth-first and top-down, is polynomial and runs in \( O(k) \) time. It has also been shown that finding all MFSs of a query is intractable and is NP-complete. However, finding \( l \) MFSs, for any fixed \( l \) \((l < k)\), can be done in an acceptable time.

Let \( Q = P_1 \land \cdots \land P_k \) be a failing query, \( T^\uparrow(Q) \) a relaxation of \( Q \), and \( SQ[j] \) a subquery of \( Q \) obtained by deleting the predicate \( P_j \) from \( Q \). Let also \( mfs(Q) = \{M_1, \ldots, M_n\} \) be the set of MFSs of \( Q \) and \( \{P_{n_1}, \ldots, P_{n_u}\} \) be the predicates that appear in at least one MFS of \( Q \). To characterize the set of MFSs of \( T^\uparrow(Q) \) with respect to that of \( Q \), we introduce the following propositions (Bose et al., 2009c), whose proofs are straightforward.

**Proposition 10.2.** If \( T^\uparrow(Q) = SQ[j] \land T^\uparrow_j(P_j) \) with \( j \in \{n_1, \ldots, n_u\} \), then the MFSs of \( T^\uparrow(Q) \) must be searched in \( mfs(Q) \) by substituting \( P_j \) with \( T^\uparrow_j(P_j) \) in each element of \( mfs(Q) \).

**Proposition 10.3.** If \( T^\uparrow(Q) = SQ[j] \land T^\uparrow_j(P_j) \) with \( j \notin \{n_1, \ldots, n_u\} \), then \( mfs(Q) \) is also the set of MFSs of \( T^\uparrow(Q) \).

**Example 10.1.** Consider a query \( Q = P_1 \land P_2 \land P_3 \land P_4 \) and assume that \( mfs(Q) = \{P_1 \land P_3, P_1 \land P_4\} \). Then,

- If \( T^\uparrow(Q) = T^\uparrow_1(P_1) \land SQ[1] = T^\uparrow_1(P_1) \land P_2 \land P_3 \land P_4 \), the MFSs of \( T^\uparrow(Q) \) are searched in \( \{T^\uparrow_1(P_1) \land P_3, T^\uparrow_1(P_1) \land P_4\} \).
- If \( T^\uparrow(Q) = SQ[2] \land T^\uparrow_2(P_2) = P_1 \land T^\uparrow_2(P_2) \land P_3 \land P_4 \), the MFSs of \( T^\uparrow(Q) \) are the MFSs of \( Q \).

As pointed out in Bose et al. (2007a), local query modification of a query \( Q = P_1 \land \cdots \land P_k \) (where all conditions involved in \( Q \) are of the same importance for the user) leads to an ordering (\( \prec \)) between the modified queries obtained from \( Q \). That ordering can be defined on the basis of the
Empty and Plethoric Answers

The set of modified queries related to $Q$ (i.e., $T^i(P_1) \land \cdots \land T^i(P_k)$) can then be organized in a lattice. For instance, the lattice associated with the weakening of the query $P_1 \land P_2$ is given in Figure 10.4. In practice, and in order to find a relaxed query related to $Q = P_1 \land \cdots \land P_k$ that returns a non-empty answer set, one has to deal with the following three main issues when using a local query modification:

- define an intelligent way to exploit the lattice of weakened queries,
- guarantee the property of equal relaxation effect for all fuzzy predicates,
- study the extent to which the user has to intervene in the relaxation process.

The two latter issues have been discussed in Bosc et al. (2007a). In the following, we show how the scanning of the lattice can be done in an efficient way by exploiting the MFSs of the original failing query.

**Scanning the lattice.** In general, search techniques in a lattice are time-consuming, and result in algorithms whose time complexity is exponential. To make the search over the lattice more efficient, the MFSs of the original failing query may be exploited. The corresponding search technique consists of two steps:

1. **Enumerating $l$ MFSs of $Q$**. To do this, one may use the algorithm proposed in Godfrey (1997) which is designed for computing $l$ MFSs.
in an acceptable time (when \( l \) is not too large). Such an algorithm can easily be adapted in the case of fuzzy queries with no main changes, as pointed out in Bosc et al. (2007a). From Propositions 10.2 and 10.3, this algorithm has to be run only once, since the MFSs of any relaxed query \( T^\uparrow(Q) \) can be deduced from those of \( Q \).

(2) An intelligent search technique. Information about MFSs allows the provision of an intelligent search technique over the lattice by avoiding the need to evaluate some nodes. Indeed, the node in the lattice that preserves at least one MFS of its father node (that is, the node from which it is derived) does not have to be evaluated (since we are certain that it fails). This technique is sketched in Algorithm 10.2.

**Algorithm 10.2.** MFS-based search for a relaxed query with non-empty answers.

```plaintext

\begin{verbatim}
let \( Q = P_1 \land P_2 \land \cdots \land P_k \); 
mfs(Q) = \{P_1 \land P_2 \land \cdots \land P_{m_1}, \ldots, P_1 \land P_{m_2} \land \cdots \land P_{m_k}\}; 
found := false; i := 1; 
while (i \leq \omega \cdot k) and (not found) do 
begin 
level(i) := \{Q_1^i, \ldots, Q_n^i\}; 
for each req in level(i) do 
begin 
modif := true; 
for each a_mfs in mfs(father(req)) do 
begin 
modif := modif \land (a_mfs \notin req) end; 
if modif then 
if evaluate(req) then 
level(i) = \emptyset; 
found := true 
endif; 
endif; 
if level(i) \neq \emptyset then compute mfs(req) endif; 
i := i + 1; 
end; 
if found then return req;
end.
\end{verbatim}
```

---

*Fuzzy Preference Queries to Relational Databases*
In this algorithm:

- **level(i)** stands for the set of relaxed queries in level $i$ of the lattice,
- **father($Q$)** is the set of nodes from which $Q$ can be derived, i.e., $Q$ is an immediate relaxed variant of any query contained in **father($Q$)**. For instance, in Figure 10.4, if $Q = T(P_1) \land T(P_2)$, then **father($Q$) = \{T(P_1) \land P_2, P_1 \land T(P_2)\}**,
- **evaluate($Q$)** is a function that evaluates $Q$ against the database. It returns **true** if $Q$ produces some answers, **false** otherwise,
- **$\omega$** represents the maximal number of relaxation steps for each predicate $P_j$ ($j = 1, k$). Hence, the maximal relaxation of a query $Q = P_1 \land \cdots \land P_k$ is the modified query given by $T^{(\text{max})}(Q) = T_1^{(\omega)}(P_1) \land \cdots \land T_k^{(\omega)}(P_k)$. This implies that the lattice is bounded and maximally contains $\omega.k$ levels.

As can be seen, to find the relaxed query with non-empty answers, several candidate queries must be evaluated against the database. One way to substantially reduce the cost of this evaluation is, first, to evaluate the lower bound of relaxed queries, i.e., $T^{(\text{max})}(Q)$ and store the resulting items; second, to evaluate each candidate query on the result of $T^{(\text{max})}(Q)$.

10.2.3 Case-based reasoning approach

10.2.3.1 Introduction and principle

The main problem with the type of approach presented in the previous section (i.e., that based on query relaxation) is that it introduces high combinatorics in the case of conjunctive queries — even though the notion of the MFS can be exploited in order to limit these combinatorics, as we just showed — since one has to scan the lattice of possible relaxed queries (which can be huge) and process many of them (which is expensive) before finding a non-failing one.

In this section, we aim to avoid this combinatory explosion by exploring a *case-based reasoning* approach (Bosc et al., 2009a; Pivert et al., 2010). The idea is to take advantage of the queries previously submitted to the system in order to efficiently find a non-failing query as similar as possible to a failing one. We consider conjunctive queries of the form “$Q = P_1 \land P_2 \land \cdots \land P_n$”, where each $P_i$ is of the form “$A_i \text{ in } E_i$”, $A_i$ denoting an attribute and $E_i$ a fuzzy set which can be either discrete or continuous. The essential idea is to look for a “good” global substitute to a failing
query $Q$ among queries previously submitted to the system (and stored in a repository — also called query cache in the following — $D^+$) whose sets of answers are known to be non-empty. Such an approach implies the availability of a resemblance measure over every attribute domain involved in the database considered, so as to define the notion of semantic proximity between queries. With this approach, a non-empty answer is guaranteed in one step, since only one query — known to be non-failing — needs to be processed. Note that this approach is relevant mainly for databases in which deletions and updates are rare or non-existent (a typical example is an archiving system). Otherwise the repository would have to be updated too, which would be a complex and expensive task.

10.2.3.2 Predicate substitution

Let $P = (A \text{ in } E)$ be a failing predicate with respect to a relation $r$ ($\forall t \in r, t.A \notin E$), where $A$ is an attribute and $E$ is a finite set or an interval, and $P' = (A' \text{ in } E')$ a non-failing predicate with respect to $r$ ($\exists t \in r, t.A \in E'$). We denote by $\text{prox}$ the proximity relation defined over the domain of $A$.

**Boolean case.** We first deal with the case where the failing query only involves Boolean conditions.

**Case where $E$ and $E'$ are finite sets.** In order to assess the “quality” of $P'$ as a substitute to $P$, a measure is needed. It is not, strictly speaking, a proximity measure, since the symmetry property is not desired here. Indeed, one wants to know whether $P'$ is a good substitute to $P$, but not the reciprocal. Several possible substitutivity measures (denoted by $sbs_i$ later on) are discussed below.

**1st idea:** Assess the extent to which every element from $(E' - E)$ resembles at least one element from $E$:

$$sbs_1(P, P') = \inf_{x' \in (E' - E)} \sup_{x \in E} \text{prox}(x, x').$$

(10.2)

The problem with this measure is that the worst element “masks” the others, as illustrated in the next example. In the following, we assume that the resemblance relation on animals depicted in Table 10.1 is available.

**Example 10.2.** $E = \{\text{hen, duck, turkey}\}$, $E'_1 = \{\text{hen, turkey, cow}\}$, $E'_2 = \{\text{cow, rooster}\}$. We have $sbs_1(P, P'_1) = sbs_1(P, P'_2) = 0$, but since there are neither hens nor turkeys in the database — otherwise $P$ would not be a failing predicate — it seems reasonable to claim that $E'_2$ should be a better
Empty and Plethoric Answers

Table 10.1 Proximity relation over attribute animal

<table>
<thead>
<tr>
<th></th>
<th>rooster</th>
<th>hen</th>
<th>duck</th>
<th>turkey</th>
<th>cow</th>
</tr>
</thead>
<tbody>
<tr>
<td>rooster</td>
<td>1</td>
<td>0.9</td>
<td>0.6</td>
<td>0.7</td>
<td>0</td>
</tr>
<tr>
<td>hen</td>
<td>0.9</td>
<td>1</td>
<td>0.6</td>
<td>0.7</td>
<td>0</td>
</tr>
<tr>
<td>duck</td>
<td>0.6</td>
<td>0.6</td>
<td>1</td>
<td>0.5</td>
<td>0</td>
</tr>
<tr>
<td>turkey</td>
<td>0.7</td>
<td>0.7</td>
<td>0.5</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>cow</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Substitute than $E'_1$. However, in the computation of $sbs_1(P, P'_2)$, the element cow “masks” the element rooster.

2nd idea: Assess the extent to which there is an element from $(E' - E)$ which resembles at least one element from $E$:

\[
sbs_2(P, P') = \sup_{x' \in (E' - E)} \sup_{x \in E} \text{prox}(x, x').
\] (10.3)

Here, the difficulty is that the “winning set” may include elements which are very distant from those desired by the user.

**Example 10.3.** $E = \{\text{hen}\}$, $E'_1 = \{\text{hen}, \text{cow}, \text{turkey}\}$, $E'_2 = \{\text{duck}\}$. Here, $E'_2$ should win, since it includes only elements close to the desired ones, contrary to $E'_1$, which includes “cow”. However, it is $E'_1$ which wins, since $sbs_2(P, P'_1) = 0.7$ while $sbs_2(P, P'_2) = 0.6$.

3rd idea: Mix the quantitative and the qualitative aspects by measuring the average resemblance degree between an element from $(E' - E)$ and an element from $E$. For each element $x'$ from $(E' - E)$, the corresponding measure looks for the maximal proximity between $x'$ and an element $x$ from $E$, computes the sum of these maximal proximities, and divides this sum by the number of elements present in $(E' - E)$:

\[
sbs_3(P, P') = \frac{\sum_{x' \in (E' - E)} \sup_{x \in E} \text{prox}(x, x')}{|E' - E|}.
\] (10.4)

**Example 10.4.** $E = \{\text{hen, duck, turkey}\}$, $E'_1 = \{\text{hen, turkey, cow}\}$, $E'_2 = \{\text{cow, rooster}\}$. We have: $sbs_3(P, P'_1) = 0$ and $sbs_3(P, P'_2) = 0.45$.

Since measure $sbs_3$ appears the most satisfactory, it will be used in the following.

**Case where $E$ and $E'$ are intervals.** We first consider the simple case where proximity is defined in a Boolean manner:

\[
\text{res}(x, y) = 1 \text{ if } |x - y| \leq \delta, 0 \text{ otherwise}.
\] (10.5)
Consider two intervals: $E = [m, M]$ — from $P$ — and $E' = [m', M']$ — from the potential substitute $P'$. Using the tolerance value $\delta$, interval $E$ is extended into $E'' = [m - \delta, M + \delta] = [m'', M'']$. It is necessary to compute the extent to which $E'$ is close to $E''$, and the expression of the substitutivity measure becomes:

$$sbs_3(P, P') = \frac{|(E' - E) \cap E''|}{|E' - E|}.$$  \hfill (10.6)

The case where proximity is defined by means of a fuzzy tolerance indicator $Z$ (see Section 10.2.2.1) is slightly more complex. In such a situation, interval $E$ is dilated into a fuzzy set $E''$ according to the calculus described in Section 10.2.2.2 (see also Bosc et al., 2005) and the definition of $sbs_3(P, P')$ becomes:

$$sbs_3(P, P') = \frac{\int_{x \in (E' - E)} \mu_{E''}(x)}{|E' - E|}.$$  \hfill (10.7)

Case where $E$ is an interval and $E'$ is a set. When $P$ involves an interval $E$ and $P'$ involves a set $E'$, $sbs_3(P, P')$ is written as:

$$sbs_3(P, P') = \sum_{x' \in E'} \sup_{x \in E} \text{prox}(x, x').$$  \hfill (10.8)

On the other hand, the dual case (set in $P$ and interval in $P'$) is much more tricky and cannot be captured by Expression (10.7) defining $sbs_3$ when the attribute domain is continuous. Consequently, we introduce the constraint that a finite set can only be replaced by another finite set.

**Fuzzy case.** We now move to the case where value constraints are expressed by means of fuzzy predicates. Consider a conjunctive fuzzy query $Q = P_1 \land \cdots \land P_n$, where any predicate $P_i$ is of the form $(A_i \text{ is } T_i)$ and $T_i$ is a fuzzy term. Here, the fact that $Q$ returns an empty answer set means that there does not exist any element $x$ in the database such that $\top_{i=1,\ldots,n}(\mu_{T_i}(x)) > 0$, where $\top$ denotes a triangular norm generalizing the conjunction. This fact can be expressed by saying that the support of the query relative to the database is empty.

In order to deal with this kind of query, measure $sbs_3$ needs to be generalized by taking into account the resemblance between the degrees originating from the two fuzzy terms considered. The generalized measure obtained, which can also be seen as a variant of the interchangeability
measure proposed in Bosc and Pivert (1997), is defined as:

\[
sbs_3(E, E') = \sum_{x' \in sp(E' - E)} w(x') \times \sup_{x \in E} \min(\text{prox}(x, x'), \Psi(\mu_{E'}(x'), \mu_E(x)))
\]

(10.9)

where \(sp(E)\) denotes the support of \(E\), function \(\Psi\) assesses the resemblance between two degrees in the unit interval — it can be defined, e.g., as \(\Psi(a, b) = 1 - |a - b|\) — and:

\[
w(x') = \mu_{E' - E}(x') = \min(\mu_{E'}(x'), 1 - \mu_E(x')).
\]

The weight \(w(x')\) captures the fact that it is all the more important to find a good substitute to \(x'\) as \(x'\) strongly belongs to \(E' - E\). It is straightforward to show that, if the sets are crisp, this expression is equivalent to Expression (10.8).

The above definition can be directly extended to the case of continuous fuzzy sets by replacing the sum by an integral.

**Example 10.5.** Consider the fuzzy sets: \(E = \{1/\text{rapeseed}, 0.8/\text{cabbage}, 0.3/\text{wheat}\}\) and \(E' = \{0.4/\text{rapeseed}, 0.3/\text{cabbage}, 0.4/\text{corn}, 0.7/\text{broccoli}\}\). Using the most common definition of the difference between fuzzy sets, i.e.,

\[
\mu_{A - B}(x) = \min(\mu_A(x), 1 - \mu_B(x)),
\]

one obtains:

\(E' - E = \{0.2/\text{cabbage}, 0.4/\text{corn}, 0.7/\text{broccoli}\}\).

For “cabbage”, the supremum equals:

\(\sup(\min(0.2, 0.3), \min(1, 0.5), \min(0.2, 1)) = 0.5\),

for “corn”, it equals:

\(\sup(\min(0.4, 0.4), \min(0.1, 0.6), \min(0.8, 0.9)) = 0.8\),

and for “broccoli” we obtain the degree 0.9. Hence, the final substitutivity degree equals:

\[
\frac{0.2 \times 0.5 + 0.4 \times 0.8 + 0.7 \times 0.9}{0.2 + 0.4 + 0.7} = 0.81.
\]
Conjunctive query replacement strategy

Let $Q$ be a failing user query, and consider the set $S_Q$ of predicates \{\(A_i \in E_i\)\} from $Q$. We assume a repository $D^+$ of non-failing queries. Each query $Q'$ of $D^+$ is associated with the set $S_{Q'}$ of its predicates \{\(A_i \in E'_i\)\}. The problem is to find the best substitute to $Q$ among all queries of $D^+$.

**Definition 10.3 (substitute to a query).** A substitute to a query $Q$ is a query $Q'$ from $D^+$ such that:

i) $Q'$ is addressed to the same relation(s) as $Q$,

ii) $Q'$ shares at least one attribute from its “where” clause with that from $Q$,

iii) $Q'$ involves at least one predicate (\(A_i \in E'_i\)) which is not subsumed by (\(A_i \in E_i\)).

Item ii) of the definition above guarantees semantic proximity between $Q$ and $Q'$ while item iii) is based on the following remark. Even if query $Q'$ involves a predicate which is subsumed by its counterpart in $Q$, query $Q'$ can be an interesting substitute to $Q$ since the other predicates must also be taken into account. For instance, if query $Q = (A \in \{\text{rabbit}, \text{hen}\} \land B \in \{\text{wheat}, \text{cabbage}\})$ returns an empty answer, it is still possible for query $Q' = (A \in \{\text{rabbit}\} \land B \in \{\text{wheat}, \text{oats}\})$ to return a non-empty answer, whereas the predicate on $A$ in $Q'$ is subsumed by that in $Q$. However, for a query $Q'$ to be a possible substitute, it is necessary for $Q'$ to involve at least one predicate which is not subsumed by the corresponding predicate in $Q$ (but note that if this were not the case, the answer to $Q'$ would be empty — since the answer to $Q$ is empty — and $Q'$ would therefore not be in $D^+$).

The approach that we propose consists of the following three steps:

1. select the candidate queries from the repository $D^+$ (and adapt these queries; see the algorithm below),
2. compute the proximity degrees between the queries retained and the user query $Q$, through the measure $sbs_3$,
3. determine the closest substitute to $Q$ and process it.

**Replacement mechanism.** It is worth noting that the predicates from $Q'$ which are strictly subsumed by those from $Q$ can be replaced by their counterparts in $Q$. Indeed, it is not possible for such a predicate from $Q$ (taken individually) to have been the reason for the query failure. Moreover,
for every predicate $A_i \in E_i$ from $Q$ which is not “covered” by $Q'$, i.e., which concerns an attribute on which there is no constraint in $Q'$, one may compute the proximity between $P$ and the entire domain of the attribute considered.

The conjunctive combination of proximities related to the atomic predicates can be performed by means of a triangular norm (for instance the minimum), so as to obtain the overall proximity between two queries. Note that alternative solutions could also be possible; for instance, one might use a mean operator. The substitution algorithm is outlined below.

Let $Q'$ be a query from $D^+$ which is addressed to the same relation(s) as $Q$ and shares at least one attribute from its where clause with that in $Q$. The five steps of the algorithm are:

1. replace the select clause from $Q'$ by that from $Q$,
2. remove from $Q'$ every predicate that concerns an attribute absent from the where clause in $Q$,
3. replace every predicate from $Q'$ which is strictly subsumed by the corresponding one from $Q$ by the latter,
4. for the other predicates, compute the proximity between the predicate from $Q'$ and the corresponding one from $Q$, by means of measure $sbs_3$, and replace the predicate from $Q'$ by its union with that from $Q$. In the case of the predicates from $Q$ which are not covered by $Q'$, compute their substitutivity degree relative to the entire domain of the attribute involved,
5. aggregate the local proximities by means of a triangular norm (the idea being to assess the extent to which every predicate of the substitute query is close to the corresponding predicate from the initial failing query).

In the case of a fuzzy query, the notion of a subsumption can be based on the inclusion between fuzzy sets proposed by Zadeh, i.e., $E \subseteq F \iff \forall x, \mu_E(x) \leq \mu_F(x)$.

**Detailed example.** Let $Q$ be the following failing user query:

```sql
select #id from Farms
where veg in {"corn", "rapeseed"} and
city in {"Lannion", "Caouennec", "Prat"} and area in [60, 100];
```

Assume that the domain of $veg$ is: \{corn, rapeseed, sunflower, wheat, cabbage, broccoli, potato, rutabaga\} and that the associated proximity relation
Table 10.2 Proximity relation over attribute veg

<table>
<thead>
<tr>
<th>co</th>
<th>ra</th>
<th>su</th>
<th>wh</th>
<th>ca</th>
<th>br</th>
<th>po</th>
<th>ru</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.4</td>
<td>0.3</td>
<td>0.8</td>
<td>0.1</td>
<td>0.1</td>
<td>0.6</td>
<td>0.4</td>
</tr>
<tr>
<td>0.4</td>
<td>1</td>
<td>0.9</td>
<td>0.6</td>
<td>0.2</td>
<td>0.2</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>0.3</td>
<td>0.9</td>
<td>1</td>
<td>0.5</td>
<td>0.1</td>
<td>0.1</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>0.8</td>
<td>0.6</td>
<td>0.5</td>
<td>1</td>
<td>0.2</td>
<td>0.1</td>
<td>0.5</td>
<td>0.4</td>
</tr>
<tr>
<td>0.1</td>
<td>0.2</td>
<td>0.1</td>
<td>0.2</td>
<td>1</td>
<td>0.9</td>
<td>0.6</td>
<td>0.7</td>
</tr>
<tr>
<td>0.1</td>
<td>0.2</td>
<td>0.1</td>
<td>0.1</td>
<td>0.9</td>
<td>1</td>
<td>0.4</td>
<td>0.6</td>
</tr>
<tr>
<td>0.6</td>
<td>0.1</td>
<td>0.3</td>
<td>0.5</td>
<td>0.6</td>
<td>0.4</td>
<td>1</td>
<td>0.8</td>
</tr>
<tr>
<td>0.4</td>
<td>0.1</td>
<td>0.3</td>
<td>0.4</td>
<td>0.7</td>
<td>0.6</td>
<td>0.8</td>
<td>1</td>
</tr>
</tbody>
</table>

is given in Table 10.2. Let \( Q'_1 \) be the following query from \( D^+ \):

\[
\text{select} \#\text{name from Farms} \\
\text{where veg in \{“wheat”, “rapeseed”, “sunflower”\} and} \\
\text{city in \{“Lannion”, “Prat”\} and area = 125 and animal in} \\
\text{\{“cow”, “pig”\};}
\]

The query \( Q''_1 \) obtained by adapting \( Q'_1 \) according to the above algorithm is:

\[
\text{select} \#\text{id from Farms} \\
\text{where veg in \{“corn”, “rapeseed”, “wheat”, “sunflower”\}} \\
\text{and city in \{“Lannion”, “Caouennec”, “Prat”\} and} \\
\text{(area in [60, 100] or area = 125);} \\
\]

The degree computed by \( sbs_3 \) for the substitution of \{corn, rapeseed\} by \{wheat, rapeseed, sunflower\} equals:

\[
\frac{\max(0.8,0.6) + \max(0.3,0.9)}{2} = 0.85.
\]

Assume that the proximity over the areas is based on a fuzzy tolerance indicator \( Z \) with a triangular membership function of support \([-50, 50] \). The substitution of \([60, 100] \) by 125 is assigned the degree 0.5 (i.e., the proximity degree between 100 and 125; see Expression (10.8)). Finally, the degree computed for \( Q''_1 \) using the t-norm minimum is:

\[
\min(0.85,0.5) = 0.5.
\]

Now consider another query, denoted by \( Q'_2 \), from \( D^+ \):

\[
\text{select} \#\text{name from Farms} \\
\text{where city = “Caouennec” and area in [80, 180] and} \\
\text{animal in \{“sheep”, “goat”\};}
\]
Altering $Q_2'$ according to the algorithm yields $Q''_2$:

```
select #id from Farms
where city in {"Lannion", "Caouennec", "Prat"} and area in [60, 180];
```

For the condition on attribute `veg`, we have:

$$sbs_3(\text{veg} \in \{\text{corn, rapeseed}\}, \text{veg} \in \text{domain(veg)}) = (0.9 + 0.8 + 0.2 + 0.2 + 0.6 + 0.4)/6 = 0.52.$$  

For the condition on attribute `area`, we have (according to Expression 10.7):

$$sbs_3(\text{area} \in [60, 100], \text{area} \in [80, 180]) = ((150 - 100)/2)/(180 - 100) = 25/80 = 0.31.$$  

Thus, the degree attached to $Q''_2$ is $\min(0.52, 0.31) = 0.31$ and $Q''_1$ is a better substitute for $Q$ than is $Q''_2$.

**Remark 10.1.** In the event of tied results, the cardinality of the result of each candidate query could be taken into account so as to break these ties, provided that these cardinalities are stored in $D^+$.  

### 10.3 Plethoric Answer Problem

#### 10.3.1 Introduction

The plethoric answer problem (PAP) in the context of fuzzy queries can be stated as follows. Denote by $\Sigma^0_Q = \{\mu_1/x_1, \ldots, \mu_n/x_n\}$ the fuzzy set of tuples whose satisfaction degree with respect to a fuzzy query $Q$ is strictly positive. Assume that $Q$ involves a number $k$ of expected answers. Let:

$$\mu_{\max}(Q) = \sup_{x_i \in \Sigma^0_Q} \mu_i$$

and:

$$\Sigma^{0}_{Q, \max}(Q) = \{x_i \in \Sigma^0_Q | \mu_i = \mu_{\max}(Q)\}.$$  

If $|\Sigma^0_{Q, \max}(Q)| \gg k$, then we are in the presence of a plethoric answer problem. This issue often arises because the user query is too general. In other words, the fuzzy requirements in the query are not restrictive enough. To overcome this problem, one can refine the query in order to make it more specific, so as to return a reasonable set of items. The first approach described below, based on predicate modification, assumes that...
\( \mu_{\text{max}}(Q) = 1 \) (i.e., there are too many answers which fully satisfy the user’s requirement).

10.3.2 **Approach based on predicate strengthening**

10.3.2.1 **Principle**

The first way to refine the query consists of intensifying the query’s fuzzy constraints in order to reduce the set \( \Sigma^1_Q \). To achieve this task, a fundamental required property of the intensification mechanism is to significantly shrink the cores of the fuzzy sets associated with the conditions of the query.

As in the case of query relaxation, query intensification can then be performed by applying a basic transformation \( T^i \) on all or some of the query’s predicates. This transformation can be applied iteratively if necessary. Three basic properties are required for any transformation \( T^i \) when applied to a predicate \( P \) (\( T^i(P) \) representing the intensified predicate), as described in Bosc et al. (2008):

- **IC1**: \( T^i \) does not increase the membership degree for any element of the domain, i.e., \( \forall u \in \text{domain}(A), \mu_{T^i(P)}(u) \leq \mu_P(u) \) where \( A \) denotes the attribute concerned by \( P \),
- **IC2**: \( T^i \) must shrink the core \( C(P) \) of the fuzzy predicate \( P \), i.e., \( \text{core}(T^i(P)) \subset \text{core}(P) \),
- **IC3**: \( T^i \) preserves the left-hand (resp. right-hand) spread of the fuzzy predicate \( P \), i.e., if \( P = (A, B, a, b) \), \( T^i(P) = (A', B', a', b') \) with \( A < A' \), \( B > B' \), \( a = a' \) and \( b = b' \), and \( A' - A < a \) and \( B - B' < b \).

The second property allows for reduction of the width of the core, and then effectively decreases the number of answers with degree 1 (i.e., the set \( \Sigma^1_Q \)). The third property guarantees that the data excluded from the core of \( P \) remain in its support, i.e., that no data that was initially fully acceptable is then rejected by the intensification.

10.3.2.2 **Erosion operation**

Let \( Z \oplus X = F \) be an equation where \( X \) is the unknown variable. Solving this equation was extensively discussed in Dubois and Prade (1983). It was demonstrated that the greatest solution of this equation is given by \( \bar{X} = F \ominus (-Z) = F \ominus Z \) (since \( Z = -Z \)), where \( \ominus \) is the extended Minkowski...
subtraction defined by Dubois and Prade (1983, 1988):

\[
\mu_{F \ominus Z}(r) = \inf_s (\mu_Z(r - s) \Rightarrow_r \mu_F(s))
\]

\[
= \inf_s (\mu_{E[Z]}(s, r) \Rightarrow_r \mu_F(s)),
\]

(10.10)

where \( \Rightarrow_r \) is an R-implication generated by the t-norm \( \top \), defined by

\[
\Rightarrow_r \mu(a, v) = \sup \{ \lambda \in [0, 1] | \top(a, \lambda) \leq v \}
\]

for \( a, v \in [0, 1] \). We make use of the same t-norm \( \top = \min \) as in the dilation operation (see Section 10.2.2.3), which implies that \( \Rightarrow_r \) is Gödel’s implication.

Let \( E[Z]_r = \{ s | \mu_{E[Z]}(s, r) > 0 \} \) be the set of elements that are somewhat close to \( r \) in the sense of \( E[Z] \). The expression above can then be interpreted as the degree of inclusion of \( E[Z]_r \) in \( F \). This means that \( r \) belongs to \( F \ominus Z \) if all the elements \( s \) that are close to \( r \) are in \( F \). Hence, the inclusion \( F \ominus Z \subseteq F \) holds. This operation is very useful for intensifying the meaning of vague terms from natural language. Now, eroding the fuzzy set \( F \) by \( Z \) results in the fuzzy set \( F_Z \) defined by \( F_Z = F \ominus Z \).

**Lemma 10.2.** Given a fuzzy set \( F \) and a proximity relation \( E[Z] \), for \( F_Z = F \ominus Z \) we have \( F_Z \subseteq F \).

If \( F = (A, B, a, b) \) and \( Z = (-z, z, \delta, \delta) \), then \( F \ominus Z = (A + z, B - z, a - \delta, b - \delta) \) provided that \( a \geq \delta \) and \( b \geq \delta \). Figure 10.5 illustrates this operation. In the crisp case, \( F \ominus Z = [A, B] \ominus [-z, z] = [A + z, B - z] \) (while \( F \oplus Z = [A - z, B + z] \)).

Again, and in order to satisfy the needs of some practical applications, only one constituent part of a fuzzy set \( F \) (either the core or the support) must be affected when applying an erosion operation. We denote by core erosion (resp. support erosion) the eroding transformation that affects only

\[
\begin{align*}
\text{Fig. 10.5} & \quad \text{Erosion operation}
\end{align*}
\]
the core (resp. support) of \( F \). The following proposition allows such results to be obtained.

**Proposition 10.4.** Let \( F \) be a fuzzy set and \( E[Z] \) a proximity relation. We have:

- core erosion obtained using the family of tolerance indicators of the form \( Z = (-z, z, 0, 0) \),
- support erosion obtained using the family of tolerance indicators of the form \( Z = (0, 0, \delta, \delta) \).

By this proposition, if \( F = (A, B, a, b) \) then core erosion (resp. support erosion) leads to \( F_Z = (A + z, B - z, a, b) \) (resp. \( F_Z = (A, B, a - \delta, b - \delta) \)).

### 10.3.2.3 Erosion of an atomic query

In the following, we show how the notion of the parameterized absolute proximity relation \( E[Z] \) can provide the basis for an intensification transformation \( T^{\downarrow} \). We first investigate the case of an atomic query. Let \( P \) be a fuzzy predicate and \( E[Z] \) be a proximity relation parameterized by a tolerance indicator \( Z \) of the form \( (-z, z, 0, 0) \). Making use of the erosion operation, the predicate \( P \) can be transformed into a restricted fuzzy predicate \( P' \) defined as follows:

\[
P' = T^{\downarrow}(P) = P_Z = P \ominus Z.
\]

Now, if \( P = (A, B, a, b) \) then \( T^{\downarrow}(P) = (A + z, B - z, a, b) \); see Figure 10.5. Properties IC1–IC3 straightforwardly hold.

**Principle of the approach.** Let \( Q = P \) be a flexible query containing a single fuzzy predicate \( P \). Assume that \( \Sigma^1_Q \) is too large. In order to reduce the cardinality of \( \Sigma^1_Q \), we transform \( Q \) into \( \bar{Q}_1 = T^{\downarrow}(P) = P \ominus Z \). This intensification mechanism can be applied iteratively until the database returns a manageable set of answers to the modified query \( \bar{Q}_n = T^{\downarrow(n)}(P) = P \ominus nZ \). An implicit measure of nearness such that \( \bar{Q}_k \) is nearer to \( Q \) than \( \bar{Q}_l \) if \( k < l \) is then induced by this intensification strategy for atomic queries.

Let us take a look at the subset of \( \Sigma^1_Q \) resulting from the intensification process. The items of that subset (which are the answers to the revised query \( \bar{Q}_n \)) can be viewed as the best answers to the original query \( Q \), since they constitute the typical values (i.e., prototypes) of the concept expressed by the fuzzy set associated to \( Q \).
Empty and Plethoric Answers

Controlling the intensification. Limiting the amplitude of an intensification process does not appear as crucial as in the case of query relaxation. Indeed, the intensification process only aims at reducing a large set of answers, not at finding alternative ones. It is, however, worth emphasizing that the query refinement process must stop when the upper bound and the lower bound of the core (of the modified query \( Q_i \)) are equal to \((A + B)/2\). Indeed, the TMF associated with \( Q_i \) is \((A + iz, B - iz, a, b)\).

Now, since \( A + iz \leq B - iz \) holds, we have \( i \leq (B - A)/2z \). This means that the maximal query refinement is obtained when the core is reduced to a singleton.

Note that the risk of obtaining an empty set of answers during the process is excluded when \( a > z \) and \( b > z \) (since the data that have been eliminated from \( \Sigma_{Q_{i-1}} \) still belong to the support of \( Q_i \)). Hence, those data are answers to \( Q_i \). Otherwise, it is possible to back up and try another variation (for instance, by adjusting the tolerance parameter \( Z \)).

Intensification algorithm. Algorithm 10.3 formalizes this intensification approach.

**Algorithm 10.3.** Atomic query intensification.

\[ Q := P; \]
\[ \text{let } Z = (-z, z, 0, 0) \text{ be a tolerance indicator; } \]
\[ i := 0; Q_i := Q; \]
\[ \text{compute } \Sigma_{Q_i}; \]
\[ \text{while } (| \Sigma_{Q_i} | \text{ is too large and } i \leq (B - A)/2z) \text{ do } \]
\[ \begin{align*}
& i := i + 1; \\
& Q_i := T^{(i)}(P) := P \ominus iZ; \\
& \text{compute } \Sigma_{Q_i}; \\
& \end{align*} \]
\[ \text{return } \Sigma_{Q_i}; \]

Particular cases. Note that, for some kinds of atomic queries to be intensified, the property of symmetry of the tolerance indicator \( Z \) is not required. Consider, for instance, the query \( Q = P \), where \( P = (0, 25, 0, 10) \), which represents the concept **young** in a given context. Intensifying \( P \) boils down to reducing the width of the core \( C(P) \) and thus to come closer to the typical values of \( P \). Here, it is the left part of \( C(P) \) which contains the typical values of the concept **young**. The intensification transformation
must therefore only affect the right part of \( C(P) \) and preserve entirely its left part. The appropriate form of \( Z \) that allows this transformation is \((0, z, 0, 0)\), which leads to \( T^i(P) = P \ominus Z = (0, 25 - z, 0, 10)\).

Consider now a query \( Q \) of the form \( Q = \neg P \), where \( \neg \) stands for the negation \( (\mu_{\neg P}(u) = 1 - \mu_P(u), \forall u) \), and assume that the query results in plethoric answers. To solve this problem, one may apply the intensification mechanism proposed above to \( Q \), i.e., shrink the core of \( \neg P \). It is easy to check that this transformation is equivalent to extending the support of \( P \).

This means that applying \( T^\downarrow \) to \( Q \) boils down to applying \( T^\uparrow \) to \( P \). So, we have \( T^\downarrow(\neg P) = T^\uparrow(P) \).

10.3.2.4 Intensifying a conjunctive query

Let \( Q = P_1 \land \cdots \land P_k \) be a conjunctive fuzzy query. Assume that the set of answers \( \Sigma^1_Q \) to \( Q \) is too large. Then, obviously, each subquery of \( Q \) also results in plethoric answers.

**Lemma 10.3.** If \( Q = P_1 \land \cdots \land P_k \) results in plethoric answers, then each predicate \( P_i \) of \( Q \) also results in plethoric answers.

This means that, if the plethoric answer problem arises in a conjunctive query \( Q \), it then arises in each proper subquery of \( Q \). In that case, examining the subqueries of \( Q \) is useless.

In the following, we consider a global modification strategy which, given a set of intensification transformations \( \{T^\downarrow_1, \ldots, T^\downarrow_k\} \), leads to the set of intensified queries:

\[
\{T^\downarrow_i(P_1) \land \cdots \land T^\downarrow_i(P_k)\}.
\]

The ordering \( (\prec) \) introduced in Section 10.2.2.4 may also be defined for intensified queries. Letting \( Q' \) and \( Q'' \) be two intensified queries of \( Q \), we say that

\[
Q' \prec Q'' \text{ if and only if } \sum_{i=1}^k \text{count}(T^\downarrow_i \text{ in } Q') < \sum_{i=1}^k \text{count}(T^\downarrow_i \text{ in } Q'').
\]

This ordering allows for the introduction of a semantic distance between queries. For that semantic distance to make sense, it is desirable that the set of transformations \( \{T^\downarrow_1, \ldots, T^\downarrow_k\} \) should fulfill the property of equal intensification effect (EIE) on all predicates (Bosc et al., 2008). There are several ways to define this property. A possible definition is to consider the ratio of the lengths of the cores associated with the original and the modified fuzzy
predicates. This ratio must be of the same magnitude for all the predicates involved in $Q$, when the transformations $\{T_1^i, \ldots, T_k^i\}$ are applied. Letting $\Delta(P_i, T_i^j(P_i))$ denote this ratio when $T_i^j$ is applied to $P_i$, we have:

$$\Delta(P_i, T_i^j(P_i)) = \frac{L(\text{core}(T_i^j(P_i))))}{L(\text{core}(P_i))},$$

where $L(\text{core}(P_i))$ (resp. $L(\text{core}(T_i^j(P_i)))$) stands for the length of $\text{core}(P_i)$ (resp. $\text{core}(T_i^j(P_i)))$. A simple calculus enables the following (with $P_i = (A_i, B_i, a_i, b_i)$ and $Z_i = (-z_i, z_i, 0, 0))$:

$$\Delta(P_i, T_i^j(P_i)) = 1 - 2z_i/(B_i - A_i).$$

Now, given $k$ predicates $P_1, \ldots, P_k$, the EIE property for a set of transformations $\{T_1^i, \ldots, T_k^i\}$ can be expressed as follows:

$$\Delta(P_1, T_1^j(P_1)) = \cdots = \Delta(P_k, T_k^j(P_k)).$$

Thus, to start the intensification process of a conjunctive query $Q = P_1 \land \cdots \land P_k$, it is necessary to initialize the tolerance value $z_i$ associated with each predicate $P_i$. Using the EIE property described above, this initialization reduces to setting only one value $z_l(1 \leq l \leq k)$, with the other values being automatically deduced. Algorithm 10.4 sketches this query intensification technique.

**Algorithm 10.4.** Conjunctive query intensification.

let $Q = P_1 \land \cdots \land P_k$; /* with $P_i = (A_i, B_i, a_i, b_i)$ */
/* Initialization step */
choose a predicate $P_l (1 \leq l \leq k)$;
(1): set the tolerance indicator $Z_l$ associated with $P_l$;
/* $Z_l = (-z_l, z_l, 0, 0)$ */
j := 1;
while $(j \leq k)$ do
begin
  if $(j \neq l)$ then
    compute the indicator $Z_j$ associated with $P_j$;
    /* using the EIE property */
  endif;
  j := j + 1;
end;
/* Intensification process */
i := 0;
$Q_i := Q$;
impossible := false;
compute $\Sigma_{Q_i}$;

while (not impossible) and ($|\Sigma_{Q_i}|$ is too large) do
begin
  $i := i + 1$;
  for $j = 1$ to $k$ do
  begin
    if ($i \leq (B_j - A_j)/2z_j$)
    then compute $T_j^{(i)}(P_j) := P_j \ominus i.Z_j$
    else impossible := true
    endif;
    if (not impossible) then
    $Q_i := T_1^{(i)}(P_1) \land \cdots \land T_k^{(i)}(P_k)$;
    compute $\Sigma_{Q_i}$;
    endif;
  end;
end;
if (impossible) then adjust the tolerance indicator $Z_i$; goto (1) endif;
return $\Sigma_{Q_i}$;

10.3.3 Approach based on query expansion

We now present an alternative strengthening strategy, based on query augment-ation and predicate correlation, which is mainly applicable to what may be called under-specified queries. An under-specified query $Q$ typically involves a few predicates (between one and three) to describe an expected set of answers that can be more precisely described by properties not specified in $Q$.

For example, consider a user looking for “second-hand cars which are very recent”. The answer set to this query can be reduced through the integration of additional properties like low mileage, high security and high comfort level, i.e., properties usually possessed by very recent cars.

The query strengthening approach described below aims at identifying correlation links between additional properties and an initial query. These additional correlated properties are suggested to the user as candidates for the strengthening of the initial query. This interactive process is iterated until the result is of an acceptable size for the user and corresponds to what he/she was really looking for.
10.3.3.1 Fuzzy cardinalities and association rules

Fuzzy cardinalities. In the context of flexible querying, fuzzy cardinalities appear to be a convenient formalism to represent the number of tuples from a relation that satisfy a fuzzy predicate to various degrees. It is considered that these various membership degrees are defined by a finite scale $1 = \sigma_1 > \sigma_2 > \cdots > \sigma_f > 0$. Such fuzzy cardinalities can be computed incrementally, and maintained for each linguistic label and for the diverse conjunctive combinations of these labels. Fuzzy cardinalities are represented by means of a possibility distribution (Dubois and Prade, 1985a) such as:

$$F_{P_i} = \frac{1}{\sigma_1} + \frac{1}{\sigma_2} + \cdots + \frac{1}{\sigma_f}$$

where $1 > \sigma_1 \geq \cdots \geq \sigma_f > \sigma_{f+1} = 0$ for a predicate $P_i$. Here, without loss of information, we use a more compact representation:

$$F_{P_i} = \frac{1}{c_1} + \frac{\sigma_2}{c_2} + \cdots + \frac{\sigma_f}{c_f},$$

where $c_j, j = 1, \ldots, f$ is the number of tuples in the relation concerned that are $P_i$ with a degree at least equal to $\sigma_j$. For the computation of cardinalities concerning a conjunction of $q$ fuzzy predicates, such as $F_{P_1 \land P_2 \land \cdots \land P_q}$, the calculation takes into account the minimal satisfaction degree obtained by each tuple $t$ for the concerned predicates, $\min(\mu_{P_1}(t), \mu_{P_2}(t), \ldots, \mu_{P_q}(t))$.

Association rules and correlation. Given two predicates $P_1$ and $P_2$, an association rule denoted by $P_1 \Rightarrow P_2$ aims at quantifying the fact that tuples that are $P_1$ are also $P_2$ ($P_1$ and $P_2$ can be replaced by any conjunction of predicates). As suggested in Bosc et al. (2001), fuzzy cardinalities can be used to quantify the confidence of such an association by means of either a scalar or a fuzzy set detailing the subsequent confidence for each $\sigma$-cut. The first representation of the confidence, as a scalar, is used in the following, as it is more convenient to compare fuzzy sets using scalar measures. The confidence of an association rule $P_1 \rightarrow P_2$, denoted by $conf(P_1 \rightarrow P_2)$, is thus computed as follows:

$$conf(P_1 \rightarrow P_2) = \frac{\Gamma_{P_1 \land P_2}}{\Gamma_{P_1}}.$$
fuzzy cardinalities. With the data from Example 10.6 below, the scalar 
version of $\Gamma_{\text{recent}} = 1/6 + 0.6/7 + 0.2/8$ is $\Gamma_{\text{recent}} = 1 \times 6 + 0.6 \times (7 - 6) + 0.2 \times (8 - 7) = 6.8$.

**Example 10.6.** Consider a relation named *secondHandCars* with schema

$(\text{brand}, \text{model}, \text{type}, \text{years}, \text{mileage}, \text{optionLevel}, \text{securityLevel}, \text{horsePower})$

containing advertisements for second hand cars. A sample of its extension

is given in Table 10.3. We assume that the finite scale of degrees used for
the computation of the fuzzy cardinalities is

$$\sigma_1 = 1 > \sigma_2 = 0.8 > \sigma_3 = 0.6 > \sigma_4 = 0.4 > \sigma_5 = 0.2 > 0.$$ 

From Table 10.3, we can compute the cardinalities of the predicates recent
and low-mileage (Figure 10.6) and of their conjunction:

- $F_{\text{recent}} = 1/6 + 0.8/6 + 0.6/7 + 0.4/7 + 0.2/8$,
- $F_{\text{low-mileage}} = 1/3 + 0.8/4 + 0.6/4 + 0.4/4 + 0.2/5$,
- $F_{\text{low-mileage} \land \text{recent}} = 1/3 + 0.8/4 + 0.6/4 + 0.4/4 + 0.2/5$.

From this dataset, one can deduce that the rule $P_{\text{recent}} \rightarrow P_{\text{low-mileage}}$, 
expressing that recent cars have a low mileage, has a confidence of

$$\frac{1 \times 3 + 0.8 \times (4 - 3) + 0.2 \times (5 - 4)}{1 \times 6 + 0.6 \times 1 + 0.2 \times 1} \approx 0.59,$$

and that the rule $P_{\text{low-mileage}} \rightarrow P_{\text{recent}}$ has a maximal confidence of 1 as
all low-mileage cars are also recent in Table 10.3.
10.3.3.2 Shared vocabulary

As mentioned in Ozawa and Yamada (1995), fuzzy sets can be used to represent what the authors call a “macro expression of the database”. Contrary to the approach presented in Ozawa and Yamada (1995), where this knowledge is computed by means of a fuzzy clustering process, it is, in the approach presented here, defined a priori by means of a Ruspini partition of each attribute domain. Recall that a Ruspini partition (Ruspini, 1969) is composed of fuzzy sets, where a set, say \( P_i \), can only overlap with its predecessor \( P_{i-1} \) or/and its successor \( P_{i+1} \) (when they exist), and for each tuple \( t \), \( \sum_{i=1}^{m_i} \mu_{P_i}(t) = 1 \), where \( n_i \) is the number of partition elements of the concerned attribute. These partitions are specified by an expert during the database design step and represent “common sense partitions” of the domains, instead of the result of an automatic process which may be difficult to interpret. Indeed, we assume that the predefined fuzzy sets involved in a partition constitute a shared vocabulary, and that these fuzzy sets are used by the users when formulating their queries. Moreover, it is assumed that the predicates that are added to the initial query also belong to this vocabulary.

Consider a relation \( r \) containing \( w \) tuples \( \{t_1, t_2, \ldots, t_w\} \) defined on a set \( Z \) of \( q \) categorical or numerical attributes \( \{Z_1, Z_2, \ldots, Z_q\} \). A shared predefined vocabulary on \( r \) is defined by means of partitions of the \( q \) domains. A partition \( \mathcal{P}_i \), associated with the domain of attribute \( Z_i \) is composed of \( m_i \) fuzzy predicates \( \{P_{i,1}^P, P_{i,2}^P, \ldots, P_{i,m_i}^P\} \), such that \( \forall z_i \in \mathcal{Z}(Z_i), \sum_{j=1}^{m_i} \mu_{P_{i,j}^P}(z_i) = 1 \). A predefined predicate is denoted by \( P_{i,j}^P \), which corresponds to the \( j \)th element of the partition defined on attribute \( Z_i \). Each \( \mathcal{P}_i \) is associated with a set of linguistic labels \( \{L_{i,1}^P, L_{i,2}^P, \ldots, L_{i,m_i}^P\} \).
each of them corresponding to an adjective which gives the meaning of the fuzzy predicate.

A query $Q$ to this relation $r$ is composed of fuzzy predicates chosen among the predefined predicates which form the partitions. A predicate involved in a user query is said to be specified, is denoted by $P_{k,l}^s$, and corresponds to the $l$th element of the partition associated with the domain of $Z_k$.

If $Q$ leads to a plethoric answer set, $Q$ can be strengthened in order to obtain a more restrictive query $Q'$ such that $\Sigma^\mu_{\text{max}}(Q') \subset \Sigma^\mu_{\text{max}}(Q)$. Query $Q'$ is obtained through the integration of additional predefined fuzzy predicates chosen from the shared vocabulary on $r$.

As an example, consider again the relation secondHandCars introduced in Section 10.3.3.1. A common-sense partition and labelling of the domain of attribute years is illustrated in Figure 10.7.

10.3.3.3 Strengthening steps

Correlation-based ranking of the candidates. In the approach we propose, the new conjuncts to be added to the initial query are chosen from among a set of possible predicates pertaining to the attributes of the schema of the database queried. This choice is mainly made according to the conjuncts’ correlation with the initial query. A user query $Q$ is composed of $n, n \geq 1$, specified fuzzy predicates, denoted by $P_{k_1,l_1}^s, P_{k_2,l_2}^s, \ldots, P_{k_n,l_n}^s$, which come from the shared vocabulary associated with the database. The first step of the strengthening approach is to identify the predefined predicates that are most correlated with the initial query $Q$.

Fig. 10.7 A partition of the domain of attribute years
The notion of correlation is used to qualify and quantify the extent to which two fuzzy sets (one associated with a predefined predicate $P_{i,j}$, the other associated with the initial query $Q$) are somewhat “semantically” linked. The degree of this correlation is denoted by $\mu_{cor}(P_{i,j}, Q)$. Roughly speaking, a predicate $P_{i,j}$ is considered somewhat correlated with a query $Q$ if the group of items characterized by $P_{i,j}$ is somewhat similar to $\Sigma_Q$. For instance, it may be noticed that a fuzzy predicate “highly powerful engine” is more correlated to a query aimed at retrieving “fast cars” than cars having a “low consumption”. Adding predicates that are correlated to the user-specified predicates makes it possible to preserve the scope of the query (i.e., the user’s intent) while making it more demanding. It is worth mentioning that the term correlation used in this approach means a mutual semantic relationship between two concepts, and does not have the meaning it has in statistics — where it represents similarities between series variations — but rather corresponds to the notion of set correlation.

As illustrated in Section 10.3.3.1, fuzzy association rules express such a semantic link that can be quantified by their confidence. Thus, if we consider a query $Q$ and a predefined predicate $P_{i,j}$, we compute the confidence of the association rules $Q \rightarrow P_{i,j}$ and $P_{i,j} \rightarrow Q$ according to the fuzzy cardinalities $F_Q$, $F_{P_{i,j}}$ and $F_{Q \wedge P_{i,j}}$. We then quantify the correlation degree between $Q$ and $P_{i,j}$, denoted by $\mu_{cor}(P_{i,j}, Q)$, as:

$$
\mu_{cor}(P_{i,j}, Q) = \top \left( \text{conf}(Q \rightarrow P_{i,j}), \text{conf}(P_{i,j} \rightarrow Q) \right),
$$

where $\top$ stands for a t-norm (minimum is used in the experimentation described in Section 10.3.3.5). Thus, this correlation degree is both reflexive ($\mu_{cor}(Q, Q) = 1$) and symmetrical ($\mu_{cor}(P_{i,j}, Q) = \mu_{cor}(Q, P_{i,j})$).

Using this measure, it is possible to identify the predefined predicates that are most correlated to an under-specified query $Q$. In practice, we only consider the $\eta$ most correlated predicates to a query, where $\eta$ is a technical parameter (set to 5 in the experimentation). This limitation is motivated by the fact that a strengthening process involving more than $\eta$ iterations, i.e., the addition of more than $\eta$ predicates, could lead to important modifications to the scope of the initial query. Those $\eta$ predicates that are most correlated to $Q$ are denoted by $P_{Q,1}^\eta, P_{Q,2}^\eta, \ldots, P_{Q,\eta}^\eta$.

**Reduction-based reranking of the candidates.** We have shown above how to retrieve the $\eta$ predicates most correlated with an initial query $Q$. The second step of this strengthening process aims at reranking those
η predicates according to their reduction capability. As discussed in Section 10.3.1, it is assumed that the user specifies a value for the parameter k which defines his/her expected number of answers. Let \( F_{Q \land P_{Q}^g} \), \( g = 1, \ldots, \eta \), be the fuzzy cardinality of the result set when \( Q \) is augmented with \( P_{Q}^g \). \( P_{Q}^g \) is all the more interesting for strengthening \( Q \) as \( Q \land P_{Q}^g \) contains a \( \sigma_i \)-cut (\( \sigma_i \in [0, 1] \)) with a cardinality \( c_i \) close to \( k \) and a \( \sigma_i \) close to 1. To quantify how interesting \( P_{Q}^g \) is, we compute for each \( \sigma_i \)-cut of \( F_{Q \land P_{Q}^g} \) a strengthening degree which represents a compromise between its membership degree \( \sigma_i \) and its associated cardinality \( c_i \). The global strengthening degree assigned to \( F_{Q \land P_{Q}^g} \), denoted by \( \mu_{\text{stren}}(F_{Q \land P_{Q}^g}) \), is the maximal strengthening degree of its \( \sigma_i \)-cuts:

\[
\mu_{\text{stren}}(F_{Q \land P_{Q}^g}) = \sup_{1 \leq i \leq \eta} \top \left( 1 - \frac{|c_i - k|}{\max(k, \lvert \Sigma_Q^{\sigma_i} \rvert - k)} , \sigma_i \right),
\]

where \( \top \) stands for a t-norm (minimum in our experimentation). This reranking of the predicates that are most correlated with \( Q \) can be carried out using the fuzzy cardinalities associated with each conjunction \( Q \land P_{Q}^g \), \( g = 1, \ldots, \eta \).

**Example 10.7.** To illustrate this reranking strategy, consider a user query \( Q \) resulting in a PA problem (\( |\Sigma_Q^{\text{max}}| = 123 \) and \( |\Sigma_Q^{\sigma_i} | = 412 \)), where \( k \) has been set to 50. As an example, consider the following candidates \( P_{Q}^1, P_{Q}^2, P_{Q}^3, P_{Q}^4, P_{Q}^5 \) and the respective fuzzy cardinalities:

- \( F_{Q \land P_{Q}^1} = \{1/72 + 0.8/74 + 0.6/91 + 0.4/92 + 0.2/121\}, \mu_{\text{stren}}(F_{Q \land P_{Q}^1}) \approx 0.94, \)
- \( F_{Q \land P_{Q}^2} = \{1/89 + 0.8/101 + 0.6/135 + 0.4/165 + 0.2/169\}, \mu_{\text{stren}}(F_{Q \land P_{Q}^2}) \approx 0.9, \)
- \( F_{Q \land P_{Q}^3} = \{1/24 + 0.8/32 + 0.6/39 + 0.4/50 + 0.2/101\}, \mu_{\text{stren}}(F_{Q \land P_{Q}^3}) \approx 0.93, \)
- \( F_{Q \land P_{Q}^4} = \{1/37 + 0.8/51 + 0.6/80 + 0.4/94 + 0.2/221\}, \mu_{\text{stren}}(F_{Q \land P_{Q}^4}) \approx 0.96, \)
- \( F_{Q \land P_{Q}^5} = \{1/54 + 0.8/61 + 0.6/88 + 0.4/129 + 0.2/137\}, \mu_{\text{stren}}(F_{Q \land P_{Q}^5}) \approx 0.99. \)

According to the problem definition (\( k = 50 \)) and the fuzzy cardinalities above, the following ranking is suggested to the user: 1) \( P_{Q}^5 \), 2) \( P_{Q}^4 \), 3) \( P_{Q}^1 \), 4) \( P_{Q}^3 \), 5) \( P_{Q}^2 \). Of course, to make this ranking more intelligible to
the user, the candidates are proposed alongside their associated linguistic labels (for an example, see the concrete example about used cars given in Section 10.3.3.5).

### 10.3.3.4 Strengthening process

**Precomputed knowledge.** Because the predicates specified by the user and those that may be added to the initial query are chosen among the predicates that form the domain partitions, one can precompute some useful knowledge that will make the strengthening process faster. Below, we discuss how to compute and maintain precomputed knowledge, stored in two tables.

The first table contains precomputed fuzzy cardinalities. It is of course impossible to precompute the fuzzy cardinalities for all possible conjunctions of predefined predicates, because the number of conjunctions grows exponentially with the size of the shared vocabulary. It is not even useful to store the fuzzy cardinalities of each possible conjunction, as the approach mainly focuses on under-specified queries and produces strengthened queries that contain only correlated predicates. Indeed, queries involving non-correlated predicates are less likely to return a plethoric answer set. Thus, this table only contains the fuzzy cardinalities of conjunctions involving sufficiently correlated predicates. A threshold $\gamma$ is used to prune some branches of the exploration tree of all possible conjunctions.

This table can be easily maintained as the fuzzy cardinalities can be updated incrementally. Moreover, as we will see, this precomputed knowledge is used to give interesting information — in constant time — about any under-specified query, without computing the query. In particular, it provides the size of the answer set associated with a query and the predicates that can be used to strengthen it. An index computed on the string representation of each conjunction makes this fast access possible.

The second table stores the correlation degrees between each sufficiently correlated conjunction of predefined predicates and each specified atomic predicate. For each conjunction of predicates, the $\eta$ most correlated predefined predicates are ranked in decreasing order of their correlation degree. This table is checked to generate the candidates that may be used to augment the query.

Both tables have to be updated after each (batch of) modification(s) performed on the data, but these updates involve a simple incremental calculus.
Interactive strengthening process. Recall that a query \( Q \) is assumed to be composed of fuzzy predicates chosen among the predefined predicates that form the shared vocabulary. First, the table of fuzzy cardinalities is checked, in order to test whether its fuzzy cardinality is available. If this is the case, it is then established whether the user is faced with a plethoric answer problem according to the value he/she has assigned to \( k \). If so, up to \( \eta \) candidates are retrieved — still in constant time — that are then reranked according to \( k \) and presented to the user. Finally, as illustrated in Section 10.3.3.5, the user can decide to process the initial query, to process one of the suggested strengthened queries, or to ask for another strengthening iteration of a strengthened query.

Using the aforementioned tables, this approach guides users in an interactive way from an under-specified query to a more demanding but semantically close query that returns an acceptable number of results.

10.3.3.5 Experimentation

We consider a relation secondHandCars containing 10,604 advertisements about used cars. This relation has the schema: (idAd, model, year, mileage, optionLevel, securityLevel, comfortLevel, horsePower, engineSize, price). The following shared vocabulary, composed of 41 fuzzy predicates, has been predefined to query this relation and to strengthen under-specified queries. The quadruple \((a, b, c, d)\) associated with each linguistic label defines its trapezoidal membership function (where \([a, d]\) represents the support and \([b, c]\) the core).

**year:** vintage \((28, 30, +\infty, +\infty)\), very old \((19, 20, 28, 30)\), old \((12, 14, 19, 20)\), medium \((6, 7, 12, 14)\), recent \((3, 4, 6, 7)\), very recent \((1, 2, 3, 4)\), last model \((0, 0, 1, 3)\)

**mileage:** very low \((0, 0, 20,000, 30,000)\), low \((20,000, 30,000, 60,000, 70,000)\), medium \((60,000, 70,000, 130,000, 150,000)\), high \((130,000, 150,000, 260,000, 300,000, +\infty, +\infty)\), very high \((260,000, 300,000, +\infty, +\infty)\)

**optionLevel:** very low \((0, 0, 1, 2)\), low \((1, 2, 4, 5)\), medium \((4, 5, 6, 7)\), high \((6, 7, 10, 12)\), very high \((10, 12, +\infty, +\infty)\)

**securityLevel:** very low \((0, 0, 1, 2)\), low \((1, 2, 4, 5)\), medium \((4, 5, 6, 7)\), high \((6, 7, 8, 9)\), very high \((8, 9, +\infty, +\infty)\)

**comfortLevel:** very low \((0, 0, 1, 2)\), low \((1, 2, 4, 5)\), medium \((4, 5, 6, 7)\), high \((6, 7, 8, 9)\), very high \((8, 9, +\infty, +\infty)\)

**horsePower:** very low \((0, 0, 30, 50)\), low \((30, 50, 70, 80)\), high \((70, 80, 120, 140)\), very high \((120, 140, +\infty, +\infty)\)
Engine Size: very small (0, 0, 0.8, 1.0), small (0.8, 1.0, 1.4, 1.6), big (1.4, 1.6, 2.0, 2.4), very big (2, 2.4, +∞, +∞)
Price: very low (0, 0, 2,000, 2,500), low (2,000, 2,500, 6,000, 6,500), medium (6,000, 6,500, 12,000, 13,000), high (12,000, 13,000, 20,000, 22,000), very high (20,000, 22,000, 35,000, 40,000), extremely high (35,000, 40,000, +∞, +∞)

According to these partitions, fuzzy cardinalities of conjunctions involving correlated predicates have been computed in order to populate the table of fuzzy cardinalities and then to identify the η = 5 predicates that are most correlated to each partition element. On a laptop with a classical configuration (Intel Core 2 Duo 2.53 GHz with 4Go 1067 MHz of DDR3 RAM), it took about 8 s to compute the useful fuzzy cardinalities and to store them in a dedicated table. This process can be made even faster using indexes defined on the relevant relation (i.e., secondHandCars). Using a low correlation threshold of 0.05, only 7,436 conjunctions involve predicates that are sufficiently correlated to be considered as possible under-specified queries among the 604,567 possible conjunctions that can be constructed using the shared vocabulary. Among the 7,436 conjunctions stored in the table of fuzzy cardinalities, 41 concern one predicate (one for each partition element), while there are 170 fuzzy cardinalities for conjunctions of two predicates, 382 for three predicates, 695 for four predicates, 1,040 for five predicates, 1,422 for six predicates, 1,808 for seven predicates, and 1,878 for eight predicates.

Example of an under-specified query. To illustrate the strengthening method, consider an example of a query Q composed of fuzzy predicates chosen from the shared vocabulary, involving a quantitative threshold k = 50:

\[
Q = \text{select } 50 \ast \text{ from secondHandCars where year is old}.
\]

From a string representation of Q and the table of fuzzy cardinalities, we directly check whether Q corresponds to an under-specified query. If this is the case, its fuzzy cardinality is presented to the user (obviously in a more linguistic and user-friendly way than given here): \(F_Q = \{1/179 + 0.8/179 + 0.6/179 + 0.4/323 + 0.2/323\}\).

At the same time, predicates corresponding to properties correlated to the initial query are suggested, if they exist, and are ranked in decreasing order of their reduction capability with respect to k. In this example, the
following candidates are suggested, showing the fuzzy cardinality of the corresponding strengthened queries:

1. **mileage is medium** \((\mu_{\text{cor}}(Q, P_{\text{mileage,medium}}) = 0.11))**:
   
   \[ F_{Q \land P_{\text{mileage,medium}}} = \{1/24 + 0.8/27 + 0.6/28 + 0.4/72 + 0.2/77\}. \]

2. **mileage is very high** \((\mu_{\text{cor}}(Q, P_{\text{mileage,very high}}) = 0.19))**:
   
   \[ F_{Q \land P_{\text{mileage,very high}}} = \{1/7 + 0.8/7 + 0.6/8 + 0.4/18 + 0.2/19\}. \]

3. **mileage is high** \((\mu_{\text{cor}}(Q, P_{\text{mileage,high}}) = 0.37))**:
   
   \[ F_{Q \land P_{\text{mileage,high}}} = \{1/101 + 0.8/106 + 0.6/110 + 0.4/215 + 0.2/223\}. \]

For each candidate query \(Q')\, the user can decide either to process \(Q')\ (i.e., retrieve the results) or to repeat the strengthening process on \(Q')\. If the latter option is chosen, the table of fuzzy cardinalities is checked in order to retrieve strengthening candidates for \(Q')\ (i.e., properties correlated to \(Q')\) and their associated fuzzy cardinalities that are ranked according to \(k\). In this example, even if \(k\) (50) items can be ranked and returned to the user according to this strengthened query, we suppose that the user selects:

\[ Q' = (\text{year is old}) \text{ and } (\text{mileage is medium}) \]

for second strengthening step. The following candidates are suggested along with their fuzzy cardinalities:

1. **optionLevel is low** \((\mu_{\text{cor}}(Q', P_{\text{optionLevel,low}}) = 0.34))**:
   
   \[ F_{Q' \land P_{\text{optionLevel,low}}} = \{1/18 + 0.8/20 + 0.6/21 + 0.4/46 + 0.2/51\}. \]

2. **optionLevel is very low** \((\mu_{\text{cor}}(Q', P_{\text{optionLevel,very low}}) = 0.15))**:
   
   \[ F_{Q' \land P_{\text{optionLevel,very low}}} = \{1/6 + 0.8/7 + 0.6/7 + 0.4/22 + 0.2/22\}. \]

**Remarks about this experimentation.** From this first experimentation on a concrete database, we observe that this query strengthening approach, based on correlation links, gives the users interesting information about data distributions and the possible queries that can be formulated in order to retrieve coherent answer sets. By “coherent answer set”, we mean a group of items that share correlated properties and that may correspond to what the user was looking for without knowing initially how to retrieve it.
Moreover, thanks to the precomputed knowledge tables, it is not necessary to process correlated queries (i.e., queries containing correlated predicates) to provide the user with interesting information about the size of their answer sets and the predicates that can be used to strengthen them.

This experimentation shows that the predicates suggested for strengthening the queries are meaningful and coherent according to the initial under-specified queries. Below are some examples of suggested augmented queries $Q'$ starting from under-specified queries $Q$:

- $Q = (\text{year is old})$ and $(\text{mileage is high})$ and $(\text{optionLevel is low})$:
  intensified after two iterations into $Q'' = (\text{year is old})$ and $(\text{mileage is high})$ and $(\text{optionLevel is low})$ and $(\text{comfortLevel is very low})$ and $(\text{securityLevel is low})$,
  with $|\Sigma_{Q}^{\mu_{\text{max}}(Q')}| = 63$ and $|\Sigma_{Q''}^{\mu_{\text{max}}(Q')}| = 26$.

- $Q = \text{year is recent}$:
  intensified after two iterations into $Q'' = (\text{year is recent})$ and $(\text{mileage is low})$ and $(\text{optionLevel is high})$,
  with $|\Sigma_{Q}^{\mu_{\text{max}}(Q')}| = 4060$ and $|\Sigma_{Q''}^{\mu_{\text{max}}(Q')}| = 199$.

- $Q = \text{comfortLevel is high}$:
  intensified after one iteration into $Q' = (\text{comfortLevel is high})$ and $(\text{optionLevel is high})$,
  with $|\Sigma_{Q}^{\mu_{\text{max}}(Q')}| = 180$ and $|\Sigma_{Q'}^{\mu_{\text{max}}(Q')}| = 45$.

- $Q = \text{year is very old}$:
  intensified after two iterations into $Q'' = (\text{year is very old})$ and $(\text{optionLevel is low})$ and $(\text{comfortLevel is very low})$,
  with $|\Sigma_{Q}^{\mu_{\text{max}}(Q')}| = 35$ and $|\Sigma_{Q''}^{\mu_{\text{max}}(Q')}| = 6$.

Finally, we found that the parameter $\eta$ was useless in this applied context, because the correlation threshold used during the computation of fuzzy
cardinalities already restricts the size of the correlation table to the most correlated conjunction of predicates. However, for some applied contexts involving many more predefined predicates and a low correlation threshold, $\eta$ can certainly be useful.

### 10.4 Conclusion

In this chapter, we have presented several approaches aimed at dealing with situations where a conjunctive fuzzy query returns either an empty or a plethoric answer set. This sample of techniques shows that these two problems are not as related as one might think. Indeed, each problem has strong distinctiveness. When a query $Q$ returns no answer, one may either:

- relax $Q$; but then one has to look for the minimal failing subqueries of $Q$ which cause this absence of answers, so as to target the problematic part(s) of the query (and also in order to provide the user with some explanations), or
- use a case-based reasoning approach and look for a query similar to $Q$ whose answer is known to be non-empty.

On the other hand, when a query $Q$ returns too many answers, one may think of either:

- strengthening $Q$ by making some of its atomic predicates more selective, or
- augmenting $Q$ with additional predicates in order to reduce its scope; but this latter choice requires some rationale for choosing the predicates to be added.

Despite the peculiarities of each problem, it appears that one type of approach, among those outlined above, is able to solve both problems in a somewhat symmetric way: the approach based on a fuzzy tolerance relation, which leads to dilation of the atomic predicates in the empty answer case, and to their erosion in the plethoric answer situation. However, even though the tools used are symmetric (dilation/erosion), the resulting algorithms are quite different, as we have shown in this chapter.

Among the perspectives for future research, it would be interesting to extend the scope beyond selection queries, and consider, e.g., the
empty/plethoric answer problems in the context of division queries. In the case where the result is empty, a solution is to make the division tolerant (see Section 7.3). However, the plethoric answer situation is more problematic, and the weakening/strengthening of stratified division queries is also an open issue.
Chapter 11

Conclusion

To the best of our knowledge, this book is the first extensive monograph devoted to fuzzy preference queries. The main objectives we pursued were:

- to show that fuzzy set theory constitutes a suitable and highly expressive framework for modeling database preference queries,
- to study the algorithmic aspects of the evaluation of fuzzy queries, in order to demonstrate that this framework offers a good trade-off between expressivity and efficiency,
- to tackle some specific issues somewhat related to fuzzy querying, such as bipolarity (Chapter 8) and cooperative answering (Chapter 10).

Two families of fuzzy queries have been considered:

- Unipolar fuzzy queries, leading to the computation of a single score for each tuple of the result, which makes it possible to rank-order the answers to a query (a complete preorder is obtained). We have shown that the framework considered, namely the SQLf language, makes it possible to express a wide range of preference queries while preserving the main aspects of the SQL standard.
- Bipolar fuzzy queries, made of a pair (constraint, wish), acknowledging the fact that the human mind often makes decisions on the basis of both positive and negative information. Negative preferences correspond to constraints, since they specify those values or objects that have to be rejected (i.e., those that do not satisfy the constraints), while positive preferences correspond to wishes, as they specify those objects that are more desirable than others. As shown in Chapter 8, the satisfaction of the constraint and of the wish can be used to order the tuples of the result by means of lexicographic order (with priority given to the constraint).
Fuzzy queries have many potential application domains. Indeed, soft querying turns out to be relevant in many contexts, such as information retrieval, in particular on the Web (many commercial systems, e.g., Google or Yahoo, use a technique to rank-order the answers), yellow pages, classified ads, image or multimedia retrieval. One may assume that the richer the semantics of stored information (images or video being a good example), the more difficult it becomes for the user to characterize his/her search criteria in a crisp way, i.e., using Boolean conditions. In this kind of situation, fuzzy queries which involve imprecise descriptions (or goals) and vague terms, may provide a convenient means for expressing information needs. For instance, the flexible route-planning system described in Mokhtari et al. (2009) illustrates the relevance and usefulness of fuzzy queries in the context of intelligent transportation systems. It is important to emphasize that, even though relational data have been assumed throughout this book, many results can be transposed to other contexts, such as information retrieval or multimedia database querying.

Among the topics which have not been dealt with in this book but are closely related to its subject, it is appropriate to mention:

- The handling of “competitive conditional preferences”. This corresponds to a new type of database query introduced in Bosc et al. (2011), whose format is somewhat outside the scope of an SQL-like language since it involves conditional requirements (which is more in the spirit of a procedural programming language). The idea is to consider preference clauses structured as a tree, of the type “preferably $P_1$ or ... or $P_n$; if $P_1$ then preferably $P_{1,1}$ or ...; if $P_2$ then preferably $P_{2,1}$ or ...”, where the $P_i$’s are not exclusive (hence the notion of competition).
- The modeling of contextual preferences. In Hadjali et al. (2010), a fuzzy-rule-based model is proposed for the representation of contextual preferences in a database querying framework. The idea is to augment a query with fuzzy preferences deduced from information regarding the current context of the user. We have not presented this type of approach here since it is not a purely relational one (it implies coupling a DBMS with a rule-based system, which is somewhat beyond the scope of this book).

Considering the benefits of the fuzzy-set-based approach (particularly in terms of expressivity), one may wonder why this approach is not more
prominent in the database world when it comes to database preference queries. In our opinion, the two main reasons are related to:

- The strong priority given to efficiency over expressivity by the database community (hence the popularity of Skyline queries, top-k queries, etc). Obviously, there is still some effort required in order to convince mainstream database researchers that fuzzy queries may be efficiently processed, and we hope this book will contribute to that effect.

- The relative difficulty for an end-user to specify fuzzy membership functions and choose appropriate fuzzy operators. However, as discussed in Chapter 5, it is perfectly possible to conceive user-friendly interfaces to make this task as easy as it is in qualitative approaches such as those based on Pareto-order (e.g., Preference SQL; Kießling and Köstler, 2002).

To conclude, let us say a few words about a confusion which has not helped make work about fuzzy queries particularly “readable” for the mainstream database community in the past, namely the use of the expression “fuzzy databases”. This term actually covers two orthogonal issues: fuzzy queries on the one hand, and uncertain data handling on the other hand. Unfortunately, many research papers about “fuzzy databases” mix both issues, and it is often quite difficult to discern what the fundamental issues are; not to mention the fact that very few papers about fuzzy — in the sense of uncertain — databases have a solid theoretical basis. Even though these two issues have a non-empty intersection, we believe it wise to treat them separately in a comprehensive way before investigating that intersection. Both preference queries and uncertain databases are rather old issues (in each case, the first works date back to the eighties), but it is only quite recently that they have become “hot topics” in the database community. Many uncertain database models based on probability theory have been proposed, particularly in the last decade (see, e.g., Benjelloun et al., 2008; Dalvi and Suciu, 2004; Antova et al., 2007 for some recent ones, and Bose and Pivert, 2010 for an overview and a critical discussion). However, it is quite clear that not all uncertain information is of a probabilistic nature. A typical example is that of a person who witnesses a car accident and who does not remember for sure the model of the car involved. In such a case, it seems reasonable to model the uncertain value by something more qualitative than a probability distribution — which, by the way, would have to be artificially normalized. Several qualitative approaches based on possibility theory (Zadeh, 1978; Dubois and Prade, 1988) — whose strong ties with
fuzzy set theory must be emphasized here — have been proposed in the recent years, among which are Bosc et al. (2003a), Bosc and Pivert (2005, 2007b,a), and Bosc et al. (2009d, 2010c). However, not much work has yet been done on the subject of preference queries to possibilistic databases (we mention Bosc and Pivert, 2004; Bosc et al., 2010b), and many research issues remain to be investigated in that domain. In particular, there is an open question over the extent to which fuzzy queries, such as those studied in this book, could be adapted to a possibilistic database context.
Bibliography


Fuzzy Preference Queries to Relational Databases


Bibliography

Processing in Databases (Information Science Reference, Hershey, PA), pp. 97–114.


Fuzzy Preference Queries to Relational Databases


Bibliography


Index

ACCESS 106, 128
aggregate function 24, 27, 28, 104, 131, 160–182, 252, 253, 254, 256, 257
alpha-cut 35, 39, 74, 114, 115, 121, 161, 171, 257
answer
  cooperative 268, 276
  empty 267–287
  plethoric 267, 287–306
antidivision 19, 29, 30, 185, 207
antonymy 232
ARES 78
association rule 251, 258, 261, 263, 264, 265, 295
bipolarity 221–249

cardinality 35, 60, 99, 140, 258, 259
Cartesian product
  of bipolar relations 226, 238, 246
  of fuzzy relations 93, 188, 189
  of fuzzy sets 43
  of usual relations 8, 9, 12, 22
  of usual sets 43
case-based reasoning 279, 306
ceteris paribus 85, 86
characteristic function 33, 34
commensurability 84, 109
compatible attributes 6, 8, 15, 26
conjunction 38, 187, 244, 257
  noncommutative 53, 54, 188, 195
  weighted 41, 92, 108
CP-net 85, 86, 88
DBMS 2, 7, 21, 106, 114, 121, 219, 258
derivation 114, 117, 121, 122, 125, 127, 129
difference
  of bipolar relations 232–235, 246
  of fuzzy relations 93, 100, 102, 103, 104
  of fuzzy sets 41, 42, 62
  of usual relations 8, 10, 11
dilation 271–273, 289, 306
discrimin 40
disjunction 38
  noncommutative 53, 54
  weighted 41, 92
distance 78, 79, 81
division 17–19, 29, 185–220
  algebraic expression 19
  bipolar 240, 247
  in SQL 29
  in SQLf 192
  of bipolar relations 240, 247
  of fuzzy relations 185, 211, 239
  of usual relations 11, 17, 18, 19, 29, 239

327
primitivity 190, 244
stratified 198–211
tolerant 193–197, 213–215
domain linearization 87
erosion 288–290, 306
extended relational algebra 92, 93, 222, 224
extension principle 71
FQUERY 128
FSQL 128
fuzzy cardinality 295, 300, 302–304
fuzzy integral 68, 70, 132–138
fuzzy measure 68, 69, 74, 135, 136
fuzzy ordering 107
fuzzy partition 111, 251, 252, 253, 254, 261, 264
fuzzy pattern matching 140
fuzzy predicate 3, 91, 92, 93, 106, 108, 109, 110
fuzzy quantifier 64, 73, 74, 100, 104, 106
fuzzy relation 92, 93, 96, 102, 103, 121
fuzzy set 2, 3, 31, 33, 34
  alpha-cut 35, 39, 74, 114, 115, 121, 161, 257
cardinality 35, 60, 99, 140, 258, 259
  complement 36, 43
  composition 36
  core 35, 106, 273, 290
  height 35, 62, 99
  intuitionistic 223, 228, 248
  normalized 35, 73
  support 35, 106, 259, 261, 273, 290
twofold 222, 227
graded equality 63
horizontal quantification 150
implication
  contraposition 54, 55
  fuzzy 50, 52, 54, 55, 56, 220, 240
  Gödel 52, 53, 56, 57, 58, 188, 289
  Goguen 52, 53, 56, 57, 58
  Kleene-Dienes 51, 56, 61, 188
  Łukasiewicz 51, 52, 54, 56, 57, 58, 189
  Reichenbach 51, 56, 61, 63
  Rescher-Gaines 52, 55, 56, 57, 58
  R-implication 52, 53, 54, 55, 56, 58, 188, 189, 191, 194, 259, 260
  S-implication 51, 54, 55, 56, 57, 62, 188, 189, 191, 195
inclusion
  graded 59, 60, 61, 62, 65, 187, 220, 256, 259
  regular 58
  strong 60
  tolerant 64, 66, 193, 194
integral
  Choquet 70, 132, 135, 139, 147, 157, 182
  fuzzy 68, 70, 132–138
  Lebesgue 70
  Sugeno 70, 74, 132, 135, 139, 141, 142, 155, 161, 182
interface 106, 112, 121, 129
intersection
  of bipolar relations 224, 246
  of fuzzy relations 93, 102, 103, 104
  of fuzzy sets 36
  of usual relations 8, 10
  weighted 41
involutivity 36, 226, 229, 230, 231
join
  antijoin 17, 99
  equijoin 16, 99, 100
  natural join 16
  of bipolar relations 238, 247
  of fuzzy relations 93, 95, 103, 116, 122
  of usual relations 11, 14, 15, 21, 22
  selfjoin 16
  semijoin 17
  theta-join 11, 14
Index

laws
de Morgan 37, 48, 232
double distributivity 39
excluded middle 38, 39
idempotency 38
non-contradiction 38
leximin 40, 88

membership function 31, 33, 106, 107,
110, 111, 112, 115
modifier 91, 92
MULTOS 79

negation 92, 119, 167–168, 226
nested query 25, 95–102, 104, 105,
119, 123, 127, 150, 173–175
nonsymmetric operators 45

operator/operation
algebraic 8
binary 8, 14
combination 14, 24, 25
nonsymmetric 45
relational 8
set-oriented 8, 26
unary 8, 11, 12

order
lexicographic 204, 222, 234
Pareto 83, 88, 108
partial 84
ordinal scale 199, 201, 208
outranking 81
OWA 44, 45, 92, 132, 142–149,
157–160

Pareto order 83, 88, 108
possibilistic logic 88, 89
possibility distribution 140
preference 2, 3,
contextual 109
juxtaposition 82, 83
nesting 82
query 2, 77, 85
relation 77, 86, 87
Preference SQL 84, 108
PREFERENCES 82

projection
of bipolar relations 237, 247
of fuzzy relations 93, 95, 116, 122
of usual relations 11, 12, 13, 22

quantified statement/proposition 73,
131–160, 220, 263
quantifier 64, 73, 74, 263
absolute 73, 75, 133, 144, 146
existential 73, 75
fuzzy 64, 73, 74, 100, 104, 106,
131–160
negation 147
relative 73, 75, 133, 145
universal 73, 265

query
augmentation 294
bipolar 221–249
contextual 108
equivalences 20, 25, 94, 96, 98,
100, 101, 102, 103, 104, 105,
125, 127
evaluation-processing 114, 121,
122, 129, 152–160, 176–182,
211–219, 246, 262, 265
expansion 294
fuzzy 91
intensification 275, 288, 290–293
relaxation 268–279, 291
skyline 83, 84, 85
top-k 81
under-specified 294, 299,
301–303, 305
quotient 17, 186, 188, 194, 195, 197,
205, 210, 242

relation 5
attribute 6
cardinality 6
degree 6
domain 5
key 6
schema 6
relational algebra 1, 8, 22, 92, 93,
222, 224
antijoin 17, 99
bipolar 221–249
Cartesian product 8, 9, 12, 22, 93, 226, 238
difference 8, 10, 11, 26, 30, 93, 232–235
division 11, 17, 18, 19, 29
equijoin 16, 99, 100
intersection 8, 10, 26, 93, 224
join 11, 14, 15, 21, 22, 93, 238
natural join 16
projection 11, 12, 13, 21, 22, 93, 99, 235
selfjoin 16, 23
semijoin 17
union 8, 9, 93, 225
relational database 5
relational model 5
selection
of bipolar relations 235, 246
of fuzzy relations 93, 95, 99, 108, 116, 122
of usual relations 11, 13, 21, 22
similarity 78, 81
Skyline 83, 84, 85
SQL 1, 21, 91
base block 22, 25
nesting 25
partitioning 27, 28, 29, 30
subqueries 25, 26, 29, 96–102, 150
SQLf 92, 94, 110, 114, 149, 252
base block 95
nesting 95–102, 104, 105, 119, 123, 127, 150, 173–175
partitioning 104
query evaluation/processing
strengthening 48, 49
subquery
SQL 25, 26, 29, 110

SQLf 95–102, 104, 105, 119, 123, 127, 150, 173–175
Sugeno integral 70, 74
summarization 251, 254, 255, 263, 264
support
of a fuzzy relation 191
of a fuzzy set 35, 259, 261
trade-off operators 44, 45, 108
mean 44, 45
OWA 44, 45, 92, 132, 142–149, 157–160
weighted mean 44, 45, 92
triangular conorm (t-conorm) 36, 37, 54, 92, 224
Hamacher 39
Lukasiewicz 38
maximum 37, 38, 39
probabilistic 38
Weber 37, 51
Zadeh 39
triangular norm (t-norm) 36, 37, 54, 92, 224
Hamacher 39
Lukasiewicz 38
minimum 37, 38, 39, 40
probabilistic 38
Weber 37
Zadeh 39
union
of bipolar relations 225, 246
of fuzzy relations 93, 102
of fuzzy sets 36
of usual relations 9
weighted 41
unit interval 33, 34, 36, 39, 50, 56
VAGUE 78
winnow 84, 89, 248