Fuzzy Logic Programming
Based on $\alpha$-Cuts

Thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

by

Rafee E. Ebrahim, B.Sc. Eng., MBA

Faculty of Computing Sciences & Engineering
De Montfort University

February 2003
BEST COPY AVAILABLE.

VARIABLE PRINT QUALITY
PAGE MISSING IN ORIGINAL
Abstract

This research provides foundations to Fuzzy Logic Programming based on $\alpha$ cuts. This is done via a sound and complete formal development of several Fuzzy Logic Programming systems, namely, FLP1, FLP2 and FLP3.

FLP1 is presented with its model theory, proof theory and fixpoint theory which are shown to be equivalent. The more general system FLP2 is founded on a new notion of pseudo-complementarity as well as a doubting factor. It is studied with its model theory, proof theory and fixpoint theory. A system meta-interpreter is presented to emphasize the results.

In FLP3, disjunctive logic programming for which fixpoint theory and model-state semantics are extended to fuzzy disjunctive logic programs with new theorems proved. Soundness & Completeness results are obtained for all FLP1, FLP2 and FLP3.

It was particularly observed that the $\alpha$-cut approach to fuzzy logic programming provided soundness/completeness results that are as robust as that of classical logic programming. This was due to the preservation of the Principle of Sufficiency of Herbrand models. Yet, the development differed in many ways than that of classical logic programming. There are syntactic differences as well as semantic which is given using Herbrand interpretations. New fixpoint operators are defined which are used to reach the desirable characteristics.

Furthermore, a framework that incorporates relational databases, logic programming together with fuzzy set theory is developed. The result is a foundation for fuzzy relational databases, fuzzy logic programming and fuzzy deductive databases. The relational model is extended to handle fuzziness through the introduction of two original notions: domain-associated fuzziness and tuple-associated fuzziness, then the Fuzzy Relational Model is formally proposed. Relational algebra is extended with fuzzy algebraic operations establishing the Fuzzy Relational Algebra. Fuzzy tuple relational calculus and fuzzy domain relational calculus are formally developed. It is also shown that fuzzy tuple calculus is reducible to fuzzy domain calculus, fuzzy relational algebra to fuzzy tuple calculus and fuzzy domain calculus to the fuzzy relational algebra. It was found that the correspondence between the new notion of pseudo-complementarity in fuzzy logic programming on the one hand and that of the tuple-associated fuzziness in fuzzy relational databases on the other adds to and emphasizes the isomorphism between the model-theoretic and proof-theoretic views of fuzzy databases. Finally, the theory of fuzzy logic programming, fuzzy logic founded on a new notion of pseudo-complementarity forms a logical reconstruction for Fuzzy Expert Systems through a formal system of computational fuzzy logic.
## Contents

Acknowledgments ................................................................. 2  
Abstract ................................................................................. 3  
Contents .................................................................................. 4  
Chapter 1: Introduction ............................................................. 7  
1.1 Context and Objectives ......................................................... 7  
1.2 Contributions ..................................................................... 11  
1.3 Structure of the Thesis ......................................................... 12  
2.1 Implemented Systems of Fuzzy Logic Programming ................. 15  
2.2 Application of Fuzzy Set Theory to Databases ....................... 17  
Chapter 3: Fuzzy Logic Programming: FLP1 ................................ 23  
3.1 Syntax of FLP1 ................................................................. 24  
3.2 Semantics of FLP1 ............................................................ 26  
3.3 Proof Theory ..................................................................... 39  
3.4 Soundness and Completeness of FLP1 .................................. 42  
3.5 Negation in Fuzzy Logic Programming .................................. 46  
3.6 Application to Fuzzy Constraint Satisfaction Problem ............. 47  
Chapter 4: Fuzzy Logic Programming Founded on a New of Pseudo-complementarity: FLP2 .................................................. 51  
4.1 Syntax & Semantics ........................................................... 51  
4.2 Model Semantics & Fixpoint Semantics ................................ 53
4.3 Proof Theory .................................................. 55
4.4 Soundness & Completeness of FLP2 .................................. 57
4.5 A System Meta-interpreter ............................................. 58
4.6 Computational Complexity ............................................. 61

Chapter 5: Fuzzy Disjunctive Logic Programming: FLP3 ............... 64
5.1 Syntax of FLP3 .................................................... 65
5.2 Model-State Semantics & Fixpoint Theory ............................ 67
5.3 Procedural Semantics .................................................. 70
5.4 Soundness of Fuzzy SLI Resolution ................................... 73
5.5 Completeness of Fuzzy SLI Resolution ................................ 73
5.6 Negation in Fuzzy Disjunctive Logic Programming ............... 75

Chapter 6: The Fuzzy Relational Model ................................... 82
6.1 Motivating the Fuzzy Relational Model .............................. 82
6.2 The Structural Part of the Model .................................... 83
6.3 The Fuzzy Relational Algebra ....................................... 85
6.4 The Fuzzy Tuple Relational Calculus ................................ 90
6.5 Fuzzy Tuple Calculus Expressions .................................. 92
6.6 The Fuzzy Domain Relational Calculus .............................. 96
6.7 Reduction of Fuzzy Tuple Calculus to Fuzzy Domain Calculus ....... 98
6.8 Reduction of Fuzzy Relational Algebra to Fuzzy Tuple Calculus .... 99
6.9 Reduction of Fuzzy Domain Calculus to Fuzzy Relational Algebra .... 100
6.10 Examples ............................................................. 102
Chapter 7: Fuzzy Deductive Databases ............................................. 107

7.1 Background ........................................................................... 107
7.2 Fuzzy Databases: Two Views .................................................... 108
7.3 The Coupling Approach ............................................................ 110
7.4 The Integration Approach ........................................................ 111
7.5 Negation in Fuzzy Deductive Databases ............................... 117
7.6 Implementation Issues .............................................................. 118

Discussion & Conclusion .............................................................. 121

Appendix ....................................................................................... 125

X.1 Some Fuzzy Mathematical Concepts ...................................... 125
X.2 Basic Set-Theoretic Operations on Fuzzy Sets ........................ 127
X.3 Basic Logic Programming ....................................................... 133

Bibliography ............................................................................... 143

Publication: Fuzzy Logic Programming, Fuzzy Sets & Systems ........... 159
Chapter 1

Introduction

1.1 Context and Objectives

Logic is the science of valid arguments. Crucial to a valid argument is the notion of a consequence. The notion of a consequence takes the logician's mind from a belief of the truth of a certain proposition to another. An example of a valid argument is: All British are European, X is British. Then, X is European. This is a form of a valid argument. The argument is not only valid for any X, but also the form of the argument is valid for any property other than British and other than European. For instance: All British are programmers. X is British. Then X is a programmer. Here, the conclusion is false. However, the argument is valid. The conclusion is false since the first proposition is false. So, logic studies the science of valid arguments: common forms of arguments, discovering common properties in deducing a conclusion from given premises.

There are two notions of studying a logical consequence which is deducing conclusions from a given set of premises. One is the notion of syntax with its deeper issues of proof theory. The other is that of semantics with its deeper issues of model theory (Lloyd [107]). The syntax specifies all the symbols used in a particular logic system and how well-formed formulae are formulated. Deeper proof-theoretic issues define the notion of consequence fully syntactically. Through proof theory, new propositions are deduced from old ones by mere form and syntax regardless of their meaning.

On the other hand, semantics is concerned with the meaning of the symbols of the language. The meaning of the well-formed formulae and their truth. Through model theory, new propositions are deduced from old ones by judging their truth.
Soundness and completeness of the system dictates that these two notions of consequence must be equivalent. In other words, whatever is deduced syntactically, must also be proven semantically. Completeness of the system dictates that whatever is true from a semantic point of view must also be provable syntactically.

What is Logic Programming & Computational Logic: Bearing in mind the notions above: syntax & semantics, proof theory & model theory, soundness & completeness, many logicians became interested in mechanizing human thought. Apart from the ancient Greek philosophers and logicians, probably the earliest was Leibniz (see e.g. Blasius et al. [19]) who thought through the creation of a formal language: *lingua characteristica*, human thought can be formulated and instead of two good-willed philosophers arguing, they can submit their argument in this formal language and through a particular calculus, a conclusion for their argument can be found and justified.

Three hundred years passed after Leibniz without even approaching this stage. Nevertheless, a great effort has been achieved under which all the automated deduction systems of today have their foundation (Blasius et al. [19]). In addition, it led to the discovery of the limitations of all computing machinery and based the foundations for computer science through the positive/negative celebrated results of completeness, incompleteness and undecidability (Blasius et al. [19]).

Until 1972, logic was used in computer science only as a specification language. Then, a procedural meaning for logic was discovered which made it possible to be used as a programming language. The programming language PROLOG with its different variations was the outcome of this effort (Lloyd [107]).

What is Fuzziness: fuzziness is a theory of graded concepts. So, if a set is considered in its classical sense, an element either belongs to the set or does not belong to it. In fuzzy set theory, an element may belong to a set to a
certain degree. In classical logic, a proposition may be true or false. In fuzzy logic, a proposition may be true to some degree. Fuzzy set theory and fuzzy logic provide a precise framework for dealing with vague concepts, their representation, manipulation and inference (Zimmermann [155]).

Despite the success of fuzzy logic and fuzzy set-based methods in solving many problems specially in the area of fuzzy control (Zimmermann [155]), some critics cast serious doubts about the foundations of fuzzy logic. Elkan [52] proved that fuzzy logic collapses to two-valued logic and that fuzzy logic is impossible. This was based on faulty assumptions and misunderstood mathematical reasoning. Yet, it generated a big discussion as seen in the large number of papers in a separate IEEE fuzzy logic symposium issue [156].

In 1998, Hajek demonstrated very powerfully that fuzzy logic is possible in his book: “Metamathematics of Fuzzy Logic” (Hajek [85]), several fuzzy logic systems with their syntax, semantics, model theory and proof theory were presented.

In the same year, The author published the paper: “Fuzzy Logic Programming” - a system for fuzzy logic programming was presented, its syntax, semantics, model theory, proof theory and fixpoint theory together with their equivalence proven. This system forms the foundation for other systems proposed in this thesis.

The work of Hajek established that fuzzy logic can indeed have its own rigorous system of syntax, semantics and deduction. The work in this thesis adds to it by presenting deductive fuzzy logic systems in automated deduction. This is done by presenting several systems of fuzzy logic programming. The work builds and invalidates critics doubts and claims against fuzzy logic. In fuzzy logic and fuzzy set-based methods a distinction is made between two directions (Dubois et al. [46]):

1. degrees of partial truth which are allowed to be truth-functional and which pertain to gradual (or fuzzy) propositions, and
2. degrees of uncertainty which cannot be compositional with respect to all the connectives when attached to classical propositions.

An example of this difference is between fuzzy logic and possibilistic logic (Dubois et al. [46]). The problem tackled in this thesis is the development of a family of fuzzy logic programming systems based on $\alpha$ cuts. A predicate in such a logic is a fuzzy one with partial truth value attached to it. After the cut is made, it becomes classical. So, the direction of this research is that of item 1 above. Still, it compares nicely to that of item 2 (possibilistic logic). Roughly speaking, a predicate in the systems presented in this thesis is interpreted fuzzily and becomes classical after the cut, while a predicate in possibilistic logic is interpreted classically with a weight attached to it. The analogy between the systems presented in this thesis and possibilistic logic enhances these two directions of fuzzy logic and fuzzy set theory-based methods.

The application of these fuzzy logic programming systems to fuzzy relational databases and fuzzy deductive databases is also presented. The fuzzy researcher who needs a motivation to logic programming is advised to consult the appendix as well as the logic programming researcher who needs a motivation to fuzzy logic.

FLP1 is a direct extension of classical logic programming based on $\alpha$ cuts. FLP2 extends FLP1 with a doubting factor as well as an original notion of pseudo-complementarity. FLP3 extends those two systems with disjunctive information. In the development, a notation which parallels that of classical logic programming is used as the theory of fuzzy logic programming is developed. The advantage of the fuzzy logic programming approach to other ad-hoc systems employing fuzzy logic is the soundness and completeness results obtained for the system. This means if the system is going to be a real implementation of the systems FLP1, FLP2 and FLP3, the user is quite sure that the system produces correct answers and that all correct answers can be produced. While some of the other
approaches to fuzzy logic programming may enjoy this advantage as well, the reader will notice that the systems presented in this thesis do have a very well understood semantics (by the advantage of the cut/new fixpoint theory) and their behaviour is supported by the soundness and completeness results obtained.

1.2 Contributions

Outlined below are the contributions of the work or what is original is sketched. In this respect, the following points are noted:

1. **Fuzzy Logic Programming:** The work in this area resulted in the author’s publication [50]. This paper forms the foundation of the Fuzzy Logic Programming part of this thesis: system FLP1 which is further built on cuts. Original theorems are proved showing the equivalence between the declarative semantics, minimal model semantics, fixpoint semantics (new fixpoint operators defined) and procedural semantics for fuzzy logic programs.

2. **Pseudo-complementarity:** In the more generalized class of fuzzy logic programming systems FLP2, the system FLP1 is extended to deal with the case of inference when the goal and the clause head are not complementary. The notion of pseudo-complementarity is introduced and a new system is defined with a doubting factor doubting the rule to a degree.

3. **A Fuzzy Meta-Interpreter Developed:** An original fuzzy meta-interpreter was developed in IC-Prolog. The system can be associated with any Prolog program and given a fuzzy goal, the system responds with the truth level for that goal.

4. **Fuzzy Disjunctive Logic Programming:** the results are extended to disjunctive logic programming: FLP3. Results were developed establishing model-state semantics and fixpoint theory for fuzzy disjunctive logic programs. Soundness and completeness results are obtained for FLP1, FLP2 and FLP3.
5. **New Fuzzy Database Notions:** Two new notions of fuzzy databases are introduced. They are: domain-associated fuzziness and the tuple-associated fuzziness. They offer a structured and an easier way to perceive fuzziness within relational database systems. At the same time, conventional query languages are ready to work as the classical relational data structure has been retained.

6. **Fuzzy Database Theorems Proved:** a rigorous and a coherent mathematical theory of fuzzy databases is developed for the fuzzy relational model, algebra and calculus. Some new and original theorems are proven: reduction of fuzzy tuple calculus to fuzzy domain calculus, reduction of fuzzy relational algebra to fuzzy tuple calculus and reduction of fuzzy domain calculus to fuzzy relational algebra.

The result being that the fuzzy logic practitioner and engineer has a solid theoretical background if systems built around FLP1, FLP2 and/or FLP3 are employed. Engineers as well as fuzzy technologists would be doing what they are doing better. Furthermore, the soundness results secure that all inference done by the system is correct. The completeness results ensure that every possible information that is correct can be shown. This is not the situation in most of the applications of fuzzy logic where there are no formal results as most of these systems are not based on any formal system of logic. This has been noted by Martin et al. [116] specially when they say:“whatever fuzzy control is doing, it certainly is not logic” (Martin et al. [116] pp. 4).

**1.3 Structure of the Thesis**

The structure of the thesis is as follows:

1. Chapter 2: Systems of Fuzzy Logic Programming, a critical survey of the area is presented motivating research in this direction.
Chapter 3: Fuzzy Logic Programming, FLP1, the system FLP1 is presented, its syntax, declarative semantics, minimal model semantics, fixpoint semantics and procedural semantics are presented. The equivalence of these semantics is shown. The soundness and completeness of the system are proven.

Chapter 4: Fuzzy Logic Programming Founded on a new Notion of Pseudo-Complementarity, FLP2, where the system of FLP1 is extended with this new notion that enables the system to make inference in some cases where the system of FLP1 cannot, thus the system FLP2 is obtained. FLP2 syntax, semantics, soundness and completeness are proven.

Chapter 5: Fuzzy Disjunctive Logic Programming, FLP3, where the systems of FLP1 and FLP2 are extended to deal with disjunctive information. The syntax, semantics, soundness and completeness of the new system are proven.

Chapter 6: Fuzzy Relational Model, where two new notions of fuzziness that capture fuzziness in a fuzzy database system: domain-associated fuzziness, and tuple-associated fuzziness are introduced. The fuzzy relational algebra, fuzzy relational tuple calculus, fuzzy relational domain calculus are presented and proven equivalent.

Chapter 7: Fuzzy Deductive Databases, where concepts from fuzzy logic programming systems FLP1, FLP2 and FLP3 together with the Fuzzy Relational Model presented are incorporated.

Discussion: at the end of the thesis, a discussion on the relationship of this work to others and possible future work based on this thesis is presented.

Appendix: this appendix contains an introduction to fuzzy set theory concepts as well as that of classical logic programming followed by bibliography.
Chapter 2

Systems of Fuzzy Logic Programming

A Survey

Several deductive systems of fuzzy logic has been proposed, e.g. Gödel logic, Lukasiewicz logic and Product logic (Hajek [85]). Logic programming systems usually use resolution so the survey will be confined to the systems using resolution.

The first attempt for using resolution in fuzzy logic was carried out by Lee [103]. Lee's formulae are syntactically identical to that of classical logic, but semantically they are different. An interpretation may assign a truth value to a formula which if \( \geq 0.5 \), the formula is considered valid, otherwise inconsistent. Lee [103] proved a property of his system which is that for any two clauses \( C_1 \) and \( C_2 \) if \( \min(C_1, C_2) = a > 0.5 \) and \( \max(C_1, C_2) = b \) under some interpretation \( I \), then \( a \leq R(C_1, C_2) \leq b \) for each resolvent \( R \) of \( C_1 \) and \( C_2 \).

The problem of the above approach is that they do not express the truth values at a syntactic level unlike Novak [122], where a weight is introduced at the front of the formula or in operator fuzzy logic as Liu et al. [105] where a weight is introduced as a prefix to the formula. This prefix weight may have its values not only as a real number between \([0,1]\) but also in a lattice. The logic is truth functional not only with the respect to the usual connectives but also when they are prefixed with the valuation operator.
On the theoretical side, there has been the contributions by van Emden [141] who introduces weights in the Horn clauses of logic programming. Fixpoint theory is developed in the fuzzy case as in the classical case. This approach was followed by the author in [50] and in the systems presented in this thesis. It has also been adopted by Vojtas [144] where a system of a collection of fuzzy connectives is introduced. The syntax, semantics, model theory and proof theory are handled in a manner which is similar to that of the author in [50]. In this thesis, new fuzzy logic programming systems are developed based on cuts. Now, the implemented systems of Fuzzy Logic Programming are presented.

2.1 Implemented Systems of Fuzzy Logic Programming

1. Lee’s results have been implemented by Shen et al. [136] and Mukaidono et al. [118]. This work resulted in a more general form of resolution. They defined the fuzzy resolvent which is $R(C_1, C_2) \lor (l \land \lnot l)$, where $(C_1, C_2)$ is the classical resolvent and $l$ is a literal. The result of their work is LbFP, which is a fuzzy Prolog system based on Lukasiewicz implication.

2. An FPROLOG interpreter by Martin et al. [115,116]: It is familiar that a logic program (a set of first-order logic statements in clausal form) could be read procedurally or declaratively. In fuzzy logic programming the procedural reading is similar to the ordinary case except that a truth value has to be computed from the truth values of fuzzy procedures used. The declarative reading is such that the goal is shown to be consistent to some degree with the database. The solution is an instance in which the query is true to some degree, given by a suitable combination of the truth values of clauses used in the derivation of the solution. FPROLOG has been implemented in Lisp, for the facilities of list processing, symbol manipulation, construction of arbitrary data structures and garbage collection. It required a list-based syntax rather the common Edinburgh syntax. This
is the reason why it is based on the micro-Prolog syntax. The interpreter uses a standard that represents complex terms by structure sharing. The default is to take the minimum of truth values but it can be overridden by any combination of them by a combination operator and even to redefine the values corresponding to “true” and “false”. If only solutions with a truth value greater than some threshold are required, the system can backtrack as soon as it uses a fact with a truth value lower than the threshold, rather than solving the query and then finding that the truth value is lower than the threshold. The “not” operator is also extended so that if a goal succeeds with a value $\lambda$, its negation will have the truth value $1 - \lambda$. This may correspond to success or failure according to the values of $\lambda$ and the threshold. FPROLOG can run in conjunction with Fril to call on breadth-first search mechanism, and to use Fril base relations as though they were part of the FPROLOG database. The FPROLOG was rewritten using the C language to achieve maximum portability. Later on, the Fril system can deal with fuzziness and probability and enables the user to run Prolog like code which is executed in a depth-first manner.

3. f-Prolog Interpreter by Li et al. [104]: An f-Prolog interpreter was implemented using IF/Prolog which is very similar to the Edinburgh syntax which provides ready-made facilities for matching and backtracking. In addition, IF/Prolog provides a complete SQL interface to the relational database Oracle. Both of the above systems have the capability of backtracking as soon as the system uses a certain value which is lower than the defined threshold. Martin calls it partial failure of the query, Li calls it generalized failure of the query. The system also has a linguistic extension which can make it produce answers in a sort of “definite, possible, very possible, fairly possible”, which is computed using a possibility distribution and associating it with the final answer. Also, it incorporates built-in fuzzy
comparison operators like "approximately equal" and "much greater than".

4. Prolog-ELF

Ishizuka and Kanai [91] implemented Lee's definitions and results. So, no syntactic difference from the classical logic. But at the semantic level formulae can take a truth value $\geq 0.5$ and the standard min/max operations are used for conjunction and disjunction and $1 - \mu$ for negation. No uncertainty is allowed in attribute values. Multiple proof paths are combined using $\text{max}$ considering that as the favourable solution.

The disadvantage of these systems approach is that they are not based on soundness/completeness proof unlike that of van Emden and Vojtas. The approach presented in this thesis is in the spirit of van Emden. Due to the use of cuts, the principle of sufficiency of Herbrand models is preserved in addition to the soundness/completeness proofs.

2.2 Application of Fuzzy Set Theory to Databases

Fuzzy set theory can extend the expressiveness of relational databases. Queries can include vague and imprecise terms. This could be more natural in the context of many applications (Kerre [96]). Also, fuzzy set theory can extend deductive databases with capabilities of reasoning with uncertainty. Uncertainty management is a fundamental issue in expert systems (Kandel [98]) for which the systems of fuzzy logic programming is providing theoretical background. The application of fuzzy set theory to databases is surveyed.

The underlying mathematical structure for relational databases is ordinary set theory. The relational algebra can be understood through set-theoretic operators. The relational calculus as a query language can be considered as a variation of the predicate calculus (set theory is a model of the predicate calculus). If databases are to deal with vague, imprecise and uncertain or even unknown data which could often be the case, fuzzy set theory as a mathematical structure offers
ways toward solution for these problems (vagueness, uncertainty), meanwhile subsuming ordinary relational databases (Kerre [96]).

In extending relational databases to fuzzy relational databases, fuzzy relational algebraic operators are defined in various ways. Relational calculus can be understood with the help of fuzzy logic rather than ordinary first-order logic. In the following, the developments of the application of fuzzy set theory to databases are surveyed.

2.2.1 Theoretical Developments in Fuzzy Databases

Umano [140] proposes a possibility distribution (definition X.2.8) for a fuzzy relational model in which fuzzy data are represented by the possibility distribution itself. A fuzzy relational algebra is defined using the traditional set-theoretic operators. Projection, join, restriction and division are newly defined. Within this framework, queries like "Get all pairs whose ages are approximately equal", "Find all persons who are about 36 years old"... could be run.

Prade et al. [124] introduced an extra element e to indicate that a value for an attribute is inapplicable, i.e. precise and imprecise values as well as uncertainty concerning the value of an attribute are all represented by possibility distributions. They have also fuzzified the concept of functional dependencies between two attributes using fuzzy proximity relations. Based on the dual concept of possibility and necessity measures, they have generalized the relational calculus to handle queries that include fuzzy comparators "approximately equal", "much greater than".

Buckles [24] developed a framework that differs from the classical relational model in two ways:

1. The attribute values for the tuples need not be single-valued, but still they are ordinary subset of the underlying domain.

2. He required that each domain is provided with a similarity relation which
in most cases max-min transitive and indicates to what extent two elements of the same domain set can be considered as interchangeable.

The "possible" values of an attribute are represented by a crisp collection of elements that are interchangeable with respect to the similarity threshold, the concept of redundant tuples is defined. Redundant tuples are eliminated by taking the union of their corresponding components. The removal of redundant tuples gave rise to a unique result, due to the max-min similarity condition of the similarity relation which is \( s(x, y) \) of a given domain \( D_i \) maps every pair of the domain onto the unit interval \([0,1]\):

\[
s : D_i \times D_i \rightarrow [0,1]
\]

It is a generalization of the equivalence relation, it is:

- reflexive \( s(a, a) = 1 \)
- symmetric \( s(a, b) = s(b, a) \)
- max-min transitive \( s(a, c) \geq \max \{\min[s(a, b), s(b, c)]\} \forall b \in D_i \)

Buckles [24] gave an example of a baseball team with various similarities and dissimilarities between pairs of positions. Shenoi et al. [137] showed using proximity relations (similarity relations as above with the removal of max-min transitivity condition) provides database users with more freedom to express their value structures. Meanwhile, preserving the nice characteristics of their model which hold for classical databases: no two tuples have identical interpretation and each relational operation has a unique result.

Buckles [24] also extended the fuzzy database with fuzzy numbers. He cannot extend the similarity relation employed in the case of discrete sets in the case of continuous sets because the transitivity property fails to hold, i.e. there is no transitivity property that effects partitioning of the domain set in a manner that guarantees uniqueness of relation representation. He solved this problem by introducing the concepts of \( \alpha \)-proximity relations (\( \alpha \) is a convexity condition). He
proved that removing redundant tuples leads to well-defined relational commands such as select, project, join... etc.

The two above approaches [possibility distributions, similarity relations] were the main ideas of extending the relational model using fuzzy set theory. That is to say, fuzzy databases. It will be clear in the following chapters how these ideas can be improved. The fuzzy relational model proposed includes both of the above features without sacrificing any of the desirable properties of the relational model.

Functional dependencies have also been fuzzified by Raju et al. [129,130] and Bosc et al. [21]. The functional dependency between sets of attributes have been derived and found to have the same reflexive, augmentation and transitive property like that of classical relational databases. They have developed a set of sound and complete inference rules. Furthermore, they studied the problem of lossless join decomposition of fuzzy relations. Later on, Bosc et al. [21] studied fuzzy functional dependencies in the context of fuzzy database design and the role they play in capturing the semantics of databases and how to achieve redundancy elimination.

Dealing with the problem of null values, the earliest attempt in classical databases was Codd’s extension of the relational model (extending the relational model to capture more meaning, in [33]), where null values with the third value “unknown” i.e. were introduced. Thus, an answer to a query divides a relation into three parts, a true result, a false result and a maybe result, contrary to true-false in the traditional relational model.

Li et al. [104] proposed the use of the special values UNDEFINED, UNDECIDED and NULL. They showed a way to handle these in their query languages FSQL (a fuzzified version of SQL) and f-Prolog (a fuzzified version of Prolog).

The above survey shows that no unified framework to incorporate fuzzy databases with fuzzy logic programming was proposed. One contribution of this
research is to fill in this gap in the literature and provide a foundation for fuzzy deductive databases.

2.2.2 Implemented and Proposed Systems of Fuzzy Databases

A Fuzzy Relational Language (FSQL): The system FSQL developed by Li and Liu [104], FSQL (Fuzzy Structured Query Language) can handle fuzzy and incomplete data appropriately. A WITH clause is added to the familiar SELECT/FROM/WHERE of standard SQL. The WITH clause determines the threshold value which is a real number in the closed interval [0,1]. The authors went a step further and allowed a set of linguistic qualifiers that could be used in the WHERE and the WITH clause, quoting their example:

```
SELECT Name
FROM candidates
WHERE Age=ABOUT 25
WITH VERY possible
```

The system also includes the usage of fuzzy comparison operators which is implemented via possibility distributions. Fuzzy sorting is included by the extension of the ORDER-BY clause of standard SQL. This facility is implemented by defining comparisons between fuzzy values. The definition of the comparison is also based on possibility distributions.

Bosc et al. [20] addressed the issue of fuzzy querying against a usual database. To deal with a specific application they have modified the language SQL to include some fuzzy querying capabilities. They suggested further work in how to deal with fuzzy relations. They concluded that their experiment reveals to be a good validation for their approach.

Kacprzyk et al. [94] introduced the idea of fuzzy linguistic quantifiers in queries. Imprecise queries such as “find the record in which almost all, much more than 75 percent”, etc. can be run.

In an attempt to classify fuzzy databases or to establish a taxonomy for fuzzy
databases, Ronki et al. [133] considered the approaches:

1. The *similarity relations* approach (Buckles [24]).

2. The *fuzzy logic* approach.

3. The *possibility distribution* approach.

4. The *fuzzy linguistics* in query system.

While Kerre [96] classified fuzzy database systems as:

1. Fuzzy-relation-based.

2. Similarity-based.

3. Possibility-based.

4. Extended-Possibility-based.

5. Combined.

This taxonomy is helpful in comparing the different approaches taken to fuzzy databases. In this chapter, one can see several approaches to fuzzy logic programming. Most of them lacked soundness/completeness proofs and none of them was closer to classical logic as the one proposed here based on $\alpha$-cuts. In the following chapters, several fuzzy logic programming systems will be developed followed by a proposal for a fuzzy relational model. Further, no attempt has been made to incorporate fuzzy logic programming and fuzzy databases together under a unified framework. This is done when the final chapter of the thesis is reached: fuzzy deductive databases. In the discussion, a comparison between the approach adopted in this thesis and other approaches is made. In the following chapter, the first system of fuzzy logic programming FLP1 is presented.
Chapter 3

Fuzzy Logic Programming: FLP1

Fuzzy Logic Programming is developed with the goal to provide a theoretical background for fuzzy expert systems. With the system of fuzzy logic programming that is presented the researcher and the practitioner would have the theoretical results that the system produces only correct answers and all the possible correct answers. A new notion of satisfiability and logical consequence is introduced. A minimal model is proposed. The equivalence between the declarative, minimal model, fixpoint and procedural semantics for fuzzy logic programs is proven through a series of theorems.

In the following section, the syntax of the first fuzzy logic programming system FLP1 is introduced. In section 3.2, the semantics of the syntax presented in section 3.1 is presented. The declarative semantics is linked to the minimal-model semantics and is shown equivalent in this section. Satisfiability is defined, then Herbrand interpretations. The fixpoint closure operator is defined in this section and the fixpoint characterization of the least Herbrand model is shown. At this point, the equivalence between the declarative, minimal-model, fixpoint semantics of fuzzy logic programs is reached. In section 3.3, a proof procedure for the fuzzy logic programming system presented earlier is proposed by modifying the classical procedure of SLD-resolution. The SLD-derivation and the SLD-refutation are defined. The concept of a correct answer and that of a computed answer are also defined. In section 3.4, soundness and completeness results are reached by showing that every instance of a correct answer is an instance of a computed answer and vice versa. In section 3.5, it is shown that fuzzy logic can
be very useful in handling negation in logic programming. In section 3.6, the fuzzy logic programming system presented is applied to the Fuzzy Constraint Satisfaction Problem together with an illustrative example.

3.1 Syntax of FLP1

The alphabet of FLP1 has the following different kinds of symbols:

1. **Constants.**

2. **Variables.**

3. **Function Symbols.**

4. **Predicate Symbols.**

5. **Classical Connectives.**

6. **Quantifiers:** FLP1 has $\forall$ and $\exists$ as classical logic programming systems.

7. **Punctuation symbols:** (, and brackets) in writing formulae.

In order to define the set of well-formed formulae in FLP1 an atomic formula should be defined. Atomic formulae are formed using terms. So, a term will be defined inductively as follows:

1. A variable is a **term**.

2. A constant is a **term**.

3. If $f$ is an $n$-ary function symbol and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a **term**.

4. **Terms** can only be constructed by applying the above rules a finite number of times.
Steps 1 and 2 above form the basis step of this inductive definition, step 3 the induction and step 4 the closure. An atomic formula in $\text{FLP1}$ can now be defined. The definition of the atomic formula will be the basis for the inductive definition of the well-formed formulae of $\text{FLP1}$. The following definition defines a cut for a fuzzy set.

**Definition 3.1.1:** The (crisp) set of elements that belong to the fuzzy set $A$ at least to the degree $\alpha$ is called the $\alpha$-level set or $\alpha$-cut:

$$A_\alpha = \{ x \in X | \mu_A(x) \geq \alpha \}$$

**Example 1.1.1**[155]: A realtor wants to classify the offers to his clients. One indicator of comfort is the number of bedrooms in a house. Let $X = \{1, 2, 3, \ldots, 10\}$. Then the fuzzy set “comfortable type of house for a 4-person family” may be described as:

$$A = \{(1, .2), (2, .5), (3, .8), (4, 1), (5, .7), (6, .3)\}$$

the possible $\alpha$-level sets are listed:

- $A_{0.2} = \{1, 2, 3, 4, 5, 6\}$
- $A_{0.5} = \{2, 3, 4, 5\}$
- $A_{0.8} = \{3, 4\}$
- $A_1 = \{4\}$

In the notation, $\mu$-cut is used instead of $\alpha$ since $\alpha$ is used in a different context of pseudo-complementarity in $\text{FLP2}$. But when $\mu$ is subscripted, it denotes a membership function.

**Definition 3.1.2:** An atomic formula $F$ is a pair $< p(t_1, \ldots, t_n), \mu >$ where $t_1, \ldots, t_n$ are terms and $\mu$ is a term (constant, variable or a function called annotation) whose value would be taken in a linear order in $[0, 1]$.

The set of well-formed formulae can be defined inductively as follows:

**Basis:** any atomic formula is a *formula*.

**Induction:**
1. If $F$ and $G$ are formulae, then $(\neg F), (F \land G), (F \lor G)\ and (G \leftarrow F)$ are formulae as well. Since the pair $<p, \mu>$ is crisp, so all the connectives in the systems presented in this thesis are classical connectives, and so are the quantifiers.

2. If $F$ is a formula and $x$ is a variable, then $\forall xF$ and $\exists xF$ are formulae.

The same definitions (as in classical logic) for the scope of a quantifier and bound and/or free occurrence of a variable are used. The language of FLP1 may include term equations that define the cut operation as a function.

**Definition 3.1.3:** A *program clause* of FLP1 is a clause of the form:

$$< A(t_1, \ldots, t_n), \mu_A > \leftarrow < B_1(t_1, \ldots, t_n), \mu_{B_1} >, \ldots, < B_n(t_1, \ldots, t_n), \mu_{B_n} >$$

which contains precisely one atom (atomic formula) in its consequent. It is to be noted that the number of terms within the atoms $t_i$ could be different. It is assumed that all the variables in the clause are universally quantified. Note, function terms can appear only in the head of a clause and not in its body and may depend on the logical variables as well as the annotation variables in the clause.

**Definition 3.1.4:** A *program* is a finite set of program clauses.

**Definition 3.1.5:** A *goal* is a clause of the form:

$$\leftarrow < B_1(t_1, \ldots, t_n), \mu_{B_1} >, \ldots, < B_n(t_1, \ldots, t_n), \mu_{B_n} >$$

### 3.2 Semantics of FLP1

In this section, the semantics of the FLP1 whose syntax was presented in the previous section is presented. Given the same formula, one can have a different meaning in FLP1 system and classical logic programming. As a start, truth values of fuzzy predicates are to be taken in a linear order in $[0, 1]$. The meaning of the connectives $\text{AND}, \text{OR}, \text{NOT}$ and the implication is the same as classical connectives since $<p, \mu>$ is crisp.
Semantics for the symbols of the languages are given by an *interpretation*. Then, satisfiability, validity, consistency and models are defined. The counterparts of Herbrand universe, Herbrand base, Herbrand interpretation and Herbrand model are introduced. The fuzzy program clause is defined with its meaning. The logical consequence is defined to provide the declarative semantics. Minimal-model semantics is established and it is shown that the atoms in the minimal model are the ones that are logical consequence of the fuzzy logic program. So, the proved theorems show the equivalence between the declarative semantics and the minimal model semantics. A new fixpoint closure operator is defined and a theorem to characterize the minimal model as the least fixpoint of this closure operator is proven.

**Definition 3.2.1:** Let $P$ be a set of wff in FLP1, the *Herbrand universe* $U_P$ for $P$ is the set of all ground terms (appearing in the logical part), which can be formed out of the constants and function symbols appearing in $P$.

**Definition 3.2.2:** Let $P$ be a set of wff in FLP1, the *fuzzy Herbrand base* $FB_P$ for $P$ is the set of all ground atoms (after removing the annotation part) which can be formed out by using the predicate symbols from $P$ with ground terms from the Herbrand universe as arguments.

**Definition 3.2.3:** The *Herbrand interpretation* $I$ is defined as $I : FB_P \rightarrow [0,1]$.

**Definition 3.2.4:** Let $F$ be a formula in FLP1 and $I$ an Herbrand interpretation for this formula, then $F = \langle p(t_1, \ldots, t_n), \mu \rangle$ (where $\mu$ is a constant) is *true under* $I$ if $I$ interprets $p$ to the fuzzy relation $r$ and $r(t_1, \ldots, t_n) \geq \mu$.

**Definition 3.2.5:** Let $I$ be an Herbrand interpretation of a language of FLP1 and let $F$ be a formula in this language. Let $F$ be the atomic formula $\langle p(t_1, \ldots, t_n), \mu \rangle$, where $\mu$ is a constant for the moment. Then:

1. $F$ is *satisfiable* in $I$ if there exists terms $t_1, \ldots, t_n$ such that $\langle p(t_1, \ldots, t_n), \mu \rangle$ is true under $I$.

2. $F$ is *valid* in $I$ if for all terms $t_1, \ldots, t_n$, $\langle p(t_1, \ldots, t_n), \mu \rangle$ is true under $I$.  

27
An Herbrand interpretation for an atomic formula \(< p(t_1, \ldots, t_n), \mu >\) which makes it true is an Herbrand model of the formula.

**Definition 3.2.6:** A variable assignment for an annotation variable in the formula \(F = < p(t_1, \ldots, t_n), \mu >\) is an assignment \(V = \{ \mu / \mu_c \}\) for the variable \(\mu\) assigning \(\mu_c\) which is a constant in \([0, 1]\).

**Definition 3.2.7:** A formula \(F = < p(t_1, \ldots, t_n), \mu >\) is satisfiable under an Herbrand interpretation \(I\) and a variable assignment \(V\) if \(I\) assigns the fuzzy relation \(r\) to \(p\) and \(V\) assigns the constant \(\mu_c \in [0, 1]\) to \(\mu\) and \(r(t_1, \ldots, t_n) \geq \mu_c\).

**Definition 3.2.8:** A formula \(F = < p(t_1, \ldots, t_n), f(x_i, \mu_i) >\) is satisfiable under an Herbrand interpretation \(I\) if \(I\) assigns the fuzzy relation \(r\) to \(p\) and \(f(x_i, \mu_i)\) evaluates to \(\mu_c \in [0, 1]\) and \(r(t_1, \ldots, t_n) \geq \mu_c\). The satisfiability of a clause is the same as classical logic since the clause is a classical one.

**Definition 3.2.9:** Let \(P\) be a fuzzy program and \(F\) be a closed formula. Then, \(F\) is a logical consequence of \(P\) if, for every interpretation \(I\) of \(P\), \(I\) is a model for \(P\) implies that \(I\) is a model for \(F\).

**Example 3.2.1:** This example clarifies the relation between the syntax and semantics of FLP1. Let us consider two interpretations \(I_1\) and \(I_2\) on a same universe \(D\). Interpretation \(I_1\) maps \(p\) to a relation \(r_1 : D \times D \rightarrow \{0, 1\}\) (i.e. classical relation), while \(I_2\) maps \(p\) to a relation \(r_2 : D \times D \rightarrow [0, 1]\). Now, one can say that predicate \(p\) is classical under the interpretation \(I_1\) and \(p\) is fuzzy under the interpretation \(I_2\). Then given \(\mu \in [0, 1]\) as a cut, indeed the universe \(D \times D\) is cut into two parts: \(\{(a, b) \in D \times D\) such that \(r_2 : D \times D \rightarrow [0, 1]\) \(\geq \mu\}\) and \(\{(a, b) \in D \times D\) such that \(r_2(a, b) < \mu\}\). Let's consider \(I_2\) as above, with \(D = \{a, b, c\}\), which maps \(p\) to the following fuzzy relation \(r_2 : D \times D \rightarrow [0, 1]\):

\[
\begin{align*}
r_2(a, a) &= 0.1 \\
r_2(a, b) &= 0.5 \\
r_2(a, c) &= 0.8 \\
r_2(b, a) &= 0.2
\end{align*}
\]
\[ r_2(b, b) = 0.3 \\
\[ r_2(b, c) = 0.4 \\
\[ r_2(c, a) = 0.2 \\
\[ r_2(c, b) = 0.3 \\
\[ r_2(c, c) = 0.1 \\
\]
and which maps object constants \( m, n \) to elements of \( D \), \( a \) and \( b \) respectively:

1. \( < p(m, n), 0.3 > \), where \( m \) and \( n \) are object constants. Under \( I_2, p(m, n) \) gets the truth value \( r_2(a, b) = 0.5 \). Since \( 0.5 \geq 0.3 \). So, \( I_2 \) satisfies the atomic formula \( < p(m, n), 0.3 > \), hence \( I_2 \) is a model of \( < p(m, n), 0.3 > \).

2. \( < p(m, y), 0.3 > \), where \( m \) is an object constant and \( y \) is a variable. Then in this case, we need a “variable assignment” for \( y \) to compute the truth value for \( p(m, y) \). Let us consider two possible variable assignments: \( v_1 \), such that \( v_1(y) = a \) and \( v_2 \), such that \( v_2(y) = c \). Then, it turns out that the truth value of \( p(m, y) \) under \( I_2 \) and \( v_1 \) is \( r_2(a, a) = 0.1 \), hence since \( 0.1 < 0.3 \), we say that \( < p(m, y), 0.3 > \) is not satisfied under \( I_2 \) and \( v_1 \). However, under \( I_2 \) but with \( v_2, p(m, y) \) gets a truth value of \( r_2(a, c) = 0.8 \), hence \( p(m, y) \) is satisfied under \( I_2 \) and \( v_2 \).

3. \( (\forall y) < p(m, y), 0.3 > \). Here, one has to evaluate \( p(m, y) \) for all possible value assignments for \( y \), so \( v_1(y) = a, v_2(y) = c \), and \( v_3(y) = b \), and one has respectively, \( r_2(a, a) = 0.1, r_2(a, c) = 0.8 \) and \( r_2(a, b) = 0.5 \). But, by definition, \( (\forall y) < p(m, y), 0.3 > \) is satisfied by \( I_2 \) iff it is satisfied for every instance of \( y \), hence since for \( v_1(y) = a \) it is not, then \( (\forall y) < p(m, n), 0.3 > \) is not satisfied under \( I_2 \).

**Example 3.2.2:** Let \( F \) be the formula \( < p(m, n), \mu > \), where \( \mu \) is a variable. Let \( I_2 \) be the interpretation as above which maps \( m \) to \( a \) and \( n \) to \( b \) and \( p \) to the fuzzy relation above and \( V \) the variable assignment \( V = \{ \mu / 0.3 \} \) then one has: \( p \) is 0.5-true under \( I_2 \) and \( F \) is satisfiable under \( I_2 \) and \( V \).
A statement which has a model is called a consistent statement in classical logic. Similarly, a set of statements are called consistent if they have a model. In other words, if they can be made true under some interpretation. The same case holds for FLP1.

Example 3.2.3: Suppose we have the following formulae:

\[ < \text{Tall}(\text{John}), 0.8 > \]
\[ < \text{Tall}(\text{George}), 0.7 > \]
\[ < \text{Tall}(\text{Angela}), 0.6 > \]
\[ < \text{Tall}(\text{Angelina}), 0.5 > \]

1. Suppose we have the wff clause:

\[ < \text{GoodBasketBallPlayer}(x), 1 > \leftarrow < \text{Tall}(x), 0.65 > \]

which expresses that somebody is a good basketball player (with degree 1) whenever his degree of tallness is not smaller than 0.65. Then with the above facts, we would get as only logical consequences:

\[ < \text{GoodBasketBallPlayer}(\text{John}), 1 > \]
\[ < \text{GoodBasketBallPlayer}(\text{George}), 1 > \]

2. Suppose one considers the wff clause:

\[ < \text{GoodBasketBallPlayer}(x), \mu > \leftarrow < \text{Tall}(x), \mu > \]

where \( \mu \) is now a “cut”-variable, which expresses that somebody is a good basketball player as much as he is tall. Then, one would have as consequences:

\[ < \text{GoodBasketBallPlayer}(\text{John}), 0.8 > \]
\[ < \text{GoodBasketBallPlayer}(\text{George}), 0.7 > \]
\[ < \text{GoodBasketBallPlayer}(\text{Angela}), 0.6 > \]
\[ < \text{GoodBasketBallPlayer}(\text{Angelina}), 0.5 > \]
3. Suppose one considers the wff clause:

\(< \text{GoodBasketBallPlayer}(x), f(\mu) > \leftarrow < \text{Tall}(x), \mu >\)

where again \(\mu\) is a "cut"-variable, and \(f(\mu)\) is a cut term, for instance:

(a) \(f(\mu) = \mu\), if \(\mu \geq 0.65\).

(b) \(f(\mu) = 0\), if \(\mu < 0.65\).

Such a clause expresses that somebody is a good basketball player as much as he is tall, provided his degree of tallness is not smaller than 0.65. In such a case:

\(< \text{GoodBasketBallPlayer}(John), 0.8 >\)

\(< \text{GoodBasketBallPlayer}(George), 0.7 >\)

**Example 3.2.4** Let's consider to have the height as facts, not the degree to which somebody is tall as this being more natural. Then, from this data, one can compute their degree of tallness by means of some suitable membership function for "being tall". Suppose \(g_{\text{tall}} : [0, 250]\text{cm} \rightarrow [0, 1]\), is the preferred membership function for tall, which gives for any height in cm, the degree of tallness. For instance, \(g_{\text{tall}}(190) = 0.8, g_{\text{tall}}(180) = 0.7, g_{\text{tall}}(170) = 0.6\) and \(g_{\text{tall}}(160) = 0.5\). Now, consider the following program:

\(< \text{GoodBasketBallPlayer}(x), g_{\text{tall}}(y) > \leftarrow < \text{height}(x, y), 1 >,\)

\(< \text{height}(John, 190), 1 >\)

\(< \text{height}(George, 180), 1 >\)

\(< \text{height}(Angela, 170), 1 >\)

\(< \text{height}(Angelina, 160), 1 >\)

Then, with the goal:

\(?\cdash < \text{GoodBasketBallPlayer}(x), z >\)

Then, one has:

\(< \text{GoodBasketBallPlayer}(John), 0.8 >\)
< GoodBasketBallPlayer(George), 0.7 >
< GoodBasketBallPlayer(Angela), 0.6 >
< GoodBasketBallPlayer(Angelina), 0.5 >

and if the rule is changed to: < GoodBasketBallPlayer(x), f(y) > ←< height(x, y), 1 >

where the function f is defined as:

1. \( f(y) = g_{tall}(y) \), if \( g_{tall}(y) \geq 0.65 \)

2. \( f(y) = 0 \), otherwise.

one has the following consequences:

< GoodBasketBallPlayer(John), 0.8 >
< GoodBasketBallPlayer(George), 0.7 >
< GoodBasketBallPlayer(Angela), 0 >
< GoodBasketBallPlayer(Angelina), 0 >

Note that the cases considered in this example cannot be formulated in the theory of annotated logic programs Kifer et al. [98'] but the previous examples could be. What is new in this example is including the annotation term to be a function which is a membership function depending on logical variables appearing in the body of the clause.

**Theorem 3.2.1:** Let \( S \) be a set of clauses and suppose \( S \) has a model. Then \( S \) has an Herbrand model.

**Proof:** Let \( I \) be an interpretation of \( S \). Another Herbrand interpretation \( I' \) can be defined as:

\[ I' = \{ < p(t_1, \ldots, t_n), \mu > \in B_S : p(t_1, \ldots, t_n) \text{ is true in } I \} \]

Another interpretation \( I' \) can be constructed with terms only from the Herbrand base then it is straightforward to show that, if \( I \) is a model, \( I' \) is an Herbrand model.
Definition 3.2.10: If $FM_1$ is a model and $FM_2$ is a model, then $FM_1 \cap FM_2$ contains the atoms in both of $FM_1$ and $FM_2$.

Theorem 3.2.2: (Model Intersection Property)
Let $P$ be a fuzzy program and $\{FM_i\}_{i \in I}$ be a non-empty set of Herbrand models for $P$. Then $\cap_{i \in I} M$ is an Herbrand model for $P$.

Proof: For two models $FM_1$ and $FM_2$, $FM_1 \cap FM_2$ is a model from the above definition which can be extended to any number of finite models.

Theorem 3.2.3: Let $P$ be a fuzzy program and $F$ be a closed formula. then $F$ is a logical consequence of $P$ iff $P \cup \{\neg F\}$ is unsatisfiable.

Proof: Suppose that $F$ is a logical consequence of $P$. Let $I$ be an interpretation of $P$ and suppose $I$ is a model of $P$. Then $I$ is a model for $F$. Hence, $I$ is not a model for $P \cup \{\neg F\}$. Thus $P \cup \{\neg F\}$ is unsatisfiable. Conversely, suppose $S \cup \{\neg F\}$ is unsatisfiable. Let $I$ be any interpretation of $P$. Suppose $I$ is a model for $P$. Since $P \cup \{\neg F\}$ is unsatisfiable, $I$ cannot be a model for $\neg F$. Thus, $I$ is a model for $F$ and so $F$ is a logical consequence of $P$.

Theorem 3.2.4: Let $P$ be a fuzzy program of FLP1 and $FM_P$ be the minimal model for $P$. Then,

$$FM_P = \{A \in FB_P : A \text{ is a logical consequence of } P\}$$

Proof: $A$ is a logical consequence of $P$
if $P \cup \{\neg A\}$ is unsatisfiable, by Theorem 3.2.3.
if $P \cup \{\neg A\}$ has no Herbrand models.
if $\neg A$ cannot be satisfied by any Herbrand model.
if $A$ can be satisfied by all Herbrand models.
if $A \in FM_P$.

Definition 3.2.11: Let $P$ be a fuzzy definite program, the mapping $FT_P : [0,1]^{FB_P} \to [0,1]^{FB_P}$ (denoting the set of fuzzy subsets of $FB_P$) is defined as follows. Let $I$ be an Herbrand interpretation, then:
\[ FT_P(I) = \{ A \in \mu_A FB_P : \mu_A = \text{sup}\{ f(x_i, \mu_{B_i}) \} | A \leftarrow B_1, \ldots, B_n \text{ is a ground instance of a clause in } P \text{ and } \{B_1, \ldots, B_n\} \subseteq I \}, x_i \text{ are the variables appearing in } B_i. \]

The function \( f(x_i, \mu_{B_i}) \) can be a function of the logical variables as in previous examples or can be a function in the truth values of the body as in the following examples which could be taken as the minimum function, i.e. \( f(x_i, \mu_{B_i}) = \text{min}(\mu_{B_i}) \). The reader is strongly advised to compare the definition of the closure operator and compare it with the classical case as well as comparing the soundness/completeness proofs.

**Example 3.2.5:**

\[ \langle \text{GoodCreditCustomer}(x), \mu_1 \rangle \leftarrow \langle \text{BalanceLevel}(x), \mu_2 \rangle, \]
\[ \langle \text{OrderFrequency}(x), \mu_3 \rangle \]

which reads as a customer is to be considered as having good credit as his balance level and order frequency, whichever is lower.

\[ \langle \text{BalanceLevel}(John), 0.7 \rangle \leftarrow\]
\[ \langle \text{OrderFrequency}(John), 0.6 \rangle \leftarrow\]
\[ \langle \text{BalanceLevel}(Richard), 0.8 \rangle \leftarrow\]
\[ \langle \text{OrderFrequency}(Richard), 0.7 \rangle \leftarrow\]
\[ \mu_1 = f(\mu_2, \mu_3) = \text{min}(\mu_2, \mu_3) \]

Consider the goal \( \leftarrow \text{Good-Credit-Customer}(Richard, \mu) \) which results into the minimum of 0.8 and 0.7 which are the truth values of the two subgoals. \( \mu_1 = \text{min}(0.8, 0.7) = 0.7 \).

**Example 3.2.6:**

\[ \langle \text{MatureStudent}(x), \mu_1 \rangle \leftarrow \langle \text{Student}(x), 1 \rangle, \langle \text{AgeAbout21}(x), \mu_2 \rangle \]

which reads a student is a mature student as much as his age is near to 21 years.

\[ \langle \text{AgeAbout21}(John), 0.9 \rangle \leftarrow\]
\[ \langle \text{AgeAbout21}(Peter), 0.4 \rangle \leftarrow\]
\[ \langle \text{Student}(John), 1.0 \rangle \leftarrow \]
\(<\text{Student}(\text{Peter}), 1.0>\leftrightharpoons\mu_1 = \mu_2\)

Here, there are three predicate symbols, namely \text{Student}, \text{MatureStudent} and \text{AgeAbout21}. Now, given the goal \(\leftarrow \text{MatureStudent}(\text{John}), \mu\), which will unify the head of the first rule with unification \((x=\text{John}, \mu = \mu_1)\). Thus, resulting into two sub-goals, the first \text{Student}(\text{John}) which succeeds with 1. The other sub-goal is \(\leftarrow \text{AgeAbout21}(\text{John}), \mu_2\) which succeeds with the value \(\mu_2 = 0.9\) for John. Then \(\mu_1 = \mu_2 = 0.9\).

\textbf{Example 3.2.7:}

\(R1: \left< p(x, y), \mu_{p_1} \right> \leftrightharpoons \left< q(x), \mu_{q_1} \right>, \left< r(y), \mu_r \right>\)

\(R2: \left< p(x, y), \mu_{p_2} \right> \leftrightharpoons \left< q(x), \mu_{q_2} \right>, \left< s(y), \mu_s \right>\)

\(R3: \left< q(m), 0.3 \right> \leftrightharpoons\)

\(R4: \left< r(x), \mu_r \right> \leftrightharpoons \left< t(x), \mu_t \right>\)

\(R5: \left< s(n), 1 \right> \leftrightharpoons\)

\(R6: \left< t(n), 0.4 \right> \leftrightharpoons\)

\(\mu_{p_1} = \min(\mu_{q_1}, \mu_r)\)

\(\mu_{p_2} = \min(\mu_{q_2}, \mu_s)\)

\(\mu_r = \mu_t\)

Consider the fuzzy goal \(\left< p(m, n), 0.3 \right>\) which unifies with the first fuzzy rule giving the two fuzzy sub-goals:

1. \(\left< q(m), \mu_{q_1}, \mu_{q_1} \geq 0.3\right>\)

2. \(\left< r(n), \mu_r, \mu_r \geq 0.3\right>\)

The fuzzy sub-goal (1) unifies with R3 and succeeds while the second fuzzy subgoal unifies with R4 and results with another two fuzzy sub-goals with the second being \(\mu_r \geq 0.3\) resulting in the goal \(\left< t(n), 0.3 \right>\) which succeeds when unifying with R6. As a result, the original goal \(\leftarrow p(m, n, 0.3)\) succeeds as far as
matching with rule R1 is considered. When matching with rule R2, two fuzzy sub-goals are generated, they are:

1. \( \leftarrow< q(m), \mu_{q_2}, \mu_{q_2} \geq 0.3 \)

2. \( \leftarrow< s(n), \mu_s, \mu_s \geq 0.3 \)

The first successfully matches with R3 and the second as well with R5. So, the original fuzzy goal succeeds in this case.

Now consider the fuzzy goal \( \leftarrow< p(m, n), 0.2 > \) when matching with R1 two fuzzy sub-goals are generated, namely:

1. \( \leftarrow< q(m), \mu_{q_1}, \mu_{q_1} \geq 0.2 \)

2. \( \leftarrow< r(n, \mu_r), \mu_r \geq 0.2 \)

The first fuzzy sub-goal of (1) \( \leftarrow< q(m), \mu_{q_1} \) unifies with R3 giving \( \mu_q = 0.3 \) and as a result the second fuzzy sub-goal \( \mu_q \geq 0.2 \) succeeds. For the second fuzzy sub-goal \( \leftarrow< r(n), \mu_r >, \mu_r \geq 0.2 \), one has only rule R4 which unifies successfully resulting in the goal \( \leftarrow< t(n), 0.2 > \) which succeeds when unifying with R6. As a result, the original fuzzy goal \( \leftarrow< p(m, n), 0.2 > \) succeeds.

When matching with R2, two fuzzy sub-goals are generated, namely:

1. \( \leftarrow< q(m), \mu_q >, \mu_q \geq 0.2 \)

2. \( \leftarrow< s(n), \mu_s >, \mu_s \geq 0.2 \)

The first fuzzy sub-goal matches with R3 and succeeds. The second fuzzy subgoal matches with R5 and succeeds.

Now consider a fuzzy goal with a variable \( \mu \), i.e. \( \leftarrow< p(m, n), \mu > \), matching with R1, the following is obtained:

1. \( \leftarrow< q(m), \mu_q >, \mu_q \geq \mu \)
2. \( \leftarrow < r(n), \mu_r >, \mu_r \geq \mu \)

The first matches with R3 and \( \mu_q = 0.3 \), thus solving \( \mu \leq 0.3 \). The second will unify with rule R4 then rule R6 returning \( \mu \leq 0.4 \). The original goal succeeds with \((\mu \leq 0.3) \land (\mu \leq 0.4)\). Thus \( \mu \leq 0.3 \). When matching with rule R2, two fuzzy sub-goals are generated:

1. \( \leftarrow < q(m), \mu_q >, \mu_q \geq \mu \)
2. \( \leftarrow < s(n), \mu_s >, \mu_s \geq \mu \)

The first matches with R3 giving \( \mu \leq 0.3 \). The second matches with R5 giving \( \mu \leq 1 \). The original goal succeeds with \( (\mu \leq 0.3) \land (\mu \leq 1) \lor [(\mu \leq 0.3) \land (\mu \leq 0.4)] \). Thus \( \mu \leq 0.3 \).

The following lemmas demonstrate results for \( FT_P \) similar to that developed for possibilistic logic programming (Dubois [40]) and many-valued lattice-based logic programming (Subrahmanian [138']).

**Lemma 3.2.1:** \( FT_P \) is monotonic.

**Proof:** Immediately from the definition of \( FT_P \) that \( I' \subseteq I \) implies \( FT_P(I') \subseteq FT_P(I) \).

**Lemma 3.2.2:** \( FT_P \) is well-defined.

**Proof:** \( \forall I, FT_P(I) \) is bounded by 1 and hence convergent.

**Lemma 3.2.3:** \( FT_P \) has a least fixpoint \( \lfloor FT_P \rfloor = FT_P \uparrow \omega \).

**Proof:** Subrahmanian [158] proves the result with the only required property being monotonocity.

**Example 3.2.8:** Let \( P \) be the following program:

\[< p, \mu_p > \leftarrow < q, \mu_q >, < r, \mu_r >\]

\[< q, \mu_q > \leftarrow < t, \mu_t >, < r, \mu_r >\]

\[< t, 1.0 >\]

\[< r, 0.8 >\]
\[ \mu_p = f(\mu_q, \mu_r) = \min(\mu_q, \mu_r) \]

\[ \mu_q = f(\mu_t, \mu_r) = \min(\mu_t, \mu_r) \]

Then, one has the following:

\[ \text{FT}_p \uparrow 0 = \{(p, 0), (q, 0), (t, 0), (r, 0)\} \]

\[ \text{FT}_p \uparrow 1 = \{(p, 0), (q, 0), (t, 1), (r, 0.8)\} \]

\[ \text{FT}_p \uparrow 2 = \{(p, 0), (q, 0.8), (t, 1), (r, 0.8)\} \]

\[ \text{FT}_p \uparrow \omega = \text{FT}_p \uparrow 3 = \{(p, 0.8), (q, 0.8), (t, 1), (r, 0.8)\} \]

**Theorem 3.2.5:** (Interpretations which are models can be characterized through \( \text{FT}_p \))

Let \( P \) be a fuzzy program and \( I \) be an Herbrand interpretation for \( P \). Then \( I \) is a model for \( P \) iff \( \text{FT}_p(I) \subseteq I \).

**Proof:** \( I \) is a model for \( P \) iff for every ground clause \( A \leftarrow A_1, \ldots, A_n \), one has:

\[ \text{model}(I, (A_1, \ldots, A_n)) \Rightarrow \text{model}(I, A) \Rightarrow \text{FT}_p(I) \subseteq I \]

where the notation \( \text{model}(I, A) \) means interpretation \( I \) is a model for \( P \) even when the atom \( A \) is added to it.

**Theorem 3.2.6:** (Fixpoint Characterization of the Least Herbrand Model)

Let \( P \) be a fuzzy definite program. Then:

\[ \text{FM}_P = \text{lfp}(\text{FT}_p) = (\text{FT}_p) \uparrow \omega \]

**Proof:** \( \text{FM}_P \) is the minimal model which is the intersection of any non-empty set of Herbrand models = the greatest lower bound of the lattice of the power set of Herbrand models. One has:

1. \( \text{FM}_P = \text{glb}\{I : I \text{ is an Herbrand model of } P\} \), by definition.

2. \( \text{FM}_P = \text{glb}\{\text{FT}_p(I) \subseteq I\} \), by theorem 3.2.5.

3. \( \text{FM}_P = \text{lfp}(\text{FT}_p) \), by theorem X.3.1.

4. \( \text{FM}_P = \text{FT}_p \uparrow \omega \) by lemma 3.2.3. ■
3.3 Proof Theory

As a start, the definitions of a fuzzy SLD-derivation and a fuzzy SLD-refutation that will be used later on to show the soundness and the completeness of the system will be presented. Then the fuzzy procedural interpretation is discussed as compared to the classical case establishing the fuzzy procedural semantics.

Definition 3.3.1: Let $G \leftarrow A_1, \ldots, A_m, \ldots, A_k$ and $C \leftarrow A \leftarrow B_1, \ldots, B_q$. Then $G'$ is derived from $G$ and $C$ using mgu $\theta$ if the following conditions hold, ($G'$ is the fuzzy resolvent of $G$ and $C$): (see appendix for the definition of most general unifiers).

1. $A_m$ is an atom called the selected atom in $G$.
2. $\theta$ is an mgu of $A_m$ and $A$.
3. $G'$ is the goal $\leftarrow (A_1, \ldots, A_{m-1}, B_1, \ldots, B_q, A_{m+1}, \ldots, A_k)\theta$
4. $\mu_A \geq \mu_{A_m}$ or equivalently $\mu_{A_{\text{rule}}} \geq \mu_{A_{\text{goal}}}$, $\mu_{A_{\text{goal}}}$ must be a constant. In attempting to satisfy a goal with $\mu$ as a variable, the system must respond with the threshold. This is done by translating the goal with a variable into a group of goals with constants to find out the value of the threshold.

It is to be noted that the computation rule $R$ by which an atom is selected must be within condition 4 above. Obviously, the computation rule selects an atom of those satisfying condition 4 as others have failed before being attempted. As such, the fuzzy computation rule is a classical rule with the intelligence of discovering the failure of a sub-goal without attempting it.

Now, the proof procedure will be presented. Many refutation procedures have been used, the one used mostly in Prolog systems which is SLD-resolution was selected, which stands for Linear resolution with Selection function for Definite clauses. The classical SLD-resolution is due to Kowalski [101].
Definition 3.3.2: Let $P$ be a fuzzy program and $G$ a goal. A fuzzy SLD-derivation of $P \cup \{G\}$ consists of (finite or infinite) sequence $G_{\omega} = G, G_1, \ldots$ of fuzzy goals, a sequence $C_1, C_2, \ldots$ of fuzzy program clauses of $P$ and a sequence $\theta_1, \theta_2, \ldots$ of mgu's such that each $G_{i+1}$ is derived from $G_i$ and $C_{i+1}$ using $\theta_{i+1}$.

Definition 3.3.3: A fuzzy SLD-refutation of $P \cup \{G\}$ is a finite fuzzy SLD-derivation of $P \cup \{G\}$ which has the empty clause as the last goal in the derivation. If $G_n = \text{empty clause}$, the refutation is said to have the length $n$. The empty clause is derived from $\leftarrow < A(t_1, \ldots, t_n), \mu_{A_{\text{goal}}} >$ and $< A(t_1, \ldots, t_n), \mu_{A_{\text{fact}}} > \leftarrow$ with $\mu_{A_{\text{fact}}} \geq \mu_{A_{\text{goal}}}$.

Definition 3.3.4: Let $P$ be a fuzzy program and $G$ a fuzzy goal. A fuzzy computed answer $\theta$ for $P \cup \{G\}$ is the substitution obtained by restricting the composition $\theta_1 \ldots \theta_n$ to the variables of $G$, where $\theta_1 \ldots \theta_n$ is the sequence of mgu’s used in the fuzzy SLD-refutation of $P \cup \{G\}$.

Definition 3.3.5: Let $P$ be a fuzzy program, $G$ a fuzzy goal $\leftarrow A_1, \ldots, A_k$ and $\theta$ be an answer for $P \cup \{G\}$. $\theta$ is a fuzzy correct answer for $P \cup \{G\}$ if $\forall (A_1 \land \ldots \land A_k) \theta$ is a logical consequence of $P$.

In the following, the procedural interpretation in the fuzzy system is compared to the classical case. In classical logic programming, the procedural interpretation is given by the following:

1. $\leftarrow A_1, \ldots, A_n (n \geq 0)$, is a goal statement in which atoms are questions that need to be answered.

2. $A \leftarrow B_1, \ldots, B_n$ is a procedure declaration or a method for question answering. To answer $A$ all the $B_i$'s must be answered.

3. $A \leftarrow$ is a fact that answers $\leftarrow A$ without the need for subsequent derivations.

In FLP1, one has the following fuzzy procedural interpretation:
1. \[ \leftarrow < A_1(t_1, \ldots, t_n), \mu_{A_1} >, < A_2(t_1, \ldots, t_n), \mu_{A_2} >, \ldots, < A_n(t_1, \ldots, t_n), \mu_{A_n} > \] is a goal which will not be satisfied until all the sub-goals within it are satisfied.

If the \( \mu_{A_i} \) in the sub-goal was a constant, then for the sub-goal to succeed, the truth value deduced from the program must be equal to or higher that that of the sub-goal. If the \( \mu_{A_i} \) in a fuzzy sub-goal was a variable, so for it to succeed an answer substitution should be found.

2. The clause \[ \leftarrow < A(t_1, \ldots, t_n), \mu_A > \leftarrow < B_1(t_1, \ldots, t_n), \mu_{B_1} >, < B_2(t_1, \ldots, t_n), \mu_{B_2} >, \ldots, < B_n(t_1, \ldots, t_n), \mu_{B_n} > \]

is interpreted as a fuzzy rule for question answering. To answer \( A \), all the \( B_i's \) must have an answer first. And, it is true that:

\[ \mu_A = f(x_i, \mu_{B_i}) \]

3. The fact:

\[ \leftarrow < A(t_1, \ldots, t_n), \mu_A > \]

where \( \mu_A \) must be a constant interpreted as an assertion. The fuzzy goal \[ \leftarrow < A(t_1, \ldots, t_n), \mu_{A_{goal}} > \] succeeds as follows:

(a) \( \mu_{A_{goal}} \) is a variable, it returns the constant \( \mu_{A_{fact}} \).

(b) \( \mu_{A_{goal}} \) is a constant, it returns success only if \( \mu_{A_{fact}} \geq \mu_{A_{goal}} \).

The fuzzy SLD-derivation proceeds as follows:

Given the fuzzy goal:

\[ \leftarrow < A_1(t_1, \ldots, t_n), \mu_{A_1} >, < A_2(t_1, \ldots, t_n), \mu_{A_2} >, \ldots, < A_n(t_1, \ldots, t_n), \mu_{A_n} > \]

corresponding to the \( n \) fuzzy sub-goals:

\[ \leftarrow < A_1(t_1, \ldots, t_n), \mu_{A_1} > \]

\[ \leftarrow < A_2(t_1, \ldots, t_n), \mu_{A_2} > \]
up to the fuzzy subgoal:

\[ \leftarrow \langle A_n(t_1, \ldots, t_n), \mu_{A_n} \rangle \]

Any of the \( A_i \)'s may be selected to be answered first. Let it be \( A_1 \). Suppose that there is a fuzzy clause in the database:

\[ \langle A(t_1, \ldots, t_n), \mu_A \rangle \leftarrow \langle B_1(t_1, \ldots, t_n), \mu_{B_1} \rangle, \ldots, \langle B_n(t_1, \ldots, t_n), \mu_{B_n} \rangle \]

such that \( \theta \) is the most general unifier of \( A \) and \( A_1 \). Then, this new stack of fuzzy sub-goals is obtained:

\[ \leftarrow [\langle B_1(t_1, \ldots, t_n, \mu_{B_1} \rangle, \langle B_2(t_1, \ldots, t_n, \mu_{B_2} \rangle, \ldots, \langle B_n(t_1, \ldots, t_n), \mu_{B_n} \rangle] \theta \]

\[ \leftarrow \langle A_2(t_1, \ldots, t_n), \mu_{A_2} > \theta \]

up to the fuzzy sub-goal:

\[ \leftarrow \langle A_n(t_1, \ldots, t_n), \mu_{A_n} > \theta \]

The derivation is reiterated till the empty clause is reached.

### 3.4 Soundness & Completeness of FLP1

A logic system is said to be *sound* if it produces correct answers. A system is called *complete* if it can compute all the correct answers. Soundness and completeness for fuzzy SLD-resolution amounts for showing that every instance of a *fuzzy correct answer* is an instance of a *fuzzy computed answer*, and every instance of a *fuzzy computed answer* is an instance of a *fuzzy correct answer*. In previous sections, it was shown through a series of theorems that the declarative, minimal-model, fixpoint semantics are equivalent. This section adds to them the fuzzy procedural semantics.

**Theorem 3.4.1:** (Soundness)

Let \( P \) be a fuzzy program and \( G \) a fuzzy goal. Then every fuzzy computed answer for \( P \cup \{G\} \) is a fuzzy correct answer for \( P \cup \{G\} \).
The proof proceeds by induction on the length of the refutation, the following should be shown:

1. The argument is true for the first step in the derivation, i.e. the argument is correct for a goal of the form $\langle A(t_1, \ldots, t_n), \mu_A, \rangle$. This forms the basis of the inductive argument.

2. It must be shown that if the argument is true for derivations with fewer than $k$ steps then it is true for the following step $k$.

To show (1), the necessary and sufficient conditions for a goal to succeed are (in terms of proof theory):

1. There exists a matching clause head.

2. $\mu_{A_{rule}} \geq \mu_{A_{goal}}$.

The necessary and sufficient condition for the fuzzy goal $A$ to succeed (in terms of model theory) is to be in the minimal model which necessarily implies being a logical consequence of the fuzzy program. This condition implies (1) above, i.e. there exists a matching clause head. Furthermore, to be a logical consequence of a fuzzy logic program implies there exists a clause $C$ within the fuzzy logic program such that: $C : \langle A(t_1, \ldots, t_n), \mu_A \rangle \leftarrow \langle A(t_1, \ldots, t_n), \mu_{A_{goal}} \rangle$ and $\mu_A \geq \mu_{A_{goal}}$. This establishes the basis of the inductive argument.

By the induction hypothesis, the argument holds for fewer than $k$ steps. The fuzzy goal must be of the form:

$\langle A_1(t_1, \ldots, t_n), \mu_{A_1}, \rangle, \ldots, \langle A_m(t_1, \ldots, t_n), \mu_{A_m}, \rangle, \ldots, \langle A_n(t_1, \ldots, t_n), \mu_{A_n}, \rangle$

Given that $A_m$ is the selected subgoal of the previous goal stack. By the induction hypothesis, there exists an input clause:

$\langle A(t_1, \ldots, t_n), \mu_A \rangle \leftarrow \langle B_1(t_1, \ldots, t_n), \mu_{B_1}, \rangle, \ldots, \langle B_q(t_1, \ldots, t_n), \mu_{B_q}, \rangle$
such that:

\[
\begin{align*}
&< A_1(t_1, \ldots, t_n), \mu_A > \land \ldots \\
&< A_{m-1}(t_1, \ldots, t_n), \mu_{A_{m-1}} > \land \\
&< B_1(t_1, \ldots, t_n), \mu_{B_1} > \land \ldots \\
&< B_q(t_1, \ldots, t_n), \mu_{B_q} > \land \\
&< A_{m+1}(t_1, \ldots, t_n), \mu_{A_{m+1}} > \land \\
&< A_n(t_1, \ldots, t_n), \mu_{A_n} > \theta_1 \ldots \theta_k \\
\end{align*}
\]

is a logical consequence of the program, where \( \theta_1 \ldots \theta_k \) is the sequence of mgu's used in the refutation. The induction hypothesis necessitates that the sequence \( \theta_1 \ldots \theta_k \) must include substitution for the \( \mu \) values which must be within the minimal model for derivations with fewer than \( k \) steps. In other words the sequence \( \theta_1 \ldots \theta_k \) must involve substitutions which make the selected atom with \( (\theta_1 \ldots \theta_k)A_m \) is a logical consequence of the fuzzy logic program where \( A_m \) unifies with \( A \) the clause head under the condition \( \mu_A \geq \mu_{A_m} \) or equivalently \( \mu_{A_{\text{rule}}} \geq \mu_{A_{\text{goal}}} \) is ensured in every step in the derivation. This establishes the result that the new subgoals are all within the range of logical consequence. This establishes the induction step of the proof. ■

**Definition 3.4.1:** Let \( P \) be a fuzzy program. The **success set** of \( P \) is the set of all \( A \in FB_P \) such that \( P \cup \{ \leftarrow A \} \) has a fuzzy SLD-refutation.

**Theorem 3.4.2:** The success set of a fuzzy program is equal to its least Herbrand model.

**Proof:** First, it will be shown that the success set of a fuzzy program is contained in its least Herbrand model. If \( A \in FB_P \) and has a refutation, then by theorem 3.4.1, \( A \) is a logical consequence of \( P \). By theorem 3.2.4, \( A \) is in the least Herbrand model of \( P \).

Second, to show that \( A \subseteq S \) (the success set). Let \( A \in FM_P \), then
1. A is a logical consequence, (equivalence between declarative semantics and minimal model semantics).

2. A has a refutation, (induction argument on the length of the refutation using $FTP$).

3. $A \in S$, (definition of $S$).

**Theorem 3.4.3: (Completeness)**

Let $P$ be a fuzzy program and $G$ a fuzzy goal. For every fuzzy correct answer $\theta$ for $P \cup \{G\}$, there exists a refutation for $P \cup \{G\}$.

**Proof:** If $G$ is the fuzzy goal $\leftarrow A_1, \ldots, A_n$ that has a fuzzy correct answer, then:

1. Let $\theta$ be any correct answer for $P \cup \{G\}$, then:

2. $\forall((A_1 \land \ldots \land A_k)\theta)$ is a logical consequence. (by definition of a correct answer), then:

3. $\forall((A_1 \land \ldots \land A_k)\theta)$ is in the minimal model, then:

4. $\forall((A_1 \land \ldots \land A_k)\theta)$ is in the success set, then:

5. $P \cup \{G\}$ has a refutation, (by theorem 3.4.2).

**Example 3.4.1:** Several fuzzy SLD-refutations are discussed for the following fuzzy program and the goal: $\leftarrow < p(x, b), 0.5 >$, then the following fuzzy SLD-refutations are obtained:

1. $< p(x, z), \mu_{p_1} > \leftarrow < q(x, y), \mu_q >, < p(y, z), \mu_{p_2} >$

2. $< p(x, x), 0.6 > \leftarrow$

3. $< q(a, b), 0.3 > \leftarrow$
When the goal unifies with 2, a success is reached. When unified with the first clause, two goals are derived \(< q(x, y), \mu_q >, < p(y, b), \mu_p >\). Unifying the two goals with 3 produces \(< q(a, b), 0.3 >, < p(b, b), \mu_p >\). The first of these is resolved and the second succeeds with 2. If the goal \(< p(b, b), \mu_p >\) is considered with clause 1, it fails.

3.5 Negation in Fuzzy Logic Programming

A transformation that transforms any negative goal to a definite goal and a negative fuzzy program to a definite fuzzy program is introduced. It will be clear that after this transformation that Fuzzy SLD-resolution can be used normally to find an answer for this goal.

For the goal \(< A(t_1, \ldots, t_n), \mu_{A\text{goal}} >\) to succeed, there are two cases to consider:

1. \(\mu_{A\text{goal}}\) is a constant, where there must be a matching fact with \(\mu_{A\text{fact}} \geq \mu_{A\text{goal}}\).

2. \(\mu_{A\text{goal}}\) is a variable, in this case the goal succeeds if a value is returned for \(\mu_{A\text{goal}}\) which is the threshold.

Consider the goal \(< A\). If \(\mu_{A\text{goal}}\) is a constant, the system answers “Yes” or “No”. If \(< A\) is “Yes”, then \(< \neg A\) is “No” and vice versa. If \(\mu_{A\text{goal}}\) is a variable the system answers with the threshold. In both cases, if \(A\) is established, then \(\neg A\) follows.

Given the goal \(< \neg < A(t_1, \ldots, t_n), \mu_{\neg A} >, \neg < A(t_1, \ldots, t_n), \mu_A >\) is tried. If \(< A(t_1, \ldots, t_n), \mu_A >\) succeeds with a value for \(\mu_A\), then \(\neg A\) succeeds with \(\mu_{\neg A} = 1 - \mu_A\). If \(A\) fails, then negation as finite failure must be considered, then \(A\) must be a ground atom of \(FB_\mu\) and if \(< A\) fails finitely for \(\mu_A = 0.2\), then \(< \neg A\) succeeds with \(\mu_{\neg A} = 0.8\). The special case when \(\mu_A = 0.5\) where \(\mu_A = \mu_{\neg A}\) is noted. This can be called graded negation by failure.
The result is, in addition to the extra-expressiveness of fuzzy logic programs relative to non-fuzzy classical logic programs, they offer a very attractive solution for the problem of negation in fuzzy logic programming. Some authors, e.g. Minker [117] tried the use of 3-valued logic, one can see that fuzzy logic offers a direct way for handling negation.

3.6 Application to Fuzzy Constraint Satisfaction Problem

Possibility theory had been used to generalize Constraint Satisfaction Problems. Soft constraints can be represented. There also can be preference among feasible solutions and priority ordering among constraints, Dubois et al. [48]. In the following we shall extend the example of Dubois et al. [48] to handle even more complex constraints (Conditional ones - with IF-then style) and show that they can be represented and solved in the framework of the fuzzy logic programming system presented.

Example 3.6.1:

A course must involve 7 sessions, namely $x$ lectures, $y$ exercise sessions and $z$ training sessions ($C_1$). There must be about 2 training sessions ($C_2$), i.e. ideally 2, possibly 1 or 3. Dr. B, who gives the exercise part of the course, wants to manage 3 or 4 sessions ($C_3$). Prof A, who gives the lectures, wants to give about 4 lectures ($C_4$), i.e., ideally 4 lectures, possibly 3 or 5). The request of Dr. B is less important than the one of Prof. A and is itself less important than the imperative constraints $C_1$ and $C_2$. In this example, flexibility is modelled using a five level scale $L = (\alpha_0 = 0 < \alpha_1 = c(\alpha_3) < \alpha_2 = c(\alpha_2) < \alpha_3 = c(\alpha_1) < \alpha_4 = 1)$, where $c$ is the order-reversing operation. The priorities of $C_3$ and $C_4$ are respectively $\alpha_2$ and $\alpha_3$ ($\alpha_2 < \alpha_3$). The domain of variables $x, y$ and $z$ is the set $\{0, 1, 2, 3, 4, 5, 6, 7\}$. The following model can be used:

$C_1$: Classical hard constraint

$$\mu_{R_1}(x, y, z) = 1 \text{ if } x + y + z = 7$$
$$\mu_{R_1}(x, y, z) = 0 \text{ otherwise}$$
$C_2$: Soft constraint
\[ \mu_{R_2}(z) = 1 \text{ if } z = 2 \]
\[ \mu_{R_2}(x) = \alpha_3 \text{ if } z = 1 \text{ or } z = 3 \]
\[ \mu_{R_2}(z) = 0 \text{ otherwise} \]
This can be written in FLP1 as:
\[ < p(2), 1 > \]
\[ < p(1), \alpha_3 > \]
\[ < p(3), \alpha_3 > \]
\[ < p(x), 0 > \text{ if } x \neq 1, 2, 3 >, \text{ where } \neq \text{ is a classical predicate written infix.} \]

It is obvious that all of the three above are facts, and in FLP1, we can represent more complex constraints containing rules. So, constraint 2 can become "If the sessions are in the morning, they should be mostly 3 and if in the evening they should be mostly 2".

\[ < p(3), 1 > \text{ if } \text{Timing(Morn)}, 0.8 >, \text{ mostly being represented by 0.8.} \]
\[ < p(2), 1 > \text{ if } \text{Timing(Even)}, 0.8 > \]

$C_3$: Prioritized constraint $Pr(C_3) = \alpha_2$
\[ \mu_{R_3}(y) = 1 \text{ if } y = 3 \text{ or } y = 4 \]
\[ \mu_{R_3}(y) = c(\alpha_2) = \alpha_2 \text{ otherwise.} \]

$C_4$: Soft and prioritized constraint $Pr(C_4) = \alpha_3$
\[ \mu_{R_4}(x) = 1 \text{ if } x = 4 \]
\[ \mu_{R_4}(x) = \max(\alpha_3, c(\alpha_3)) = \alpha_3 \text{ if } x = 3 \text{ or } x = 5 \]
\[ \mu_{R_4} = c(\alpha_3) = \alpha_1 \text{ otherwise} \]

**Conclusion**

An original treatment of fuzzy logic programming based on cuts has been presented. It was based on the discovery of the primary properties that fuzzy logic should offer in addition to standard logic. The syntax together with the declarative semantics, minimal model semantics, fixpoint semantics and procedural semantics for fuzzy logic programs have been presented. The equivalence be-
tween these semantics have been shown. The final result is the soundness and completeness of the fuzzy logic programming system presented. An original attractive treatment for negation in fuzzy logic programming was shown as well. The significance of this direction of research is that intuitive fuzzy expert systems can be founded on a sound theoretical background. The designer of such a system has a proof of correctness of each result produced by the system as well as another proof that his system produces all correct answers. The final result is the soundness and completeness of the system was reached due to the following theorems:

1. The existence of an Herbrand model for a set of fuzzy clauses having a model.

2. Model Intersection Property.

3. The relationship between the minimal model and declarative semantics.


5. The Soundness of the Fuzzy Logic Programming System - any answer produced by the system is correct.


In comparing FLP1 with that of Van Emden's [141]. It is to be noted here that in [141] a given set of rules are specified by a function $val$ taking as arguments a variable-free atomic formula $A$, and an interpretation $I$ and having as a result $val(A, I)$, the value of the membership function for $I$ at the argument $A$. The careful reader would soon notice that such a definition of function $val$ has a circular reference within itself. The reason is that $I$ itself is a structure which has as a part of it such a function for an atom $A$ (cf. Hajek [85] definition 1.4.1 pp.
15). Only if we consider instead of \( I \) the subset of the Herbrand base associated with it, the definition will be correct and it can be considered as a simplification of notation.

In the next chapter, additional extensions are added to this system. One extension is a doubting factor \( f \) attached to a rule. Another extension is being able to infer under a new notion of pseudo-complementarity.
Chapter 4

Fuzzy Logic Programming
Founded on a New Notion of Pseudo-complementarity: FLP2

In the previous chapter a fuzzy logic programming system was presented and it was proven that its model theory, proof theory and fixpoint theory are equivalent. The system of the previous chapter is extended by a doubting factor \( f \) as well as a new notion of pseudo-complementarity. A system meta-interpreter is presented to emphasize the concepts.

4.1 Syntax & Semantics

As a start, the syntax of the previously given system is extended with a factor doubting the rule to a degree; rule-associated fuzziness. The inference system of the previous chapter was capable of inference with fuzzy predicates. In this chapter, the system is extended to cope with fuzzy rules that are true to a certain degree. In other words, there would be a factor affecting the implication from the premises to the conclusion. The notion of pseudo-complementarity is introduced to enhance the fuzzy logic programming system inference.

**Definition 4.1.1:** A formula of FLP1 is a formula of FLP2 and a clause of FLP1 is a clause of FLP2. In addition, introducing the *rule doubting factor* \( f \in [0, 1] \), the syntax of a fuzzy clause will be slightly changed to:

\[
< A, \mu_A > \leftarrow (f) \leftarrow < A_1, \mu_{A_1} >, \ldots, < A_n, \mu_{A_n} >
\]
A similar syntax of the above in Li et al. [104] but with neither proofs of soundness and completeness like that of the previous chapter nor a notion of pseudo-complementarity as the one introduced here. The doubting factor has a multiplicative effect as seen in the following examples.

In the new system, the concept of logical consequence should be extended a bit further than the previous chapter to be more general.

**Definition 4.1.2:** (Satisfaction) Let $F$ be a formula in FLP2 and $I$ an Herbrand interpretation for this formula, then $F = < p(t_1, \ldots, t_n), \mu >$ (where $\mu$ is a constant) is true under $I$ if $I$ interprets $p$ to the fuzzy relation $r$ and $r(t_1, \ldots, t_n) \geq \mu$ or $r(t_1, \ldots, t_n) \approx \mu$.

**Definition 4.1.3:** Let $n$ be the number of divisions of $[0, 1]$ (a finite natural number) then $m$ the *tolerance level in one step inference* $m = |\mu - \mu_m|$ where $r(t_1, \ldots, t_n) \geq \mu$ in FLP1. An upper bound must be set on how many times the application of that tolerance level could be applied.

**Example 4.1.1:** Let $F$ be the formula: $F = < p(m, n), 0.3 >$ and $r : D \times D \rightarrow [0, 1]$ where $D = \{a, b, c\}$ and:

- $r(a, a) = 0.1$
- $r(a, b) = 0.29$
- $r(a, c) = 0.3$
- $r(b, a) = 0.4$
- $r(b, b) = 0.3$
- $r(b, c) = 0.5$
- $r(c, a) = 0.6$
- $r(c, b) = 0.7$
- $r(c, c) = 0.8$

Let $I$ be an interpretation that maps $p$ onto $r$ and $m$ onto $a$ and $n$ onto $b$, then $F$ is satisfied under $I$ in FLP2 but not in FLP1 with the tolerance level $0.3 - 0.29 = 0.01$. 

52
**Definition 4.1.4:** An Herbrand interpretation which satisfies the formula is an **Herbrand model** of the formula. From above, an Herbrand model for a formula in FLP2 may not be a model in FLP1. It will be noted that Herbrand models of FLP2 will be called **proximate models.** One has,

\[
\text{Models}_{FLP1} \subseteq \text{Proximate Models}_{FLP2}
\]

If \( A \in \text{Proximate Models}_{FLP2} \), and \( A \notin \text{Models}_{FLP1} \), then, \( A \) is called \( A \)-proximate. Example 4.1.1 illustrates this case.

**Definition 4.1.5** Let \( P \) be a fuzzy program in FLP2 and \( A \) an atom. Then \( A \) is a **proximate logical consequence** of \( P \) if for every interpretation \( I \), if \( I \) is a model for \( P \), it is a proximate model for \( A \).

**Example 4.1.2:** Let's have an extension of example 3.2.4 in FLP2, where \( g_{tall} : [0,250] \text{cm} \rightarrow [0,1] \) and \( g_{tall}(190) = 0.8, g_{tall}(180) = 0.7, g_{tall}(170) = 0.6, \) and \( g_{tall}(160) = 0.5. \) Now, the clause in FLP2:

\[
< \text{GoodBasketBallPlayer}(x), g_{tall}(y) > \leftarrow (0.9) < \text{height}(x,y), 1 >
\]

\[
< \text{height}(John, 190), 1 >
\]

\[
< \text{height}(George, 180), 1 >
\]

\[
< \text{height}(Angela, 170), 1 >
\]

\[
< \text{height}(Angelina, 160), 1 >
\]

Then, with the goal:

\[
? < \text{GoodBasketBallPlayer}(x), z >
\]

One has: \(< \text{GoodBasketBallPlayer}(John), 0.8 \times 0.9 >\)

\(< \text{GoodBasketBallPlayer}(George), 0.7 \times 0.9 >\)

\(< \text{GoodBasketBallPlayer}(Angela), 0.6 \times 0.9 >\)

\(< \text{GoodBasketBallPlayer}(Angelina), 0.5 \times 0.9 >\)

**4.2 Model Semantics & Fixpoint Semantics**

In the fuzzy logic programming system of the previous chapter, the minimal model played a major role in the theory as it contains atoms that are the logical
consequence of the fuzzy program in the sense of the previous chapter. The characterization of this minimal model with fixpoint operators was important in the theory. This theory will be extended to cope with the new Herbrand models according to the new definition of satisfaction.

**Theorem 4.2.2: Principle of Sufficiency of Herbrand Models**

Let $S$ be a set of clauses and suppose $S$ has a model. Then $S$ has an Herbrand model. Further, $S$ has an Herbrand model in FLP2.

**Proof:** It is straightforward to establish the theorem by finding a value to which is approximately equal to $\mu$. If this is not possible, obviously any model is proximate to itself. ■

**Theorem 4.2.3:** Let $P$ be a program and $F$ a closed formula, then $F$ is proximate logical consequence of $P$ iff $P \cup \{\neg F\}$ is unsatisfiable in FLP2.

**Proof:** $F$ is a proximate logical consequence of $P$. Then for every interpretation $I$ of $P$, if $I$ is a model of $P$, then it is a proximate model for $F$. Then, $I$ is not a model for $\neg F$. Then $P \cup \{\neg F\}$ is unsatisfiable in FLP2. Now, let $P \cup \{\neg F\}$ is unsatisfiable in FLP2. Then, for every interpretation $I$ of $P$, if $I$ is a model of $P$, then it is not a proximate model for $P \cup \{\neg F\}$. Then, $I$ is a proximate model for $F$. Then, $F$ is a proximate logical consequence of $P$.

**Theorem 4.2.4:** The Equivalence between the Declarative $\&$ Minimal Model Semantics: Let $P$ be a fuzzy program. Then if $M_\alpha$ is the minimal proximate model, it is true that:

$$M_\alpha = \{A \in FB_P : A \text{ is a proximate logical consequence of } P\}$$

**Proof:** $A$ is a proximate logical consequence of $P$:

- iff $P \cup \{\neg A\}$ is unsatisfiable in FLP2
- iff $\neg A$ is false in all proximate models
- iff $A$ is true in all proximate models.
- iff $A \in M_\alpha$. ■

The special case of $\mu = 0.5$ where a model can be proximate to $F$ and $\neg F$ at the
Definition 4.2.1: Here, we need to redefine the $FT_P$ mapping in the $\alpha$ sense as follows: $FT_P(I) = \{A \text{ or } A - \text{proximate} \in FB_p, \mu_A = f * \sup\{f(x_i, \mu_{B_i}), A \leftarrow B_1, \ldots, B_n \text{ is a ground instance of a clause in } P \text{ and } \{B_1, \ldots, B_n\} \subseteq I\}, f$ is the doubting factor. The tolerance level is applied only once not to allow transtivity.

Theorem 4.2.5: (Pre-fixpoint Theorem) Let $I$ be an Herbrand interpretation for a fuzzy program in FLP2. Then $I$ is a proximate model iff $FT_P(I) \subseteq I$.

Proof: Let $A \in I$ be any atom in $I$. $I$ is a proximate model for $P$. By definition of $FT_P$, there exists a ground clause such that $FT_P(I) = A$. Then, $FT_P(I) \subseteq I$.

Now, let $FT_P(I) \subseteq I$. Let $A \in FT_P(I)$, then $A \in I$. Then by the definition of $FT_P$, there exists a ground instance of a clause in $I$ such that $A$ is proximate logical consequence of this clause. Then, $I$ is a proximate model for $P$. ■

Theorem 4.2.6: Fixpoint Characterization of the Minimal proximate model.

$$M_\alpha = \text{lfp}(FT_P) = FT_P \uparrow \omega$$

Proof:

$$M_\alpha = \text{glb}\{I : I \text{ is an Herbrand proximate model}\}. \text{(definition of } M_\alpha\)$$  
$$M_\alpha = \text{glb}\{FT_P(I) \subseteq I\} \text{ (theorem 4.2.5)}$$  
$$M_\alpha = \text{lfp}(FT_P) \text{ theorem X.3.1}$$  
$$M_\alpha = FT_P \uparrow \omega \text{ (lemma 3.2.3). ■}$$

By now, the equivalence of the declarative, minimal model and fixpoint semantics has been established. Then, proof theory is introduced.

4.3 Proof Theory

Definition 4.3.1 Pseudo-complementarity

Two atoms (one goal and the other is a clause head or a fact) $< p(t_1, \ldots, t_n), \mu_p >$, $< q(t_1, \ldots, t_n), \mu_q >$ are said to be \textit{pseudo-complementary} when $\mu_p$ and $\mu_q$ are within the tolerance level of Def. 4.1.3.
Definition 4.3.2: Let \( G \) be \( \leftarrow A_1, \ldots, A_m, \ldots, A_k \) and \( C \) be \( A \leftarrow B_1, \ldots, B_q \). Then \( G' \) is derived from \( G \) and \( C \) using mgu \( \theta \) if the following conditions hold, \( (G' \) is the \( \alpha \)-proximate or simply the proximate resolvent of \( G \) and \( C )\):

1. \( A_m \) is an atom called the selected atom in \( G \).
2. \( \theta \) is an mgu of \( A_m \) and \( A \).
3. \( G' \) is the fuzzy goal \( \leftarrow (A_1, \ldots, A_{m-1}, B_1, \ldots, B_q, A_{m+1}, \ldots, A_k)\theta \)
4. \( \mu_A \geq \mu_{A_m} \), i.e., \( \mu_{A_{rule}} \geq \mu_{A_{goal}} \), or otherwise \( A \) and \( A_m \) should be pseudo-complementary.

The considerations of \( \mu \) being a variable and a constant and the independence of the computation rule is as in the previous chapter.

Definition 4.3.3: Let \( P \) be a fuzzy program and \( G \) a fuzzy goal. A proximate SLD-derivation of \( P \cup \{G\} \) is a fuzzy SLD-derivation as in the previous chapter whose first derived goal \( G_1 \) can be a proximate resolvent of \( G \). Pseudo-complementarity in a proximate SLD-derivation can be considered only with the initial resolvent.

Definition 4.3.4: A proximate SLD-refutation of \( P \cup \{G\} \) is a finite proximate SLD-derivation of \( P \cup \{G\} \) which has the empty clause as the last goal in the derivation. The empty clause is derived from \( \leftarrow \langle A(t_1, \ldots, t_n), \mu_{A_{goal}} \rangle \) and \( \langle A(t_1, \ldots, t_n), \mu_{A_{fact}} \rangle \leftarrow \) with \( \mu_{A_{fact}} \geq \mu_{A_{goal}} \) or \( \mu_{A_{fact}} \approx \mu_{A_{goal}} \). This last condition was not possible in the system FLP1.

Definition 4.3.5: Let \( P \) be a fuzzy program and \( G \) a fuzzy goal. A proximate computed answer \( \theta \) for \( P \cup \{G\} \) is the substitution obtained by restricting the composition \( \theta_1 \ldots \theta_n \) to the variables of \( G \), where \( \theta_1 \ldots \theta_n \) is the sequence of mgu's used in the proximate SLD-refutation of \( P \cup \{G\} \).
Definition 4.3.6: Let $P$ be a fuzzy program, $G$ a fuzzy goal $\leftarrow A_1, \ldots, A_k$ and $\theta$ be an answer for $P \cup \{G\}$. $\theta$ is said to be a proximate correct answer for $P \cup \{G\}$ if $\forall (A_1 \land \ldots \land A_k) \theta$ is a proximate logical consequence of $P$.

4.4 Soundness & Completeness of FLP2

Theorem 4.4.1: (Soundness)

Let $P$ be a fuzzy program and $G$ a fuzzy goal. Then every proximate computed answer for $P \cup \{G\}$ is a proximate correct answer for $P \cup \{G\}$.

Proof: The proof proceeds by induction on the length of the proximate refutation. The following basis and induction steps should be shown:

1. The argument is true for the first step in the derivation, i.e. the argument is correct for a fuzzy goal of the form $\leftarrow A(t_1, \ldots, t_n), \mu_A >$. If the resolvent was fuzzy then theorem 3.4.1 the previous chapter establishes the result. If the resolvent is proximate, then there exists a fuzzy clause such that $\mu_A \approx \mu_{A_{\text{goal}}}$ this guarantees that the goal is in the minimal model which establishes the result.

2. The induction step is established directly from theorem 3.4.1 in the previous chapter as the proximate SLD-derivation differs from the fuzzy SLD-derivation only in the first derived goal (definition 4.3.3).

Definition 4.4.1: Let $P$ be a fuzzy program. The proximate success set of $P$ is the set of all $A \in FB_P$ such that $P \cup \{\leftarrow A\}$ has a proximate SLD-refutation.

Theorem 4.4.2: The proximate success set of a fuzzy program is equal to its minimal $\alpha$-model.

Proof: First, if $A$ is the proximate success set so it has a proximate refutation and by theorem 4.4.1 $A$ is a proximate logical consequence. By theorem 4.2.4 it is in the minimal $\alpha$-model. Second, it must be shown that the minimal $\alpha$-model is contained in the success set of $P$. Suppose $A$ is the minimal $\alpha$-model
of $P$. By theorem 4.2.4 $A \in FT_p \uparrow n$ implies that $P \cup \{ \leftarrow A \}$ has a proximate SLD-refutation and hence $A$ is the proximate success set. The same induction argument of theorem 3.4.2 in the previous chapter holds and the result follows.

**Theorem 4.4.3:** (Completeness)

Let $P$ be a fuzzy program and $G$ a fuzzy goal. For every proximate correct answer $\theta$ for $P \cup \{ G \}$, there exists a refutation for $P \cup \{ G \}$.

**Proof:** If $G$ is the fuzzy goal $\leftarrow A_1, \ldots, A_n$ that has a proximate correct answer, then:

1. Let $\theta$ be any proximate correct answer for $P \cup \{ G \}$, then:

2. $\forall((A_1 \land \ldots \land A_n)\theta)$ is a proximate logical consequence. (by definition of a correct answer), then:

3. $\forall((A_1 \land \ldots \land A_n)\theta)$ is in the minimal $\alpha$-model, then:

4. $\forall((A_1 \land \ldots \land A_n)\theta)$ is in the proximate success set, then:

5. $P \cup \{ G \}$ has a proximate refutation. (by theorem 4.4.2)

### 4.5 A System Meta-interpreter

In this section, a meta-interpreter is presented to the fuzzy logic programming system discussed. The meta-interpreter is implemented in IC-Prolog. Given the rule:

$\langle p_1(x), \mu_{p_1} \rangle \leftarrow \langle q(x), \mu_{q_1} \rangle$.

It can be read declaratively or procedurally:

1. The declarative reading states that: for a certain value of the variable $x$, $p_1$ should be true to a level $\mu_{p_1} \geq \mu_{q_1}$.

2. The procedural reading states that: for a fuzzy goal $\leftarrow \langle p_1(m), 0.3 \rangle$ to succeed, the fuzzy subgoal $\leftarrow \langle q(m), 0.3 \rangle$ must succeed. Further, for the fuzzy goal $\leftarrow \langle p(m), 0.4 \rangle$, the fuzzy sub-goal $q(m, 0.4)$ must succeed.
So, as far as execution is concerned, both values of $\mu$ are instantiated in the fuzzy rule with the same constant level in the goal and then attempt succeeding the fuzzy sub-goal. Then, using the meta-interpreter, the rule is rewritten as follows:

$$R1 : < p_1(x), \mu_{p_1} > \leftarrow < q_1(x), \mu_{q_1} >$$

as

$$R1' : p1(X, Mp1) : -q(X, Mp1).$$

Now, consider the fuzzy goal $\leftarrow < p_1(m), V >$, where $V$ is a variable. Now, the system is queried to what maximum level this fuzzy goal can be satisfied. This is done via the meta-interpreter predicate $solve(A)$ which becomes $\leftarrow solve(p1(m, V))$. The system predicates functor and arg are used.

When rewriting the fuzzy logic programs in IC-Prolog or standard Prolog, care should be taken as the semantics associated with fuzzy logic programs are different than that of standard Prolog. For instance, given the fact $< q(m), 0.3 > \leftarrow$, in fuzzy logic programming, it is considered as a fuzzy fact. $q$ is said to be true to a level $\mu$ where $0 < \mu \leq 0.3$. In standard Prolog, the goal $\leftarrow q(m, 0.25)$ would return the answer “No”. So, to write a fuzzy fact in Prolog, it should be written as:

$$q(m, Mq) : -(Mq \leq 0.3), (Mq > 0)$$

During execution within the Prolog model, the answers conform to the given semantics.

Now, the extended rules are extended with a factor $f \in [0, 1]$ doubting the rule:

$$< p_1(x), \mu_{p_1} > \leftarrow (0.9) - < q(x), \mu_q > .$$

For the goal $\leftarrow < p_1(x), 0.3 >$ to succeed, the fuzzy goal $\leftarrow q(x, \mu_q)$ must succeed at least with the value $0.3/0.9$. To do this in standard Prolog, the fuzzy fact and the fuzzy rule are rewritten as follows:
\[ p_1(X, Mp1) : \neg q(X, Mp1). \]

\[ q(m, Mq) : -(Mq \leq 0.3/0.9), (Mq > 0). \]

which will lead to the intended meaning.

Now, if the predicate \( q \) happens to be in the body of two fuzzy rules with different \( f \) factors, a different rewriting of the facts is required. For instance, one obtains the following two rules and two facts:

1. \( R1 : < p_1(x), \mu_{p_1} > \leftarrow (0.9) - < q(x), \mu_q > \)
2. \( R2 : < p_2(x,y), \mu_{p_2} > \leftarrow (0.7) - < q(x), \mu_q >, < s(Y), \mu_s > . \)
3. \( Fact1 : < q(m), 0.3 > \leftarrow \)
4. \( Fact2 : < s(n), 0.4 > \leftarrow \)

If this fuzzy logic program is rewritten in Prolog, one gets:

1. \( R1' : p_1(X, Mp1) : \neg q(X, Mp1). \)
2. \( R2' : p_2(X, Y, Mp2) : \neg q(X, Mp2), s(Y, Mp2). \)

and the two fuzzy facts:

1. \( Fact1' : q(m, Mq) : -(Mq \leq 0.3/0.9), (Mq > 0) \)
2. \( Fact2' : s(n, Ms) : -(Ms \leq 0.4/0.7), (Ms > 0). \)

If a fuzzy goal matches with \( R1' \), then \( Fact1' \), this would be fine. But if a fuzzy goal matches with \( R2' \), the \( q \) fuzzy subgoal must have \( f = 0.7 \) not 0.9. Thus, given the same predicate occurring in the body of two fuzzy rules with different \( f \) factors, it should be renamed when rewriting. As a result, the predicate \( q \) is renamed in \( R2 \) to \( h \), and one obtains two fuzzy facts \( Fact1' \) and \( Fact2'' \) corresponding to Fact 1 in the original program:

1. \( R1' : p_1(X, Mp1) : \neg q(X, Mp1). \)
2. \( R2' : p_2(X, Y, Mp2) : \neg h(X, Mp2), s(Y, Mp2). \)
3. \( Fact1' : q(m, Mq) : -(Mq \leq 0.3/0.9), (Mq > 0). \)
4. \( Fact1'' : h(m, Mh) : -(Mh \leq 0.3/0.7), (Mh > 0). \)
5. \( Fact2' : s(n, Ms) : -(Ms \leq 0.4/0.7), (Ms > 0). \)
In the following, a code listing for the meta-interpreter is presented and a rewritten fuzzy logic program in IC-Prolog that was tested with the results expected from the semantics for fuzzy logic programming. The [0, 1] interval has been assumed as [0,100], i.e. one hundred increments.

\[
\begin{align*}
p1(X, Mp1) & : - q(X, Mp1). \\
p2(X, Y, Mp2) & : - q(X, Mp2), s(Y, Mp2). \\
p3(X, Y, Z, Mp3) & : - s(Y, Mp3), t(Z, Mp3), q(X, Mp3). \\
q(m, Mp2) & : (Mp2 \leq 4/9), (Mp2 > 0). \\
s(n, Mpr) & : (Mpr \leq 3/7), (Mpr > 0). \\
t(l, Mpr) & : (Mpr \leq 1/2), (Mpr > 0). \\
solve(A, 0). \\
solve(A, X) & : X > 0, functor(A, F, N), F = A, arg(N, A, H), arg(N, A, X), A!, solve(A, Z). \\
solve(A) & : solve(A, 100). \\
nt(A) & : solve(A), functor(A, F, N), arg(N, A, H), Y is 100-H, write(Y).
\end{align*}
\]

The \textit{solve} predicate finds the threshold if the goal contained variables. For a negated goal not containing variables the built-in \textit{not} predicate would produce the right answer. If the negated goal contained variables, the \textit{nt} predicate above gives the threshold.

### 4.6 Computational Complexity

The following theorem establishes the complexity of the fuzzy logic programming systems presented FLP1 and FLP2.

**Theorem 4.6.1** The computational complexity of FLP1 and FLP2 is precisely as that of classical logic.

**Proof:** We prove the result over the structure of the Herbrand universe and Herbrand base. Let \(< p, \mu >\) be a formula of FLP1 or FLP2, and \(p\) the classical syntactical counterpart of \(p\), i.e. by removing the cut attached to it. Then a
bijection can be easily established from the classical Herbrand base to the fuzzy counterpart.

The following example clarifies the above theorem.

**Example 4.6.1** Let $P$ be the classical logic program (Nilsson et al. [121]) extended as:

\[
\begin{align*}
p(f(X)) & \leftarrow q(X, g(X)) \\
q(a, g(b)) & \\
q(b, g(b)).
\end{align*}
\]

Then the Herbrand universe for this program is given by:

\[
\begin{align*}
U_0 &= \{a, b\} \\
U_{i+1} &= \{f(x) \mid x \in U_i\} \cup \{g(x) \mid x \in U_i\}
\end{align*}
\]

then,\n
\[
U_P = \bigcup_{i=0}^{\infty} U_i
\]

or extending it as:

\[
U_P = \{a, b, f(a), f(b), f(f(a)), f(f(b)), \ldots\}
\]

and\n
\[
B_P = \{p(x) \mid x \in U_P\} \cup \{q(x, y) \mid x, y \in U_P\}
\]

or extending it as:

\[
B_P = \{p(a), p(b), p(f(a)), p(f(b)), p(f(f(a))), \ldots,\}
\]

\[
q(a), q(b), q(f(a)), q(f(b)), q(f(f(a))), \ldots\}
\]

Now, take the above program to be in FLP1 or FLP2, it will look like:

\[
\begin{align*}
&< p(f(X)), \mu_p > \leftarrow < q(X, g(X)), \mu_q > \\
&< q(a, g(b)), \mu_1 > \\
&< q(b, g(b)), \mu_2 >
\end{align*}
\]

then, by the definition of $FB_P$ (removing the annotation part):

\[
FB_P = \{p(a), p(b), p(f(a)), p(f(b)), p(f(f(a))), \ldots,\}
\]

62
\[ q(a), q(b), q(f(a)), q(f(b)), q(f(f(a))), \ldots \]

One has \( B_P = FB_P \), thus having the same complexity structure. Further, the cardinality of \( B_P \) is \( \omega n \) where \( n \) is the number of distinct function symbols in addition to the number of distinct constants \( n_c \).

\[
Car(U_P) = Car(B_P) = n_c + n_f, n_f = \omega n,
\]

This result can be compared to that of the complexity of Possibilistic Logic which is more complex than that of classical logic. See Dubois et al. [45] for the proof that the satisfiability problem is a logarithmic number of calls to that of classical propositional logic. In the following chapter, Fuzzy Disjunctive Logic Programming is presented.
Chapter 5

Fuzzy Disjunctive Logic Programming: FLP3

Fuzzy disjunctive logic programs syntax and their declarative semantics are defined. The result is that fuzzy logic can embrace the additional expressiveness of disjunctive information. For the new system, the model-state and fixpoint semantics are worked out. The procedural semantics founded on the new Fuzzy SLI Resolution is developed. The equivalence of these approaches of semantics is shown together with the soundness and completeness - all is generalization of Lobo et al. [112]. This chapter together with the two previous ones provide a logical reconstruction and a theoretical background for Fuzzy Expert Systems.

As a start, the basic definitions are introduced. The purpose is to show that the development of fuzzy logic programming could be extended to fuzzy disjunctive logic programming. The result is an inference system which is capable not only of handling imprecise data but also capable of handling indefinite rules and facts.

Overview of the Chapter

In the following subsection, the syntax of fuzzy disjunctive logic programs is presented. In section 5.2, the model-state semantics and fixpoint semantics where a fixpoint operator is defined for fuzzy disjunctive logic programs. The equivalence of these semantics to each other is shown through a series of theorems. In section 5.3, procedural semantics is introduced by modifying the SLI resolution
properly to cope with fuzziness including pseudo-complementarity in the sense of the previous chapter. In section 5.4, the soundness of the Fuzzy SLI resolution is reached and in section 5.5 the completeness. Then in section 5.6, it will be particularly pointed out that fuzzy logic offers a very attractive way of handling negation in fuzzy disjunctive logic programming much better that that offered by the generalized closed world assumption and the extended generalized closed world assumption.

5.1 Syntax of FLP3

As a start, the definition of a fuzzy disjunctive program clause, a subclause, the fuzzy disjunctive Herbrand base, the fuzzy canonical form and the fuzzy expansion for a set of fuzzy disjunctive ground clauses are presented:

Definition 5.1.1: A term is defined inductively as:

1. A variable is a term.
2. A constant is a term.
3. If $f$ is an n-ary function symbol and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.
4. Terms can only be constructed by applying the above rules a finite number of times.

Definition 5.1.2: A fuzzy disjunctive logic program clause is a fuzzy program clause of the form:

$$< A_1(t_1, \ldots, t_n), \mu_{A_1} > \lor \ldots \lor < A_k(t_1, \ldots, t_n), \mu_{A_k} >$$

$$\leftarrow - (f) - < B_1(t_1, \ldots, t_n), \mu_{B_1} > \ldots, < B_m(t_1, \ldots, t_n), \mu_{B_m} >$$

with $k \geq 1$ and $m \geq 0$ where $A_1, \ldots, A_k, B_1, \ldots, B_m$ are fuzzy atoms (pairs with $\mu$ cuts but omitted here for ease of notation), and where $m = 0$, it is called a fuzzy disjunctive assertion. Then a subclause, a proper subclause and their fuzzy counterparts are defined.
Definition 5.1.3: A ground clause \( A = A_1 \lor \ldots \lor A_n \) is a subclause of a ground clause \( B = B_1 \lor \ldots \lor B_m \) if for each \( A_i \), \( 1 \leq i \leq n \), there is a \( B_j \) such that \( A_i = B_j \). The clause \( A \) is said to be a proper subclause of \( B \) if it is a subclause and there is a \( B_j, 1 \leq j \leq m \) such that for all \( A_i \) in \( A \), \( B_j \neq A_i \). Then, the fuzzy disjunctive Herbrand base is defined as follows.

Definition 5.1.4: Let \( L \) be a fuzzy first-order language and let \( P \) be a fuzzy disjunctive program in \( L \). The fuzzy disjunctive Herbrand base of \( L \), denoted by \( FDHB_L \) or \( FDHB_P \) is the set of all fuzzy disjunctive ground clauses which can be formed using distinct ground atoms from the Herbrand universe (after removing the cut part) of \( L \) or \( P \) such that no two logically equivalent (definition 5.2.1 below) clauses are in the set. The fuzzy canonical form for a set \( S \) of fuzzy ground clauses is defined.

Definition 5.1.5: Let \( S \) be a set of fuzzy ground clauses. Then the fuzzy canonical form of \( S \), denoted by \( fcan(S) \), is defined as follows: \( fcan(S) = \{ C \in S | \nexists C' \in S \text{ such that } C' \text{ is a proper subclause of } C \} \). Then, the fuzzy expansion of a set of ground clauses is defined.

Definition 5.1.6: Let \( L \) be a fuzzy first-order language and let \( P \) be a fuzzy disjunctive logic program in \( L \). Let \( S \) be a set of fuzzy ground clauses in \( L \). The fuzzy expansion of \( S \), denoted by \( fexp(S) \), is defined as follows: \( fexp(S) = \{ C \in FDHB_L | C \in S \text{ or } \exists C' \in S \text{ such that } C' \text{ is a subclause of } C \} \). Then, a fuzzy model-state is defined.

Definition 5.1.7: Let \( L \) be a fuzzy first-order language and Let \( P \) be a fuzzy disjunctive logic program in \( L \). A fuzzy state is a subset of the fuzzy disjunctive Herbrand base.

Definition 5.1.8: Let \( L \) be a fuzzy first-order language and let \( P \) be a disjunctive logic program in \( L \). A fuzzy expanded state for \( L \) is a fuzzy state such that \( S = fexp(S) \), where \( fexp(S) \) is given by definition 5.1.6.

Example 5.1.1: Let \( S \) be the set \( S = \{ p \lor q, p, q \} \), \( fcan(S) = \{ p, q \} \).
Example 5.1.2: Let $P$ be the program:

\[
\begin{align*}
p(f(x)), r(x) &\leftarrow q(x, g(x)) \\
q(a, g(b)) \\
q(b, g(b)).
\end{align*}
\]

Then $U_P = \{ a, b, f(a), f(b), g(a), g(b), f(f(a)), f(g(a)), \ldots \}$,

\[
FDHP = \{ p(a), r(a), q(a, a), p(b), r(b), q(b, b), p(a) \lor r(a), p(a) \lor q(a, a),
\]

\[
p(f(a)) \lor r(a), \ldots \}.
\]

Let $S = \{ q(a, g(b)), r(a) \}$, then:

\[
\exp(S) = \{ q(a, g(b)), r(a), q(f(a), f(g(b))), r(f(a)), r(g(a)), q(a, g(b)) \lor r(a),
\]

\[
q(a, g(b)) \lor q(f(a), f(g(b))), r(a) \lor q(f(a), f(g(b))), \ldots \}.
\]

Example 5.1.3: A Fuzzy Disjunctive Logic Program clause may look like:

\[
< \text{ProjectManager}(X), \mu_1 >, < \text{DivisionManager}(X), \mu_2 > \leftarrow
\]

\[
< \text{Experience}(X, 10), \mu_3 >
\]

which means that if $X$ has ten years of experience, he can be either a Project Manager (to a certain degree) or a Division Manager (to a certain different or same degree).

5.2 Model-State Semantics & Fixpoint Theory

In the beginning, the fuzzy model-state and the proximate model-state for a set of closed formulae are defined. Then, the Fuzzy Model-State Intersection Property is proven. Then, a theorem showing that the minimal model-state contains the clauses that are logical consequences of the program is proven. Then, a closure operator suitable for these programs is defined and its fixpoint characterization theorem is proven.

Definition 5.2.1: Let $F_1$ and $F_2$ be two formulae, they are logically equivalent if they have identical truth values under every possible Herbrand interpretation.

Definition 5.2.2: Let $L$ be a fuzzy first order language. Let $S$ be a set of closed formulae of $L$. A fuzzy model-state for $S$ is a fuzzy expanded state $ST$ such that:

1. Every Herbrand model of $ST$ is a Herbrand model of $S$. 

67
2. Every minimal Herbrand model of $S$ is contained in a minimal Herbrand model of $ST$.

A model-state $ST$ of a set of fuzzy closed formulae is \textit{minimal} if no state $S'$ of $S$ which is a proper subset of $ST$ is a model-state of $S$.

\textbf{Definition 5.2.3}: Let $L$ be a fuzzy first order language. Let $S$ be a set of closed formulae of $L$. A \textit{proximate model-state} in the sense of the previous chapter for $S$ is a fuzzy expanded state $ST$ such that:

1. Every Herbrand model of $ST$ is Herbrand model of $S$ or proximate to it as in the previous chapter.

2. Every minimal Herbrand model of $S$ is contained in a minimal Herbrand model of $ST$ or otherwise a proximate model is contained.

Then, a theorem which is analogous to the model intersection property is proven.

\textbf{Definition 5.2.4}: Let $M_1$ be a model-state and $M_2$ be a model-state, then $M_1 \cap M_2$ is a model-state. $FMS_P = \cap \{ M | M \text{ is a model-state} \}$.

\textbf{Theorem 5.2.1}: (Fuzzy Model-State Intersection Property)

Let $P$ be a fuzzy disjunctive logic program and $\{FMS \}_{i \in N}$ be a non-empty set of fuzzy model-states of $P$, where $N$ denotes the set of natural numbers. Then, $
\cap_{i \in N} FMS$ is a model-state of $P$.

\textbf{Proof}: For model-state $FMS_1$ and model-state $FMS_2$, $FMS_1 \cap FMS_2$ is a model-state. For $i$ model-states, $\cap(FMS_i)$ is a model-state. And for a finite number of model states $\cap(FMS_i)$ is an model-state. 

\textbf{Definition 5.2.5}: Let $S$ be a set of closed formulae and $F$ be a closed formula of a fuzzy first-order language $L$. $F$ is said to be a \textit{logical consequence} of $S$ if, for every interpretation $I$ of $L$, $I$ is a model-state for $S$ implies that $I$ is a model-state for $F$. We can also have proximate logical consequence in the sense of the
previous chapter. The following theorem establishes the equivalence between the declarative semantics and the model-state semantics.

**Theorem 5.2.2:** Let $P$ be a fuzzy disjunctive logic program. Then if $FMS_P$ is the fuzzy minimal model-state, one obtains:

$$FMS_P = \{ A \in FDHB_P : A \text{ is a logical consequence of } P \}.$$  

**Proof:** $A$ is a logical consequence of $P$

- if $P \cup \{ \neg A \}$ is unsatisfiable.
- if $P \cup \{ \neg A \}$ has no Herbrand models.
- if $\neg A$ is false w.r.t. to all Herbrand models.
- if $A$ is true w.r.t. to all Herbrand models.
- if $A$ is true w.r.t. all Herbrand models of any model state of $P$.

if $A \in FMS_P$.  

**Definition 5.2.6:** Let $C$ be a clause. If two or more atoms of $C$ have a most general unifier $\theta$, then $C\theta$ is called a *factor* of $C$. If $C$ is a ground clause then its *minimal factor*, denoted by $\text{fac}(C)$, is the subclause $C'$ of $C$ such that every atom in $C'$ is distinct and $C \leftarrow C'$ and $C \rightarrow C'$.

**Definition 5.2.7:** Let $P$ be a fuzzy disjunctive logic program and let $D$ be a fuzzy Herbrand state of $P$. The closure operator $FT_P : [0,1]^{FDHB_P} \rightarrow [0,1]^{FDHB_P}$ is defined: $FT_P(D) = \{ C \in FDHB_P| C' \leftarrow B_1, B_2, \ldots, B_n \text{ is a fuzzy ground instance of a fuzzy program clause in } P \text{ and } B_1 \lor C_1, \ldots, B_n \lor C_n \text{ are in } D \text{ and } C'' = C' \lor C_1 \lor \cdots \lor C_n, \text{ where } \forall i, 1 \leq i \leq n, C_i \text{ can be null, and } C \text{ is the smallest factor of } C'' \}$. The cut happens between the negatively occurring $B_i's$ (Lloyd [107] pp. 7) and the positive $B_i's$ in a pseudo-complementarity sense and $\mu_C = \sup\{f(x_i, \mu_{B_i})\}$.

This fixpoint operator formalises the hyperresolution rule of Lobo et al. [112].

**Theorem 5.2.3:** $f\text{can}(FMS_P) = f\text{can}(\text{lf}(FT_P^S)) = f\text{can}(FT_P^S \uparrow \omega)$

**Proof:** $f\text{can}(MS_P) = f\text{can}(\text{glb}\{FMS : FMS \text{ is a model-state of } P\})$

$$= f\text{can}(\text{glb}\{FMS : FT_P(FMS) \subseteq FMS\})$$, by theorem 3.10 of [112]
= \text{fcan}(lfp(FT_p)) \text{ (by theorem X.3.1)}
= \text{fcan}(FT_p \uparrow \omega) \text{ (by lemma 3.2.3).} 

5.3 Procedural Semantics
At the beginning, a linear derivation and a t-clause are defined. Then the new fuzzy truncfac derivation on which the Fuzzy SLI derivation is based is introduced. At the end of the section, the Fuzzy SLI refutation is presented.

Definition 5.3.1: Let \( S \) be a set of clauses and let \( C_0 \in S \). A linear derivation of \( C_n \) from \( S \) with top-clause \( C_0 \) is a finite sequence of clauses \( C_0, \ldots, C_n \) such that \( C_{i+1} \) is either a factor of \( C_i \), or a resolvent of \( C_i \) and a clause \( B_i \) for some \( i, 0 \leq i \leq n - 1 \), where \( B_i \) is either a factor of a clause in \( S \) or a clause \( C_j \) for some \( j, 0 \leq j < i \).

Definition 5.3.2: A t-clause \( C \), is an ordered pair \( < C, m > \) where:
1. \( C \) is a labeled tree whose root is labeled with the distinguished symbol \( \epsilon \), and whose other nodes are labeled with literals; and
2. \( m \) is marking (unary) relation on the node such that every non-terminal node in \( C \) is marked. Figure 5.1 clarifies the t-clause notation.

Definition 5.3.3: Let \( L \) be an unmarked literal in a t-clause.

\[ \delta_L = \{ N : \text{where } N \text{ is a marked literal and an ancestor of } L \} \]

\[ \gamma_L = \{ M : \text{where } M \text{ is an unmarked literal and a sibling of an ancestor of } L \} \].

The set \( \gamma_L \) captures the literals to be checked for fuzzy factoring and the set \( \delta_L \) to capture those to be checked against pseudo-complementarity defined below.

Referring to figure 5.3:

\[ \gamma_p = \{ < p_1(Z,b), \mu >, < q_1(l(X), \mu ) > \} \]

\[ \gamma_q = \{ < r(a,b), \mu >, < s(c), \mu > \} \]

\[ \delta_p = \{ < s_1(g(a), f(b)), \mu >, < t(a,b), \mu >, < \neg s(a), \mu >, < r(Z,b), \mu > \} \]

\[ \delta_q = \{ < \neg s(a), \mu >, < \neg q(Z), \mu >, < \neg s_2(g(b,a)), \mu >, < p_2(f(b,a)), \mu > \} \]

The following condition ensures that there are no loops in the t-clause.
Definition 5.3.4: A t-clause is said to satisfy the *admissibility condition* (AC) if for every occurrence of every unmarked literal $L$ in it the following conditions hold:

1. No two literals from $\gamma_L$ and $L$ have identical atoms (modulo variable renaming).
2. No two literals from $\delta_L$ and $L$ have identical atoms (modulo variable renaming).

Figure 5.4 shows an example of two t-clauses one satisfying the admissibility condition and another that does not. The following condition ensures that truncation (defined below) happens as soon as possible.

Definition 5.3.5: A t-clause is said to satisfy the *minimality condition* (MC) if there is no marked literal which is a terminal node.

Figure 5.5 shows an example of two t-clauses one satisfying the minimality condition and another that does not.

Definition 5.3.6: Let $C_o$ be a t-clause. The t-clause $C_n$ is a *fuzzy truncation derivation* (truncation, fuzzy pseudo-complementarity and fuzzy pseudo-factoring) of $C_o$ when there is a sequence of t-clauses $C_o, C_1, \ldots, C_n$ and substitutions $\theta_{o},\theta_{1},\ldots,\theta_{n-1}$ such that for all $i, 0 \leq i < n, C_{i+1}$ is obtained from $C_i$ by either fuzzy pseudo-factoring, fuzzy pseudo-complementarity or t-truncation with substitution $\theta_{i}$.

$C_{i+1}$ is obtained from $C_i$ by *fuzzy pseudo-factoring* iff:

1. $C_i$ is $(\alpha_1 L \alpha_2 M \alpha_3)$ or $C_i$ is $(\alpha_1 M \alpha_2 L \alpha_3)$;
2. $M \theta_i \simeq L \theta_i$ where $\theta_i$ is a substitution in the sense of pseudo-complementarity.
3. $L$ is in $\gamma_M$ (that is, $L$ is an unmarked sibling of an ancestor of $M$);
4. $C_{i+1}$ is $(\alpha_1 L \alpha_2 \alpha_3) \theta_i$ or $C_{i+1}$ is $(\alpha_1 \alpha_2 L \alpha_3) \theta_i$. 

71
$C_{i+1}$ is obtained from $C_i$ by fuzzy pseudo-complementarity iff:

1. $C_i$ is $(\alpha_1(L^*\alpha_2M\alpha_3)\alpha_4)$;

2. $L\theta_i \simeq \neg M\theta_i$ where $\theta_i$ is a substitution (most general and non-variable);

3. $L$ is in $\delta_M$;

4. $C_{i+1}$ is $(\alpha_1(L^*\alpha_2\alpha_3\alpha_4))\theta_i$

$C_{i+1}$ is obtained from $C_i$ by t-truncation with $\theta_i$ equal to the identity substitution iff either $C_i$ is $(\alpha(L^*)\beta)$ and $C_{i+1}$ is $(\alpha\beta)$ or $C_i$ is $(\epsilon^*)$ and $C_{i+1}$ is empty. Figures 6, 7, 8 show these cases.

**Definition 5.3.7:** Let $C_i = (\epsilon^*\alpha_1L\beta_1)$ be a t-clause. Let $B_{i+1} = (\epsilon^*\alpha_2M\beta_2)$ be another t-clause from $C_i$. Then $C_{i+1}$ is fuzzy t-derived from $C_i$ and $B_{i+1}$ using the substitution $\theta_i$ if the following conditions hold:

1. $L\theta_i' \simeq \neg M\theta_i'$ where $\theta_i'$ is a substitution (most general);

2. $C_{i+1}'$ is $(\epsilon^*\alpha_1(L^*\alpha_2\beta_2)\beta_1)\theta_i'$

3. $C_{i+1}'$ is either a fuzzy truncac-derivation of $C_{i+1}'$ with substitution $\theta_i''$ or directly $C_{i+1}'$ and for this case $\theta_i'' = \epsilon$;

4. $\theta_i = \theta_i', \theta_i''$;

5. $C_{i+1}$ must satisfy the admissibility and minimality conditions.

**Definition 5.3.8:** Let $S$ be an input set of t-clauses and let $C$ be a t-clause in $S$. A Fuzzy SLI derivation of a t-clause $E$ from $S$ with top t-clause $C$ is a sequence of t-clauses $(C_1, \ldots, C_n)$ such that:

1. $C_1$ is either $C$ or a fuzzy truncac derivation of $C$, and $C_n$ is $E$;

2. For all $i$, $1 \leq i \leq n - 1$, $C_{i+1}$ is t-derived from $C_i$ and a t-clause $B_{i+1}$ in $S$.  

72
Definition 5.3.9: Let $S$ be an input set of t-clauses and let $C$ be a t-clause in $S$. A Fuzzy SLI refutation from $S$ with top t-clause $C$ is a fuzzy SLI derivation of the empty clause. The empty clause is derived from $\leftarrow A_{\text{goal}}(t_1, t_2, \ldots, t_n)$ and $A_{\text{fact}}(t_1, t_2, \ldots, t_n))$ $\leftarrow$ with $\mu_{A_{\text{fact}}} \geq \mu_{A_{\text{goal}}}$ or $\mu_{A_{\text{fact}}} \approx \mu_{A_{\text{goal}}}$.

5.4 Soundness of Fuzzy SLI Resolution

The soundness of Fuzzy SLI resolution is proven in this section. The definitions of the fuzzy t-model of a clause and a set of t-clauses are needed first.

Definition 5.4.1: Let $P$ be a fuzzy disjunctive logic program. An Herbrand interpretation $M$ of $P$ is said to be a model of a t-clause $C$ or a t-model of $C$, if and only if there exists an unmarked literal $L$ in $C$ such that for all $K \in \delta_L, M \models K$ and $M \models L$.

Definition 5.4.2: Let $P$ be a fuzzy disjunctive logic program. An Herbrand interpretation $M$ of $P$ is said to be a t-model of a set $S$ of t-clauses (written $M \models S$) if and only if $M$ is a t-model of every t-clause in $S$.

Theorem 5.4.1: (Soundness of Fuzzy SLI resolution) Let $S$ be a set of input fuzzy t-clauses. If $C$ is derivable from $S$ by a fuzzy SLI derivation then for every t-model $M$, if $M \models S$ then $M \models C$.

Proof: Let $S^*$ and $C^*$ be $S$ and $C$ after the cut operation is applied, then by theorem 4.4 of Lobo et al. [112], one obtains: if $M \models S^*$, then $M \models C^*$, but every step in the SLI-derivation is applied to crisp atoms (all non-variable substitutions), then we have: $M \models S$, then $M \models C$.

5.5 Completeness of Fuzzy SLI Resolution

As a start, a mapping that represents the SLI resolution is defined then it is shown that its least fixpoint is equivalent to that of the closure operator defined earlier $FT^S_P$.

Definition 5.5.1: Let $C_o$ be a t-clause and $C_n$ a fuzzy trufac derivation or a fuzzy t-derivation of $C_o$, the mapping $T$ is defined such that:

$$ T(C_i) = C_{i+1} $$
Theorem 5.5.1: For any \( m, n \in \omega \), \( \text{lft}(T) \uparrow n = \text{lft}(FT_p^S) \uparrow m \), where \( m, n \) may be different.

**Proof:** The mapping \( FT_p^S \) defines the hyperresolution rule. Its original definition by Lobo et al. [112] is by multiple cuts between complementary literals. The definition extends it to be between pseudo-complementary literals.

This will lead us to a stronger relationship between both mappings as in the following theorem.

Theorem 5.5.2: \( lft(T) = lft(T_p^S) \)

**Proof:** By the above theorem for a large enough \( n \), i.e. \( n = \omega \), one has \( T \uparrow \omega = T_p^S \uparrow \omega \) which leads to the proof of the theorem using theorem 1.2 (Ebrahim, [50]).

**Definition 5.5.2:** Let \( P \) be a fuzzy disjunctive logic program. The *success set* of \( P \) is all the goals \( A \) such that \( P \cup \{\leftarrow A\} \) has a fuzzy SLI-refutation.

**Theorem 5.5.3:** The success set of a fuzzy disjunctive logic program is equal to its minimal model-state.

**Proof:** First, it will be shown that the success set of a fuzzy program is contained in its minimal model-state. If the goal \( A \) has a Fuzzy SLI-refutation, then by the soundness theorem \( A \) is a logical consequence of \( P \). By the fixpoint characterization theorem, \( A \) is in the minimal model-state. It must be shown that the minimal model-state of \( P \) is contained in the success set of \( P \). Suppose \( A \) is in the minimal model-state of \( P \). By the fixpoint characterization theorem and theorem 5.5.2, \( A \in FT_p^S \uparrow n \) for some \( n \in \omega \). It is proven by induction on \( n \) that \( A \in FT_p^S \uparrow n \) implies that \( P \cup \{\leftarrow A\} \) has a fuzzy SLI-refutation and hence \( A \) is in the success set. Suppose that \( n = 1 \). Then \( A \in FT_p^S \uparrow 1 \) means that \( A \) is a ground instance of a unit fuzzy clause in \( P \), i.e. \( P \cup \{\leftarrow A\} \) has a refutation. Suppose the result holds for \( n - 1 \). Let \( A \in FT_p^S \uparrow n \). By the definition of \( FT_p^S \), there exists a ground instance of a fuzzy clause \( B \leftarrow B_1, \ldots, B_k \) such that \( A \simeq B\theta \) and \( \{B_1\theta, \ldots, B_k\theta\} \subseteq FT_p^S \uparrow (n - 1) \), for some \( \theta \). By the induction hypothesis,
\( P \cup \{ \leftarrow (B_1, \ldots, B_k) \theta \} \) has a refutation for \( i = 1, \ldots, k \). Because each \( B_i \theta \) is ground, these refutations can be combined into a single fuzzy SLI refutation of \( P \cup \{ \leftarrow (B_1, \ldots, B_k) \theta \} \). Thus \( P \cup \{ \leftarrow A \} \) has a fuzzy SLI refutation. ■

**Theorem 5.5.4 (Fuzzy SLI Resolution Completeness)** If \( \{ \theta_1, \ldots, \theta_n \} \) is a correct answer for \( P \cup \{ G \} \), there exists an SLI computed answer \( \{ \sigma_1, \ldots, \sigma_k \} \) from an SLI refutation.

**Proof:** If \( G \) is the fuzzy goal \( \leftarrow A_1, \ldots, A_n \) that has a fuzzy correct answer, then:

1. Let \( \theta \) be any correct answer for \( P \cup \{ G \} \), then:
2. \( \forall((A_1 \land \ldots \land A_k) \theta)) \) is a logical consequence. (by definition of a correct answer), then:
3. \( \forall((A_1 \land \ldots \land A_k) \theta)) \) is in the minimal model-state, then:
4. \( \forall((A_1 \land \ldots \land A_k) \theta)) \) is in the success set, (theorem 5.5.3), then:
5. \( P \cup \{ G \} \) has a refutation. (by theorem 5.5.3) ■

5.6 Negation in Fuzzy Disjunctive Logic Programming

In this section, the treatment of negation in fuzzy logic programming will be presented and it will be fruitful in fuzzy disjunctive logic programming. The model-theoretic and proof-theoretic definitions of the generalized closed world assumption used for negation in disjunctive logic programming are given.

**Definition 5.6.1: (Semantic Definition of the GCWA)**

Let \( P \) be a disjunctive program. The model-theoretic definition of the generalized closed world assumption is given by: \( GCWA(P) = \{ \neg A \mid A \in HB_P \text{ and } A \text{ is not in any minimal Herbrand model of } P \} \).

**Definition 5.6.2: (Syntactic Definition of the GCWA)**

Let \( P \) be a disjunctive logic program. The proof-theoretic definition of the generalized closed world assumption is given by: \( GCWA(P) = \{ \neg A \mid A \in HB_P \text{ and for all } K \text{ positive (possibly null) ground clauses, } P \models A \lor K \text{ implies } P \models K \} \).
The definitions of the generalized closed world assumptions for disjunctive logic programs to cope with the inconsistency that arises if one is to use the closed world assumption. The closed world assumption defines a success set and anything else would be in the failure set. The generalized closed world assumption defines a success set, a failure set and an unknown set. This makes it more suitable for disjunctive logic programs. In fuzzy disjunctive logic programming, fuzzy logic offers a very attractive solution to deal with negation using the transformations developed in the section on negation in fuzzy logic programming in chapter 3. No any other special treatment is needed. Only if the system encounters negation in front of a crisp atom, the generalized closed world assumption as a meta-rule could be invoked. One can view how more expressive is fuzzy logic especially in its handling of negation. The syntactic handling of negation is substituted by its semantic truth values. Thus, one can say that fuzzy logic offers a better treatment of negation in various logic programming systems.

Conclusion

Fuzzy Disjunctive Logic Programming has been presented with its syntax, and different approaches to semantics which have been shown equivalent. This chapter together with the previous two provide a sound theoretical background for fuzzy expert systems where the designer of the system has the confidence that his system produces only correct answers and all the correct answers possible. The importance of this is also very clear.
S = \{ \langle \text{getto}(X,Y), \mu \rangle \lor \langle \text{getto}(X,Z), \mu \rangle \leftarrow \langle \text{path}(X,Y,\text{via}(Z)), \mu \rangle \\
\langle \text{Path}(\text{was},\text{ny},\text{via}(\text{phil})), \mu \rangle \lor \langle \text{at}(\text{was}), \mu \rangle \\
\langle \text{at}(X), \mu \rangle \leftarrow \langle \text{getto}(Y,X), \mu \rangle \} \\

\text{Fig. 5.1 t-notation of the above set of disjunctive clauses}
Fig. 5.2 t-derivation of the above set of disjunctive clauses
Fig. 5.3 Literals to be checked against factoring and ancestry resolution

Fig. 5.4 The first t-clause satisfies the admissibility condition whereas the other does not.
Fig. 5.5 The first t-clause satisfies the minimality condition whereas the other does not.

Fig. 5.6 The second t-clause can be obtained from the first after t-factoring.

Fig. 5.7 The second t-clause can be obtained from the first by t-ancestry
Fig. 5.8 The second t-clause can be obtained from the first by t-truncation
Chapter 6

The Fuzzy Relational Model

A new fuzzy relational model founded on two new notions of Domain-Associated Fuzziness and Tuple-Associated Fuzziness is presented. It is shown that the new model with these two new notions captures most of the features demonstrated in fuzzy models in previous work. Independent fuzzy relational algebra, fuzzy tuple relational calculus and the fuzzy domain relational calculus are presented for the new model. Their equivalence is proven in a sequence of original theorems.

6.1 Motivating the Fuzzy Relational Model

Fuzzy set theory can extend the classical relational model without losing any of its simplicity or elegance. In this section, it will be intuitively shown that this is possible while in later sections it will be demonstrated formally. Let's discuss the first concept of Domain-Associated Fuzziness. If domain values are allowed to be fuzzy sets, one ends up having each element in a domain should be a pair. The pair consists of the domain value and its membership value. Instead, fuzzy sets will be allowed to happen over the domain. In a particular situation, there could be more than one fuzzy set over the domain, each one would induce its own membership values to domain elements. These membership values should be stored separately and not in the relational table. They should be stored in a separate definition of each fuzzy set. Since this is data about data, it can be considered as a fuzzy extension to the data dictionary. This approach will maintain compatibility with existing relational database systems as it does not affect the atomicity of the values in the table.
Second, the concept of *Tuple-Associated Fuzziness* relates to associating a membership value for each tuple in the relation. This value may be computed depending on one or more domain value or it might be independent.

### 6.2 The Structural Part of the Model

In this section, the above concepts of *Domain-Associated Fuzziness* and *Tuple-Associated Fuzziness* will be formalized:

**Domain-Associated Fuzziness**

**Definition 6.2.1:** The universe of a fuzzy relational database denoted by $U$, is a finite nonempty set of elements $A_1, A_2, \ldots, A_n$ called the attribute names or simply attributes, i.e.

$$U = \{A_1, A_2, \ldots, A_n\}$$

**Definition 6.2.2:** A domain $DOM(A_i)$ for an attribute $A_i$ takes values from a finite crisp set. One or more fuzzy set can occur over this domain inducing a pair, i.e. the *domain value* and its *membership value*. In the table, only the domain value is stored. The membership functions are stored separately.

**Example 6.2.1:** Let *Student-ID* and *Age* be attributes for the table *Students*. The values for *Age* could be numbers denoting the student’s age. One can have several fuzzy sets defined on the *Age* domain: *Young*, *Moderately Young*, *Old*, *Moderately Old*. For each fuzzy set, a separate view can be computed, i.e. the table *Young-Students* as a crisp table after applying the cut operation. Such a table would look like an asserted fact in the fuzzy logic programming systems developed earlier.

**Tuple-associated Fuzziness**

These are values $\mu \in [0, 1]$ associated with each tuple indicating its relevance with the concept of the fuzzy relation. These values may depend on values assigned by fuzzy sets defined on the relation domain. For instance, one can have a tuple in the
relation CUSTOMER[ID, Name, AssetValue, Balance]. A useful fuzzy relation can be CREDIT-WORTHINESS. As far as domain-associated fuzziness is concerned, several fuzzy sets [Low, Average, Good, High, VeryHigh] can be defined over the domains of the attributes AssetValue and Balance. The tuple-associated fuzziness would define a value for each tuple in the CREDIT-WORTHINESS fuzzy relation. This membership value may depend on membership values that a particular tuple may have for the attributes AssetValue and Balance. It may be followed by a cut operation.

In the following, it will be shown that these two simple notions capture all other generalizations of other fuzzy models mentioned in chapter 2 while in the meantime maintaining the objective of being fully compatible with existing relational database management systems which require table values to be atomic.

First, the fuzzy-relation based approach will be considered: clearly, the tuple-associated fuzziness captures its concepts. In addition, the tuple-associated fuzziness provides a means via which this membership value can be computed.

Second, the similarity-based approach: the domain-associated fuzziness captures this concept and more. If a similarity relation is associated on the domain, this can be achieved by defining a fuzzy set over the domain. So, the domain-associated fuzziness is more general than the similarity-based approach as it allows more than one fuzzy set to be defined over the domain.

Third & Fourth, the possibility-based approach and the extended-possibility based approach with similarity and closeness relations can deviate a lot from the objective of being compatible with existing systems as non-atomic values are allowed in the tables.

Fifth, the combined approaches: the approach is definitely a combined one but it differs from that of Kerre [95] in that they allow a possibility and a necessity measure.
6.3 The Fuzzy Relational Algebra

The manipulative part of the fuzzy relational model which is essentially founded on fuzzy set theory will be introduced in this section. In the following, the fuzzy algebraic operations are presented. These operations are considered prior to the cut operation when the relations are fuzzy, otherwise classical operations apply. (The following are fuzzy generalizations to that of Yang [148]).

The Fuzzy Relational Algebraic Operations

Some fuzzy algebraic operations {e.g. union, intersection and difference} need their operands to be union-compatible. The following definition captures what is meant by two relations being union-compatible.

Two relations \( r(R) \) and \( s(S) \) in a database are \textit{union-compatible} if they have the same set of attributes, i.e. if \( R = S \).

1. Fuzzy Union

The \textit{union} of two union-compatible fuzzy relations \( r(R) \) and \( s(S) \) written as \( r \cup s \) is the relation over \( R \) consisting of each tuple belonging to either \( r \) or \( s \) and if both, its membership value is taken to be the maximum or any other appropriate t-norm (Zimmerman [155]):

\[
r \cup s = \{t \mid t \in r \text{ or } t \in s \}
\]

where \( t \) is a tuple either in \( r \) or in \( s \) or in both and if in both, its degree of membership is \( \max(t_r, t_s) \) or \( t\text{-norm}(t_r, t_s) \).

Since the fuzzy union operation is defined in terms of the fuzzy tuples of the operand relations, tuple-associated fuzziness must be considered while domain-associated fuzziness is irrelevant to this operation.

2. Fuzzy Intersection

The \textit{intersection} of two union-compatible relation \( r(R) \) and \( s(S) \) written as \( r \cap s \), is the fuzzy relation over \( R \) consisting of all tuples belonging to both
Since the fuzzy intersection operation is defined in terms of the fuzzy tuples of the operand relations, tuple-associated fuzziness must be considered. The membership value of the resultant fuzzy tuple will be evaluated according to an appropriate t-norm which could be the minimum. Domain-associated fuzziness is irrelevant to this operation.

3. Fuzzy Difference

The fuzzy difference of two union-compatible fuzzy relations \( r(R) \) and \( s(S) \) written as \( r - s \) is the fuzzy relation over \( R \) consisting of each tuple belonging to \( r \) but not to \( s \):

\[
 r - s = \{ t | t \in r \text{ and } t \notin s \}
\]

As above, domain-associated fuzziness is irrelevant to this operation.

4. Fuzzy Projection

Let \( U_j \) be a nonempty subset of \( U \), \( r_j(U_j) \) be a fuzzy relation over \( U_j \) and \( X \) be a nonempty subset of \( U_j \). The fuzzy projection of \( r_j \) onto \( X \), written as \( \Pi_X(r_j) \) or \( r_j[X] \) is the fuzzy relation over \( X \) consisting of the \( X \)-value of each tuple in \( r_j \):

\[
 \Pi_X(r_j) = \{ t[X] | X \subseteq U_j \text{ and } t \in r_j(U_j) \}
\]

where the membership value of the resultant tuples will be re-evaluated (via a t-conorm like max.) according to the columns retained.

5. Fuzzy Selection

To select some certain fuzzy tuples from a fuzzy relation, this should be done according to a specific fuzzy condition. This fuzzy condition should be a fuzzy formula defined recursively as below:
(a) $A_j Q v B_k$, $A_j Q v c$ and $c Q v A_j$ are formulae where $A_j$ and $B_k$ are compatible attributes in $U$, $c$ is an element in $\text{DOM}(A_j)$ and $\nu$ is a comparison operator (arithmetic or order) $\{=, \neq, <, \leq, >, \geq\}$ and $Q$ is a fuzzy modifier which could be applicable to a certain fuzzy comparison operator like “much less than”, “much greater than”, “more or less equal”, “very” or “approximately”. Their definition is via a possibility distribution, e.g.: “$x$ is much greater than $a$” $\triangleq_{\text{def}} 1/[(x-a \geq 30)] + 0.8/30 < (x-a) \leq 25] + 0.6/25 < (x-a) \leq 20] + 0.5/20 < (x-a) \leq 15]$. 

(b) If $G$ and $H$ are formulae, their fuzzy conjunction, disjunction and negation are formulae as well.

(c) Nothing else is a formula.

So, the selection of a relation $r_j(U_j)$ under a fuzzy formula $F$ that is applicable to $r_j$ is defined as the subset of $r_j$, written as $\sigma_F(r_j) = \{t | t \in r_j \text{ and } t \text{ satisfies } F\}$.

6. Fuzzy Theta-Join

Let $r(R)$ and $s(S)$ be two fuzzy relations and $A_j Q v B_k$ be a fuzzy formula applicable to the complex product $r \ast s$. The theta-join of $r(R)$ and $s(S)$ on the attributes $A_j$ in $R$ and $B_k$ in $S$, written as $r[A_j Q v B_k]s$, is the set consisting of every string $t_r t_s$ for some $t_r$ in $r$ and some $t_s$ in $s$ such that $t_r(A_j)Qv t_s(B_k)$ is evaluated to some truth value:

$$r[A_j Q v B_k]s = \{t_r t_s | t_r \in r, t_s \in s, \text{ and } t_r(A_j)Qv t_s(B_k) = \mu \text{ (some truth value)}\}$$

This truth value can be fuzzy. Or the Fuzzy Theta-Join can be expressed in terms of complex product and selection as:

$$r[A_j Q v B_k]s = \sigma_{A_j Q v B_k}(r \ast s)$$
when the complex product \( r \cdot s \) is not a fuzzy relation (because of the existence of duplicated columns, including names and values) the selection is undefined.

The theta-join could be viewed as a partial function from the complex product \( r \cdot s \) to \( r \cdot s \) such that for each argument \( t_r t_s \) in \( r \cdot s \) is still \( t_r t_s \) if \( t_r(A_j) \land t_s(B_k) \) evaluated to some truth value which is the tuple’s membership in the resultant fuzzy relation.

7. Fuzzy Natural Join

(Comment: Fuzzy Natural join by the redefinition of the relation “Equal”)

If the fuzzy formula \( F \) contained the \( \nu \) operator to be equal, i.e. an equi-join, a mechanism should be applied to avoid the repeated column values repeated in the equi-join which is the theta-join and \( \nu \) is equality. Fuzzy natural join overcomes this problem as defined below:

The natural join or simply a join of \( r \) and \( s \) written as \( r \ri 	imes s \) is the relation over the scheme \( R(S - R \cap S) \) consisting of all fuzzy tuples such that for each fuzzy tuple \( t \) in the fuzzy join, there exists some tuple \( t_r \) in \( r \) and some tuple \( t_s \) in \( s \) satisfying \( t[R] = t_r \) and \( t[s] = t_s \), i.e.

\[
(\exists \, (t_r, t_s) \in r \times s) \land (t[R] = t_r \land t[S] = t_s)
\]

when \( R \) and \( S \) are disjoint, the join of \( r \) and \( s \) is identical to the complex product of \( r \) and \( s \), i.e.

\[
r \ri \times s = r \cdot s \quad (\text{if } R \cap S = \emptyset)
\]

8. Fuzzy Complex Product

The cartesian product of fuzzy sets is defined as follows: Let \( A_1, A_2, \ldots, A_n \) be fuzzy sets in \( X_1, X_2, \ldots, X_n \). The cartesian product is then a fuzzy set
in the product space $X_1 \times X_2 \times \cdots \times X_n$ with the membership function:

$$\mu_{A_1 \times A_2 \times \cdots \times A_n} = \min\{\mu_{A_i}(x_i) | x_i \in X_i\}$$

The complex product of two non-empty fuzzy relations $r(R)$ and $s(S)$ (for disjoint $R$ and $S$) written as $r \ast s$ is the fuzzy set consisting of the string $t_r t_s$ for each pair of strings $(t_r, t_s)$ in the fuzzy cartesian product $r \times s$:

$$r \ast s = \{t | t = t_r t_s \text{ and } (t_r, t_s) \in r \times s\}$$

The membership value attached to the output fuzzy tuple could be evaluated using an appropriate $t$-norm.

6.3.1 The Fuzzy Relational Algebra as a Formal System

The Fuzzy Relational Algebra (FRA) is:

$$FRA = \{U, DOM, Mem, dom, fDb, fdb, Q, \Omega, O\}$$

$U$: the universe of attribute names.

$DOM = \{DOM(A_i) | i \in \{1, 2, \ldots, n\}\}$, the set of domains.

$Mem$: a collection of membership functions $\mu(A_i)$ over $DOM(A_i)$ wherever the domain-associated fuzziness is applicable.

$dom : U \rightarrow DOM$.

$fDb = \{U_j | 1 \leq j \leq m\}, U_j \subseteq U$ and $U = U_1 \cdots U_m$.

$fdb = \{r_j | 1 \leq j \leq m\}$, each $r_j$ is a fuzzy relation with scheme $U_j$.

$Q = \{\text{very, fairly, more or less, } \ldots\}$, list of linguistic modifiers.

$\Omega = \{=, \neq, <, \leq, >, \geq\}$, the set of arithmetic operators which can be fuzzified by elements of $Q$.

$O$: the fuzzy algebraic operators as defined above which can be performed on $DOM$, $\Omega$ could be arithmetic or order operations modified by $Q$.  

89
In the following sections, the theory of the fuzzy tuple relational calculus, fuzzy domain relational calculus will be developed. Then, it is shown that the fuzzy tuple calculus can be reduced to the fuzzy domain calculus, the fuzzy relational algebra can be reduced to the fuzzy tuple calculus and the fuzzy domain calculus to the fuzzy relational algebra.

### 6.4 The Fuzzy Tuple Relational Calculus

The fuzzy tuple calculus is defined as a formal system, then the evaluation of fuzzy tuple calculus expressions is discussed. Then, the safe fuzzy tuple calculus expression is defined.

#### The Elements of the Fuzzy Tuple Calculus Expressions

The fuzzy tuple relational calculus will be established as a formal system. The concept of a fuzzy tuple calculus expression over a fuzzy formula is introduced. So, a legal fuzzy formula is defined. As a start, the elements of the fuzzy tuple calculus are introduced.

**Definition 6.4.1:** The *fuzzy tuple relational calculus* or simply (fuzzy tuple calculus) is an 8-tuple \( fG_t = (U, \text{DOM}, \text{Mem}, \text{dom}, fDb, fdb, \Omega, Q) \):

- **U**: universe of attributes.
- **DOM** = \( \{\text{DOM}(A_i)|i \in \{1, 2, \ldots, n\}\} \), the set of domains.
- **Mem**: a collection of membership functions \( \mu(A_i) \) over \( \text{DOM}(A_i) \) wherever the domain-associated fuzziness is applicable.
- **dom** : \( U \rightarrow \text{FDOM} \).
- **fDb** = \( \{U_j|1 \leq j \leq m\}, U_j \subseteq U \) and \( U = U_1 \cdots U_m \).
- **fdb** = \( \{r_j|1 \leq j \leq m\} \), each \( r_j \) is a fuzzy relation with scheme \( U_j \).
- **Q** = \( \{\text{very, fairly, more or less,}\ldots\} \), list of linguistic modifiers.
- **\( \Omega \)** = \( \{=, \neq, <, \leq, >, \geq\} \), the set of arithmetic or order operators that can be fuzzified by elements from \( Q \).

In order to define the set of the legal fuzzy formulae, the definition of a fuzzy
atom is needed.

**Definition 6.4.2:** *Fuzzy atoms* in the fuzzy tuple calculus are defined as follows:

1. Truth values, denoted by *true* and *false*, or any other truth value in between are *fuzzy atoms*.

2. A fuzzy tuple variable $x$ belonging to a fuzzy relation $r_j(U_j)$, written as $r_j(x)$ is a *fuzzy atom* where $r_j$ is in a database $fdb$, and $r_j$ and $x$ have the scheme $U_j$.

3. $x(A_j)Qvy(A_k)$ is an atom where $x$ and $y$ are (not necessarily distinct) fuzzy tuple variables, $A_j$ and $A_k$ are (not necessarily distinct) compatible attributes in $U$, $\nu$ a comparison operator from $\Omega$ which could be modified with elements from $Q$.

4. $cQvx(A)$ and $x(A)Qvc$ are *fuzzy atoms* where $c$ is a constant in $DOM(A)$, $x(A)$ is the $A$-component of tuple variable $x$, and $\nu$ is in $\Omega$.

Now, the definition of legal fuzzy formulae is presented.

**Definition 6.4.3:** *Formulae, freedom* and *boundedness* of fuzzy tuple variables in fuzzy formulae are defined as follows:

1. Every fuzzy atom is a fuzzy formula and any tuple variable occurs in a fuzzy atom is free.

2. If $F$ is a fuzzy formula, then the negation of $F$, denoted by $\neg F$ is a formula. Any fuzzy tuple variable occurring in $\neg F$ is free or bound as it is free or bound in $F$.

3. If $F$ and $G$ are formulae, then their conjunction $F \land G$ and their disjunction $F \lor G$ are formulae. Any free variable in either of them or both is still free in either or both. If a variable is free in one of them and bound in the other it remains the same.
4. If \( x \) with scheme \( R \) is a free tuple variable occurring in a fuzzy formula \( F \), then \( \forall x(R)F(x) \), \( \exists x(R)F(x) \) are formulae. The quantified fuzzy tuple variable that is originally free in \( F \) becomes bound, any other tuple variable \( y \neq x \) occurring in \( F \) is free or bound as it is free or bound in \( F \).

5. Parentheses may be used as needed.

6. Nothing else is a fuzzy formula.

6.5 Fuzzy Tuple Calculus Expressions

In this section, a fuzzy tuple calculus expression and its evaluation are defined. In order to evaluate these expressions, an interpretation to fuzzy formulae over such a calculus must be given. First, the definition of a fuzzy tuple calculus expression is presented.

**Definition 6.5.1:** A fuzzy tuple calculus expression over the fuzzy tuple calculus \( fC_t \):

\[
E := \{ x(R) | F(x) \}, \text{ where:}
\]

1. \( F \) is a fuzzy legal formula relative to \( fC_t \).
2. \( x \) is the only free variable in \( F \).
3. \( R \) is a subset of \( U \) and is the scheme for \( x \), and

So, the expression is logical formula defined over a variable on a subset of the scheme of attributes. Now, the fuzzy tuple calculus expression is evaluated.

**Definition 6.5.2:** Evaluation of fuzzy tuple calculus expressions

Let \( F(x) \) be a legal fuzzy formula with the free tuple variable \( x \) in a fuzzy tuple calculus expression \( E_t := \{ x(R) | F(x) \} \). Then, \( F(x) \) with a fuzzy tuple \( t \) denoted by \( F(x \leftarrow t) \), is the fuzzy formula obtained by modifying each fuzzy atom in \( F \) involving a free occurrence of \( x \) as follows:
1. If $F(x) := r(x)$ where $r(x)$ is a fuzzy atom denoting a fuzzy tuple variable $x \in r$, then replace $r(x)$ by the atom true (or $\mu$ true) if $t \in \mu r$ or by the atom false if $t \not\in r$. (By $\in\mu$, i.e. belongs to the relation $r$ with degree of membership $\mu$).

2. If $F(x) := x(A_j)Qv_y(A_k)$ or $y(A_k)Qv_x(A_j)$ where the right side of := is a fuzzy atom with $x \neq y$, then $x(A_j)$ is replaced by the constant $c_1$ where $t(A_j) = c_1$ and $y(A_k)$ by $t(A_k) = c_2$ for some $c_1, c_2$ in $DOM(A_j), DOM(A_k)$.

3. If $F(x) := x(A_j)Qv_x(A_k)$, the entire fuzzy atom $x(A_j)Qv_x(A_k)$ is replaced by the fuzzy atom true (or $\mu$ degree of truth) $c_1Qv_c$ where $t(A_j) = c_1$ for some $c_1$ in $DOM(A_j)$ and $t(A_k) = c_2$ for some $c_2$ in $DOM(A_k)$ and by the atom false, otherwise.

4. If $F(x) := x(A)Qv_c$ or $cQv_x(A)$ where the right side of := is a fuzzy atom, then the entire fuzzy atom is replaced by the atom true (or $\mu$ degree of truth) if $c_1Qv_c$ or $cQv_c$ where $t(A) = c_1$ for some $c_1$ in $DOM(A)$, and by the atom false otherwise.

In order to avoid an infinite or a very large relation in the interpretation, the Active Fuzzy Domain is introduced and it includes the union of all the domain values of all the attributes at a given moment of time together with the associated fuzzy sets defined on them (i.e. their membership functions). Of course, by the insertions, deletions, this active domain changes with time. Considering the active domain is better than considering the whole domain of values as one will not have an infinite or a very large relation. So, it is clear that Active Domains are points in time of the original full-value domains.

Fuzzy formulae over fuzzy tuple calculus expressions can contain variables that are in the full-value domain and not in the active domain, the following definition of an Extended Active Fuzzy Domain captures this idea.
Definition 6.5.3: Let $F$ be a legal fuzzy tuple calculus formula in a fuzzy tuple calculus expression $fE_t := \{x(R)|F(x)\}$, and let $A$ be an attribute occurring in $F$. The extended active fuzzy domain or simply $EAFDOM(A, F)_x$, is the union of the active domain $AFDOM$ and the set of constants occurring in $F$ and $DOM(A)$ not in $AFDOM(A)$. For a set of attributes $R$, $EAFTUP(R, F)$ (or simply $EAFTUP(R)$ when the fuzzy formula $F$ is understood) is considered be the set of all possible tuples $t$ with scheme $R$ such that $t(A) \in EAFDOM(A, F)$ for each $A$ in $R$.

By using $EAFDOM(A, F)$ and $EAFTUP(R, F)$ to interpret a fuzzy formula $F$, it is possible to avoid an infinite or a very large relation. $EAFTUP$ is called the extended active fuzzy complex product.

The discussion is limited to active fuzzy domains and the fact that an attribute occurring in the fuzzy formulae may contain a constant that is not in the active domain, the extended active fuzzy domain is needed. Based on this, the limited interpretation of a fuzzy formula is defined.

Definition 6.5.4: The limited fuzzy interpretation of a closed fuzzy tuple calculus formula $F$ denoted by $i(F)$, is defined recursively as follows; by using only extended active fuzzy domains relative to the fuzzy formula $F$:

1. The limited fuzzy interpretation $i(F)$ is true to the degree $\mu$ if $F$ is true to the degree $\mu$.

2. If $F$ is $\neg G$, then $G$ must be a closed legal fuzzy formula. The limited fuzzy interpretation $i(F)$ is evaluated using an appropriate negation function (most commonly $\mu_F = 1 - \mu_G$).

3. If $F$ is $G \land H$ or $G \lor H$, then both $G$ and $H$ must be closed legal formulae, then $i(F)$ is evaluated using an appropriate $t$-norm or $t$-co-norm for conjunction and disjunction, respectively.

94
4. If $F$ is $\exists x(R)G$, $\forall x(R)G$, then $G$ must be a legal fuzzy formula, and $x$ must be the only fuzzy free tuple variable occurring in $G$. If $F$ is $\exists x(R)G$, then $i(F)$ is true (or $\mu$ degree of truth) if there is at least one tuple $t$ in $EAFTRP(R,F)$ such that $i(G(x \leftarrow t))$ is true (or $\mu$ degree of truth), and false otherwise. If $F$ is $\forall x(R)G$, then $i(F)$ is true (or $\mu$ degree of truth) if, for every tuple $t$ in $EAFTRP(R,F)$, $i(G(x \leftarrow t))$ is true (or $\mu$ degree of truth), false otherwise.

**Definition 6.5.5:** Let $fE_t := \{x(R)|F(x)\}$ be a fuzzy tuple calculus expression over the fuzzy tuple calculus $fC_t$. The value of $fE_t$ on the fuzzy database $fdb$ under the limited fuzzy interpretation denoted by $fE_t(fdb)$, is the fuzzy relation with scheme $R$ consisting of those fuzzy tuples $t$ in $EAFTRP(R,F)$ such that each fuzzy tuple $t$ satisfying $i(F \leftarrow t) = \mu$ ($\mu$ the degree of truth that $F$ reaches).

The limited fuzzy interpretation ensures that all relations derived or attempted for derivation are finite.

**Safety of Fuzzy Tuple Calculus Expressions**

**Definition 6.5.6:** A fuzzy tuple calculus expression is defined as safe if the following conditions are satisfied:

1. $i(F(x \leftarrow t)) = \mu$ degree of truth $\Rightarrow t \in \mu EAFTRP(R,F)$.

   The above implication indicates that the interpretation of the fuzzy formula $F$ is to be taken true to a degree of $\mu$ if $t$ is assigned to the variable $x$ in $F$, this means that $t$ belongs to the fuzzy extended active domain with a degree of membership of $\mu$. The same concept is used in the following.

2. For each subformula of $F(x)$ of the form $\exists y(S)G(y,z_1,\ldots,z_k \leftarrow t_k)$,

   $i(G(y \leftarrow t, z_1 \leftarrow t_1,\ldots,z_k \leftarrow t_k)) = \text{true to a degree } \mu \Rightarrow t \in \mu EAFTRP(S,G)$

3. For each subformula of $F(x)$ of the form $\forall y(S)G(y,z_1,\ldots,z_k)$. 
\( i(G(y \leftarrow t, z_1 \leftarrow t_1, \ldots, z_k \leftarrow t_k)) = \text{true to a degree } \mu \Rightarrow t = EAFTUP(S, G) \)

where \( \mu \) denotes degree of truth or degree of membership as appropriate.

The formulae without the limited interpretation may be considering a very large relation and maybe infinite one. Though this could not be the case in practice, this treatment makes the theory closer to practice, by considering the active domain as the database that is actually stored.

6.6 The Fuzzy Domain Relational Calculus

As a start, fuzzy atoms, the fuzzy domain calculus expressions and their interpretations are defined. The interpretations are limited to the fuzzy tuples in the extended active fuzzy domain. Those interpretations will be called limited. Then, the safeness of fuzzy domain calculus expressions is discussed.

Definition 6.6.1: Fuzzy atoms:

Fuzzy atoms for the fuzzy domain calculus are defined as follows:

1. If \( r_j(U_j) \) is a fuzzy relation in a fuzzy database, \( fd_b \), with scheme \( U_j = A_1A_2\ldots A_k \), then \( r_j(a_1, a_2, \ldots, a_k) \) is a fuzzy atom where each \( a_i \) for \( 1 \leq i \leq k \), is either a fuzzy domain variable with scheme \( A_j \) or a constant in \( DOM(A_k) \).

2. If \( a \) and \( b \) are fuzzy domain variables with the same scheme, \( \nu \) an arithmetic or order operator, \( Q \) a fuzzy modifier and \( c \) a constant in \( DOM(U) \) whose scheme is identical to that of \( a \), then \( aQvb, aQvc \) and \( cQva \) are all fuzzy atoms.

3. The truth values, \( \text{true, false} \) and \( \mu \in [0, 1] \) are all fuzzy atoms.

4. Nothing else is a fuzzy atom.

Now, the definition of a fuzzy domain calculus expression is introduced.

Definition 6.6.2: A fuzzy domain calculus expression \( fE_d \) over \( fC_d \) has the form:

96
\[ f E_d := \{a_1(A_1) \ldots a_k(A_k)\} F(a_1, \ldots, a_k), \text{ where:} \]

1. \( F \) is a legal fuzzy domain calculus formula with free domain variables 
   \( a_1, a_2, \ldots, a_k \).

2. \( A_1, A_2, \ldots, A_k \) are attributes in \( U \) and

So, the expression here has a set of domain variables, cf. def. 6.5.1, then see how this affects the first reduction theorem (theorem 6.7.1). The value of a fuzzy domain calculus expression under a limited fuzzy interpretation is the fuzzy relation over the scheme \( A_1 \ldots A_k \) of those fuzzy tuples of the form \( c_1 \ldots c_k \) such that \( c_j \in EAFDOM(A_j, F) \) for each \( j \) such that \( 1 \leq j \leq k \) and

\[
i(F(a_1 \leftarrow c_1, a_2 \leftarrow c_2, \ldots, a_k \leftarrow c_k)) = \mu
\]

where \( \mu \) is a specific truth value which is the membership degree of the tuple in the table.

**Definition 6.6.3:** Safeness of fuzzy domain calculus expressions:

A fuzzy domain relational calculus is **safe** if the following conditions are satisfied:

1. For constants \( c_1, c_2, \ldots, c_k \) the implication:

\[
i(F(a_1 \leftarrow c_1, \ldots, a_k \leftarrow c_k)) = \mu \Rightarrow c_j \in_\mu EAFDOM
\]

\( \mu L.H.S. \) : degree of truth.

\( \mu R.H.S. \) : degree of membership.

The above implication indicates that the interpretation of the fuzzy formula \( F \) is to be taken true to a degree of \( \mu \) if \( c_j \) is assigned to the free variable \( a_i \) in \( F \), then this means that \( c_j \) belongs to the extended active fuzzy domain with a degree of membership of \( \mu \). The same concept is used in the following.
2. For each fuzzy subformula $F$ of the form $\exists a(A_j)G(a)$, the implication:

\[ i(G(a \leftarrow c)) = \mu \Rightarrow c \in \mu EAFDOM(A_j, G) \]

3. For each subformula of the form:

$\forall a(a_j)G(a)$:

\[ c = EAFDOM(A_j, G) \]

6.7 Reduction of Fuzzy Tuple Calculus to Fuzzy Domain Calculus

The following theorem establishes the equivalence between the fuzzy tuple calculus and the fuzzy domain calculus.

**Theorem 6.7.1:** If $fE_t$ is a fuzzy tuple calculus expression, then there is an expression $fE_d$ in the fuzzy domain calculus equivalent to $fE_t$, that is for a fuzzy database $f\text{db}$, $fE_d(f\text{db}) = fE_t(f\text{db})$.

**Proof:** Any fuzzy tuple calculus expression can be converted into a fuzzy domain calculus expression recursively as follows:

1. Any fuzzy atom $r(x)$ in $F$ is replaced by $r(a_1, a_2, \ldots, a_k)$ where $a_j$ is the domain variable.

2. Any fuzzy atom $x(A_i)Q\nu c$ or $cQ\nu x(A_j)$ for $c$ being a component of another tuple variable or a constant is replaced by $a_jQ\nu c$ or $cQ\nu a_j$ where $a_j$ is the domain variable denoting the $A$-component of the tuple variable $x$.

3. Any atom $x(A_i)Q\nu y(B_j)$ is replaced by $a_iQ\nu b_j$.

4. A quantified subformula $\exists x(S)G$ is replaced by $\exists a_1(A_1)\exists a_2(A_2) \ldots \exists a_k(A_k)G$.

5. A quantified subformula $\forall x(S)G$ is replaced by $\forall a_1(A_1)\forall a_2(A_2) \ldots \forall a_k(A_k)G$. 

98
6.8 Reduction of Fuzzy Relational Algebra to Fuzzy Tuple Calculus

It will be shown that for any fuzzy algebraic expression over the fuzzy relational algebra, there is an equivalent fuzzy tuple calculus expression over the fuzzy tuple calculus.

**Theorem 6.8.1:** If \( fE_a \) is a fuzzy algebraic expression over the fuzzy relational algebra, then there is an expression \( fE_t \) in the fuzzy tuple calculus equivalent to \( fE_a \), that is for a fuzzy database \( fdb \), \( fE_a(fdb) = fE_t(fdb) \).

**Proof:** The proof proceeds by induction on the number of occurrences of operators in \( fE_a \).

**Basis:** No operators. Then \( fE_a \) is a constant fuzzy relation, then:

\[
fE_t := \{ x(R) | [x(R) = t_1(R)] + [x(R) = t_2(R)] + \cdots + [x(R) = t_p(R)] \}
\]

**Induction:** Suppose the theorem holds for any fuzzy algebraic expression with fewer than \( k \) operators. There are five cases to consider:

1. **Fuzzy Union** (analogous to fuzzy disjunction)

\[
fE_a := fE_{a1} \cup fE_{a2}
\]

By the induction hypothesis, these two fuzzy algebraic expressions have equivalent fuzzy tuple calculus expressions \( \{x(R)|G(x)\} \) and \( \{y(R)|H(y)\} \), then \( fE_a \) is equivalent to:

\[
fE_t := \{ z(R)|G(z) \vee H(z) \}
\]

2. **Fuzzy Difference**

The fuzzy difference would be analog to the fuzzy logical negation so long as the same negation function is used throughout:

\[
fE_a := fE_{a1} - fE_{a2}
\]

\[
fE_t := \{ z(R)|G(z) \land \neg H(z) \}
\]

where \( G \) and \( H \) are as above and \( \neg H(z) \) is the fuzzy negation of \( H(z) \).
3. Fuzzy Complex Product

\[ f_{E_a} := f_{E_{a1}} \ast f_{E_{a2}} \]

By the induction hypothesis \( f_{E_{a1}} \) is equivalent to \( \{x(R)|G(x)\} \) and \( f_{E_{a2}} \) is equivalent to \( \{y(S)|H(y)\} \), then \( f_{E_a} \) will be equivalent to:

\[ f_{E_t} := \{z(RS)|\exists x(R)\exists y(S) [G(x) \land H(y)] \land [z(R) = x(R)] \land [z(S) = y(S)]\} \]

4. Fuzzy Projection

\[ f_{E_a} := \Pi_{A_1 A_2 \ldots A_i} (r) \]

\[ f_{E_t} := \{z(A_1 \ldots A_i)|\exists x(R) \left[ r(x) \land [z(A_1) = x(A_1)] \ldots [z(A_i) = x(A_i)] \right]\} \]

5. Fuzzy Selection

\[ f_{E_a} := \sigma_F (r) \]

\[ f_{E_t} := \{z(R)|r(z) \land G\} \]

where \( G \) is the fuzzy formula obtained from \( F \) by replacing each attribute \( A_i \) occurring in \( F \) by \( z(a_i) \), i.e. the \( A_i \)-component of the fuzzy tuple variable \( z \).

6.9 Reduction of Fuzzy Domain Calculus to Fuzzy Relational Algebra

**Theorem 6.9.1:** Let \( f_{E_d} \) be a fuzzy domain calculus expression, there is an algebraic expression \( f_{E_a} \) equivalent to \( f_{E_d} \). That is \( f_{E_d}(f_{db}) = f_{E_a}(f_{db}) \).

**Proof:** \( f_{E_d} \) is of the form:

\[ f_{E_d} := \{a_1(A_1)a_2(A_2)\ldots a_n(A_n)|F(a_1, a_2, \ldots, a_n)\} \]

where \( F \) may have a number of subformulae say \( G_1, G_2 \). Then for a subformula \( G \) of \( F \) the fuzzy domain relational calculus expression would be:

\[ f_{E_d} := \{b_1(B_1)b_2(B_2)\ldots b_n(B_m)|G(b_1, b_2, \ldots, b_m)\} \]
Now, although the \( a_{v,s} \) are distinct and so are the \( b_{v,s} \), the \( A_{v,s} \) and the \( B_{v,s} \) need not be. The following recursive cases are considered for every fuzzy subformula in \( F \) above. It will be shown that each one of them will have an equivalent algebraic expression:

1. The fuzzy subformula \( G \) is an atom of the for \( aQvb \), \( aQvc \) or \( cQva \) where

\[
f_{E_d} := \{a(A)b(B)|aQvb\}
\]

\[
f_{E_{a}} := \sigma_{A_{Qvb}A}| \times |B, a \neq b
\]

\[
f_{E_{a}} := \sigma_{A_{Qvb}A}, a = b
\]

and if \( c \) is a constant

\[
f_{E_{a}} := \sigma_{A_{Qvc}A}
\]

2. The fuzzy subformula \( G \) is \( r(a_1, a_2, \ldots, a_m) \)

\[
f_{E_{a}} := \Pi_{A_{1}A_{2}A_{m}} \sigma_{a_{i} = A_{i}}(r)
\]

3. The fuzzy subformula \( G \) is of the form \( G := \neg H \), then \( f_{E_{g}} := \neg f_{E_{h}} \)

Given that \( EAFTUP \) is the extended active fuzzy complex product containing all the possible fuzzy tuples at a given moment of time, then if the fuzzy relational algebraic expression corresponding to \( f_{E_{h}} \) is a fuzzy relation \( R \), the fuzzy relational algebraic expression corresponding to \( f_{E_{g}} \) would be the complement of \( R \) with respect to \( EAFTUP \).

4. The fuzzy subformula \( G \) of the form:

\[
G := H_1 \land H_2
\]

then one gets:

\[
f_{E_{a_{g}}} := f_{E_{H_{1}}} \cap f_{E_{H_{2}}}
\]
5. The fuzzy subformula $G$ is of the form:

$$G := H_1 \vee H_2$$

$$fE_{ag} := fE_{H_1} \cup fE_{H_2}$$

6. The fuzzy subformula $G$ is of the form $\exists a(A)H$

$$fE_{ag} := \Pi_{sch(fE_h) \rightarrow sch(a)}(fE_h)$$

where $fE_h$ is a fuzzy algebraic expression equivalent to fuzzy domain calculus expression corresponding to the fuzzy subformula $H$.

7. The fuzzy subformula $G$ is of the form

$$G := \forall a(A)H$$

$fE_g$ is obtained from $fE_h$ via appending all instances of the fuzzy domain variable $a$ found in EAFTUP to the fuzzy relation $fE_h$.

6.10 Examples

Example 6.10.1: Given the following two FRIL relations [17]:

Population(City, Inhabitants), Near(City1, City2)

(Population

(Gloucester 300000)
(Cam 6000)
(Bristol 550000)
(Bath 200000)
('Bradford-on-Avon' 20000)
(Avonmouth 350000)
(Almondsbury 80000) )

(Near
<table>
<thead>
<tr>
<th>Location Comparison</th>
<th>Similarity</th>
</tr>
</thead>
<tbody>
<tr>
<td>('Bradford-on-Avon' 'Bradford-on-Avon')</td>
<td>1.0</td>
</tr>
<tr>
<td>(Cam 'Bradford-on-Avon')</td>
<td>0.1</td>
</tr>
<tr>
<td>(Bristol 'Bradford-on-Avon')</td>
<td>0.7</td>
</tr>
<tr>
<td>(Gloucester 'Bradford-on-Avon')</td>
<td>0.1</td>
</tr>
<tr>
<td>('Bradford-on-Avon' Gloucester)</td>
<td>0.1</td>
</tr>
<tr>
<td>('Bradford-on-Avon' Avonmouth)</td>
<td>0.4</td>
</tr>
<tr>
<td>('Bradford-on-Avon' Almondsbury)</td>
<td>0.6</td>
</tr>
<tr>
<td>('Bradford-on-Avon' Bristol)</td>
<td>0.7</td>
</tr>
<tr>
<td>(Cam Gloucester)</td>
<td>0.7</td>
</tr>
<tr>
<td>(Cam Avonmouth)</td>
<td>0.1</td>
</tr>
<tr>
<td>(Cam Almondsbury)</td>
<td>0.5</td>
</tr>
<tr>
<td>(Cam Bristol)</td>
<td>0.4</td>
</tr>
<tr>
<td>(Bristol Bristol)</td>
<td>1.0</td>
</tr>
<tr>
<td>(Bristol Gloucester)</td>
<td>0.2</td>
</tr>
<tr>
<td>(Bristol Avonmouth)</td>
<td>0.7</td>
</tr>
<tr>
<td>(Bristol Almondsbury)</td>
<td>0.9</td>
</tr>
<tr>
<td>(Gloucester Gloucester)</td>
<td>1.0</td>
</tr>
<tr>
<td>(Gloucester Avonmouth)</td>
<td>0.1</td>
</tr>
<tr>
<td>(Gloucester Almondsbury)</td>
<td>0.3</td>
</tr>
<tr>
<td>(Gloucester Bristol)</td>
<td>0.2</td>
</tr>
<tr>
<td>('Bradford-on-Avon' Cam)</td>
<td>0.1</td>
</tr>
<tr>
<td>('Bradford-on-Avon' Bath)</td>
<td>0.9</td>
</tr>
<tr>
<td>(Cam Cam)</td>
<td>1.0</td>
</tr>
<tr>
<td>(Cam Bath)</td>
<td>0.3</td>
</tr>
<tr>
<td>(Bristol Cam)</td>
<td>0.4</td>
</tr>
<tr>
<td>(Bristol Bath)</td>
<td>0.8</td>
</tr>
<tr>
<td>(Gloucester Cam)</td>
<td>0.7</td>
</tr>
<tr>
<td>(Gloucester Bath)</td>
<td>0.2</td>
</tr>
</tbody>
</table>
The solution of the following query in this chapter framework is presented. The query is: "Which towns have small populations and are near towns with large populations". The solution follows the following steps.

1. Evaluate the fuzzy relation $R_1 = \text{SmallPopulation}(\text{City}, \text{Inhabitants}, \text{Degree})$, using the fuzzy set Small defined on the domain of the attribute Inhabitants. Further, a cut operations happens at a certain level and a
classical relation is retained. In the Fuzzy Relational Algebra presented

\[ R_1 = \sigma_{\text{Inhabitants}=\text{Small}}(\text{Population}) \]

2. Evaluate the fuzzy relation \( R_2 = \text{LargePopulation}(\text{City}, \text{Inhabitants}, \text{Degree}) \), using the fuzzy set \text{Large} defined on the domain of the attribute \text{Inhabitants}. The definition of the two fuzzy sets \text{Small}, \text{Large} illustrates \textit{domain-associated fuzziness}. The resultant degrees of membership (third column) illustrate \textit{tuple-associated fuzziness}. A similar cut operation is performed to retain a classical relation. (Note that cuts are not necessarily needed, they are included for consistency with the earlier systems of FLP1, FLP2 and FLP3). \[
R_2 = \sigma_{\text{Inhabitants}=\text{Large}}(\text{Population})
\]

3. The required result of the query is the fuzzy relation: \text{near}(R_1, R_2).

**Example 6.10.2:** The previous fuzzy relational algebraic expression:

\[ fE_a = R_1 = \sigma_{\text{Inhabitants}=\text{Small}}(\text{Population}) \]

The equivalent fuzzy tuple calculus expression is:

\[ fE_t := \{ z(R) | r(z) \land G \} \]

where \( r(z) = \text{Population}(z) \), \( G \) is the fuzzy formula \( \text{Inhabitants}=\text{Small} \)

**Example 6.10.3:** For the previous fuzzy tuple calculus expression:

\[ fE_t := \{ z(R) | r(z) \land G \} \]

the equivalent fuzzy domain calculus expression is:

\[ fE_d := \{ a(\text{City})b(\text{Inhabitants}) | r(\text{City}, \text{Inhabitants}) \land (b = \text{small}) \} \]
Example 6.10.4: For the previous fuzzy domain calculus expression:

\[ f_{Ed} := \{a(City)b(Inhabitants)\} | r(City, Inhabitants) \land (b = small) \]

the equivalent fuzzy algebraic expression is (using theorem 6.9.1 - item 1):

\[ f_{Ea} = \sigma_{\text{inhabitants=small}}(\text{Population}) \]

Note that \( f_{Ea} = R_2 \) of example 6.10.1, thus confirming all the equivalence theorems.
Chapter 7

Fuzzy Deductive Databases

Following the previous chapters research in the areas of fuzzy logic programming and fuzzy databases. The two popular views of deductive databases are extended to fuzzy deductive databases. The proof-theoretic view [chapters 3-5] and the model-theoretic view [chapter 6] are presented via two schemes. One is of Coupling fuzzy logic programming systems to fuzzy relational databases, the other is of Integration of both systems. Several algorithms are developed in this regard.

7.1 Background

When one speaks of fuzzy deductive databases, that does not only mean the product of the merger of two new technologies i.e. fuzzy relational databases and fuzzy logic programming but it is also meant the semantic framework covering both of them; essentially that of fuzzy logic.

In this chapter, it will be demonstrated that fuzzy logic and fuzzy set theory can capture more semantics of real-life situations than traditional tools. Fuzzy logic and fuzzy set theory can capture uncertainty in data as seen in the fuzzy relational model [chapter 6]. It can also capture uncertainty and imprecision in rules as shown in fuzzy logic programming [chapters 3-5]. Inference rules exist by which inference can be undertaken if none of the rule hypothesis exactly match the situation at hand. This was done by introducing the notion of pseudo-complementarity [chapter 4].
Here, it will be clear that fuzzy logic and fuzzy set theory can have a great rôle in the design and modelling of databases. Their use can lead to much better systems. Apart from the above issues, all thinking (when trying to perceive a real world state) is done in natural language which is full of uncertainty and imprecision. One can argue that many adjectives and predicative expressions are linguistic labels denoting fuzzy sets. The denotation function is essentially a possibility distribution. Also, one can follow up the argument and say that noun and noun phrases which denote entities that could be fuzzy. In other words, there is no sharp boundary to tell whether a given instance is a member of the class denoted by the entity or not. Verbs denote relationships which definitely may be fuzzy. So, one can say that since fuzzy set theory is much more closer to natural language (by which the perception, selection and thinking) than classical set theory, then it must be a better modelling tool for the construction of better databases. In the following, the two popular views for fuzzy databases are presented. These views are standard for classical deductive databases. This view is enhanced in the case of fuzzy deductive databases.

7.2 Fuzzy Databases: Two Views
There are two common views of databases: the model-theoretic view and proof-theoretic view. In the model-theoretic view, a database is a model of its integrity constraints and an answer to a query should make the query true in the model given by the data in the database.

In the proof-theoretic view, the database is a first-order theory (in fuzzy logic w.r.t. fuzzy deductive databases - FLP1, FLP2 and FLP3). Answering a query involves proving the query to be a logical consequence of the database. In dealing with fuzzy deductive databases, the proof-theoretic view presented in [chapters 3-5] is considered, as the model-theoretic view had been dealt with in [chapter 6].

In the proof-theoretic view of fuzzy databases, three systems of fuzzy logic were developed (subsuming a fragment of first-order logic). Fuzzy databases can
be seen as models of their soft integrity constraints (fuzzy logic formulae) and answering a query could be finding out a certain level of truth value.

Fuzzy deductive databases can use either FLP1 [chapter 3] or FLP2 [chapter 4] with their inference rules as discussed earlier.

The correspondence between similar concepts of fuzzy relational databases and fuzzy logic programming can be seen through the following:

1. The concept of a fuzzy relation corresponds to the concept of a fuzzy predicate.

2. The concept of an attribute in fuzzy relational databases corresponds to the concept of a predicate argument in FLP1/FLP2.

3. The concept of a fuzzy tuple in fuzzy relational databases corresponds to the concept of a fuzzy fact.

4. The concept of a fuzzy view in fuzzy relational databases corresponds to the concept of a fuzzy rule in fuzzy logic programming. As a rule is corresponding to a view in classical deductive databases.

5. The concept of a fuzzy query in fuzzy relational databases corresponds to the concept of a fuzzy goal in fuzzy logic programming.

6. The concept of a soft constraint in fuzzy relational databases corresponds to the concept of a fuzzy rule to be enforced over the facts.

In the following, two approaches for Fuzzy Deductive Databases will be presented. The first is coupling fuzzy logic programming with fuzzy relational databases and the second is the integration of both.

1. The Coupling Approach: An interface between two existing systems to provide a single system image. Both systems preserve their individuality. The interface provides procedures required for bringing data from the persistent
database system into the logic programming execution environment in order to evaluate queries or validate constraints (Ceri et al. [29]). Several research prototypes used this approach (in the classical non-fuzzy sense): PRO-SQL, EDUCE, ESTEAM, BERMUDA, CGW and Quintus-Prolog (Ceri et al. [29]).

2. **The Integration Approach**: A single system is developed for providing fuzzy logic programming on top of a mass-memory fuzzy database system. This requires building new data structures and algorithms specifically for this purpose.

### 7.3 The Coupling Approach

The coupling approach is easier to achieve but is much less efficient than the integration approach. Coupling systems use existing Prolog systems coupled to relational databases, whereas integrated systems use Datalog which is an evolution of Prolog built specifically to act as a database language. It uses breadth-first instead of depth-first, thus providing set-oriented query answers instead of the tuple-at-a-time of Prolog. Also, it is not sensitive to the order of the rules in the logic program. Furthermore, it does not have special predicates needed by the Prolog programmer to control the execution environment (like the cut), neither does it have function symbols. Within the coupling approach, the interface could have various degrees of sophistication, two cases are distinguished, Ceri et al. [29]:

1. **Loose Coupling**: The interaction between the fuzzy logic programming and fuzzy database environments takes place at compile-time or at load-time (with interpreters), i.e. all facts are extracted from the database before the activation of the rule. That is why it is sometimes call *static coupling*.

2. **Tight Coupling**: Where the interaction between the logic programming and
the database system takes place in the frame of execution of each rule, that is why it is called *dynamic coupling* (Ceri et al. [29]).

**F-Prolog and FSQL**

In both approaches to coupling, the interaction between the two systems is mostly through base conjunctions. A base conjunction is a sequence of database predicates and arithmetic predicates. It can be a part of a Prolog rule or through the composition of Prolog rules, e.g. Ceri et al. [29]:

\[ bc_1 : db_1(X_1, a, X_3), db_2(X_3, X_4, X_5), (X_4 = b), db_3(X_5, X_6, X_7), (X_6 = c) \]

Each base conjunction corresponds to a join query that is to be evaluated on the database. One can propose a formulation that presents a logical canonical form in which queries are transformed into logic expressions and then evaluated in the F-Prolog inference system. The author agrees that this could be useful but the conversion from logic to algebraic formalism allows one to use the various classical optimization techniques and well-established results, thus leading to a more efficient system. Ceri et al. [29] presents an algorithm for translating Datalog goals into positive relational algebra (a subset of relational algebra that does not contain difference). The algorithm could be used in conjunction with coupling or with the integration approaches. A *fuzzy base conjunction* is a sequence of fuzzy database predicates and fuzzy arithmetic predicates:

\[ fb_{bc_1} : db_1(X_1, a, X_3, \mu_1), db_2(X_3, X_4, X_5, \mu_2), (X_4 \simeq b), db_3(X_5, X_6, X_7, \mu_3), (X_6 \simeq c) \]

In determining fuzzy base conjunctions, truth values are associated within fuzzy atoms. Pseudo-complementarity maybe taken into consideration. In arithmetic predicates, “equal” is replaced by “approximately equal”.

**7.4 The Integration Approach**

The integration approach for combining fuzzy relational databases and fuzzy logic programming proposes building new algorithms and data structures to serve the
purposes of these new systems. The coupling approach only provides an interface between the two systems.

When discussing fuzzy logic programming, one can see a fuzzy deductive database as a collection of fuzzy rules and fuzzy facts as developed in earlier chapters [3-5]. Yet, for database purposes, FuzzyDatalog is much better than FuzzyProlog. The difference is the same as between Datalog and Prolog which is Datalog goals are computed using the breadth-first strategy which produces the set of all answers, rather than with the depth-first strategy which produces answers with a tuple-at-a-time approach. The following algorithms provide a basis for an implementation of FuzzyDatalog. As a start, the basic notions of a fuzzy deductive database are defined.

**Definition 7.4.1:** A fuzzy deductive database is a quintuple:

\[ FDB = \langle DOM, Mem, \sim, P, I \rangle \]

where:

- **DOM**: Each \( DOM_i \) represents a domain of values in a fuzzy relation in the database.

- **Mem**: Each \( Mem_{(i,j)}(A) \) is the fuzzy membership function for the \( j^{th} \) fuzzy set which is defined over the \( i^{th} \) domain. \( Mem_{(i,j)}(A) \) is a function from \( DOM_i \) to the interval \([0, 1]\), i.e. \( Mem_{(i,j)}(A) : DOM_i \rightarrow [0, 1] \). It is to be noted that \( Mem \) is not necessarily one-to-one nor it is necessarily onto. An example of \( A \) could be “Young”, “Old” over the domain of the variable \( \text{Age} \).

- \( \sim_i \) is the “approximately equal” relationship defined on the \( i^{th} \) domain, i.e. \( \sim_i : DOM_i \rightarrow DOM_i \)

- **P**: A finite set of axioms expressible in the language or meta-rules not expressible in the language.

- **I**: a finite set of fuzzy formulae expressing soft integrity constraints. The set \( I \) represents important information about a database.
The main concern of this section is the set $P$ which forms the axioms defining what constitutes a fuzzy database statement and how the database could be queried. In classical databases, there is a set of basic axioms: the domain closure axiom, the unique name axiom, and the equality axiom. These axioms assume that the database is finite, all constants can be named, different constant symbols stand for different objects and equality stands for the identity relation.

In addition to the axioms in classical deductive databases, in the case of fuzzy deductive databases, the following additional axioms must be included, concerning $\text{Mem}$ and $\simeq$:

1. $\text{Mem}_{(i,j)}(A) \in [0, 1]$
2. $\text{Mem}_{(i,j)}(A) \cup \text{Mem}^c_{(i,j)}(A) = 1 - \text{Mem}_{(i,j)}(A) \cap \text{Mem}^c_{(i,j)}(A)$
3. $\text{Mem}_{(i,j)}(A) \cap \text{Mem}^c_{(i,j)}(A) = 1 - \text{Mem}_{(i,j)}(A) \cup \text{Mem}^c_{(i,j)}(A)$
   \hspace{1cm} where $\text{Mem}^c$ is the fuzzy complement of $\text{Mem}$.
4. $\simeq (x, x)$: "approximately equal" is reflexive.
5. $\simeq (x, y) \Rightarrow \simeq (y, x)$: "approximately equal" is symmetric.
6. $\simeq (x, y) \land \simeq (y, z) \Rightarrow \simeq (x, z)$: this is not always true as "approximately equal" may or may not be transitive.

Two different cases are distinguished:

1. A fuzzy relational database: In this case, $P$ consists of ground atomic formulae only.

   Example 7.4.1: $C$ contains the constants $a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8$, the unary predicate symbol $R_1$, the binary predicate symbol $R_2$ and the ternary predicate symbol $R_3$.

   \hspace{1cm} $p_1 : < R_1(a_1), \mu_{p_1} >$
p_2 : < R_1(a_2), \mu_{p_2} > \leftarrow \\
p_3 : < R_2(a_1, a_3), \mu_{p_3} > \leftarrow \\
p_4 : < R_2(a_2, a_4), \mu_{p_4} > \leftarrow \\
p_5 : < R_3(a_1, a_5, a_7), \mu_{p_5} > \leftarrow \\
p_6 : < R_3(a_2, a_3, a_4), \mu_{p_6} > \leftarrow \\
p_7 : < R_3(a_8, a_1, a_7), \mu_{p_7} > \leftarrow \\

The following queries are considered:

Consider the goal \iff R_1(x), 0.8 > and \mu \in [0, 1]. In this example, \mu = 0.8, so there are two answers x = a_1, x = a_2 assuming \mu_{p_1} = 0.85 and \mu_{p_2} = 0.8. Clearly, the answer is \mu_{P}-dependent.

If given the query \iff R_1(a_3), \mu >. The answer is “No” and in this case the fuzzy relational database handles the situation as the non-fuzzy relational database since the query represents a fact which is not a part of the database at all. The closed world assumption can be restated in fuzzy terminology. The \textit{closed world assumption} states that one infers \neg A if one cannot prove A, i.e. A is not in \text{T} (the database theory). It can be restated in fuzzy terminology as \text{one infers} \neg A \text{ iff the degree of truth of } A \text{ is 0.}

2. \textit{A Fuzzy Horn Database:} In a fuzzy Horn database, P consists of formulae of the form:

\[ B \leftarrow A_1, \ldots, A_n \]

where the \text{A}_i's and \text{B} are all atoms. The examples on the fuzzy logic programming chapters are of this type, another recursive example is considered:

\textbf{Example 7.4.2:}

\[ p_1 : < R_1(a_1, a_3), \mu_{p_1} > \leftarrow \]
Given the query: $\leftarrow <R_2(x, y), 0.7 >$, the answer will be $<a_1, a_3 >$, $<a_3, a_4 >, <a_4, a_4 >, <a_4, a_6 >, <a_4, a_6 >, <a_4, a_6 >$. Matching first with $p_4$ then $p_1, p_2$ and $p_3$ then by recursion using $p_5$.

Assuming that $\mu_{p_1}, \mu_{p_2}, \mu_{p_3}, \mu_{R_1}, \mu_{R_2}$ are all $\geq 0.7$. The declarative, fix-point and procedural semantics and their equivalence were discussed in the chapters of fuzzy logic programming.

The algorithm **FuzzyPRODUCE** below produces the ground fuzzy facts resulting from applying a fuzzy rule to a fuzzy fact. The algorithm **FuzzyINFER1** provides all the facts that can be inferred in one step from a set of fuzzy rules by applying every rule to all ground fuzzy facts. The algorithm **FuzzyINFER** computes the set of all ground fuzzy facts which follow from a set $S$ of Fuzzy-Datalog clauses. These algorithms are modifications over the classical non-fuzzy case in Ceri et al. [29].

**Function FuzzyPRODUCE**($R, F_1, \ldots, F_n$)

*Input:* A FuzzyDatalog rule $R$ of the form $L_o \leftarrow (f) - L_1, \ldots, L_n$

and a list of fuzzy facts $F_1, \ldots, F_n$.

*Output:* The ground facts resulting from applying the fuzzy rule to the facts; the dummy symbol $\nabla$ otherwise.

BEGIN
FOR $i := 0$ TO $n$ DO $K_i := L_i$

FOR $i := 1$ TO $n$ DO
BEGIN
    $\lambda := \text{FuzzyUnifier}(K_i, F_i)$;
    IF $\lambda = \emptyset$ THEN RETURN $\emptyset$;
    ELSE FOR $j := 0$ TO $n$ DO $K_j := K_j \lambda$
END
RETURN $K_o$ with truth value $\mu_{K_o} * f$
END.

Note that the inner loop of FuzzyProduce is from 0 to $n$, while the outer loop is from 1 to $n$ as there is no $F_0$. The FuzzyUnifier algorithm is the same classical unification algorithm but modified to consider approximately equal constants as equal. The algorithms of Fontana [56] can be used instead.

Function FuzzyINFER1(S)
Input: A finite set of $S$ of FuzzyDatalog clauses.
Output: The set of all facts which can be inferred in one step from $S$ by applying every rule to all ground fuzzy facts.
BEGIN
result:= $\emptyset$
FOR each rule $R : L_o \leftarrow (f) - L_1, L_2, \ldots, L_n$ of $S$ DO
    FOR each n-tuple $< F_1, F_2, \ldots, F_n >$ of ground fuzzy facts of $S$ DO
        BEGIN
            new:= FuzzyPRODUCE($R, F_1, F_2, \ldots, F_n$);
            IF new $\neq \emptyset$ THEN result:=result $\cup \{ \text{new} \}$
        END;
RETURN result
END.
Function FuzzyINFER(S)

*Input:* A finite set $S$ of FuzzyDatalog clauses

*Output:* The set of all ground facts which follow from a set $S$ of FuzzyDatalog clauses.

BEGIN
old := ∅
new := S;
WHILE new ≠ old DO
    BEGIN
        old := new;
        new := new ∪ FuzzyINFER1(new)
    END;
    result := all facts of new; RETURN result
END.

7.5 Negation in Fuzzy Deductive Databases

In classical deductive databases, negation was usually treated in three ways Minker [117]:

1. Closed World Assumption, CWA.

2. Completed Database, CDB.

3. Negation as Finite Failure, NFF.

Negation in fuzzy logic programming in [chapter 3] was discussed and it was demonstrated that by just simple transformations how fuzzy logic offers an attractive solution to the problem of negation. Earlier in this chapter (the fuzzy relational database example), the closed world assumption was restated in fuzzy terminology. The membership function could be seen as a concrete representation
for it. The completed database is used in a fuzzy deductive database only when
negation is in the body of a rule.

7.6 Implementation Issues
In this section, various implementation issues of fuzzy relational databases are
discussed. One should have a modular approach towards the implementation of
these systems. In this modular architecture, the following is to be noted:

1. A fuzziness layer over existing relational database systems. This layer can
   have various degrees of sophistication and can be built as an integral part
   of the RDBMS or as an add-on to it.

2. A fuzziness layer over existing Prolog languages. The degree of sophistica-
   tion of this layer may vary as well.

3. Coupling & Integration of both of the above layers into fuzzy deductive
databases.

Issues in Fuzzy Relational Databases
Implementation issues regarding the tuple-associated fuzziness, domain-associated
fuzziness and t-norms are discussed. In an RDBMS, these implementation issues
should reflect on the DDL (Data Definition Language) and the DML (Data Ma-
nipulation Language) provided by the system.

Tuple-associated Fuzziness: The key-component of the fuzzy relational model
is the fuzzy relation. The fuzzy relation can be implemented in the classical
relational model. By adding a column to a classical relation that represents
the degree of compatibility between the object denoted by the tuple and the
concept denoted by the relation. So, the fuzzy relation is a table exactly like
the classical relation. The point is that it has an additional column which has
a special interpretation. This means that the database designer can use existing
database systems to design his fuzzy database. The tuple-associated fuzziness
which is represented by a value associated with each tuple could be a numerical value $\in [0,1]$.

**Domain-associated Fuzziness:** The domain-associated fuzziness is not handled in the same straightforward manner as the tuple-associated one. The domain-associated fuzziness is represented by the similarity relation that represents to what extent two elements of the domain could be considered interchangeable or fuzzy sets defined over the domain. In the case of finite values for domains, similarity relations could be represented by matrices, where for instance $a[i,j]$ could mean the degree to which the two elements of the domain could be seen as similar. For the case of countably infinite domains, the similarity relation must be represented in another way unlike the matrices that enumerate the whole domain. An analytic function representation could be the most suitable in this case. Since similarity relations are data about data, they can be considered as a *fuzzy extension to the data dictionary.*

**T-norms:** As discussed earlier, all the fuzzy notions introduced will reflect a change on the DDL and the DML provided by the RDBMS. For instance, the `CREATE TABLE` command should add a column to allow for the tuple-associated fuzziness. Furthermore, there should be an indication whether this truth value will be entered by the user or will be computed via a t-norm that is provided by the system. The database designer may choose either ways or one as a default that could be overridden.

**Fuzzy Comparisons:** One of the most crucial implementation issues of a DML whether it is algebraic-based or calculus-based is the issue of fuzzy comparisons. The concept of the semantic distance between two fuzzy sets can be used to model the way one makes fuzzy comparison.

$$\text{Identical}(A, B) = \mu$$

where $\mu = 1 - S_d(A, B)$ (Li et al. [104]) where $S_d$ is a fuzzy distance measure. So, if the semantic distance between the two fuzzy sets $A, B$ is nil, then they
are identical in the classical sense. The same line of argument can be followed regarding similarity thresholds in domain-associated fuzziness. The FuzzyDML varies whether it is algebraic-based or calculus-based. If it is algebraic-based, it should accommodate the algebraic operations as defined before. This is done by handling tuple-associated fuzziness and the manipulation of fuzzy predicates. In whatever kind of language, the FuzzyDML should provide a means of manipulation of the tuple-associated fuzziness as well as the domain-associated fuzziness regarding similarity relations.
Discussion & Conclusion

Fuzzy set theory-based methods can be classified into two main approaches (Dubois et al. [46]):

1. degrees of partial truth which are allowed to be truth-functional and which pertain to gradual (or fuzzy) propositions, and

2. degrees of uncertainty which cannot be compositional with respect to all connectives when attached to classical propositions. This distinction is exemplified by the difference between fuzzy logic and possibilistic logic.

The systems FLP1, FLP2 and FLP3 compare nicely to that of possibilistic logic as can be seen from the following:

\[ p_{[FLP]} : \text{fuzzy}, \quad p_{[PL]} : \text{crisp} \]

\[ < p, \mu >_{[FLP]} : \text{crisp}, \quad < p, \mu >_{[PL]} : \text{fuzzy} \]

As \( < p, \mu >_{[FLP]} \), \( \mu \) is a cut truncation at a tolerance level while \( < p, \mu >_{[PL]} \) is an uncertainty qualification.

Another important distinction between fuzzy logic and possibilistic logic is that of compositionality. That’s to say \( T(p \ast q) \) could be expressed by the \( T(p) \) and \( T(q) \), where \( T(p) \) stands for the truth of \( p \) and \( \ast \) is a fuzzy connective. It is to be noted that the connectives of FLP1, FLP2 and FLP3 were all classical connectives as the fuzzy atom \( p \) becomes a crisp atom after the cut operation. So, \( < p, \mu > \) is a crisp entity. This made it easier for these systems to handle refutation which is concerned only with clauses. Also, the principle of sufficiency of Herbrand models can safely be applied which is not the case if arbitrary formulae are considered.

The syntactic difference between the systems of fuzzy logic programming introduced in this thesis and possibilistic logic has been pointed out. Semantics
have been given to FLP systems using models and fixpoint theory. In possibilistic logic, semantics is given by fuzzy sets of interpretations and best models. Though in possibilistic logic semantics is given using any non-Herbrand model, Dubois et al. [45], but in possibilistic logic programming, Dubois et al. [40] introduced the concept of a gradual Herbrand interpretation as well as a fixpoint operator similar to the ones defined in this thesis to characterise interpretations which are models in their system.

The rigorous treatment of possibilistic logic programming as well as the systems presented in this thesis remain in the spirit of van Emde [141] to which a comparison was made at the end of chapter 3. Possibilistic logic is capable of deduction under partial inconsistency and it has been shown that it is fully axiomatisable (Dubois et al. [45]).

Possibilistic logic uses possibility distributions to induce an ordering over the lattice of interpretations. In the system FLP2, the concept of proximate models was used to enlarge the set of models by considering other models which are close to a model. Also, the systems of fuzzy logic programming presented in this thesis could compare to “labelled deductive systems”, of Gabbay [61]. Dubois et al. [47] allow the weights to be functions of variables involved in the clauses which enables the system of hypothetical reasoning. It also enables the system to have fuzzily restricted quantifiers as the functions appear in the expression of the weight. Dubois et al. [49] extended the system to be able to reason with imprecise, i.e. fuzzy constants. Of course, this issue raises the subject of fuzzy unification which has been done using similarity relations (Fontana et al. [55,56]).

Alsinet [2] defined a formal semantics and a sound resolution-style calculus for the system PLFC: Possibilistic Logic with Fuzzy Constants and fuzzily restricted quantifiers, where fuzzy unification is employed. Then PGL (Possibilistic Logic based on Gödel infinitely-valued logic) is defined. A complete modus ponens-style calculus for the Horn-rule fragment of PGL is defined. This logic is extended to
handle fuzzy constants using fuzzy unification. A possibilistic logic programming system based on PGL is introduced. It uses the maximum degree of possibilistic entailment and the degree of deduction as compared to the logical consequence introduced in this thesis. PGL+ is introduced which is the same PGL with fuzzy constants and this is made possible via fuzzy unification.

Then PGL+V which is a first-order possibilistic logic with fuzzy constants based on Gödel predicate logic. PLFC uses a resolution-style calculus with fuzzy unification mechanism while PGL uses a deduction procedure based on a Hilbert-style axiomatization of the logic and a generalized modus ponens inference rule. This calculus is complete only for the Horn-rule fragment of that logic.

Back to truth-functional fuzzy logics, Vojtas [144] proposes a fuzzy logic programming system that employs a wide variety of connectives, conjunctions \&_1, \&_2, \ldots, \&_k, disjunctions \lor_1, \lor_2, \ldots, \lor_l, implications \rightarrow_1, \rightarrow_2, \ldots, \rightarrow_m and aggregations @_1, @_2, \ldots, @_n. Generally, associativity and commutativity are not assumed because of this large variety of connectives. The syntactic level is not affected by this multitude of truth functions. Declarative semantics, procedural semantics, fixpoint theory, soundness and completeness of the proposed system is shown in the spirit of van Emden [141] and Lloyd [107]. This was the same approach adopted in this thesis. The cut level is the main difference between the systems proposed in this thesis and that of others. The cut approach adopted in this thesis led to the fact that these systems deal with clauses where the sufficiency of Herbrand models are assured. In other words, the minimal Herbrand model would contain the atoms that are logical consequence of the program. Furthermore, this minimal model can be characterised using the fixpoint closure operator. In the development of Vojtas [144] where the system can never be confined to clauses, there are no such minimal model semantics. Furthermore, fixpoint characterisation is not complete as in the case of the systems presented in this thesis. In fact, Vojtas fixpoint theorem shows that the minimal fixpoint of
the closure operator is a model. This is due to the complexity of the system introduced in the multitude of connectives. Fixpoint characterisation presented in this thesis characterised minimal models (which contain all logical consequences of the program) up to isomorphism.

Future work built on the results in this thesis could be adapting the systems FLP1, FLP2 and FLP3 to accommodate imprecise constants, fuzzily restricted quantifiers. Also, that would involve employing algorithms for fuzzy unification. The use of the languages of FLP1, FLP2 and FLP3 as specification languages could be a possible direction. Further, fuzzy disjunctive deductive databases could be a new area that is worth exploring. Finally, the reconstruction of fuzzy databases under an axiomatic system of fuzzy logic could be desirable and a worthwhile research direction.
Appendix

In this appendix we give some fuzzy mathematical and basic logic programming concepts to motivate the reader.

§X.1 Some Fuzzy Mathematical Concepts

Given a crisp set $A$ in a universe of discourse, say $X$, the statement $a \in A$ may be true or false and nothing in between. It is noted that that the universe of discourse $X$, together with the crisp set $A$ in $X$ could be:

1. finite.
2. infinite (countably infinite or uncountable).

The ways to describe a classical crisp set could be:

1. Enumerate (list) the elements belonging to the set.

\[ \text{Set of Days} = \{ \text{Mon, Tue, Wed, Thu, Fri, Sat, Sun} \} \]

2. Analytical description: by stating a property (a proposition or a predicate in a logical sense) which have variables which when instantiated, the property may become true or false. When true the instantiating elements belong to the set; when false they do not:

\[ A = \{ x | x^2 = 9 \} \]

3. Characteristic function:

The characteristic function is the same as the property above as for every element if mapped onto 0 means nonmembership and if it is mapped onto 1 means it is a member of the set. Thus, a characteristic function of a crisp set $A$ in a universe of discourse $X$:

\[ f : a \in A \rightarrow \{ 0, 1 \} \]
i.e. every element in the set is mapped either to 0 or to 1. For a fuzzy set the membership function allows various degrees of membership for the elements in the set:

\[ \mu : a \in A \rightarrow [0, 1] \]

Note that the co-domain of the characteristic function of a crisp set was the set \( \{0, 1\} \) while the co-domain of the membership function of a fuzzy set is the whole interval \([0, 1] \). A fuzzy set is defined in the following

**Definition X.1.1:** If \( X \) is a collection of objects denoted generically by \( x \) then a fuzzy set \( A \) in \( X \) is a set of ordered pairs:

\[ A = \{(x, \mu_A(x))|x \in X\} \]

To familiarize the reader with fuzzy mathematical concepts, the concepts are illustrated with examples from [155].

**Example X.1.1:** A realtor wants to classify the offers to his clients. One indicator of comfort is the number of bedrooms in a house. Let \( X = \{1, 2, 3, \ldots, 10\} \). Then the fuzzy set “comfortable type of house for a 4-person family” may be described as:

\[ A = \{(1, .2), (2, .5), (3, .8), (4, 1), (5, .7), (6, .3)\} \]

In the literature one finds different ways of denoting fuzzy sets:

1. A fuzzy set is denoted by an ordered set of pairs, the first element of which denotes the element and the second the degree of membership (as in the previous definition).

**Example X.1.2:** \( A = \)“real numbers considerably larger than 10”

\[ A = \{(x, \mu_A(x))|x \in X\} \]

where

\[ \mu_A(x) = \begin{cases} 0 & x \leq 10 \\ (1 + (x - 10)^{-2})^{-1} & x > 10 \end{cases} \]
Example X.1.3: $A$ = “real numbers close to 10”

\[ A = \{(x, \mu_A(x))|\mu_A(x) = (1 + (x - 10)^2)^{-1}\} \]

2. $A = \frac{\mu_A(x_1)}{x_1} + \frac{\mu_A(x_2)}{x_2} + \cdots = \sum_{i=1}^{n} \frac{\mu_A(x_i)}{x_i}$ for a discrete universe of discourse or

\[ \int_{X} \frac{\mu_A(x)}{x} \text{ for a continuous universe of discourse} \]

It is to be noted that $\sum$ does not indicate any addition operation and the $\int$ does not indicate integration. They are just a convenient notation.

Example X.1.4: $A$ = “integers close to 10”

\[ A = 0.1/7 + 0.5/8 + 0.8/9 + 1/10 + 0.8/11 + 0.5/12 + 0.1/13 \]

Example X.1.5: $A$ = “real numbers close to 10”

\[ A = \int_{R} \frac{1}{1 + (x - 10)^2}/x \]

Definition X.1.2: The (crisp) set of elements that belong to the fuzzy set $A$ at least to the degree $\alpha$ is called the $\alpha$-level set or $\alpha$-cut:

\[ A_\alpha = \{x \in X | \mu_A(x) \geq \alpha \} \]

$A'_\alpha = \{x \in X | \mu_A(x) > \alpha \}$ is called “strong $\alpha$-level set” or “strong $\alpha$-cut”.

Example X.1.6: Referring to example X.1.1, the possible $\alpha$-level sets are listed:

\[ A_{0.2} = \{1, 2, 3, 4, 5, 6\} \]
\[ A_{0.5} = \{2, 3, 4, 5\} \]
\[ A_{0.8} = \{3, 4\} \]
\[ A_{1} = \{4\} \]

§X.2 Basic Set-Theoretic Operations on Fuzzy Sets

The intersection, union and complement of fuzzy sets are defined together with some examples:
Definition X.2.1: The membership function $\mu_C(x)$ of the intersection $C = A \cap B$ is pointwise defined by:

$$\mu_C(x) = \min\{\mu_A(x), \mu_B(x)\}, \quad x \in X$$

Definition X.2.2: The membership function of the union $D = A \cup B$ is pointwise defined by:

$$\mu_D(x) = \max\{\mu_A(x), \mu_B(x)\}, \quad x \in X$$

Definition X.2.3: The membership of the complement of a fuzzy set $A$, $\mu_{\tilde{A}}(x)$, is defined by:

$$\mu_{\tilde{A}} = 1 - \mu_A(x), \quad x \in X$$

Example X.2.1: Let $A$ be the fuzzy set “comfortable type of house for a 4-person family” for the previous example and $B$ the fuzzy set “large type of house” defined as:

$$B = \{(3, .2), (4, .4), (5, .6)(6, .8), (7, 1), (8, 1)\}$$

The intersection $C = A \cap B$ is:

$$C = \{(3, .2), (4, .4), (5, .6), (6, .3)\}$$

The union $D = A \cup B$ is:

$$D = \{(1, .2), (2, .5), (3, .8), (4, 1), (5, .7), (6, .8), (7, 1), (8, 1)\}$$

The complement $B^c$ which might be interpreted as “not large type of house” is:

$$B^c = \{(1, 1), (2, 1), (3, .8), (4, .6), (5, .4), (6, .2), (9, 1), (10, 1)\}$$

Example X.2.2: Let us assume that $A = \text{"x considerably larger than 10"}$, $B = \text{"x approximately 11"}$, characterized by:

$$A = \{(x, \mu_A(x)) | x \in X\}$$
where
\[ \mu_A(x) = \begin{cases} 
0 & x \leq 10 \\
(1 + (x - 10)^2)^{-1} & x > 10 
\end{cases} \]
and
\[ B = \{(x, \mu_B(x)) | x \in X\} \]
where
\[ \mu_B(x) = (1 + (x - 11)^4)^{-1} \]
Then
\[ \mu_{A \cap B}(x) = \begin{cases} 
\min[(1 + (x - 10)^2)^{-1}, (1 + (x - 11)^4)^{-1}] & x > 10 \\
0 & x \leq 10 
\end{cases} \]
(x considerably larger than 10 and approximately 11)
\[ \mu_{(A \cup B)}(x) = \max[(1 + (x - 10)^2)^{-1}, (1 + (x - 11)^4)^{-1}] , \quad x \in X \]

In reasoning why to choose the \textit{min} operator for intersection and the \textit{max} for union and not any other operators, Bellman and Giertz felt that if these operators were denoted \( \land \) and \( \lor \), they should possess the following properties [155]. Afterwards, they proved that:

\[ \mu_{\hat{S} \land \hat{T}} = \min(\mu_{\hat{S}}, \mu_{\hat{T}}) \]
\[ \mu_{\hat{S} \lor \hat{T}} = \max(\mu_{\hat{S}}, \mu_{\hat{T}}) \]

1. \( \mu_{\hat{S}} \land \mu_{\hat{T}} = \mu_{\hat{T}} \land \mu_{\hat{S}} \)
2. \( \mu_{\hat{S}} \lor \mu_{\hat{T}} = \mu_{\hat{T}} \lor \mu_{\hat{S}} \)
3. \( (\mu_{\hat{S}} \land \mu_{\hat{T}}) \land \mu_{\hat{U}} = \mu_{\hat{S}} \land (\mu_{\hat{T}} \land \mu_{\hat{U}}) \)
4. \( (\mu_{\hat{S}} \lor \mu_{\hat{T}}) \lor \mu_{\hat{U}} = \mu_{\hat{S}} \lor (\mu_{\hat{T}} \lor \mu_{\hat{U}}) \)
5. \( \mu_{\hat{S}} \land (\mu_{\hat{T}} \lor \mu_{\hat{U}}) = (\mu_{\hat{S}} \land \mu_{\hat{T}}) \lor (\mu_{\hat{S}} \land \mu_{\hat{U}}) \)
6. \( \mu_{\hat{S}} \lor (\mu_{\hat{T}} \land \mu_{\hat{U}}) = (\mu_{\hat{S}} \lor \mu_{\hat{T}}) \land (\mu_{\hat{S}} \lor \mu_{\hat{U}}) \)
7. \( \mu_{\tilde{S}} \wedge \mu_{\tilde{T}} \) and \( \mu_{\tilde{S}} \vee \mu_{\tilde{T}} \) are continuous and nondecreasing in each component.

8. \( \mu_{\tilde{S}} \vee \mu_{\tilde{S}} \) and \( \mu_{\tilde{S}} \wedge \mu_{\tilde{S}} \) are strictly increasing in \( \mu_{\tilde{S}} \).

9. \( \mu_{\tilde{S}} \wedge \mu_{\tilde{T}} \leq \min(\mu_{\tilde{S}}, \mu_{\tilde{T}}) \)

10. \( \mu_{\tilde{S}} \vee \mu_{\tilde{T}} \geq \max(\mu_{\tilde{S}}, \mu_{\tilde{T}}) \)

11. \( 1 \wedge 1 = 1 \)

12. \( 0 \vee 0 = 0 \)

A more general definition of a fuzzy set than that is given in the first section is that of an \( L \)-fuzzy set. By contrast to the above definition, the membership function of an \( L \)-fuzzy set maps into a partially ordered set, \( L \). Since the interval \([0,1]\) is a poset (partially ordered set) if the rational numbers therein are considered. Then, the fuzzy set as defined in the previous section is a special \( L \)-fuzzy set. The definitions of similarity, proximity, \( \alpha \)-similarity and \( \alpha \)-proximity relations are also needed.

**Definition X.2.4:** A similarity relation is a mapping \( s_k : D_k \times D_k \rightarrow [0,1] \) such that for \( x, y, z \in D_k \):

\[
\begin{align*}
    s_k(x, x) &= 1 \\
    s_k(x, y) &= s_k(y, x) \\
    s_k(x, z) &\geq \max_{y \in D_k} \{\min[s_k(x, y), s_k(y, z)]\}
\end{align*}
\]

These three conditions are of reflexivity, symmetry and max-min transitivity, respectively. Dropping the max-min transitivity condition makes the relation a proximity relation as defined below.

**Definition X.2.5:** A proximity relation is a mapping \( s_k : D_k \times D_k \rightarrow [0,1] \) such that for \( x, y \in D_k \):

\[
\begin{align*}
    s_k(x, x) &= 1 \\
    s_k(x, y) &= s_k(y, x)
\end{align*}
\]
These are the conditions of reflexivity and symmetry, respectively. Similarity thresholds or the degree of closeness between elements are defined below. In the following, the concept of two elements being \(\alpha\)-similar or \(\alpha\)-proximate is defined.

**Definition X.2.6:** If \(s\) is a similarity relation on \(D\), then given any \(\alpha \in [0,1]\), two elements \(x, z \in D\) are \(\alpha\)-similar (denoted by \(xS_\alpha z\)) if and only if \(s(x,y) \geq \alpha\).

**Definition X.2.7:** If \(s\) is a proximity relation on \(D\), then given any \(\alpha \in [0,1]\), two elements \(x, z \in D\) are \(\alpha\)-proximate (denoted by \(xS^\alpha z\)) if and only if there exists a sequence \(y_1, y_2, \ldots, y_r \in D\) such that:

\[
xS_\alpha y_1S_\alpha y_2S_\alpha \ldots S_\alpha y_rS_\alpha z
\]

It is noted that \(S^\alpha\) is an equivalence relation while \(S_\alpha\) is not.

**Further Operations on Fuzzy Sets**

In the previous section it was argued that the \(\min\) and \(\max\) should be used as operators for intersection and union. In 1980, Dubois and Prade proposed the t-norms which are two-valued functions \(t : [0,1] \times [0,1] \rightarrow [0,1]\) which satisfy the following conditions [155]:-

1. \(t(0,0) = 0; t(\mu_A(x), 1) = t(1, \mu_A(x)) = \mu_A(x)\)

2. \(t(\mu_A(x), \mu_B(x)) \leq t(\mu_C(x), \mu_D(x))\) if \(\mu_A(x) \leq \mu_C(x)\) and \(\mu_B(x) \leq \mu_D(x)\) (Monotonicity)

3. \(t(\mu_A(x), \mu_B(x)) = t(\mu_B(x), \mu_A(x))\) (Commutativity)

4. \(t(\mu_A(x), t(\mu_B(x), \mu_C(x))) = t(t(\mu_A(x), \mu_B(x)), \mu_C(x))\) (Associativity)

The functions defined by t-norms define a general class of intersection operators for fuzzy sets, the \(\min\), product and bounded sum belong to this class. Analogous functions t-co-norms or s-norms are defined for a class of union operators. These functions satisfy the following analogous properties:
1. \( s(1,1) = 1; s(\mu_A(x), 0) = s(0, \mu_A(x)) = \mu_A(x) \)

2. \( s(\mu_A(x), \mu_B(x)) \leq s(\mu_C(x), \mu_D(x)) \) if \( \mu_A(x) \leq \mu_C(x) \) and \( \mu_B(x) \leq \mu_D \)
(Monotonicity)

3. \( s(\mu_A(x), \mu_B(x)) = s(\mu_B(x), \mu_A(x)) \) (Commutativity)

4. \( s(\mu_A(x), s(\mu_B(x), \mu_C(x))) = s(s(\mu_A(x), \mu_B(x)), \mu_C(x)) \) (Associativity)

Typical dual pairs of t-norms and t-conorms are compiled below [131]:

1. **Drastic Product**:
\[
t(\mu_A(x), \mu_B(x)) = \min\{\mu_A(x), \mu_B(x)\} \text{ if } \max\{\mu_A(x), \mu_B(x)\} = 1, 0 \text{ otherwise}
\]

2. **Drastic Sum**:
\[
s(\mu_A(x), \mu_B(x)) = \max\{\mu_A(x), \mu_B(x)\} \text{ if } \min\{\mu_A(x), \mu_B(x)\} = 1, 1 \text{ otherwise}
\]

3. **Bounded Difference**:
\[
t(\mu_A(x), \mu_B(x)) = \max\{0, \mu_A(x) + \mu_B(x) - 1\}
\]

4. **Bounded Sum**:
\[
s(\mu_A(x), \mu_B(x)) = \min\{1, \mu_A(x) + \mu_B(x)\}
\]

5. **Einstein Product**:
\[
t(\mu_A(x), \mu_B(x)) = \frac{\mu_A(x) \cdot \mu_B(x)}{2 - [\mu_A(x) + \mu_B(x) - \mu_A(x) \cdot \mu_B(x)]}
\]

6. **Einstein Sum**:
\[
s(\mu_A(x), \mu_B(x)) = \frac{\mu_A(x) + \mu_B(x)}{1 + \mu_A(x) \cdot \mu_B(x)}
\]

7. **Algebraic Product**:
\[
t(\mu_A(x), \mu_B(x)) = \mu_A(x) \cdot \mu_B(x)
\]

8. **Algebraic Sum**:
\[
s(\mu_A(x), \mu_B(x)) = \mu_A(x) + \mu_B(x) - \mu_A(x) \cdot \mu_B(x)
\]
9. Hammacher Product:
\[ t(\mu_A(x), \mu_B(x)) = \frac{\mu_A(x)\mu_B(x)}{\mu_A(x)+\mu_B(x)-\mu_A(x)\mu_B(x)} \]

10. Hammacher Sum:
\[ s(\mu_A(x), \mu_B(x)) = \frac{\mu_A(x)+\mu_B(x)-2\mu_A(x)\mu_B(x)}{1-\mu_A(x)\mu_B(x)} \]

11. Minimum:
\[ t(\mu_A(x), \mu_B(x)) = \min\{\mu_A(x), \mu_B(x)\} \]

12. Maximum:
\[ s(\mu_A(x), \mu_B(x)) = \max\{\mu_A(x), \mu_B(x)\} \]

Also, the definition of possibility distribution will be needed.

**Definition X.2.8:** [155] Let \( F \) be a fuzzy set in a universe of discourse \( U \) which is characterized by its membership function \( \mu_F(u) \), which is interpreted as the compatibility of \( u \in U \) with the concept labeled \( F \). Let \( X \) be a variable taking values in \( U \) and \( F \) act as a fuzzy restriction, \( R(X) \), associated with \( X \). Then the proposition “\( X \) is \( F \)”, which translates into \( R(X) = F \) associates a possibility distribution \( \pi_x \), with \( X \) which is postulated to be equal to \( R(X) \). The possibility distribution function, \( \pi_x(u) \), characterizing the possibility distribution \( \pi_x \) is defined to be numerically equal to the membership function \( \mu_F(u) \) of \( F \), that is,

\[ \mu_x = \mu_F \]

**§X.3 Basic Logic Programming**

Logic can be viewed by mathematicians as providing a precise, solid foundation of the whole discipline of mathematics. Through its definition of a system of a formal language, its syntax, semantics, axioms, rules of inference, model theory and proof theory, it can provide its user (the mathematician) of a methodology to mechanically prove theorems. Applying the rule of inference to the axioms
generates new theorems. The syntax defines which formulae are legal in the language and which are not. The semantics give meaning to these formulae usually using interpretations. An interpretation for a formula that makes it true is called a model. Through model theory, one should know the formulae that are true from a pure semantical point of view. While from proof theory, one should know the formulae that are true from a syntactical point of view, i.e. by applying the rule of inference to the given axioms. In other words, the proof theory presents a proof procedure via which it proves or refutes a given fact. This later equivalence between model theory and proof theory is translated into soundness and completeness results. By introducing theoretically sound (one that produces only correct answers) and complete (one that produces all correct answers) systems of fuzzy logic, no doubt strengthens the thesis of applying fuzzy technology and enhances the confidence that the system designer has in the system that he built. This work is into this significant direction: providing a theoretical foundation for intuitive fuzzy expert systems. Simply, by making Fuzzy Expert Systems governed by the rules of a formal language, its syntax, semantics, axioms, rules of inference, model theory, proof theory. Based on this, the designer would have the results that keeps him confident that any result his system produces is correct (soundness) and that his system produces all the correct answers (completeness).

Until 1972, logic was used as declarative language in computer science till Bob Kowalski introduced a procedural interpretation of logic. The idea of a procedural interpretation for a logic formula was revolutionary with the result of the programming language PROLOG: PROgramming in LOGic. In the following, a brief formal introduction to the basic concepts of logic and logic programming intuitively discussed above will be presented. For a wider background on logic programming, see [101,107]. The objective is to define a system as a formal language, its syntax, semantics, inference procedure, then show that the results produced
by this inference procedure are \textit{correct - soundness}, and that this procedure is \textit{complete - completeness}, i.e. produces \textit{all} correct answers. This has already been established for classical logic programming, which uses SLD-resolution (Linear resolution with Selection function for Definite clauses) and SLI- resolution (Linear resolution with Selection function for Indefinite clauses) - for disjunctive logic programming.

\textbf{First Order Theories:} A first order theory consists of an alphabet, a first order language, a set of axioms and a set of inference rules. The first order language consists of the well-formed formulae of the theory. The axioms are a designated subset of well-formed formulae. The axioms and rules of inference are used to derive the theorems of the theory. An alphabet and a first order language are defined:

\textbf{Definition X.3.1:} An \textit{alphabet} consists of seven classes of symbols:

1. variables.
2. constants.
3. function symbols.
4. predicate symbols.
5. connectives.
6. quantifiers.
7. punctuation symbols.

\textbf{Definition X.3.2:} A \textit{term} is defined inductively as follows:

1. A variable is a term.
2. A constant is a term.
3. If $f$ is an $n$-ary function symbol and $t_1, \ldots, t_n$ are terms, the $f(t_1, \ldots, t_n)$ is a term.

Definition X.3.3: A well-formed formula is defined inductively as follows:

1. If $p$ is an $n$-ary predicate symbol and $t_1, \ldots, t_n$ are terms, the $p(t_1, \ldots, t_n)$ is a formula (called an atomic formula or an atom).

2. If $F$ and $G$ are formulae, the so are $(\neg F), (F \land G), (F \lor G), (F \Rightarrow G)$ and $(F \Leftrightarrow G)$.

3. If $F$ is a formula and $x$ is a variable, the $(\forall x F)$ and $(\exists x F)$ are formulae.

Definition X.3.4: The first order language given by an alphabet consists of the set of all formulae constructed from the symbols of the alphabet.

Definition X.3.5: A clause is a formula of the form:

$$\forall x_1 \ldots \forall x_s (L_1 \lor \ldots \lor L_m)$$

where each $L_i$ is an atom or the negation of an atom. So, one can have the equivalent notation:

$$\forall x_1 \ldots \forall x_s (A_1 \lor \ldots \lor A_k \lor \neg B_1 \lor \ldots \lor \neg B_n)$$

where $A_1, \ldots, A_k, B_1, \ldots B_n$ are atoms and $x_1, \ldots, x_s$ are all the variables occurring in these atoms. So, one can proceed to the logically equivalent notation

$$(p \rightarrow q \equiv \neg p \lor q)$$

$$A_1, \ldots, A_k \leftarrow B_1, \ldots, B_n$$

where all variables are assumed to be universally quantified.

Definition X.3.6 A definite program clause is a clause of the form

$$A \leftarrow B_1, \ldots, B_n$$

$A$ is called the head and $B_1, \ldots, B_n$ is called the body of the program clause.
Definition X.3.7: A \textit{unit clause} is a clause of the form:

\[ A \leftarrow \]

The informal semantics of \( A \leftarrow B_1, \ldots, B_n \) is "for each assignment of each variable, if \( B_1, \ldots, B_n \) are all true, the \( A \) is true". If \( n > 0 \), the clause is conditional. Otherwise \( A \leftarrow \) is unconditional and means "for each assignment of each variable, \( A \) is true".

Definition X.3.8: A \textit{definite program} is a set of definite program clauses.

Definition X.3.9: A \textit{definite goal} is a clause of the form:

\[ \leftarrow B_1, \ldots, B_n \]

Definition X.3.10: The \textit{empty clause} is the clause with empty consequent and empty antecedent. It is a contradiction.

Definition X.3.11: A \textit{Horn clause} is a clause which is either a definite program clause or a definite goal. If the following formula is to be proven:

\[ \exists y_1 \ldots \exists y_r (B_1 \land \ldots \land B_n) \]

is a logical consequence of a logic program \( P \). The negation of the formula is added to the program and contradiction is derived. The logic programming system computes the bindings for the variables which when substituted in the formula, it becomes a logical consequence of the program. Unification (described later in this section) produces these bindings. If \( y_1, \ldots, y_r \) are the variables of the goal:

\[ \leftarrow B_1, \ldots, B_n \]

From the above definitions the syntax was defined, i.e. what constitutes a legal formula in the language. In the following, using interpretations semantics is given to all the symbols of the language. Interpretations which are true in the language are called \textit{models}. This method of defining semantics is called \textit{model semantics}. 

137
The definition of logical consequence gives another definition of semantics which is called *declarative semantics*. Through the introduction of a closure operator, another method of semantics is defined called *fixpoint semantics* which is necessary to link the declarative and model semantics to the mechanical proof procedure which is the methodology to obtain theorems defining: the *procedural semantics*. The equivalence of these semantics has been proven in classical logic programming. The equivalence paves the way for soundness and completeness results.

**Definition X.3.12:** An *interpretation* $I$ of a first order language $L$ consists of the following:

1. A non-empty set $D$, called the *domain* of the interpretation.
2. For each constant in $L$, the assignment of an element in $D$.
3. For each $n$-ary function symbol in $L$, the assignment of a mapping from $D^n$ to $D$.
4. For each $n$-ary predicate symbol in $L$, the assignment of a mapping from $D^n$ into \{true,false\}, or equivalently, a relation on $D^n$.

**Definition X.3.13:** Let $I$ be an interpretation of a first order language $L$ and let $F$ be a closed formula of $L$. Then $I$ is a *model* for $F$ if $F$ is true wrt $I$.

The following definition of logical consequence defines the declarative semantics.

**Definition X.3.14:** Let $S$ be a set of closed formulae and $F$ be a closed formula of a first order language $L$. One says $F$ is a *logical consequence* of $S$ if, for every interpretation $I$ of $L$, $I$ is a model for $S$ implies that $I$ is a model for $F$.

**Definition X.3.15:** A *ground term* is a term not containing variables. Similarly, a *ground atom* is an atom not containing variables.

**Definition X.3.16:** Let $L$ be a first order language. The *Herbrand universe* $U_L$ for $L$ is the set of all ground terms, which can be formed out of the constants and function symbols appearing in $L$. 

138
Interpretations containing the constants in the language play an important role. They are called Herbrand interpretations. This is due to the result that if a set of clauses has a model, it must have an Herbrand model. In the following the classical definitions of an Herbrand universe, Herbrand base, Herbrand interpretation and Herbrand model are given.

**Definition X.3.17:** Let \( L \) be a first order language. The *Herbrand base* \( B_L \) for \( L \) is the set of all ground atoms which can be formed by using predicate symbols from \( L \) with ground terms from the Herbrand universe as arguments.

**Definition X.3.18:** An *Herbrand interpretation* is an interpretation with \( U_L \) as the non-empty set \( D \).

**Definition X.3.19:** Let \( L \) be a first order language and \( S \) a set of closed formulae of \( L \). An *Herbrand model* for \( S \) is an Herbrand interpretation for \( L \) which is a model for \( S \).

A logic programming system can be viewed as a system that computes binding of variables which are the result of a given query. These results must be a logical consequence of the program clauses and facts. But unlike the logician who is interested in logical consequence only, the programmer is interested in the bindings of variables which are obtained via unification and substitution.

**Definition X.3.20:** A *substitution* \( \theta \) is a finite set of the form \( \{ v_1/t_1, \ldots, v_n/t_n \} \), where each \( v_i \) is a variable, each \( t_i \) is a term distinct from \( v_i \) and the variables \( v_1, \ldots, v_n \) are distinct. Each element \( v_i/t_i \) is called a *binding* for \( v_i \). \( \theta \) is called a *ground substitution* if the \( t_i \) are all ground terms. \( \theta \) is called a *variable-pure substitution* if the \( t_i \) are all variables.

**Definition X.3.21:** An *expression* is either a term, a literal or a conjunction or disjunction of literals. A *simple expression* is either a term or an atom.

**Definition X.3.22:** Let \( \theta = \{ u_1/s_1, \ldots, u_m/s_m \} \) and \( \sigma = \{ v_1/t_1, \ldots, v_n/t_n \} \) be substitutions. Then the *composition* \( \theta \sigma \) of \( \theta \) and \( \sigma \) is the substitution obtained
from the set:
\[ \{ u_1/s_1\sigma, \ldots, u_m/s_m\sigma, v_1/t_1, \ldots, v_n/t_n \} \]

by deleting any binding \( u_i/s_i\sigma \) for which \( u_i = s_i\sigma \) and deleting any binding \( v_j/t_j \)
for which \( v_j \in \{ u_1, \ldots, u_m \} \).

**Example:** Let \( \theta = \{ x/f(y), y/z \} \) and \( \sigma = \{ x/a, y/b, z/y \} \). Then:
\[ \theta\sigma = \{ x/f(b), z/y \} \]

**Definition X.3.23:** Let \( S \) be a finite set of simple expressions. A substitution \( \theta \) is called a unifier for \( S \) if \( S\theta \) is a singleton. A unifier \( \theta \) for \( S \) is called a most general unifier (mgu) for \( S \) if, for each unifier \( \sigma \) of \( S \), there exists a substitution \( \gamma \) such that \( \sigma = \theta\gamma \).

The following definitions are important in that they let us define the fixpoint semantics. Through the definition of a fixpoint operator the model semantics is linked to the procedural semantics and the semantics equivalence is proved. In this regard, the definitions of a complete lattice and some fixpoint theorems are needed.

**Definition X.3.24:** Let \( S \) be a set. A relation \( R \) on \( S \) is a subset of \( S \times S \), written infix \( (x, y) \in R \) as \( xRy \).

**Definition X.3.25:** A relation \( R \) on a set \( S \) is a partial order if the following conditions are satisfied:

1. \( xRx \), for all \( x \in S \).
2. \( xRy \) and \( yRx \) imply \( x = y \), for all \( x, y \in S \).
3. \( xRy \) and \( yRz \) imply \( xRz \), for all \( x, y, z \in S \).

**Definition X.3.26:** Let \( S \) be a set with a partial order \( \leq \). Then \( a \in S \) is an upper bound of a subset \( X \) of \( S \) if \( x \leq a \), for all \( x \in X \). Similarly, \( b \in S \) is a lower bound of \( X \) if \( b \leq x \), for all \( x \in X \).
Definition X.3.27: Let $S$ be a set with a partial order $\leq$. Then $a \in S$ is the \textit{least upper bound} of a subset $X$ of $S$ if $a$ is an upper bound of $X$ and, for all upper bounds $a'$ of $X$, it is true that $a \leq a'$. Similarly, $b \in S$ is the \textit{greatest lower bound} of a subset $X$ of $S$ if $b$ is a lower bound of $X$ and, for all lower bounds $b'$ of $X$, it is true that $b' \leq b$. The least upper bound of $X$ is unique, if it exists, and is denoted by $\text{lub}(X)$. Similarly, the greatest lower bound of $X$ is unique, if it exists, and is denoted by $\text{glb}(X)$.

Definition X.3.28: A partially ordered set $L$ is a \textit{complete lattice} if $\text{lub}(X)$ and $\text{glb}(X)$ exist for every subset $X$ of $L$.

Definition X.3.29: Let $L$ be a complete lattice and $T : L \rightarrow L$ be a mapping. $T$ is said to be \textit{monotonic} if $T(x) \leq T(y)$, whenever $x \leq y$.

Definition X.3.30: Let $L$ be a complete lattice and $X \subseteq L$. $X$ is said to be \textit{directed} if every finite subset of $X$ has an upper bound in $X$.

Definition X.3.31: Let $L$ be a complete lattice and $T : L \rightarrow L$ be a mapping. $T$ is said to be \textit{continuous} if $T(\text{lub}(X)) = \text{lub}(T(X))$, for every directed subset $X$ of $L$.

Definition X.3.32: Let $L$ be a complete lattice and $T : L \rightarrow L$ be a mapping. $a \in L$ is said to be the \textit{least fixed point} of $T$ if $a$ is a fixed point (that is, $T(a) = a$) and for all fixed points $b$ of $T$, it is true that $a \leq b$. Similarly, the greatest fixed point is defined.

Theorem X.3.1: Let $L$ be a complete lattice and $T : L \rightarrow L$ be monotonic. Then $T$ has a least fixed point, $\text{lft}(T)$. Furthermore, $\text{lft}(T) = \text{glb}\{x : T(x) = x\} = \text{glb}\{x : T(x) \leq x\}$.

Proof: See [107].

The concepts of ordinal numbers and a mapping raised to its ordinal powers will be needed. $\omega$ the first infinite ordinal which is $\omega = \{0, 1, 2, \ldots\}$, the set of all non-negative integers. A mapping can be applied $\omega$ times which is the infinite ordinal power of the mapping. The following result will also be needed.
Theorem X.3.2: Let $L$ be a complete lattice and $T : L \to L$ be continuous. Then

$$\text{ift}(T) = T \uparrow \omega$$

Proof: See [107].
Bibliography


147


database querying system based on fuzzy logic with linguistic quantifiers,
Information systems, 1989.

[94] J.Kacprzyk, A.Ziolkowksi: Database queries with fuzzy linguistic quantifiers,
IEEE Transactions on systems, man, and cybernetics, vol. SMC-16 No. 3
May/June 1986.

retrieval, Somerset, England.

[96] E.Kerre, G. Chen: An Overview of Fuzzy Data Models, in Fuzziness in

[97] W.Kim, F.H.Lochovsky (eds.): Object-Oriented Concepts, Databases, and


[98'] M.Kifer, V.S.Subrahmanian: Theory of Generalized Annotated Logic Pro-


[100] G.J.Klir, B.Yuan, Eds.: Fuzzy sets, Fuzzy Logic, and Fuzzy Systems: Se-

1979.

[102] R.C.T. Lee, and C.L.Chang: Some properties of fuzzy logic, Inform. and


[152] P.Zdzislaw: Rough Sets, Institute of Computer Sciences, Polish Academy of Sciences, P.O.Box 22, Warsaw.


Fuzzy logic programming

Rafee Ebrahim*

De Montfort University, Leicester, UK

Received July 1997; received in revised form July 1998

Abstract

We develop Fuzzy Logic Programming with the goal to provide a theoretical background for fuzzy expert systems. With the system of fuzzy logic programming that we present the researcher and the practitioner have the theoretical results that the system produces only correct answers and all the possible correct answers. A new notion of fuzzy logical consequence is introduced. A fuzzy minimal model is proposed. The equivalence between the declarative, minimal model, fixpoint and procedural semantics for fuzzy logic programs is proven through the series of the following theorems. The final result is the soundness and completeness of the system:

1. The existence of an Herbrand µ-model for a set of fuzzy clauses having a µ-model.
2. µ-Model Intersection Property.
3. The relationship between the fuzzy minimal model and the fuzzy Herbrand base.
5. The Soundness of the Fuzzy Logic Programming System – any answer produced by the system is correct.

Keywords: Approximate reasoning; Artificial intelligence; Expert systems; Fuzzy logic; Logic programming; Fuzzy logic programming; SLD-resolution; Fuzzy prolog

1. Background and previous work

1.1. Logic and fuzzy logic

Logic can be viewed by mathematicians as providing a precise, solid foundation of the whole discipline of mathematics. Through its definition of a system of a formal language, its syntax, semantics, axioms, rules of inference, model theory and proof theory, it can provide its user (the mathematician) with a methodology to mechanically prove theorems. Applying the rule of inference to the axioms generates new theorems. The syntax defines which formulae are legal in the language and which are not. The semantics give meaning to these formulae usually using interpretations. An interpretation is an assignment of an element from a given set (the domain of the interpretation) to each variable in the domain (defined rigorously below). An interpretation for a formula that makes it true is called a model. We shall see how we generalize this concept in the µ-model for fuzzy logic. Through the model theory, we should know the formulae that are
true from a pure semantical point of view. While from the proof theory, we should know the formulae that are true from a syntactical point of view, i.e. by applying the rule of inference to the given axioms. In other words, the proof theory presents a proof procedure via which it proves or refutes a given fact. This later equivalence between model theory and proof theory is translated into soundness and completeness results. By introducing theoretically sound (one that produces only correct answers) and complete (one that produces all correct answers) systems of fuzzy logic, no doubt strengthens the thesis of applying fuzzy technology and enhances the confidence that the system designer has in the system that he built. This work is into this significant direction: providing a theoretical foundation for intuitive fuzzy expert systems. Simply, by making Fuzzy Expert Systems governed by the rules of a formal language, its syntax, semantics, axioms, rules of inference, model theory, proof theory. Based on this, the designer would have the results that keeps him confident that any result his system produces is correct (soundness) and that his system produces all the correct answers (completeness). For a wider background in logic, the reader is referred to [6,8,18,20,25].

Till 1972, logic was used as a declarative language in computer science till Bob Kowalski [11] introduced a procedural interpretation of logic. The idea that a logic formula can have a procedural interpretation was revolutionary with the result of the programming language PROLOG: PROgramming in LOGic. In the following, we will provide a brief formal introduction to the basic concepts of logic and logic programming intuitively discussed above. For a wider background on logic programming, the reader is referred to [17,1,24] together with foundational work of Kowalski [11–15]. The target is to define a system as a formal language, its syntax, semantics, inference procedure, then show that the results produced by this procedure are correct (soundness), and that this procedure is complete (completeness), i.e. produces all correct answers. This is already established for classical logic, we confirm the results with respect to fuzzy logic for the first time.

First-order theories. A first-order theory consists of an alphabet, a first-order language, a set of axioms and a set of inference rules. The first-order language consists of the well-formed formulae of the theory. The axioms are a designated subset of well-formed formulae. The axioms and rules of inference are used to derive the theorems of the theory. We define what is an alphabet and what is a first-order language:

Definition 1.1. An alphabet consists of seven classes of symbols:
1. variables
2. constants
3. function symbols
4. predicate symbols
5. connectives
6. quantifiers
7. punctuation symbols

Definition 1.2. A term is defined inductively as follows:
1. A variable is a term.
2. A constant is a term.
3. If $f$ is an $n$-ary function symbol and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.

Definition 1.3. A well-formed formula is defined inductively as follows:
1. If $p$ is an $n$-ary predicate symbol and $t_1, \ldots, t_n$ are terms, then $p(t_1, \ldots, t_n)$ is a formula (called an atomic formula or an atom).
2. If $F$ and $G$ are formulae, then so are $(\neg F)$, $(F \land G)$, $(F \lor G)$, $(F \Rightarrow G)$ and $(F \iff G)$.
3. If $F$ is a formula and $x$ is a variable, then $(\forall x F)$ and $(\exists x F)$ are formulae.

Definition 1.4. The first-order language given by an alphabet consists of the set of all formulae constructed from the symbols of the alphabet.

Definition 1.5. A clause is a formula of the form
$$\forall x_1 \ldots \forall x_s (L_1 \lor \cdots \lor L_m),$$
where each $L_i$ is an atom or the negation of an atom. So, we can have the equivalent notation:
$$\forall x_1 \ldots \forall x_s (A_1 \lor \cdots \lor A_k \lor \neg B_1 \lor \cdots \lor \neg B_n),$$
where $A_1, \ldots, A_k, B_1, \ldots, B_n$ are atoms and $x_1, \ldots, x_s$ are all the variables occurring in these atoms. So, we can proceed to the logically equivalent notation
(p → q ≡ ¬p ∨ q)

A₁, ..., Aₖ ← B₁, ..., Bₙ,

where we assume that all variables are universally quantified.

Definition 1.6. A definite program clause is a clause of the form

A ← B₁, ..., Bₙ.

A is called the head and B₁, ..., Bₙ is called the body of the program clause.

Definition 1.7. A unit clause is a clause of the form A+-.

The informal semantics of A+- is “for each assignment of each variable, if B₁, ..., Bₙ are all true, then A is true”. If n > 0, the clause is conditional. Otherwise A+- is unconditional and means “for each assignment of each variable, A is true”.

Definition 1.8. A definite program is a set of definite program clauses. A definite goal is a clause of the form

← B₁, ..., Bₙ.

Definition 1.9. The empty clause is the clause with empty consequent and empty antecedent. It is a contradiction.

Definition 1.10. A Horn clause is a clause which is either a definite program clause or a definite goal.

If we want to prove that the formula:

∀y₁...∀yₙ (∃B₁ ∨ ... ∨ ¬Bₙ),

¬∃y₁...∃yₙ (B₁ ∧ ... ∧ Bₙ).

From the above definitions we defined the syntax, i.e. what constitutes a legal formula in our language. In the following, using interpretations we give semantics to all the symbols of the language. Interpretations which are true in the language are called models. This method of defining semantics is called model semantics. The definition of logical consequence gives another definition of semantics which is called declarative semantics. Through the introduction of a closure operator, another method of semantics is defined called fixpoint semantics which is necessary to link the declarative and model semantics to the mechanical proof procedure which is the methodology to obtain theorems defining: the procedural semantics. The equivalence of these semantics has been proven in classical logic programming and we show it for the first time in fuzzy logic. The equivalence leads to the soundness and completeness results.

Definition 1.11. An interpretation I of a first-order language L consists of the following:
1. A non-empty set D, called the domain of the interpretation.
2. For each constant in L, the assignment of an element in D.
3. For each n-ary function symbol in L, the assignment of a mapping from Dⁿ to D.
4. For each n-ary predicate symbol in L, the assignment of a mapping from Dⁿ into {true, false}, or equivalently, a relation on Dⁿ.

Definition 1.12. Let I be an interpretation of a first-order language L and let F be a closed formula of L. Then I is a model for F if F is true w.r.t. I.

The following definition of logical consequence defines the declarative semantics.

Definition 1.13. Let S be a set of closed formulae and F be a closed formula of a first-order language L. We say F is a logical consequence of S if, for every
interpretation \( I \) of \( L \), \( I \) is a model for \( S \) implies that \( I \) is a model for \( F \).

**Definition 1.14.** A ground term is a term not containing variables. Similarly, a ground atom is an atom not containing variables.

**Definition 1.15.** Let \( L \) be a first-order language. The Herbrand universe \( U_L \) for \( L \) is the set of all ground terms, which can be formed out of the constants and function symbols appearing in \( L \).

Interpretations containing the constants in the language play an important role. They are called Herbrand interpretations. This is due to the result that if a set of statements have a model they must have an Herbrand model [17]. We extend this result later to the fuzzy case. In the following, we give the classical definitions of an Herbrand universe, Herbrand base, Herbrand interpretation and Herbrand model.

**Definition 1.16.** Let \( L \) be a first-order language. The Herbrand base \( B_L \) for \( L \) is the set of all ground atoms which can be formed by using predicate symbols from \( L \) with ground terms from the Herbrand universe as arguments.

**Definition 1.17.** An Herbrand interpretation is an interpretation with \( U_L \) as the non-empty set \( D \).

**Definition 1.18.** Let \( L \) be a first-order language and \( S \) a set of closed formulae of \( L \). An Herbrand model for \( S \) is an Herbrand interpretation for \( L \) which is a model for \( S \).

A logic programming system can be viewed as a system that computes binding of variables which are the result of a given query. These results must be a logical consequence of the program clauses and facts. But unlike the logician who is interested in logical consequence only, the programmer is interested in the bindings of variables which are obtained via unification and substitution.

**Definition 1.19.** A substitution \( \theta \) is a finite set of the form \( \{v_1/t_1, \ldots, v_n/t_n\} \), where each \( v_i \) is a variable, each \( t_i \) is a term distinct from \( v_i \) and the variables \( v_1, \ldots, v_n \) are distinct. Each element \( v_i/t_i \) is called a binding for \( v_i \). \( \theta \) is called a ground substitution if the \( t_i \) are all ground terms. \( \theta \) is called a variable-pure substitution if the \( t_i \) are all variables.

**Definition 1.20.** An expression is either a term, a literal or a conjunction or disjunction of literals. A simple expression is either a term or an atom.

**Definition 1.21.** Let \( \theta = \{u_1/s_1, \ldots, u_m/s_m\} \) and \( \sigma = \{v_1/t_1, \ldots, v_n/t_n\} \) be substitutions. Then the composition \( \theta \sigma \) of \( \theta \) and \( \sigma \) is the substitution obtained from the set:

\[ \{u_1/s_1\sigma, \ldots, u_m/s_m\sigma, v_1/t_1, \ldots, v_n/t_n\} \]

by deleting any binding \( u_i/s_i\sigma \) for which \( u_i = s_i\sigma \) and deleting any binding \( v_j/t_j \) for which \( v_j \in \{u_1, \ldots, u_m\} \).

**Definition 1.22.** Let \( S \) be a finite set of simple expressions. A substitution \( \theta \) is called a unifier for \( S \) if \( S\theta \) is a singleton. A unifier \( \theta \) for \( S \) is called a most general unifier (mgu) for \( S \) if, for each unifier \( \sigma \) of \( S \), there exists a substitution \( \gamma \) such that \( \sigma = \theta \gamma \).

The following definitions are important in that they let us define the fixpoint semantics. Through the definition of a fixpoint operator the model semantics is linked to the procedural semantics and the semantics equivalence is proved. In this regard, we need to define what is a complete lattice and some fixpoint theorems.

**Definition 1.23.** Let \( S \) be a set. A relation \( R \) on \( S \) is a partial order if the following conditions are satisfied:
1. \( xRx \), for all \( x \in S \).
2. \( xRy \) and \( yRx \) imply \( x = y \), for all \( x, y \in S \).
3. \( xRy \) and \( yRz \) imply \( xRz \), for all \( x, y, z \in S \).

**Definition 1.24.** A relation \( R \) on a set \( S \) is a partial order if the following conditions are satisfied:
1. \( xRx \), for all \( x \in S \).
2. \( xRy \) and \( yRx \) imply \( x = y \), for all \( x, y \in S \).
3. \( xRy \) and \( yRz \) imply \( xRz \), for all \( x, y, z \in S \).

**Definition 1.25.** Let \( S \) be a set with a partial order \( \leq \). Then \( a \in S \) is an upper bound of a subset \( X \) of \( S \) if \( x \leq a \), for all \( x \in X \). Similarly, \( b \in S \) is a lower bound of \( X \) if \( b \leq x \), for all \( x \in X \).

**Definition 1.26.** Let \( S \) be a set with a partial order \( \leq \). Then \( a \in S \) is the least upper bound of a subset \( X \) of \( S \) if \( a \) is an upper bound of \( X \) and, for all upper bounds \( a' \) of \( X \), we have \( a \leq a' \). Similarly, \( b \in S \) is
the greatest lower bound of a subset \( X \) of \( S \) if \( b \) is a lower bound of \( X \) and, for all lower bounds \( b' \) of \( X \), we have \( b' \leq b \). The least upper bound of \( X \) is unique, if it exists, and is denoted by \( \text{lub}(X) \). Similarly, the greatest lower bound of \( X \) is unique, if it exists, and is denoted by \( \text{glb}(X) \).

**Definition 1.27.** A partially ordered set \( L \) is a complete lattice if \( \text{lub}(X) \) and \( \text{glb}(X) \) exist for every subset \( X \) of \( L \).

**Definition 1.28.** Let \( L \) be a complete lattice and \( T : L \rightarrow L \) be a mapping. We say \( T \) is monotonic if \( T(x) \leq T(y) \), whenever \( x \leq y \).

**Definition 1.29.** Let \( L \) be a complete lattice and \( X \subseteq L \). We say \( X \) is directed if every finite subset of \( X \) has an upper bound in \( X \).

**Definition 1.30.** Let \( L \) be a complete lattice and \( T : L \rightarrow L \) be a mapping. We say \( T \) is continuous if \( T(\text{lub}(X)) = \text{lub}(T(X)) \), for every directed subset \( X \) of \( L \).

**Definition 1.31.** Let \( L \) be a complete lattice and \( T : L \rightarrow L \) be a mapping. We say \( a \in L \) is the least fixpoint of \( T \) if \( a \) is a fixpoint (that is, \( T(a) = a \)) and for all fixpoints \( b \) of \( T \), we have \( a \leq b \). Similarly, we define the greatest fixpoint.

The next theorem is used in the proofs later in the paper:

**Theorem 1.1.** Let \( L \) be a complete lattice and \( T : L \rightarrow L \) be monotonic. Then \( T \) has a least fixpoint, \( \text{lft}(T) \). Furthermore, \( \text{lft}(T) = \text{glb}\{x : T(x) = x\} = \text{glb}\{x : T(x) \leq x\} \).

**Proof.** See [17].

We will also need the concept of ordinal numbers and a mapping raised to its ordinal powers. \( \omega \) the first infinite ordinal which is \( \omega = \{0,1,2,\ldots\} \), the set of all non-negative integers. A mapping can be applied \( \omega \) times which is the infinite ordinal power of the mapping. We shall also need the following result.

**Theorem 1.2.** Let \( L \) be a complete lattice and \( T : L \rightarrow L \) be continuous. Then

\[ \text{lft}(T) = T \uparrow \omega. \]

**Proof.** See [17].

2. Previous work

2.1. Theoretical developments

Theoretical foundations for logic programming have been provided over the last 20 years. It is marked by the first major result characterizing the least Herbrand model for a logic program (van Emden and Kowalski) using a fixpoint operator. The definition of this fixpoint operator \( T_P \) has been extended to cope with fuzzy Horn clause logic programs. Fuzzy Horn clauses are ordinary Horn clauses where every predicate is true to a certain degree rather than being true or false. [16] goes a bit further and doubts the implication with a factor. It has been shown that the least Herbrand model \( M_P \):

\[ M_P = \text{lft}(T_P) = T_P \uparrow \omega. \]

Another line of work in this area is that of [22] who builds on previous work and arrives at interesting soundness and completeness results. His work does not consider fuzzy predicates. Fuzziness is only limited to the facts and the rules. It has been noted that although various approaches of Fuzzy Prolog were proposed, there was no common interpretation for fuzzifying Prolog [23]. We argue for research to be worthwhile it has to be general and towards the discovery of the main properties of the logics themselves. That is why we find the fuzzy predicate or the fuzzy atom at the core of any fuzzy logic system. We present the declarative semantics, minimal model semantics, fixpoint semantics and procedural semantics and show their equivalence through a series of theorems.

2.2. Previous work: Implemented systems

An FPROLOG interpreter [19]: It is familiar that a logic program (a set of first-order logic statements in clausal form) could be read procedurally or declaratively. The existence of this procedural reading led to the outgrowth of logic programming (first noted by Kowalski). In fuzzy logic programming the procedural reading is similar to the ordinary case except that a truth value has to be computed from the truth values of fuzzy procedures used. The declarative
reading: the goal is shown to be consistent to some degree with the database. The solution is an instance in which the query is true to some degree, given by a suitable combination of the truth values of clauses used in the derivation of the solution.

FPROLOG has been implemented in Lisp, for the facilities of list processing, symbol manipulation, construction of arbitrary data structures and garbage collection. It required a list-based syntax rather than common Edinburgh syntax. This is the reason why it is based on the micro-prolog syntax. The Fril system at Bristol is also implemented in Lisp, thus simplifying the link between both systems. The interpreter uses a standard that represents complex terms by structure sharing [5]. The default is to take the minimum of truth values but it can be overridden by any combination of them by a combination operator and even to redefine the values corresponding to "true" and "false". As a consequence of adding cumulative truth values: if only solutions with a truth value greater than some threshold are required, the system can backtrack as soon as it uses a fact with a truth value lower than the threshold, rather than solving the query and then finding that the truth value is lower than the threshold. The "not" operator is also extended so that if a goal succeeds with a value λ, its negation will have the truth value 1 - λ. This may correspond to success or failure according to the values of λ and the threshold. FPROLOG can run in conjunction with FRIL to call on breadth-first search mechanism, and to use Fril base relations as though they were part of the FPROLOG database. The FPROLOG is being developed using the C language to achieve maximum portability.

An f-prolog Interpreter [16]: Li and Liu implemented an f-prolog interpreter using IF/prolog which is very similar to the Edinburgh syntax which provides ready-made facilities for matching and backtracking. In addition, IF/prolog provides a complete SQL interface to the relational database Oracle. Thus, simplifying the task of linking f-prolog to Oracle.

Both of the above systems [16,19] have the capability of backtracking as soon as the system uses a certain value which is lower than the defined threshold. Martin calls it partial failure of the query, Li calls it generalized failure of the query. Only the system of Li accepts doubting the implication as well as the predicates. The system also has a linguistic extension which can make it produce answers in a sort of "definite, possible, very possible, fairly possible", which is computed using a possibility distribution and associating it with the final answer. Also, it incorporates built-in fuzzy comparison operators like "approximately equal" and "much greater than".

2.3. Overview of the paper

In the following section, we introduce the syntax of the fuzzy logic programming proposed. In Section 4, we provide the semantics of the syntax presented in Section 3. The declarative semantics is linked to the minimal-model semantics and is shown equivalent in this section. Fuzzy logical consequence is defined, then fuzzy Herbrand interpretations. The fixpoint closure operator is defined in this section and the fixpoint characterization of the least fuzzy Herbrand model is proved. At this point, the equivalence between the declarative, minimal-model, fixpoint semantics of fuzzy logic programs is reached. In Section 5, a proof procedure for the fuzzy logic programming system presented earlier is proposed by modifying the classical procedure of SLD-resolution. The SLD-derivation and the SLD-refutation are defined. The concept of a correct answer and that of a computed answer are also defined. In Section 6, soundness and completeness results are reached by showing that every instance of a correct answer is an instance of a computed answer and vice versa. In Section 7, we show that fuzzy logic can be very useful in handling negation in logic programming. Section 8 is the conclusion.

3. Simple fuzzy logic programming systems

In this section, we introduce the syntax of our simple fuzzy logic programming system. We shall see that it proposes a very attractive way of implementing fuzzy prolog within prolog itself. This is one of the advantages in introducing simple fuzzy logic programming as a starting point to the theory. We start by stating the assumptions employed in our simple fuzzy logic programming system:

1. Implication is as in classical first-order logic, i.e.

\[ p \Rightarrow q \equiv \text{def} \overline{\neg p} \lor q. \]
2. *Complementarity* is taken to be between a literal and its negation in fuzzy logic $p, \neg p$. In other words, simple fuzzy logic programming systems cannot make inference except using this notion of complementarity between a literal and its complement.

3. The *quantifiers* employed will be the only classical ones $\forall$ and $\exists$.

As noted above, simple fuzzy logic programming systems form a very good starting point for the development of the theory of fuzzy logic programming. We shall start by defining the syntax of simple fuzzy logic programming systems. The alphabet of simple fuzzy logic programming systems have the following different kinds of symbols:

1. **Constants**: We shall take constants to be elements of a crisp set.
2. **Variables**: These variables can take values and get bound to any of the constants above.
3. **Function symbols**: These are functions mapping tuples from the domain onto elements from the domain.
4. **Predicate symbols**: These are fuzzy relations on the domain which map tuples from the domain onto elements from the domain.
5. **Fuzzy connectives**: Our simple fuzzy logic programming systems include the connectives AND ($\land$), OR ($\lor$), NOT ($\neg$), implication ($\Rightarrow$) and Equivalence ($\equiv$). Their exact definition will be given in the section on semantics.
6. **Quantifiers**: Simple fuzzy logic programming systems have the same quantifiers as classical logic programming systems $\forall$ and $\exists$.
7. **Punctuation symbols**: (, and brackets) in writing formulae.

In order to define the set of simple fuzzy well-formed formulae for simple fuzzy logic programming systems, we shall first define what is an atomic formula. Atomic formulae are formed using terms. So, we shall start by defining a term inductively as follows:

1. A variable is a *term*.
2. A constant is a *term*.
3. If $f$ is an $n$-ary function symbol and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a *term*.
4. *Terms* can only be constructed by applying the above rules a finite number of times.

Steps 1 and 2 above form the basis step of this inductive definition, step 3 the induction and step 4 the closure. We now define an atomic formula in a simple fuzzy logic programming system. The definition of the atomic formula will be the basis for the inductive definition of the set of simple fuzzy well-formed formulae SFWFF.

**Definition 3.1.** If $p$ is a fuzzy predicate symbol and $t_1, \ldots, t_n$ are terms, then $p(t_1, \ldots, t_n, \mu_p)$ is a fuzzy atomic formula where $\mu_p \in [0, 1]$ is the truth value when $p$ is interpreted as a fuzzy relation. We define the set of simple fuzzy well-formed formulae inductively as follows:

- **Basis**: a fuzzy atomic formula is a fuzzy formula.
- **Induction**: 1. If $F$ and $G$ are formulae, then $(\neg F = 1 - \mu_F), (F \land G = \min(\mu_F, \mu_G)), (F \lor G = \max(\mu_F, \mu_G)), (F \Rightarrow G = \neg F \lor G)$ and $(F \equiv G)$ are fuzzy formulae as well.
- 2. If $F$ is a fuzzy formula and $x$ is a variable, then $\forall x F$ and $\exists x F$ are fuzzy formulae.

We use the same definitions (as in classical logic) for the *scope* of a quantifier and *bound* and/or *free* occurrence of a variable.

4. **Semantics of the simple fuzzy logic theory**

In this section, we provide the semantics of the simple fuzzy logic programming system whose syntax was presented in the previous section. Given the same formula, we can have a different meaning in simple fuzzy logic programming and classical logic programming. We mean their semantics are different. We start by defining what a fuzzy interpretation is. Then, the fuzzy forms of satisfiability and validity, consistency and fuzzy models which we will refer to as $\mu$-models are defined. The fuzzy counterparts of Herbrand universe, Herbrand base, Herbrand interpretation and Herbrand model are introduced. The fuzzy program clause is defined with its meaning. The fuzzy logical consequence is defined to provide the declarative semantics. Minimal-model semantics is established and it is shown that the atoms in the fuzzy minimal model are the ones that are fuzzy logical consequence of the fuzzy logic program. So, the proved theorems show the equivalence between the declarative semantics and the minimal model semantics. The fixpoint
closure operator is defined and we prove a theorem to characterize the fuzzy minimal model as the least fixpoint of this closure operator.

**Definition 4.1.** A fuzzy interpretation is:

1. A non-empty set $D$, called the domain of the interpretation. The set $D$ is taken to be a crisp non-fuzzy set.
2. For every constant in our simple fuzzy language, the assignment of an element in $D$.
3. For each $n$-ary function symbol in the language, the assignment of a mapping from $D^n$ to $D$. The interpretation as defined above is identical to that of a classical logic programming system. The difference in semantics stems from the interpretation of predicate symbols. We interpret predicate symbols as fuzzy relations on the domain.
4. For each $n$-ary predicate symbol in the language, the assignment of a mapping from $D^n$ onto $[0, 1]$ unlike the classical case which is onto $\{0, 1\}$ or equivalently a fuzzy relation on $D^n$.

We extend the satisfiability and the validity of formulae to fuzzy formulae as follows.

**Definition 4.2.** Let $I$ be an interpretation for a simple fuzzy language and let $F$ be a fuzzy formula in this language, then:

1. We say $F$ is $\mu$-satisfiable in the interpretation $I$ if $\exists(F)$ has the truth value $\mu$ w.r.t. $I$ or higher.
2. We say $F$ is $\mu$-valid in the interpretation $I$ if $\forall(F)$ has the truth value $\mu$ w.r.t. $I$ or higher.
3. We say that $F$ is $\mu$-unsatisfiable in $I$ if $\exists(F)$ cannot have a truth value $\geq \mu$.
4. We say that $F$ is $\mu$-nonvalid in $I$ if $\forall(F)$ cannot have a truth value $\geq \mu$.

An interpretation for an atomic formula $p(t_1, t_2, \ldots, t_n)$, i.e. makes it true to the value $\mu$ or higher is called a $\mu$-model of $P$.

**Definition 4.3.** Let $I$ be an interpretation of a simple fuzzy language, and let $F$ be a closed formula of this language, then $I$ is a $\mu$-model for $F$ if $F$ has the truth value $\mu$ or higher w.r.t. $I$.

If $F$ has a truth value $\mu$-max, so $F$ has a $\mu$-max-model and for any $\mu < \mu$-max $F$ has a $(\mu/\mu$-max$)$-model. If $I$ is a $\mu$-model for $F$, it may be a $\mu$-max-model or any $(\mu/\mu$-max$)$-model.

**Example 4.1.** Let $p(t_1, t_2, \ldots, t_n, 0.32)$ be an atomic formula, the interpretation consisting of the terms $(t_1, t_2, \ldots, t_n, 0.32)$ is a $(0.32)$-model. While the interpretation consisting of the terms $(t_1, t_2, \ldots, t_n, 0.31)$ is a $(0.31/0.32)$-model.

A statement which has a model is called a consistent statement in classical logic. Similarly, a set of statements are called consistent if they have a model. In other words, if they can be made true under some interpretation. In the following, we define $\mu$-consistency for fuzzy formulae.

**Definition 4.4.** Let $T$ be a simple fuzzy theory and $L$ a language of $T$. A $\mu$-model for $T$ is an interpretation for $L$ which is a $\mu$-model for each axiom of $T$. If $T$ has a $\mu$-model, we say that $T$ is $\mu$-consistent. The concept of a $\mu$-model for a fuzzy formula can easily be extended to a model of a set of fuzzy formulae. The concept of a logical consequence plays a major role in the study of any system of logic. Our new concept of a $\mu$-logical consequence captures what is meant by fuzzy logic inference.

**Definition 4.5.** Let $S$ be a set of closed formulae and $F$ be a closed formula of a fuzzy language $L$. We say $F$ is a $\mu$-logical consequence of $S$ if, for every interpretation $I$ of $L$, $I$ is a $\mu$-model for $S$ implies $(\text{mathematical implication})$ that $I$ is a $\mu$-model for $F$. We define a ground fuzzy term as it is defined for classical logic.

**Definition 4.6.** A ground fuzzy term is a term not containing variables, similarly, a ground atom is an atom not containing variables. The following definitions give the fuzzy counterparts of the Herbrand universe, Herbrand base, Herbrand interpretation, Herbrand model.

**Definition 4.7.** Let $L$ be a fuzzy language, the fuzzy Herbrand universe $U_L$ for $L$ is the set of all ground terms, which can be formed out of the constants and function symbols appearing in $L$.

**Definition 4.8.** Let $L$ be a fuzzy language, the fuzzy Herbrand base $B_L$ for $L$ is the set of all ground fuzzy atoms which can be formed by using the fuzzy
predicate symbols from \( L \) with ground terms from the fuzzy Herbrand universe as arguments.

**Definition 4.9.** The fuzzy Herbrand interpretation is given by the following:
1. The domain of the interpretation is the fuzzy Herbrand universe \( U_L \).
2. Constants in \( L \) are assigned in \( U_L \).
3. If \( J \) is an \( n \)-ary function symbol in \( L \), then the mapping from \( (U_L)^n \) into \( (U_L) \) defined by \( \langle t_1, \ldots, t_n \rangle \rightarrow J(t_1, \ldots, t_n) \) is assigned to \( J \).
4. If \( p \) is a fuzzy predicate symbol in \( L \), then a fuzzy relation on \( U_L \) is assigned to \( p \).

**Definition 4.10.** Let \( L \) be a fuzzy language and \( S \) a set of closed formulae. A fuzzy Herbrand \( \mu \)-model for \( S \) is a fuzzy Herbrand interpretation for \( L \) which is a \( \mu \)-model for \( S \).

**Definition 4.11.** A fuzzy program clause is a clause of the form
\[
A(t_1, t_2, \ldots, \mu_A) \\
← B_1(t_1, t_2, \ldots, \mu_{B_1}), \ldots, B_n(t_1, t_2, \ldots, \mu_{B_n})
\]
which contains precisely one fuzzy atom in its consequent. We note that the number of terms within the atoms \( t_i \) could be different.

**Definition 4.12.** A fuzzy program is a finite set of fuzzy program clauses.

**Definition 4.13.** A fuzzy goal is a clause of the form
\[
← B_1(t_1, t_2, \ldots, \mu_{B_1}), \ldots, B_n(t_1, t_2, \ldots, \mu_{B_n})
\]
that is, a fuzzy clause with an empty consequent. A \( \mu \)-model for an atomic formula is defined above. Now, we define a \( \mu \)-model for a fuzzy clause.

**Definition 4.14.** A \( \mu \)-model for the fuzzy clause:
\[
A(t_1, t_2, \ldots, \mu_A) \\
← B_1(t_1, t_2, \ldots, \mu_{B_1}), \ldots, B_n(t_1, t_2, \ldots, \mu_{B_n})
\]
is an interpretation for the clause that yields it with the truth value \( \mu \).

Similarly, a \( \mu \)-model for the set of fuzzy clauses:
\[
A_1(t_1, t_2, \ldots, \mu_{A_1}) \\
← B_{11}(t_1, t_2, \ldots, \mu_{B_{11}}), \ldots, B_{1n}(t_1, t_2, \ldots, \mu_{B_{1n}}),
\]
\[
A_2(t_1, t_2, \ldots, \mu_{A_2}) \\
← B_{21}(t_1, t_2, \ldots, \mu_{B_{21}}), \ldots, B_{2m}(t_1, t_2, \ldots, \mu_{B_{2m}}),
\]
up to:
\[
A_k(t_1, t_2, \ldots, \mu_{A_k}) \\
← B_{k1}(t_1, t_2, \ldots, \mu_{B_{k1}}), \ldots, B_{kl}(t_1, t_2, \ldots, \mu_{B_{kl}})
\]
is an interpretation that yields every clause of them to the truth value \( \mu \) or higher.

**Theorem 4.1.** Let \( S \) be a set of clauses and suppose \( S \) has a \( \mu \)-model. Then \( S \) has an Herbrand \( \mu \)-model.

**Proof.** Let \( I \) be an interpretation of \( S \). We define an Herbrand interpretation:
\[
I' = \{ p(t_1, t_2, \ldots, t_n, \mu_p) \in B_S \mid I \text{ is a } \mu_p\text{-model of } p(t_1, t_2, \ldots, t_n, \mu_p) \}.
\]
It is straightforward to show that, if \( I \) is a \( \mu \)-model, \( I' \) is an Herbrand \( \mu \)-model. \( \square \)

**Corollary 4.1.** \( A \) is a \( \mu \)-logical consequence of a fuzzy logic program \( P \) if and only if every Herbrand interpretation \( I \) for \( P \), if \( I \) is an Herbrand \( \mu \)-model for \( P \), it is an Herbrand \( \mu \)-model for \( A \).

**Proof.** First, suppose \( A \) is a \( \mu \)-logical consequence of \( P \). Then, by Definition 4.5, for any interpretation \( I \) if \( I \) is a \( \mu \)-model for \( P \), it is a \( \mu \)-model for \( A \). But by Theorem 4.1 there must exist \( I' \) which when being an Herbrand \( \mu \)-model for \( P \), it is an Herbrand \( \mu \)-model for \( A \). This establishes the first side of the argument. Now, we have for every interpretation \( I \), if \( I \) is an Herbrand \( \mu \)-model for \( P \), it is an Herbrand \( \mu \)-model for \( A \). Let \( M \) be an interpretation, not necessarily Herbrand which is a \( \mu \)-model of \( P \). We have
\[
M' = \{ p(t_1, \ldots, t_n, \mu_p) \in B_P \mid M' \text{ } \mu_p\text{-model for } p(t_1, \ldots, t_n, \mu_p) \}
\]
and by Theorem 4.1 $M'$ is an Herbrand $\mu$-model for $P$. And so it is for $A$. So, $M$ is a $\mu$-model for all ground instances $A'$ of $A$. As a result $M$ is a $\mu$-model for $A'$, hence for $A$. This establishes the other side of the argument. □

**Definition 4.15.** If $M_1$ is a $\mu_1$-model and $M_2$ is a $\mu_2$-model, then $M_1 \cap M_2$ contains the fuzzy atoms in both of $M_1$ and $M_2$ but to a degree $\min(\mu_1, \mu_2)$.

**Theorem 4.2 ($\mu$-Model intersection property).** Let $P$ be a fuzzy program and $\{M_i\}_{i \in I}$ be a non-empty set of fuzzy Herbrand $\mu$-models for $P$. Then $\bigcap_{i \in I} M_i$ is a fuzzy Herbrand $\mu$-model for $P$.

**Proof.** For $\mu_1$-model $M_1$ and $\mu_2$-model $M_2$, $M_1 \cap M_2$ (Definition 4.15) is a $\min(\mu_1, \mu_2)$-model.

1. For $i$ $\mu$-models, $\bigcap (M_i)$ is a $\min(\mu_1, \mu_2, ..., \mu_n)$-model.
2. For an infinite number of $\mu$-models, $\bigcap (M_i)$ is an $\inf(\mu_1, \mu_2, ...)$-model. □

**Definition 4.16.** The fuzzy minimal model $FM_P$ or $\mu$-min-model for a fuzzy program is defined as $\bigcap (M_i)$ for an infinite number of Herbrand $\mu$-models for the program.

**Theorem 4.3.** Let $P$ be a fuzzy program. Then if $FM_P$ is the fuzzy minimal model or $\mu$-min-model, we have:

1. $FM_P = \{A \in B_P: A$ is a $\mu$-min logical consequence of $P\}$.
2. When $\mu \geq \mu$-min then $FM_P \subseteq \{A \in B_P: A$ is a $\mu$-logical consequence of $P$ and $\mu \geq \mu$-min\}.
3. When $\mu > \mu$-min $FM_P \subseteq \{A \in B_P: A$ is a $\mu$-logical consequence of $P$ and $\mu > \mu$-min\}.

**Proof.** $A \in B_P$ is a $\mu$-min-logical consequence of $P$ - iff for every interpretation $I$, if $I$ is an Herbrand $\mu$-min-model for $P$, it is an Herbrand $\mu$-min-model for $A$, (by Corollary 4.1).
- if $A \in FM_P$, as $FM_P$ is a $\mu$-min-model for the fuzzy program.

Clearly, if $A$ is a $\mu$-logical consequence of $P$ with $\mu \geq \mu$-min, then it is not in $FM_P$ which establishes the second and third case. □

**Definition 4.17.** Let $P$ be a fuzzy definite program, the mapping $FT_P: 2^{B_P} \rightarrow 2^{B_P}$ is defined as follows. Let $I$ be a fuzzy Herbrand interpretation, then $FT_P(I) = \{A \in B_P: A \leftarrow A_1, ..., A_n\}$ is a ground instance of a clause in $P$ and $\{A_1, ..., A_n\} \subseteq I$, where $\mu_A \geq \mu_A$.

**Theorem 4.4 (Fixpoint Characterization of the Least Fuzzy Herbrand Model).** Let $P$ be a fuzzy definite program. Then:

$FM_P = \text{fp}(FT_P) = (FT_P)^\uparrow \omega$.

**Proof.** $FM_P$ is the minimal $\mu$-model which is the intersection of any non-empty set of Herbrand $\mu$-models = the greatest lower bound of the lattice of the power set of Herbrand $\mu$-models. Since the lattice of the power set of Herbrand $\mu$-models is a complete lattice, then we can use the following fixpoint theorems of lattices:

1. Given a complete lattice $L$ and a monotonic mapping $T: L \rightarrow L$, we have $\text{fp}(T) = \text{glb}\{x: T(x) \leq x\}$ (Theorems 1.1 and 1.2 above).
2. Given a complete lattice $L$ and a continuous $T: L \rightarrow L$, then $\text{fp}(T) = T \uparrow \omega$.

Applying 1, 2 and the continuity of $FT_P$ establishes the theorem. □

**5. Proof theory and examples**

We start by giving definitions of a fuzzy SLD-derivation and a fuzzy SLD-refutation that will be used later on to show the soundness and the completeness of the system. Then we discuss the fuzzy procedural interpretation as compared to the classical case establishing the fuzzy procedural semantics. We follow this discussion by illustrative examples.

**Definition 5.1.** Let $G$ be $\leftarrow A_1, ..., A_m, ..., A_k$ and $C$ be $A \leftarrow B_1, ..., B_q$. Then $G'$ is derived from $G$ and $C$ using mgu $\theta$ if the following conditions hold ($G'$ is the fuzzy resolvent of $G$ and $C$):

1. $A_m$ is an atom called the selected atom in $G$.
2. $\theta$ is an mgu of $A_m$ and $A$.
3. $G'$ is the fuzzy goal $\leftarrow (A_1, ..., A_{m-1}, B_1, ..., B_q, A_{m+1}, ..., A_k \theta)$.
4. $\mu_A \geq \mu_{A_k}$ or equivalently $\mu_{A_{\text{res}}} \geq \mu_{A_{\text{goal}}}$, $\mu_{A_{\text{goal}}}$ must be a constant. In attempting to satisfy a goal with
$\mu$ as a variable, the system must respond with the threshold. This is done by translating the fuzzy goal with a variable into a group of fuzzy goals with constants to find out the value of the threshold.

It is to be noted that the computation rule $R$ by which we select a fuzzy atom must be within condition 4 above. If the standard switching lemma [17] is applied with this restriction, the independence of this fuzzy computation rule is still retained. Obviously, the fuzzy computation rule selects an atom of those satisfying condition 4 as others have failed before being attempted. As such, the fuzzy computation rule is a classical rule with the intelligence of discovering the failure of a fuzzy subgoal without attempting it.

Now, we turn to the proof procedure. Many refutation procedures have been used, we have selected to fuzzify the one used mostly in Prolog systems which is SLD-resolution, which stands for Linear resolution with Selection function for Definite clauses. The classical SLD-resolution is due to Kowalski [15].

Definition 5.2. Let $P$ be a fuzzy program and $G$ a fuzzy goal. A fuzzy SLD-derivation of $P \cup \{G\}$ consists of (finite or infinite) sequence $G_n = G, G_1, \ldots$ of fuzzy goals, a sequence $C_1, C_2, \ldots$ of fuzzy program clauses of $P$ and a sequence $\theta_1, \theta_2, \ldots$ of mgu's such that each $G_{i+1}$ is derived from $G_i$ and $C_{i+1}$ using $\theta_{i+1}$.

Definition 5.3. A fuzzy SLD-refutation of $P \cup \{G\}$ is a finite fuzzy SLD-derivation of $P \cup \{G\}$ which has the empty clause as the last goal in the derivation. If $G_n = \emptyset$, the empty clause, we say the refutation has length $n$. The empty clause is derived from $\leftarrow A(t_1, t_2, \ldots, JIA_{goal})$ and $A(t_1, t_2, \ldots, JIA_{goal}) \leftarrow \mu_{A_{goal}} = \mu_{A_{goal}}$. 

Definition 5.4. Let $P$ be a fuzzy program and $G$ a fuzzy goal. A fuzzy computed answer $\theta$ for $P \cup \{G\}$ is the substitution obtained by restricting the composition $\theta_1 \ldots \theta_n$ to the variables of $G$, where $\theta_1 \ldots \theta_n$ is the sequence of mgu's used in the fuzzy SLD-refutation of $P \cup \{G\}$.

Definition 5.5. Let $P$ be a fuzzy program, $G$ a fuzzy goal $\leftarrow A_1, \ldots, A_k$ and $\theta$ be an answer for $P \cup \{G\}$. We say $\theta$ is a fuzzy correct answer for $P \cup \{G\}$ if $\forall (A_1 \wedge \ldots \wedge A_k) \theta$ is a $\mu$-logical consequence of $P$.

In the following, we compare the procedural interpretation in our fuzzy system as compared to the classical case. In classical logic programming, the procedural interpretation is given by the following:

1. $\leftarrow A_1, \ldots, A_n$ ($n \geq 0$), is a goal statement in which atoms are questions that need to be answered.
2. $A \leftarrow B_1, \ldots, B_n$ is a procedure declaration or a method for question answering. To answer $A$ all the $B_i$'s must be answered.
3. $A \leftarrow$ is a fact that answers $\leftarrow A$ without the need for subsequent derivations.

If our simple fuzzy logic programming system, we shall have the following fuzzy procedural interpretation:

1. $\leftarrow A_1(t_1, t_2, \ldots, \mu_{A_1}), A_2(t_1, t_2, \ldots, \mu_{A_2}), \ldots, A_n(t_1, t_2, \ldots, \mu_{A_n})$ is a fuzzy goal which will not be satisfied until all the fuzzy subgoals within it are satisfied. If the $\mu_{A_n}$ in a fuzzy sub-goal was a constant, then for the fuzzy sub-goal to succeed, it must be at the same truth level or higher by the database. If the $\mu_{A_n}$ in a fuzzy subgoal was a variable, so for it to succeed an answer substitution should be found.
2. The fuzzy clause $A(t_1, t_2, \ldots, \mu_{A_n}) \leftarrow B_1(t_1, t_2, \ldots, \mu_{B_1}), B_2(t_1, t_2, \ldots, \mu_{B_2}), \ldots, B_n(t_1, t_2, \ldots, \mu_{B_n})$ is interpreted as a fuzzy rule for question answering. To answer $A$, all the $B_i$'s must have an answer first. And, we have $\mu_{A_n} \geq \mu_{B_i}$.

3. The fuzzy fact:

$A(t_1, t_2, \ldots, \mu_{A_n}) \leftarrow$, where $\mu_{A_n}$ must be a constant interpreted as a fuzzy assertion. The fuzzy goal $\leftarrow A(t_1, t_2, \ldots, \mu_{A_{goal}})$ succeeds as follows:

(a) $\mu_{A_{goal}}$ is a variable, it returns the constant $\mu_{A_{fact}}$.
(b) $\mu_{A_{goal}}$ is a constant, it returns success only if $\mu_{A_{fact}} \geq \mu_{A_{goal}}$.

The fuzzy SLD-derivation proceeds as follows:

Suppose we have the fuzzy goal: $\leftarrow A_1(t_1, t_2, \ldots, \mu_{A_1}), A_2(t_1, t_2, \ldots, \mu_{A_2}), \ldots, A_n(t_1, t_2, \ldots, \mu_{A_n})$ corresponding to the $n$ fuzzy subgoals:

$\leftarrow A_1(t_1, t_2, \ldots, \mu_{A_1})$

$\leftarrow A_2(t_1, t_2, \ldots, \mu_{A_2})$

up to the fuzzy subgoal:

$\leftarrow A_n(t_1, t_2, \ldots, \mu_{A_n})$. 

We may select any of the \( A_i \)'s to be answered first. Let it be \( A_1 \). Suppose that there is a fuzzy clause in
the database:
\[
A(t_1, t_2, \ldots, \mu_A)
\]
\[
\leftarrow B_1(t_1, t_2, \ldots, \mu_{B_1}), \ldots, B_n(t_1, t_2, \ldots, \mu_{B_n}),
\]
such that \( \theta \) is the most general unifier of \( A \) and \( A_1 \).

Then, we shall have this new stack of fuzzy subgoals:
\[
\leftarrow [B_1(t_1, t_2, \ldots, \mu_{B_1}), B_2(t_1, t_2, \ldots, \mu_{B_2}), \ldots, B_n(t_1, t_2, \ldots, \mu_{B_n})] \theta
\]
\[
\leftarrow A_2(t_1, t_2, \ldots, \mu_{A_2}) \theta
\]
up to the fuzzy subgoal:
\[
\leftarrow A_n(t_1, t_2, \ldots, \mu_{A_n}) \theta.
\]

The derivation is reiterated till the empty clause is
reached.

**Example 5.1.**

Mature-Student\( (x, \mu) \) \( \leftarrow \)

Student\( (x) \), Age-About-21\( (x, \mu) \)

Age-About-21\( (\text{John}, 0.9) \) \( \leftarrow \)

Age-About-21\( (\text{Peter}, 0.4) \) \( \leftarrow \)

Student\( (\text{John}) \) \( \leftarrow \)

Student\( (\text{Peter}) \) \( \leftarrow \)

Here, we have three predicate symbols, namely
Student, Mature-Student and Age-About-21.
The \( n \)-ary predicate symbol becomes an \( n \)-ary+1 if
the predicate is a fuzzy one. This is to allow for the \( \mu \)
indicating the membership value. Obviously, Mature-
Student and Age-About-21 are fuzzy predicates. Now,
we consider the goal \( \leftarrow \) Mature-Student\( (\text{John}, \mu) \).
This will unify the head of the first rule with unification
\( (x = \text{John}, \mu = \mu) \). Thus, resulting into two subgoals,
the first Student\( (\text{John}) \) which succeeds. The
other subgoal is Age-About-21\( (\text{John}, \mu) \) which succeeds
with the value \( \mu = 0.9 \) for John.

**Example 5.2.**

Potential-Customer\( (x, \mu_1) \) \( \leftarrow \) Customer\( (x) \), \( \mu_1 \geq 0.7 \)

Top-Potential-Customer\( (x, \mu_2) \) \( \leftarrow \)

Customer\( (x) \), \( \mu_2 \geq 0.9 \)

Good-Credit-Customer\( (x, \mu_3) \) \( \leftarrow \)

Balance-Level\( (x, y, \mu_3) \), \( \mu_3 \geq 0.7 \)

Customer\( (\text{John}) \) \( \leftarrow \)

Balance-Level\( (\text{John}, 400, 0.7) \) \( \leftarrow \)

Customer\( (\text{Richard}) \) \( \leftarrow \)

Balance-Level\( (\text{Richard}, 500, 0.8) \) \( \leftarrow \)

Consider the goal \( \leftarrow \)

Good-Credit-Customer\( (\text{Richard}, \mu) \) \( \leftarrow \)

**Example 5.3.**

R1: \( p(x, y, \mu_p) \) \( \leftarrow \) \( q(x, \mu_q), r(y, \mu_r) \)

R2: \( p(x, y, \mu_p) \) \( \leftarrow \) \( q(x, \mu_q), s(y, \mu_s) \)

R3: \( q(m, 0.3) \) \( \leftarrow \)

R4: \( r(x, \mu_r) \) \( \leftarrow t(x, \mu_t) \)

R5: \( s(n, 1) \) \( \leftarrow \)

R6: \( t(n, 0.4) \) \( \leftarrow \)

Consider the fuzzy goal \( \leftarrow p(m, n, 0.3) \) which uniﬁes
with the first fuzzy rule giving the two fuzzy subgoals:
1. \( \leftarrow q(m, \mu_{q_1}), \mu_{q_1} \geq 0.3 \),
2. \( \leftarrow r(n, \mu_r), \mu_r \geq 0.3 \).

The fuzzy subgoal (1) unifies with R3 and succeeds
while the second fuzzy subgoal unifies with R4 and results
with another two fuzzy subgoals with the second
being \( \mu_r \geq 0.3 \) resulting in the goal \( \leftarrow (t, 0.3) \) which
succeeds when unifying with R6. As a result, the original
goal \( \leftarrow p(m, n, 0.3) \) succeeds as far as matching
with rule R1 is considered. When matching with rule
R2, two fuzzy subgoals are generated, they are:
1. \( \leftarrow q(m, \mu_{q_1}), \mu_{q_1} \geq 0.3 \),
2. \( \leftarrow s(n, \mu_i), \mu_i \geq 0.3 \).

The first successfully matches with R3 and the sec-
ond as well with R5. So, the original fuzzy goal suc-
ceeds in this case.

Now consider the fuzzy goal \( \leftarrow p(m, n, 0.2) \) when
matching with R1, two fuzzy subgoals are generated,
namely:
1. \( \leftarrow q(m, \mu_{q_1}), \mu_{q_1} \geq 0.2 \),
2. \( \leftarrow r(n, \mu_r), \mu_r \geq 0.2 \).
The first fuzzy subgoal of (1) \( \leftarrow q(m, \mu_q) \) unifies with R3 giving \( \mu_q = 0.3 \) and as a result the second fuzzy subgoal \( \mu_q \geq 0.2 \) succeeds. For the second fuzzy subgoal \( \leftarrow r(n, \mu_r) \), \( \mu_r \geq 0.2 \), we have only rule R4 which unifies successfully resulting in the goal \( \leftarrow (t, 0.2) \) which succeeds when unifying with R6. As a result, the original fuzzy goal \( \leftarrow p(m, n, 0.2) \) succeeds.

When matching with R2, two fuzzy subgoals are generated, namely:
1. \( \leftarrow q(m, \mu_q) \), \( \mu_q \geq 0.2 \),
2. \( \leftarrow s(n, \mu_s) \), \( \mu_s \geq 0.2 \).

The first fuzzy subgoal matches with R3 and succeeds. The second fuzzy subgoal matches with R5 and succeeds.

Now consider a fuzzy goal with a variable \( \mu \), i.e. \( \leftarrow p(m, n, \mu) \), matching with R1, we get:
1. \( \leftarrow q(m, \mu_q) \), \( \mu_q \geq \mu \),
2. \( \leftarrow s(n, \mu_s) \), \( \mu_s \geq \mu \).

The first matches with R3 and \( \mu_q = 0.3 \), thus solving \( \mu \leq 0.3 \). The second will unify with rule R4 then rule R6 returning \( \mu \leq 0.4 \). The original goal succeeds with \( (\mu \leq 0.3) \land (\mu \leq 0.4) \). Thus \( \mu \leq 0.3 \). When matching with rule R2, two fuzzy subgoals are generated:
1. \( \leftarrow q(m, \mu_q) \), \( \mu_q \geq \mu \),
2. \( \leftarrow s(n, \mu_s) \), \( \mu_s \geq \mu \).

The first matches with R3 giving \( \mu \leq 0.3 \). The second matches with R5 giving \( \mu \leq 1 \). The original goal succeeds with \( [(\mu \leq 0.3) \land (\mu \leq 1)] \lor [(\mu \leq 0.3) \land (\mu \leq 0.4)] \). Thus \( \mu \leq 0.3 \).

6. Soundness and completeness

A logic system is said to be sound if it produces correct answers. A system is called complete if it can compute all the correct answers. Soundness and completeness for fuzzy SLD-resolution amounts for showing that every instance of a fuzzy computed answer is an instance of a fuzzy correct answer and every instance of a fuzzy computed answer is an instance of a fuzzy correct answer. In previous sections, it was shown through a series of theorems that the declarative, minimal-model, fixpoint semantics are equivalent. This section adds to them the fuzzy procedural semantics.

Theorem 6.1 (Soundness). Let \( P \) be a fuzzy program and \( G \) a fuzzy goal. Then every fuzzy computed answer for \( P \cup \{ G \} \) is a fuzzy correct answer for \( P \cup \{ G \} \).

Proof. The proof proceeds by induction on the length of the refutation, we need to show that:
1. The argument is true for the first step in the derivation, i.e. the argument is correct for a fuzzy goal of the form \( \leftarrow A(t_1, t_2, \ldots, \mu_A) \). This forms the basis of the inductive argument.
2. We need to show that if the argument is true for derivations with fewer than \( k \) steps then it is true for the following step \( k \).

To show (1), we can see that the necessary and sufficient conditions for a fuzzy goal to succeed are (in terms of proof theory):
1. There exists a matching clause head.
2. \( \mu_{\text{goal}} \geq \mu_{\text{goal}} \).

The necessary and sufficient condition for the fuzzy goal \( A \) to succeed (in term of model theory) is to be in the \( \mu \)-minimal model which necessarily implies being a \( \mu \)-logical consequence of the fuzzy program. This condition implies (1) above, i.e. there exists a matching clause head. Furthermore, to be a \( \mu \)-logical consequence of a fuzzy logic program implies there exists a clause \( C \) within the fuzzy logic program such that: \( C : A(t_1, t_2, \ldots, \mu_A) \leftarrow \) for the fuzzy goal \( \leftarrow A(t_1, t_2, \ldots, \mu_{A_{\text{goal}}}) \) and \( \mu_A \geq \mu_{A_{\text{goal}}} \). This establishes the basis of the inductive argument.

By the induction hypothesis, the argument holds for fewer than \( k \) steps. The fuzzy goal must be of the form
\[ \leftarrow A_1(t_1, t_2, \ldots, \mu_{A_1}), \ldots, A_m(t_1, t_2, \ldots, \mu_{A_m}), \ldots, A_n(t_1, t_2, \ldots, \mu_{A_n}) \].

We suppose that \( A_m \) is the selected fuzzy subgoal of the previous fuzzy goal stack. By the induction hypothesis, there exists an input fuzzy clause:
\[ A(t_1, t_2, \ldots, \mu_A) \leftarrow B_1(t_1, t_2, \ldots, \mu_{B_1}), \ldots, B_q(t_1, t_2, \ldots, \mu_{B_q}) \],

such that:
\[ [A_1(t_1, t_2, \ldots, \mu_A) \land \cdots \land A_{m-1}(t_1, t_2, \ldots, \mu_{A_{m-1}}) \land B_1(t_1, t_2, \ldots, \mu_{B_1}) \land \cdots \land B_q(t_1, t_2, \ldots, \mu_{B_q}) \land A_{m+1}(t_1, t_2, \ldots, \mu_{A_{m+1}}) \land A_n(t_1, t_2, \ldots, \mu_{A_n})] \theta_1 \ldots \theta_k \]
Theorem 6.3 (Completeness). Let $P$ be a fuzzy program and $G$ a fuzzy goal. For every fuzzy correct answer $\theta$ for $P \cup \{G\}$, there exists a fuzzy computed answer $\sigma$ for $P \cup \{G\}$ and a substitution $\gamma$ such that $\theta = \sigma\gamma$.

Proof. If $G$ is the fuzzy goal $\leftarrow A_1, \ldots, A_n$ that has a fuzzy correct answer, then $\forall i A_i \in BP, A_i$ is a $\mu$-logical consequence of $P$. The proof is by contradiction. Suppose for this fuzzy goal we cannot construct $\sigma$ using the proof theory. This means that we cannot construct a $\sigma$ for a specific ground fuzzy subgoal of the form $\leftarrow A$ that is a $\mu$-logical consequence of the fuzzy logic program but $A$ is in the least Herbrand $\mu$-model by Theorem 4.3. Then by Theorem 6.2, it is in the success set and it must have a refutation. The result for the original goal follows by induction on the length of the refutation.

Example 6.1. We illustrate several fuzzy SLD-refutations for the following fuzzy program and the goal: $\leftarrow p(x, b, 0.5)$, then we have the following fuzzy SLD-refutations:

1. $p(x, z, \mu_p) \leftarrow q(x, y, \mu_q), p(y, z, \mu_p)$
2. $p(x, x, 0.6) \leftarrow$
3. $q(a, b, 0.3) \leftarrow$

When the goal unifies with the second fact, a success is reached. When unified with the first clause, two goals are derived $q(x, y, \mu_q), p(y, b, \mu_p)$. Unifying the two goals with fact number 3 produces $q(a, b, 0.3), p(h, b, \mu_p)$. The first of these is resolved and the second succeeds with fact number 2. If the goal $p(h, b, \mu_p)$ is considered with clause 1, it fails.

7. Negation in fuzzy logic programming

We shall use a transformation that transforms any negative fuzzy goal to a definite fuzzy goal and a negative fuzzy program to a definite fuzzy program. We shall see after this transformation that Fuzzy SLD-resolution can be used normally to find an answer for this goal. The only case where Fuzzy SLD-resolution cannot be used when the fuzzy goal (or sub-goal) to be proved has crisp (non-fuzzy atoms). In this case, the negation as failure rule is used to find an answer using SLDNF-resolution.

For the fuzzy goal $\leftarrow A(t_1, t_2, \ldots, \mu_{t_{\text{p}}} )$ to succeed, there are two cases to consider:
1. $\mu_{A_{\text{goal}}}$ is a constant, where there must be a matching fact with $\mu_{A_{\text{fact}}} \geq \mu_{A_{\text{goal}}}$.

2. $\mu_{A_{\text{goal}}}$ is a variable, in this case the goal succeeds if a value is returned for $\mu_{A_{\text{goal}}}$ which is the threshold. Consider the fuzzy goal $\leftarrow A$. If $\mu_{A_{\text{goal}}}$ is a constant, the system answers “Yes” or “No”. If $\leftarrow A$ is “Yes”, then $\leftarrow \lnot A$ is “No” and vice versa. If $\mu_{A_{\text{goal}}}$ is a variable the system answers with the threshold. In both cases, if $A$ is established, then $\lnot A$ follows.

So if we have the fuzzy goal $\leftarrow \lnot A(t_1,t_2,\ldots,\mu_{\lnot v})$, we try $\lnot A(t_1,t_2,\ldots,\mu_{A})$. If $\lnot A(t_1,t_2,\ldots,\mu_{A})$ succeeds with a value for $\mu_{A}$, then $\lnot A$ succeeds with $\mu_{\lnot v} = 1 - \mu_{A}$. If $A$ fails, then we must consider negation as finite failure, then $A$ must be a ground fuzzy atom of $B_P$ and if $\leftarrow A$ fails finitely for $\mu_{A} = 0.2$, then $\leftarrow \lnot A$ succeeds with $\mu_{\lnot v} = 0.8$. We note the special case when $\mu_{A} = 0.5$ where $\mu_{A} = \mu_{\lnot v}$.

The result is, in addition to the extra-expressiveness of fuzzy logic programs relative to non-fuzzy classical logic programs, they offer a very attractive solution for the problem of negation in fuzzy logic programming. Some authors, e.g. [9] tried the use of 3-valued logic, we see that fuzzy logic offers a direct way for handling negation.

8. Conclusion

An original treatment of fuzzy logic programming has been presented. It was based on the discovery of the primary properties that fuzzy logic should offer in addition to standard logic. The syntax together with the declarative semantics, minimal model semantics, fixpoint semantics and procedural semantics for fuzzy logic programs have been presented. The equivalence between these semantics has been shown. The final result is the soundness and completeness of the fuzzy logic programming system presented. An original attractive treatment for negation in fuzzy logic programming was also shown as well. The significance of this direction of research is that intuitive fuzzy expert systems can be founded on sound theoretical background. The designer of such a system has a proof of correctness of each result produced by the system as well as another proof that his system produces all correct answers.

We are currently extending the results developed for fuzzy predicates and fuzzy facts to include also fuzzy rules and to develop a system meta-interpreter for fuzzy logic programs.

References


