TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY
Julkaisu 655 • Publication 655

Ossi Nykänen

Knowledge Agents via Logic Programming and Fuzzy Reasoning

Tampere 2007

Tampereen teknillinen yliopisto. Julkaisu 655
Tampere University of Technology. Publication 655

Ossi Nykänen

Knowledge Agents via Logic Programming and Fuzzy Reasoning

Thesis for the degree of Doctor of Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB111, at Tampere University of Technology, on the 2nd of April 2007, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2007

Custodian: Professor Keijo Ruohonen
Tampere University of Technology
Tampere, Finland

Opponent: Dr. Ivan Herman
Centre for Mathematics and Computer Sciences
Amsterdam, The Netherlands

ISBN 978-952-15-1727-3 (printed)
ISBN 978-952-15-1774-7 (PDF)
ISSN 1459-2045

Errata for the PhD Thesis: "Knowledge Agents via Logic Programming and Fuzzy Reasoning"

Ossi Nykänen 1.4.2007, Tampere

Remark 1 (page 11, just above definition 2):

In the general case, it makes sense to define that if *sup*(A) = 1, then A is normal.

Remark 2 (page 13, Definition 5):

The inequality symbols are ambiguous in the definition of the case of $a_1 = a_2$. It should read $x < a_1$ and $a_1 < x$ (instead of $x \leq a_1$ and $a_1 \leq x$).

Remark 3 (page 21, second last paragraph):

The text reads "we essentially get a recursive version of the well-known max-min composition". It should read: "we essentially get a recursive version of the well-known max-min *rule* composition". (The term "max-min composition" unintentionally refers to a different, well-established concept of fuzzy relations.)

Remark 4 (page 23, Proof for the Proposition 4):

The term "describe query" should be replaced with the term "ask query".

On the last paragraph, it reads: "…the smallest firing strength…". It should read: "…the *largest* firing strength…"

Remark 5 (page 24, Proof for the Proposition 4):

On the first paragraph, it reads: "…both ω_1 and ω_2 must also be firing strengths." It should read: "…*either* ω_1 *or* ω_2 must also be a firing strength."

Remark 6 (page 27, the equation before subsection 2.6):

In the first line of the equation, the operator $\otimes$ is missing from the first term ("r_3))$r_0\vee$"). It should read: "r_3))$\otimes r_0\vee$".

Remark 7 (page 40, first paragraph in subjection 3.4.1):

It reads: "...one ore more…" while it should read: "...one *or* more…".

Acknowledgements

The author wishes to thank Mr. Osmo Kaleva for pointing out the remarks 1 and 2.

Abstract

Implementing knowledgeable applications benefits from the ability to model problems on the knowledge level. In practice, this usually means engagement with the *logical description* of problems. However, when (crisp) logic is applied as a tool in application design, modelling the fundamental vagueness of the human knowledge may soon become an obstacle.

We claim that the two worlds of *logic programming* and *fuzzy reasoning* should coincide when practical knowledge applications via logic programming are concerned.

In this thesis, we point out and analyse an approach that provides the necessary methodological and technical means achieving this goal. In particular, we consider the various aspects of systems called *fuzzy knowledge agents*. By a fuzzy knowledge agent we mean a tool that helps users to process and manage information via the logical description of the domain, benefiting from the use of fuzzy models when applicable.

The chief utility of fuzzy knowledge agents lies within their ability to formulate and encapsulate information through logical and linguistic means, providing a *logic-based end-user interface* to the underlying information on the knowledge level. In this thesis, we establish and analyse a reasoning architecture that allows realising the main steps of designing and implementing fuzzy knowledge agents, including: inception and elaboration of the domain vocabulary and the associated logical procedures (to be captured with type-1 fuzzy logic programs); appropriate modelling of the domain concepts, based on heuristic and statistical arguments (to be modelled with fuzzy sets, induced from empirical data when appropriate); and construction of the reasoning and query applications.

In addition, we consider the concept of context-aware logic programs and review the necessary technical components of fuzzy knowledge agents. Finally, we evaluate the methods and discuss their applicability with several illustrative use cases.

Preface

This thesis has been written at the Institute of Mathematics in Tampere University of Technology. The work has been supported by the Tampere Graduate School in Information Science and Engineering (TISE) and the Academy of Finland.

I am grateful to my supervisor, Professor Keijo Ruohonen, for the discussions, mentoring, and practical help. I want to sincerely thank Professors Eero Hyvönen (Helsinki University of Technology) and Matti Linna (University of Vaasa) for carefully reviewing the manuscript. I also want to thank the Head of Institute, Professor Seppo Pohjolainen for his advice and support, and for providing the pivotal scientific and institutional community that enables researchers to grow beyond their immediate limitations. I highly appreciate the support and the encouragement from all the people at the Institute of Mathematics and Hypermedia Laboratory I have been fortunate to work with and to learn from.

It is almost needless to say that I am forever in debt to my lovely wife Mari, our dear children Teo and Selina, and our beloved goddaughter Iiris. You mean the world to me.

And finally:

Tämä väitöskirja on omistettu vanhemmilleni,
Hilkalle ja Markulle, sekä veljelleni Esalle[1].

Tampere, January 2007

Ossi Nykänen

[1] This thesis is dedicated to my parents, Hilkka and Markku, and to my brother Esa.

Contents

1 Introduction **1**
- 1.1 Scope and Objective . . . 1
- 1.2 Fuzzy Knowledge Agents as Interfaces . . . 3
- 1.3 Context and Related Work . . . 4
  - 1.3.1 Towards Knowledge Personalisation . . . 5
  - 1.3.2 Logic Programming . . . 5
  - 1.3.3 Extending Crisp Logic . . . 6
  - 1.3.4 Web as a Platform of (Logical) Computing . . . 7
  - 1.3.5 Approaches to Fuzzy Modelling . . . 8
- 1.4 Organisation and Contribution . . . 9

2 Type-1 Fuzzy Logic Programs **11**
- 2.1 Fuzzy Sets . . . 11
- 2.2 Representing Fuzzy Sets . . . 12
- 2.3 Syntax . . . 15
  - 2.3.1 Simple Grammar . . . 16
  - 2.3.2 Structure of Logic Programs . . . 17
  - 2.3.3 Queries . . . 18
- 2.4 Interpretation . . . 19
- 2.5 Type-1 Fuzzy LPs in Prolog . . . 25
- 2.6 Applications . . . 27
  - 2.6.1 Basic Crisp Reductions . . . 27
  - 2.6.2 Modelling "Not" By Adding Rules . . . 29
  - 2.6.3 Systems with Subnormal Models . . . 31

3 Linguistic Reasoning and Context **33**
- 3.1 Fuzzy If-Then Rules . . . 33
  - 3.1.1 Type-1 Fuzzy LPs and Fuzzy Control Systems . . . 33
  - 3.1.2 Linguistic Variables . . . 35
- 3.2 Linguistic Construction of Fuzzy Models . . . 35
  - 3.2.1 Property Constructors . . . 35

3.2.2 Hedges 37
3.3 Capturing Fuzziness via Linguistic Description 38
3.4 Context-Aware Logic Programs 40
3.4.1 Modular Logic Programs 40
3.4.2 Weighting by Contextual Distances 42
3.5 Fuzzy Models as Query Interfaces 43
3.6 Challenge of Global Interpretation 45

4 Inductive Models and Implementations 47
4.1 Inducing Fuzzy Models 47
4.1.1 Inducing Fuzzy sets 47
4.1.2 Inducing MFs with Decision Trees 49
4.1.3 A Case Study 50
4.1.4 Statistics and Fuzziness in Modelling 52
4.2 Implementations 52
4.2.1 Case CWM 53
4.2.2 Case SWI-Prolog 55

5 Case Studies 59
5.1 Introduction 59
5.2 Decision-Support for Teaching 60
5.2.1 Inducing Models for Student Classification 60
5.2.2 Logical Formulation of the Application Domain 62
5.2.3 Rules and Queries 63
5.2.4 Components of Predictive Fuzzy Systems 64
5.3 Towards Personalised Studying 66

6 Summary of Publications 69

7 Conclusion 73

List of Publications

The thesis includes the following publications, in the reversed order of publication [68, 67, 66, 64, 63, 62]:

1. Nykänen, O. 2006. Inducing Fuzzy Models for Student Classification. *Educational Technology & Society Journal.* Volume 9, Issue 2, pp. 223-234.

2. Nykänen, O. 2006. An Approach to Logic Programming with Type-1 Fuzzy Models Using Prolog. *Proceedings of the IADIS International Conference of Applied Computing.* 25-28 February, 2006 San Sebastian, Spain, pp. 201-208.

3. Nykänen, O. 2005. Implementing Semantic Web Agents That Learn Upon Experience: A Machine Learning Module for the CWM Rule System. *Proceedings of the IADIS International Conference of Applied Computing.* 22-25 February, 2005 Algarve, Portugal, Volume 1. pp. 409-416.

4. Nykänen, O. 2004. Fuzziness as a Recognition Problem: Using Decision Tree Learning Algorithms for Inducing Fuzzy Membership Functions. *Data Mining V: Data Mining, Text Mining, And Their Business Applications.* WIT Press, UK, pp. 143-153.

5. Nykänen, O. 2004. On Implementing Fuzzy Semantic Web Agents with the CWM Rule System. *Proceedings of the IADIS WWW/Internet 2004 Conference.* 6-9 October 2004, Madrid, Spain, pp. 1075-1078.

6. Nykänen, O. 2004. An Approach for Integrating Statistical Decision-Support Data with Fuzzy Action Rules. *Proceedings of IEEE International Conference on Advanced Learning Technologies.* 30 August - 1 September 2004, Joensuu, Finland, pp. 156-160.

In addition, the main text of the thesis provides a synthesis for the work, including some previously unpublished results, namely proofs and more detailed technical discussion related to the algorithms and the case studies (for details, please see Subsection 1.4). The actual implementations of the developed algorithms with Maple, Prolog, Python, and CWM are not reproduced here; only the architecture and the main results are discussed.

The treatment related to the empirical machine learning methods and statistical considerations of the approach builds upon a study published separately as a monograph [71]. Seeking case studies from the educational domain has been motivated by authors' related work in the area (see, e.g. [74, 73, 72]), and the neutral and easy-to-understand nature of the domain. In addition to the logic programming paradigm, this thesis has been influenced by the studies and reviews related to Semantic Web and other applied query and rule systems (for details, please see, e.g. [69, 70, 65]). In particular, even if detached from the abstract treatment, the development of type-1 fuzzy logic programs has largely been motivated by the conception of fuzzy Semantic Web technologies and the related distributed logic applications.

Chapter 1

Introduction

Implementing knowledgeable applications benefits from the ability to model problems on the knowledge level. In practice, this usually means engagement with the logical description of problems. However, when Classical logic is applied as a tool in application design, modelling the fundamental vagueness of the human knowledge may soon become an obstacle. It appears that the two worlds of *logic programming* and *fuzzy modelling* should coincide when practical applications are concerned.

This study establishes and analyses the framework of systems called *fuzzy knowledge agents*[1]. In short, a fuzzy knowledge agent is a tool that helps users to process and manage information via the logical description of the application domain, benefiting from the use of fuzzy models when applicable[2]. In applications, fuzzy knowledge agents appear as components of *Representation and Reasoning Systems* which in real-world applications are in general required to manage imprecise information (see, e.g., [78]).

1.1 Scope and Objective

In this thesis, we investigate the *necessary empirical and logical components* required in designing and implementing fuzzy knowledge systems, subject to *fuzzy queries and reasoning.* The methods we will be talking about are essentially *technical* or *formal.* This has *a delimiting effect* (see [24]); while we are dealing with abstractions that enable designing and implementing useful

[1] We intentionally use the word *agent* which may either simply refer to client software, e.g., as a GUI/DB/.../Web agent, or to an actor accessing information, e.g., as a software agent. The reader is free to choose the interpretation he or she is more comfortable with.

[2] Again, the user may be a human or a computer program, depending on the interpretation.

applications, we restrict the description language and make simplifying assumptions about the nature of the associated models, following the tradition of instrumentalism [28].

When practical applications are considered, general-purpose technical means are obviously not alone *sufficient*, due to the fact that quality of the production systems largely depends upon domain-specific modelling. Further, since we are dealing with fuzzy systems, domain-specific considerations fundamentally include *both* the design decisions related to the appropriate selection of the logical vocabulary and the associated procedures, *and* the design decisions related to the universe of fuzzy models.

In this thesis, we point out and construct a reasoning architecture that allows realising the main steps of designing and implementing fuzzy reasoning systems, including:

1. Inception and elaboration of the domain vocabulary and the associated logical procedures (to be captured with type-1 fuzzy logic programs).

2. Appropriate modelling of the domain concepts, based on heuristic and statistical arguments (to be modelled with fuzzy sets, induced from empirical data when appropriate).

3. Construction of the reasoning and query applications (with concrete use cases).

We in addition evaluate and demonstrate the approach in terms of educational decision-support applications. Further, we consider the required technical components and establish a strategy of actually implementing such systems with accessible technical tools.

We may perceive the developed approach as a specialisation of the more general *knowledge engineering process*: decide what to talk about; decide on a vocabulary of predicates, functions, and constants; encode general knowledge about the domain; encode a description of the specific problem instance; and pose queries to the inference procedure and get answers [80][3]. The chief distinction lies within the challenge of fuzzy modelling and fuzzy reasoning. The increased freedom in modelling implies not only additional freedom in knowledge engineering, but also requires making additional decisions about the nature and the interpretation of the models.

The main objective of this thesis is establishing the necessary technical means for designing fuzzy knowledge agents and demonstrating their role in

[3]Note that this process might be also considered as a knowledge-intensive specialisation of the more general design and development processes (see, e.g., [23]).

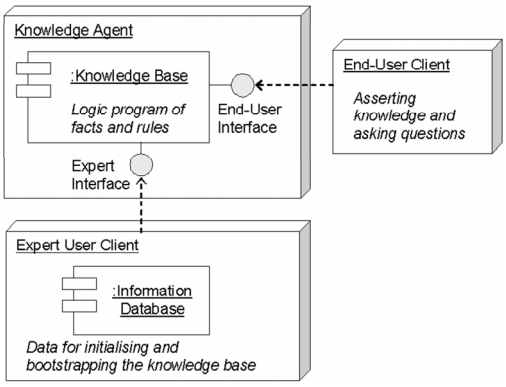

Figure 1.1: Fuzzy knowledge agents as logical information interfaces

abstract reasoning and decision-support applications. To accomplish this, we develop a new method for inducing fuzzy models, derive a novel type-1 fuzzy extension to logic programming, evaluate the approach within an educational use case, and demonstrate the approach through examples. Despite the seemingly formal nature of the treatment, the motivation behind the included studies is largely due rather practical considerations: how to capture and manage large quantities of imprecise, context-dependent data while benefiting from the understanding of logical systems in general.

We shall next provide a brief introduction to the problem domain, outline the related research, and then review the main parts of the thesis.

1.2 Fuzzy Knowledge Agents as Interfaces

Figure 1.1 characterises the basic architecture of (passive) knowledge agents, highlighting the role of different kinds of interfaces or views in accessing knowledge. In short, the knowledge agent captures the underlying (information) database with a logical description, in a form suitable to the end-user. In a way, the knowledge agent provides a conceptualisation of the relevant information. The success of this activity depends upon expert insight and evaluation.

The fuzzy knowledge agent architecture also hints the importance of logic as a communication language. Indeed, it has been claimed that mathematics has two fundamental tasks: the descriptional task and the deductional task [32]. Fuzzy knowledge agents demonstrate both, trying in practice to manage the computational requirements of the deductional task by appropriate modelling [45, 6].

The fact that we conceptualise information in terms of a logic database, includes a design decision. Alternatively, we might characterise information in terms of probabilistic models, or simply as input to some computer programs, according to an appropriate model of computation [96, 101, 78]. As usual, we consider logic merely as a tool for describing the world. In particular, we are not making the assumption that the world itself "is" logical, set-theoretic, truth-functional, etc. (see [31]). Further, while the correctness of deductions is usually proved by referring to truth values, it is important to observe that as a concept, "truth" exists only within the logical models. In applications, "true clauses" are used to record whatever information the designer considers useful.

The power and elegance of logic lies within its ability to abstract and simplify clauses. However, when practical systems are considered, managing *imprecision* becomes topical. The human description of the world is imprecise for various reasons. This imprecision originates both from the allusive nature and the complexity of the world, from the limitations of the human senses and cognition, and from the limitations of the human language. Three forms of imprecision are worth recognising in practice: *imprecision due ignorance*, *imprecision due randomness*, and *imprecision due vagueness*. In applications, imprecision due vagueness, i.e. *fuzziness*, is particular interesting since to some extent, it can be rigorously captured with logical *and* linguistic means. In applications, fuzziness may be intentional (e.g. using a relaxed intuitive terminology instead of a strict technical one) or forced (e.g., due lack of information or the high costs of obtaining it).

In an ideal setting, a fuzzy knowledge agent performs the task of an electronic sister-in law [56]. That is, knowing the user and understanding his or her contextual needs when accessing information repositories and proportioning the compiled knowledge accordingly. There is simply too much potentially useful information available to be processed without tools. Seeking efficient methods to access, filter, describe, and use information in an understandable way is thus important. Information, that can not be accessed and reasonably included to one's contextual picture of the world, simply does not matter.

1.3 Context and Related Work

An intuitive way to approach the problem domain is to consider fuzzy knowledge agents in terms of *personalisation* applications. Let us next very briefly review the main topics of interest according to the main use case: implementing context-aware knowledge agents in the educational domain.

1.3.1 Towards Knowledge Personalisation

A very attractive high-level objective of personalisation is *knowledge personalisation*. In brief, knowledge personalisation is an abstract activity of managing and composing units of distributed knowledge, based on user profiles and behaviour [19, 20]. However, since it is problematic to speak about knowledge without a decent reference system, we will next provide a brief outline of two of the potential candidates.

Technically speaking, two elements of personalisation are worth distinguishing in practise: adaptation of *presentation* and adaptation of *content*. In a simple application, personalisation might simply mean adaptation over a set of predefined information units, perhaps based on an explicit *delivery context*. In brief, this is the perspective of device independence [26]. In the advent of mobile devices, content adaptation has become particularly popular in applications related to, e.g., single-sourcing [49].

More complex applications require considerable modifications of the information content exploited in the interaction. In the context of customer-related activities, these applications include, e.g., making recommendations, searching and filtering contextual information, providing dynamic bargains, and even tailoring products for the individual customers [2]. Even if the borderline may appear a bit vague, adaptation of content should intuitively imply something more than simply, e.g., repaginating content and choosing appropriate device-depended interface components [82, 15, 21].

In brief, two complementary strategies of content adaptation may be recognised: adaptation based on semantically annotated structured components and adaptation based on logical knowledge models. While the former treats content as a hierarchy of black boxes, neatly described with appropriate metadata, the latter and the more interesting approach (at least considering this thesis) thrives to represent knowledge "as such", via a logical corpus.

1.3.2 Logic Programming

A firm basis for representing and using knowledge is established by *logic programming* (LP) (see, e.g., [86, 11, 16, 78]). In brief, modern logic programming is "usually" based on the notions of unification, resolution, and Horn clause logic, whose introduction led into the implementation of the first *Prolog* (*Programmation en Logique*) systems in the 1970s.

Horn clause logic (and thus the logic programming paradigm) is complementary to the various systems of *description logics* (DL) which consider certain tractable subsets of the more general First Order Logic [6]. A rough division of labour is that DL introduces the language for constructing con-

cepts and modelling the knowledge base including *assertions* with an associated *terminology* while Horn clause logic is used for expressing and processing rules, i.e. the *logic program.* This complementary role is genuine, i.e. considering the usual definitions, certain DL knowledge bases can not be expressed as finite Horn rule systems and vice versa.

Today, Prolog tools are available to most computer environments, including systems such as the GNU Prolog and the SWI-Prolog [27, 100]. The development of practical knowledge base applications, however, reveals a major weakness of Horn clause logic as a programming language: when envisioned as a single unit, the design process of logic programs does not scale very well to large applications [10].

Contextual logic programming (CxLP) extends logic programming with mechanisms tailored for modularization [58, 55]. A relatively recent implementation of the GNU Prolog/CX [1] provides a basis for implementing contextual logic programming using mainstream tools. In short, contextual logic programs are composed of *units* accepting unit arguments that appear as global variables within units. The topic of modular logic programming has been extensively studied, aiming to successfully integrate the so-called *programming-in-the-large* and the *programming-in-the-small* disciplines [14].

However, while the notion of context in contextual logic programming might reflect code modularity and object-oriented design, it does not as such carry the intuitive semantics of *contextual interpretation* of logic programs and knowledge. Indeed, the latter notion of context frequently appears in, e.g., linguistics [91], pervasive computing [49], and in the domains related to filtering, profiling, and matching applications in general [13, 90]. In addition, the more intuitive notion of interpretation context notably appears in the study of fuzzy systems where some progress has been made in formalising the concept [25].

Since personalisation and adaptation typically include the tasks of aggregation and transforming of (vague) information, crisp knowledge models are not always applicable. The introduction of *fuzzy methods and modelling* aims managing *imprecision* (or *vagueness*) in applications [34]. Based on the notion of fuzzy sets, the concept of vagueness genuinely complements the concept of *randomness* in modelling *indeterminacy*, establishing the two facets of the phenomenon [61].

1.3.3 Extending Crisp Logic

In short, fuzzy systems fall into two categories. While *fuzzy logic* or *fuzzy expert systems* consider fuzziness in terms of fuzzy implication and a generalisation of crisp (two or finitely-valued) logic, *fuzzy control systems* or *fuzzy rea-*

soning systems aim reproducing the behaviour of intuitive control rule groups by exploiting fuzzy models, i.e., computing with fuzzy sets [93, 38, 17]. The fact that fuzzy membership functions appear often overly precise in applications has motivated the development and application of, e.g., interval-valued and type-2 fuzzy sets [94, 53].

Due to the popularity of the fuzzy systems, native fuzzy prolog systems have also been implemented (see, e.g., [94]). Similarly, fuzzy version(s) of description logics have been developed (see, e.g., [87, 4, 77]). However, just as in the case of the emerging CxLP systems, no single fuzzy Prolog implementation has yet earned the role of the de facto standard and fuzzy DLs have not gained much popularity in applications. It is worth noticing, however, that as such, fuzzy modelling (or context-aware interpretation, for that matter) does not require adopting a native fuzzy logic shell; fuzzy models may be well captured with crisp logic programming tools (see, e.g., [39]).

In the context of educational applications, the reported soft computing experiments include applications of Bayesian and Neuro-Fuzzy methods [37, 104, 85]. These methods have been widely applied, taking many aspects of educational systems into account.

The reported applications demonstrate analysing and evaluating students based on individual assessments, students' views, education quality, grades of journals, and the performance of entire academic departments [50, 46, 88, 59, 5, 36, 92, 48]. The claimed advantages of fuzzy modelling range from faithfully evaluating students' learning and cognitive abilities to moving towards personalised education [84, 99, 40].

1.3.4 Web as a Platform of (Logical) Computing

It seems reasonable to assume that the interoperable World Wide Web technologies will have a major impact in applications related to knowledge personalisation, computing, and logic programming in general. Indeed, integrated logic programs have been developed since the introduction of the Web (see, e.g., [47]).

In brief, the *Semantic Web* (SW) provides a set of universal standards and tools for publishing and processing semantic (meta)data in applications [7, 51][4]. The well-known SW applications include search and aggregation applications, semantic portals, rule frameworks, and integrated application environments [89, 35, 3, 8, 18]. While much progress has been made, the

[4]It is worth noticing that the idea of machine-understandable semantics and logic underlines many "other" initiatives as well, including Grid Computing, Ubiquitous Computing, and Web 2.0.

dream of public large-scale applications has yet to be fulfilled [103, 105]. It is, however, expected that considerable progress might be made, e.g., in the domain of life sciences [102].

The core of the Semantic Web is defined by a set of World Wide Web Consortium (W3C) recommendations [98]. These include the Resource Description Framework (RDF), Web Ontology Language (OWL), the related RDF Schema Specification, and the RDF and OWL Semantics. The specifications most notably provide the abstract RDF data model and a system for defining RDF vocabularies, and establish a restricted version of the OWL family, the OWL DL. Since RDF exploits URI names for encoding names, the framework provides an effective strategy for the development of partially interoperable knowledge bases, without a too strict requirement of centralised co-ordination.

RDF interfaces are available to many existing Prolog systems, e.g., the SWI-Prolog (see [100]). Several other kinds of SW rule and query frameworks exist as well [79], and the plausibility of rule systems has been investigated as a subtopic in many large-scale projects, such as the *On-To-Knowledge*, *OntoWeb*, *Knowledge Web*, and the *KT Web* [75, 76, 41, 43]. General purpose rule (representation) languages, however, still remain relatively non-standardised and rule systems from different vendors are rarely interoperable [30]. Perhaps the most promising standardised representation of a rule language is the Semantic Web Rule Language (SWRL). SWRL includes a high-level abstract syntax for Horn-like rules in both the OWL DL and OWL Lite sublanguages of OWL [57]. SW rule and query technologies are currently being developed at the W3C (see [65]). However, there already exists several non-standard SW rule and query frameworks, including, e.g., *Jena*, *Sesame*, and *CWM*.

Nevertheless, despite the well-established nature of fuzzy modelling in general (see, e.g., [61]), it seems that fuzziness raises sceptic concerns among SW developers. Even if progress has been made, e.g., by developing fuzzy DL, fuzzy SWRL, and fuzzy RDF in general [87, 4, 77, 83, 52], fuzziness is not considered "necessary", but rather something that "could be quite useful and fairly easy to add" (see [29]). It would seem that these kinds of arguments apply to fuzzy deduction rather than to fuzzy description. Nevertheless, the standard set-theoretic interpretation theory of RDF (see [31]) does not easily extend to the more general truth-functional semantics in a standard manner.

1.3.5 Approaches to Fuzzy Modelling

The observation that certain applications require vague modelling and aggregating different sources of information forces the developers to extend the

syntax of logical statements and redefine the interpretation of logical rules. Considering the design, there are several options to choose from.

A straightforward way to extend crisp clauses is to associate facts with fuzzy truth values, modelled as real numbers $v \in [0,1]$, e.g., as $P(a,b) \leftarrow v$. This, however, easily leads to overly precise assertions (consider modelling a fuzzy number associated with a datatype as "4 hours") and the notion of fuzzy aggregation becomes too simple. Considering description logics, this extension leads modelling concepts (predicates) in terms of type-1 fuzzy sets (see, e.g., [87]).

Another extension is to use interval-valued truth values, e.g., $P(a,b) \leftarrow I, I \subset \epsilon([0,1])$, where $\epsilon([0,1])$ denotes the family of all closed intervals in $[0,1]$ (see, e.g., [94]).

An even more general (but also more complex) approach is to model truth values using type-1 fuzzy sets, e.g., $P(a,b) \leftarrow \widetilde{A}$ where $\widetilde{A} = \{(x, \mu_{\widetilde{A}}(x)) | x \in X\}$ itself is a fuzzy set (see, e.g., [53]). This approach clearly subsumes the point-like and the interval-valued truth values. According to the fuzzy extension principle, point-like truth values correspond to singleton fuzzy sets, and interval-valued truth values to step-wise fuzzy sets. Using type-2 (or higher order) fuzzy sets is also possible. This provides a basis for explicitly modelling, e.g., the uncertainty related to fuzzy models themselves.

The main benefit of migrating to the more expressive fuzzy models is that doing so, using overly precise models is not forced by the model architecture. In particular, using type-1 fuzzy sets enables modelling the vagueness of the assertions themselves (e.g. "(around) 4 hours"), and provides an intuitive basis for the weighted aggregation of overlapping facts. The price is that fuzziness proceeds in deduction in the spirit of the fuzzy reasoning traditionally formulated for fuzzy control systems (see, e.g., [38]). As a practical consequence, updating the knowledge base becomes much more expensive, compared to logic systems based on monotonic reasoning.

1.4 Organisation and Contribution

The chapters of the main text of the thesis establish a tour through the main parts of the study, integrating the different areas of work. The aim is providing an intuitive synthesis of the work, described in the context of the restricted use cases.

Chapter 1 (this chapter) presents the outline and the background of the work. It is mainly based on the introductions of the selected publications (and the related presentations).

Chapters 2 and 3 presents the theory and the application of type-1 fuzzy

logic programs. The text is mainly based on the publication 2. However, the chapters include a significantly more elaborated treatment of the topic. The previously unpublished parts include the more detailed definitions and the propositions of the theory, the proofs of the propositions, an elaborated discussion related to the linguistic construction of fuzzy models, and the definitions and discussion related to context-aware logic programs and context-specific interpretations.

Chapter 4 presents the process of inducing fuzzy models and implementing fuzzy knowledge agents with existing rule frameworks. The text is based on the publications 3, 4, and 5.

Chapter 5 presents a case study in the context of educational systems. The text is based on the publications 1 and 6. The previously unpublished parts include examples formulated with type-1 fuzzy logic programs, and discussion related to the use case of personalised studying.

Chapters 6 and 7 finally provide a concise, previously unpublished summary of the publications, and conclude the thesis with selected technical and application-specific notes and discussion, including an outline of the forthcoming areas of the related research.

In short, we claim that the two worlds of *logic programming* and *fuzzy reasoning* should coincide when practical knowledge applications via logic programming are concerned. In this thesis, we point out and analyse an approach that provides the necessary methodological and technical means achieving this goal.

The main contributions of the thesis may be summarised as follows:

1. Introduction and analysis of a novel type-1 fuzzy logic programming framework that bridges the two worlds of well-established Prolog programming and popular control-oriented fuzzy inference systems.

2. Introduction of context-aware logic programs and an analysis of the concepts of assertion and interpretation contexts.

3. Design and demonstration of a model-free fuzzy set induction algorithm based on well-understood decision tree learning.

4. Demonstration of an educational use case of applying the methods, including proof-of-concept implementations and empirical evaluations of the algorithms.

The thesis attempts to address the technical design of fuzzy knowledge agents from various aspects. As a results, the contributions include both analytical, empirical, and heuristic results.

Chapter 2

Type-1 Fuzzy Logic Programs

We will next establish the basic concepts and structure of fuzzy logic programs. In brief, we define and elaborate the notion of fuzzy logic databases and describe the basic properties of fuzzy logic programs. These properties include the built-in imprecision of implementations, the linearity of procedures, and the reduction of fuzzy logic programs to crisp logic programs in Prolog. The basic idea is that fuzzy logic databases provide the enabling technology for implementing (passive) knowledge agents.

The overall architecture and the terminology is inspired by logic programming and Prolog systems, and the model of computation by fuzzy control systems [16, 86, 38, 11, 78]. Of course, the basic definitions are designed so that the desired properties indeed hold.

To distinguish our fuzzy logic programming system from the other approaches, we call the following kinds of logical systems *type-1 fuzzy logic programs.*

2.1 Fuzzy Sets

Definition 1 (Fuzzy Set) *Assume an appropriate subset of* $\mathbb{R}$ *denoted by* X*. Fuzzy set* A *is a set of ordered pairs* $A = \{(x, \mu_A(x))|x \in X)\}$*. The function* $\mu_A(x) : X \rightarrow [0, 1]$ *is called the membership function of* A*.*

Assuming the definition is well-established, we may write a fuzzy set A in a more compact form simply as a function $A(x)$, $A(x) = \mu_A(x)$.

The set $\{x|A(x) > 0\}$ is called the *support* of A. If $\max(A) = 1$ we say that A is *normal*, otherwise *subnormal.*

Definition 2 (Typed Fuzzy Set) *Let* A *denote a fuzzy set. Let* D *denote a symbol of a known (data)type, drawn from the list of well-known (data)types. The pair* (A, D) *is called a typed fuzzy set.*

We consider types simply as semantic labels that identify certain classes of fuzzy sets. We say that two typed fuzzy sets *match* if they are associated with the same datatype symbol. When there is no risk of confusion, we may omit the type information.

In practice, we will use types for bookkeeping purposes when necessary, in order to prevent accidentally comparing semantically mismatching fuzzy sets. In applications, types typically establish a *type hierarchy* which enables a richer interpretation of types.

Let $f \wedge g$ denote $min(f(x), g(x))$ and $f \vee g$ denote $max(f(x), g(x))$. In the case of multiple min or max terms the parentheses may be omitted since the order of the operations is not significant. If f and g denote fuzzy sets, the operations $f \wedge g$ and $f \vee g$ may be called *fuzzy intersection* and *fuzzy union* (or *aggregation*), respectively.

Definition 3 (Firing Strength) *Let (A, D) and (B, D) denote two matching fuzzy sets. Let $w = \omega(A, B) = \bigvee_x A \wedge B$. The number w is called the firing strength of the match.*

Intuitively, the firing strength denotes the highest membership degree of the overlapping area of the fuzzy sets.

Definition 4 (Weighted Aggregation of Fuzzy Sets) *Assume a set of n pairs $\{(A_i, h_i)\}$ of matching fuzzy sets A_i and numbers $h_i \in [0, 1]$. Let*

$$\bigvee_i h_i A_i = h_1 A_1 \vee h_2 A_2 \vee ... \vee h_n A_n.$$

We say that $\bigvee_i h_i A_i$ is the weighted aggregation (or union) of the fuzzy sets $\{A_i\}$, weighted by $\{h_i\}$.

Clearly, $\bigvee_i h_i A_i$ is a fuzzy set (of the same type as the matching fuzzy sets).

2.2 Representing Fuzzy Sets

A concrete implementation of a system involving fuzzy models must encode the related fuzzy sets efficiently. Further, the representation language must be closed under the operations of *union* $(A \vee B)$ and *multiplication by a real number* $(hA, h \in [0, 1])$.

As a practical and a useful example, let us next consider a very simple approach of encoding fuzzy sets, based on membership functions with a trapezoidial base.

Definition 5 (Trapezoidial Function) *A trapezoidial function τ can be specified by four parameters $(a_1, b_1, a_2, b_2), a_1 \leq a_2$ as follows: If $a_1 < a_2$ define*

$$\tau(x; a_1, b_1, a_2, b_2) = \begin{cases} 0, & x \leq a_1 \\ \frac{b_2 - b_1}{a_2 - a_1}(x - a_1) + b_1, & a_1 \leq x \leq a_2 \\ 0, & a_2 \leq x. \end{cases}$$

If $a_1 = a_2$ we require that $b_1 = b_2$ and define

$$\tau(x; a_1, b_1, a_2, b_2) = \begin{cases} 0, & x \leq a_1 \\ b_1, & x = a_1 \\ 0, & a_1 \leq x. \end{cases}$$

Intuitively, the points $(a_1, b_1), (a_2, b_2)$ denote the two upper corners of the related trapezoid. When $a_1 < a_2$, we say that the segment

$$\{(x, y) | y = \frac{b_2 - b_1}{a_2 - a_1}(x - a_1) + b_1, a_1 \leq x \leq a_2\} \tag{2.1}$$

is the *hypotenuse* of the trapezoid. Trapezoidial functions with $a_1 = a_2, b_1 = b_2$ are called *singleton (trapezoidial) functions.*

Assume two trapezoid functions, $\tau_1 = (x; a_{11}, b_{11}, a_{12}, b_{12})$ and $\tau_2 = (x; a_{21}, b_{21}, a_{22}, b_{22})$. The firing strength of the functions can be computed by checking the co-ordinates where the segments $((a_{11}, 0), (a_{11}, b_{11}))$, $((a_{11}, b_{11}), (a_{12}, b_{12}))$, $((a_{12}, b_{12}), (a_{12}, 0))$, and $((a_{21}, 0), (a_{21}, b_{21}))$, $((a_{21}, b_{21}), (a_{22}, b_{22}))$, $((a_{22}, b_{22}), (a_{22}, 0))$ intersect. In brief, this involves a tedious process of testing the various cases, and in a certain case, solving a system of the two related linear equations (spanned by the segments (2.1)).

A function with a finite *trapezoidial base* i.e. a *piecewise trapezoidial function* can be specified as an aggregation of trapezoidial functions:

$$\Theta = \bigvee_i \tau_i, i = 1, 2, ...n. \tag{2.2}$$

We say that n denotes the *cardinality* of the base.

Assume two overlapping trapezoidial functions $\tau_1 = (a_1, b_1, a_2, b_2), \tau_2 = (c_1, d_1, c_2, d_2)$. We can alternatively represent $\tau_1 \vee \tau_2$ using at most four other non-overlapping trapezoidial functions $\tau_1 \vee \tau_2 = \tau_3 \vee \tau_4 \vee \tau_5 \vee \tau_6$. Thus, we can always *harmonise* the specification of a piecewise trapezoidial function with a base cardinality of n with another piecewise trapezoidial function with a base cardinality of max $4n$. However, harmonisation typically increases the amount of memory required for *representing* piecewise trapezoidial functions in implementations.

It is easy to see that the family of functions with trapezoidial base is closed under fuzzy union, intersection, and multiplication by a real number. Further, working with functions with a trapezoidial base is straightforward. For instance, computing the firing strength of two piecewise trapezoidial functions Θ_1 and Θ_2 can be reduced to computing the firing strength of two trapezoidial functions, i.e. going through the bases of Θ_1 and Θ_2. Many useful membership functions may be reasonably approximated with piecewise trapezoidial functions by choosing a sufficiently large base (cardinality).

While the question which important membership functions may be meaningfully captured with functions with a trapezoidial base is out of scope (because the question is related to modelling heuristics), the question of optimising the base cardinality of representations is rather pivotal. Recording a piecewise trapezoidial function consumes memory, and operations with piecewise trapezoidial functions take time, with relationship to the cardinality of the related function bases (n). In particular, deductions involving many steps (or deep proof trees) may easily yield complex and thus also slow models.

Assume a canonised piecewise trapezoidial function Θ with a base cardinality of n. We wish to optimise Θ using an other piecewise trapezoidial function Θ' with a base cardinality of $n - 1$.

The criteria of successfully optimising Θ' is now underlined by the domain of membership functions of fuzzy sets. Intuitively, we would like to preserve both the support, the shape, and the area (usually the integral) of Θ as faithfully as possible.

Two optimisation strategies are now worth elaborating:

1. Perhaps the most simple strategy is discarding the *least significant* base function τ_s with the smallest area (or with the smallest height, i.e. a modified α-cut). Of course, this approach does not preserve the support.

2. Another strategy is examining all of the adjacent trapezoids (assume a canonised function specification) and replacing two with a new trapezoid while trying to minimise the error due to the replacement. This strategy involves a variety of design decisions. (The main distinction with the previous strategy is the resulting model may preserve the support of the fuzzy set.)

In the worst case scenario (all of the base functions have the same area), the first approach may result the error of $\frac{1}{n}\%$ in the area. In the worst case scenario (all of the base functions have the same area and none are adjacent), the latter approach results the same error. However, in more well-behaving

applications, it is possible to some extent preserve the support and the shape characteristics of the piecewise trapezoidial function as a fuzzy membership function.

We may conclude these simple observations as a proposition which characterises the *upper limit of intrinsic imprecision* of concrete fuzzy systems:

Proposition 1 (Intrinsic Imprecision Lemma) *Assume Θ is a piecewise trapezoidial membership function with a base cardinality of $n \geq 2$. Approximate Θ with another piecewise trapezoidial membership function Θ', with a base cardinality of $n-1$. The proportional error of area is at most proportional to $\frac{1}{n}$.*

Note that the process of optimisation indeed requires design insight. For instance, we could always easily (blindly) optimise a base of n trapezoidial functions with a single trapezoidial function while completely preserving the area, simply by establishing a rectangle-shaped function with an equal size.

We say that the error denotes the *intrinsic* of *inner imprecision* of a concrete fuzzy system. We might assume that in well-behaving and well-known applications, the effective imprecision is typically smaller. Nevertheless, in application design it makes sense to *ensure that the intrinsic imprecision is less significant than the imprecision due fundamental vagueness of the domain and the imprecision related to the modelling strategy.*

Proposition 1 and the optimisation strategy 1 may be trivially generalised to many other types of base functions as well, in particular, functions with a Gaussian base:

$$\bigvee_i \gamma_i, \quad \gamma_i(x; c_i, d_i) = e^{-(\frac{x-c_i}{d_i})^2}.$$

Further, we may allow constructing membership functions whose base consist of a mixture of various kinds of parametric functions. The base type, however, effects not only to modelling but also the implementation, i.e. the memory requirements (base cardinality times the number of parameters) and costs of computing the firing strengths.

2.3 Syntax

We will next define the simplified syntax of type-1 fuzzy logic programs which provides the basis of modelling, queries, and the related interpretations. Later, we will relax this syntax, leaving room for more complex programs, perhaps including (crisp) procedures invisible to the fuzzy semantics.

2.3.1 Simple Grammar

Intuitively, a *fuzzy logic program* is a finite database that consists of *clauses*, each of which is *modelled* with a (typed) fuzzy set.

Clauses include *facts* and *rules*. Facts associate individual *objects* with *properties* using *base predicates* and *fuzzy evaluations*. Rules assert general statements about objects and may in addition introduce *derived predicates*, definition of which is asserted by the design of the rules.

Let us first describe a simple abstract syntax for logic programs in terms of context-free grammar[1] and then consider few examples. To improve readability, we agree that each rule of the grammar may accept one or more whitespace characters before or after it.

Definition 6 (Grammar of Simple Fuzzy Logic Programs)

```
program   ::= rule*
rule      ::= head (':-' body)? '.'
head      ::= predicate
body      ::= predicate (',' predicate)*
predicate ::= property '(' term (',' term)* ')' model
term      ::= constant | variable
property  ::= [a-z_] string # etc. encoding
model     ::= '~' constant
constant  ::= [a-z_] string | '"' (string? ' ')* '"'
variable  ::= [A-Z] string
string    ::= [a-zA-Z0-9_-:]+
```

Further, for pinpointing potential typos in programs, we may in addition require that if a variable v_k appears in the head of a rule, it appears also in the body of a rule.

The chief distinction with crisp Prolog rules is the introduction of the models; each (goal) predicate is associated with a model, e.g., `likes(john,susan)~a.`

We follow the usual abbreviation of facts and write

```
likes(john, susan)~a.
```

instead of

```
likes(john, susan)~a :- .
```

[1] The notation for the grammar follows the simple Extended Backus-Naur Form (EBNF) notation found, e.g., in [12], with the exception that we use the character # for denoting the beginning of a comment line.

Note that for simplicity, we do not (at this point) explicitly define the interpretation of *hierarchical structures* within predicates which might be considered useful in logic programs. Effectively, this excludes structures involving *functors*, for instance:

```
exhibition("Muse Rodin",address("77 Rue de Varenne",paris)).
```

Further, for simplicity, the character classes of the names are overly simplified, and so on.

2.3.2 Structure of Logic Programs

Technically, a logic program is a sequence of rules. As an example, let us consider the following type-1 fuzzy logic program P_1:

```
likes(john,susan)~a.
likes(john,theThinker)~b.
classicalScuplture(theThinker)~c.
likes(susan,X)~r0 :- rich(X)~r1,likes(X,Y)~r2,
                     classicalScuplture(Y)~r3.
```

If we ignore the fuzzy models (`a, b, c, r0, r1, r2, r3`), we get the *crisp reduction* of P_1, P_1':

```
likes(john,susan).
likes(john,theThinker).
classicalScuplture(theThinker).
likes(susan,X) :- rich(X),likes(X,Y),classicalScuplture(Y).
```

A crisp reduction is obtained simply by removing the fuzzy models from a fuzzy logic program. The main virtue of crisp reductions is that they serve as crisp logic programs in Prolog.

Intuitively, the program P_1' says that John likes Susan and also Rodin's classical sculpture, *The Thinker*. However, even if John likes a classical sculpture, we can not conclude that Susan likes John because John is not known to be rich. The interpretation of P_1 is more complex and depends on the fuzzy models assigned to the predicates.

Technically, rules consist of two parts: *head* and *body*. Series of rules with matching heads (above `likes/2`) establish *procedures*. The number 2 in `likes/2`, i.e. the number of the arguments, denotes the *arity* of the predicate. Variable-free rules with empty bodies are called *facts*.

The predicates that appear in the rule body as *goals* may be perceived as search objectives. The fuzzy models assigned to the predicates may be

perceived as *match patterns*. Goals *succeed* if appropriate matching facts can be *derived* from the database. Further, if all of the goals of the rule body *succeed*, the rule itself succeeds (or *fires*) and the head part of the rule is taken as a fact, i.e. derived by the rule. Intuitively, the rule body describes a conjunction of goals that must be succeeded (or *satisfied*) one by one, for the head to succeed.

Complex rules may include references to variables that help linking information obtainable via different predicates via *instantiation*.

2.3.3 Queries

Logic programs are applied via executing *queries*. The introduction of fuzzy models provides a basis of making two kinds of queries: ask and verify queries.

Ask queries accept a single goal without an associated fuzzy model. The goal identifies a single predicate with constants or variables as arguments. An ask query returns all the evaluations of the predicate. For instance, consider the following query:

```
?-likes(john,Y).
```

Intuitively, the describe query provides an answer to a question *"Tell me what John likes"* by returning a *result set* of the evaluations that match the query, according to the assigned fuzzy models. Considering the previous example, we might expect the query to return a result $\langle Y = susan \sim e_1, Y = theThinker \sim e_2 \rangle$ where e_1, e_2 denote fuzzy models (sets).

Verify queries accept a conjuction of goals which identifies a *proposition* whose *degree of satisfiability* is to be evaluated. A verify query returns a set of evaluations and the associated firing strengths. For instance, consider the following query:

```
?-likes(john,Y)~f,classicalScuplture(Y)~g.
```

Intuitively, the query provides an answer to a question by returning a result set of the evaluations that match the query, according to the assigned fuzzy models. Considering the example, we might expect the query to return a result $\langle Y = theThinker \sim w_1 \rangle$ where $w_1 \in [0, 1]$ denotes a firing strength. Depending on the fuzzy models f, g, the query might be read like: *"To what degree it holds that John likes very much a sculpture* (Y) *that is not particularly classical?"*

Of course, clauses and logic programs do not necessarily record "truth" nor establish semantically correct definitions concerning the object domain.

More precisely, programs simply encode certain information the (human) designer of the application for some reason considers useful. The selection of the fuzzy models may thus considerably vary upon application. (As an intuitive use case, consider e.g. applications using empirically constructed linguistic terms.)

2.4 Interpretation

Let us next consider the process of answering queries.

Definition 7 (Goal) *A goal is a triple (p, v, M) where p is a predicate symbol, v is a list of arguments, and M is a fuzzy set. Arguments may include names of individuals and variable symbols.*

A *variable* is a local symbol within a rule which is *instantiated* to matching objects or other variables during the evaluation of a rule.

We say that M is the *model* if the goal $p(v)$. A goal is called a *ground* (goal) if it includes no variable symbols. To simplify the treatment, we assume that the type of the fuzzy sets as a model is determined by the name of the related predicate symbol.

Goals are meaningful only within particular contexts, namely rules and queries.

Definition 8 (Fuzzy Rule) *A fuzzy rule is pair (h, b) where h is a goal that denotes the head of the rule and the body $b = \langle b_i \rangle$ is a list of goals.*

Definition 9 (Fuzzy Fact) *A fuzzy fact is a fuzzy rule with an empty body and a ground goal in the head.*

Rules which are not facts are called *non-trivial.*

Let s, t denote two fuzzy facts with goals $(p_s, v_s, M_s), (p_t, v_t, M_t)$. If $p_s = p_t$ and $v_s = v_t$ we say that the facts s and t *overlap.*

Definition 10 (Semantically Equivalent Facts) *Let $S = \{s_i\}, i \in I$ and $T = \{t_j\}, j \in J$ denote two sets of overlapping fuzzy facts with the associated models $\{M_i\}$ and $\{M_j\}$.*

If $\bigvee_{i \in I} M_i = \bigvee_{j \in J} M_j$, we say that S and T and semantically equivalent.

When a rule r is evaluated, the body of the rule may get satisfied with a ground argument configuration γ (i.e. a list including only individual symbols) with a firing strength $\alpha \in [0, 1]$. We say that the rule r *deduces* a fact

$((p_h, \gamma, \alpha w M_h), [])$ where p_h is the predicate symbol and M_h the model that appears in the head of the rule.

The model of the deduced fact is thus the model of the head of the rule, multiplied by the weight of the rule and the firing strength of the evaluated body.

Let us next consider the process of evaluating ask queries.

Definition 11 (Ask Query) *An ask query is a goal with an empty model. When applied to a database of rules, an ask query returns a complete list of ground argument configurations that match the query, associated with the related fuzzy models (sets).*

An *argument configuration* is simply a list of applicable arguments. A *ground* argument configuration includes only symbols of individuals.

An ask query performs the function of a Prolog interpreter when asked to retrieve all answers that satisfy the query. Alongside the processing, an ask query in addition computes matches related to the fuzzy models. Each successful reduction step returns a fuzzy model which finally evaluates each ground argument configuration with a fuzzy model.

Due to this close relationship, the implementation of queries with fuzzy logic programs can be technically based on their crisp reductions in Prolog: Assume a Prolog interpreter, a logic program P' (a crisp reduction of some fuzzy logic program P) and an ask query q. When asked to perform the query q, the Prolog interpreter executes a search, constructing a series of parameter configurations, while recursively reducing the query goal to subgoals. When a solution is found, the interpreter reports a ground argument configuration γ_i, the arity of which corresponds to the arity of the predicate of the query. (Assume that the argument configuration includes also the individuals in q, in their proper positions.)

Associate each result argument configuration γ_i with its *proof tree* Υ_i. A proof tree consists of nodes and edges that represent the goals reduced during the computation of γ_i. In brief, the root a proof tree shows the query goal with the argument configuration γ_i, and each branch recursively reports a successful reduction of the subgoals by telling which rules were applied.

Proof trees may either be constructed by a native fuzzy interpreter or they can be read from the *trace* of the related Prolog interpreter which reports of the resolvents during computation.

Algorithm 1 (Fuzzy Ask Query Machine)

Input: A finite fuzzy logic program P and an ask query q.

Output: A (potentially empty) list of solutions, i.e. pairs of successful ground attribute configurations and the related fuzzy evaluations.

1. Compute a crisp reduction of P, P'. Compute all answers to the query q from the program P', using a (standard) Prolog interpreter. This provides a finite list of solutions encoded as pairs of successful ground attribute configurations and the proof trees $\langle(\gamma_i, \Gamma_i)\rangle, i = 1, 2, ..., n$. Assume the branches of the proof trees are organised according to the order of the conjucted goals of the related rule bodies.

2. If the list of solutions is empty, halt and return an empty list [].

3. Associate each attribute configuration γ_i with a fuzzy set M_i computed from the related proof tree Γ_i via the recursive process of evaluation as follows:

   (a) Pick a leaf node t from Γ_i.

   (b) If t is a fact r, replace t with a fuzzy evaluation M_r, where M_r is the model of (the head of) r.

   (c) Repeat the above steps (a) and (b) until all leaf nodes have been replaced with fuzzy models.

   (d) If there is more than one node in the tree, pick a node t (corresponding to a rule r) with only leaf children $\{t_k\}$, each of which has already been replaced with a fuzzy model M_k. Let M_r denote the model of (the head of) r and $\{A_r^k\}$ the models of the body goals in r. Replace the subtree t with a fuzzy model

   $$(\bigwedge_k \omega(A_r^k, M_k)) \otimes M_r \tag{2.3}$$

   and repeat from d.

   (e) When there is only one node left in the tree, halt. The requested fuzzy set M_i is the fuzzy model associated with the remaining node.

4. Return a list of parameter configurations, associated with the fuzzy evaluations, $\langle(\gamma_i, M_i)\rangle$

If we choose $\otimes = \wedge$ in (2.3), we essentially get a recursive version of the well-known max-min composition. Alternatively, we may choose $\otimes = \cdot$ (multiplication).

As suspected, ask queries establish a basis of verify queries.

Definition 12 (Verify Query) *A verify query consists of a body of a fuzzy rule. When applied to a database of rules, a verify query returns a complete list of ground argument configurations that match the query, associated with the related firing strengths.*

The answer of a verify query q may be computed basically as in the case of an ask query. One strategy is adding a new temporal rule r, equipped with an appropriate head and the query as a body, $r = (h_q, q)$, to the rule database, and computing a describe query for h_q. The resulting weights may be extracted from the corresponding describe query when computing the final iteration of the weights of the models in Algorithm 1.

Due to the construction of the Algorithm 1, verify queries of type-1 fuzzy logic programs and crisp queries of Prolog programs are closely related.

Definition 13 (Meaning of a Logic Program) *Let P denote a logic program. The meaning of P, $M(P)$ is the set of facts that can be deduced from P.*

Proposition 2 (Crisp Reduction Lemma) *Let P' denote the crisp reduction of a logic program P. If fuzzy evaluations are ignored, $M(P') = M(P)$.*

Proof. The proposition follows directly from the definition of the Algorithm 1: the answers of a fuzzy query include exactly the attribute configurations of the related crisp query. □.

A rule may deduce a fact that overlaps with the facts already written in the rule database. In addition, the program may include duplicate rules which repeat certain solutions in a query. This is acceptable: we may aggregate overlapping facts before application of the rules, or alternatively, break fuzzy facts into semantically equivalent, overlapping facts without confusion. In other words, the process of computing deductions is distributive over fuzzy union.

Definition 14 (Canonised Meaning of a Logic Program) *Let P denote a type-1 fuzzy logic program. We say that the canonised (or harmonised) meaning of P, $M^H(P)$, is the set of facts where the sets of overlapping facts $s^i_k \in S^i$ with models M^i_k in P are replaced with a single fact s^i_0, model of which, M^i_0, is the aggregation of the models of the respected overlapping facts:*

$$M^i_0 = \bigvee_k M^i_k.$$

In short, the canonised meaning of a logic program does not include overlapping facts but preserves the semantics of the meaning.

Proposition 3 (Fact Canonisation Lemma) *Let P denote a logic program. The meaning of the logic program, $M(P)$, and the canonised meaning of the logic program, $M^H(P)$, are semantically equivalent.*

Proof. The statement follows directly from the definitions (4), (10), and (14). □.

Let P_1, P_2 denote two logic programs. We may aggregate the clauses of P_1 and P_2 together and establish a third logic program, P_3. To denote this, we write $P_3 = P_1 \cup P_2$. Note that the two related meanings $M(P_1) \cup M(P_2)$ and $M(P_1 \cup P_2)$ are usually different.

It is however important that adding semantically equivalent facts does not alter the canonised meaning of a logic program.

Proposition 4 (Linear Aggregation Lemma) *Let P denote a finite type-1 fuzzy logic program and f_1, f_2, f_3 three overlapping facts with models M_1, M_2, M_3, so that $M_3 = M_1 \vee M_2$.*

Let $P_1 = P \cup f_1 \cup f_2, P_2 = P \cup f_3$ denote two logic programs. It follows that $M^H(P_2) = M^H(P_1)$.

Proof. Assume M_k denotes the evaluation of f_k, $k = 1, 2, 3$. Let q denote an ask query. Let Γ_1 and Γ_2 denote the lists of answers of the describe queries $q(P_1)$ and $q(P_2)$. Let us next consider the case $\otimes = \cdot$ (multiplication); the case $\otimes = \wedge$ can be analysed in a similar manner.

Two main cases have to be examined. We have to show that $M^H(P_2) \subset M^H(P_1)$ and $M^H(P_2) \supset M^H(P_1)$.

Consider a single answer χ_3 in Γ_2, proof of which, Υ_3, includes a node f_3.

Since f_k are equal in terms of configurations, and Γ_k include all of the evaluations of all of the configurations, Γ_1 includes (otherwise identical) answers χ_1 and χ_2 where f_1 and f_2 appear in the place of f_3, respectively.

Let E_k denote the evaluation of Υ_k. We have to show that $E_3 = E_1 \vee E_2$. We sketch a proof by induction, based on the depth d of Υ_k.

Case $d = 1$: Υ_k^d includes only a single node, f_k. It gets replaced with the model M_k. Thus, we get $M_3 = M_1 \vee M_2$, as requested.

Assume the proposition holds for $d = n$.

Case $d = n + 1$: Υ_k^d includes f_k as a leaf node. Let p_k denote the parent node of f_k.

If ω_k, $\omega_k = \omega(A_k, M_k)$ is not the smallest firing strength among its siblings (i.e. the *firing weight*), it does not contribute to the model of its parent.

Assume ω_3 is a firing weight. We know that $\omega_3 = \mathtt{max}(\omega_1, \omega_2)$. Thus, both ω_1 and ω_2 must also be firing weights. We get $w_r\omega_3 M_r = w_r\omega_1 M_r \vee w_r\omega_2 M_r$ (see (2.3)) and the proposition holds for the evaluations of the parent nodes p_k.

Let us then examine the other direction of the proof.

Consider a single answer χ_1 in Γ_1, proof of which, Υ_1, includes a node f_1. Again, we may find answers χ_2 in Γ_1 and χ_3 in Γ_2 with identical configurations and proof trees Υ_2 and Υ_3, where f_2 and f_3 appear in the place of f_1, respectively.

Consider the induction proof:

Case $d = 1$: Υ_k^d includes a single node, f_k. It gets replaced with the model M_k. Thus, we get again $M_3 = M_1 \vee M_2$, as requested.

Assume the proposition holds for $d = n$.

Case $d = n + 1$: Υ_k^d includes f_k as a leaf node. Again, $\omega_3 = \mathtt{max}(\omega_1, \omega_2)$. Assume $\omega_1 \leq \omega_2$ and that ω_1 is the firing weight (the case $\omega_2 < \omega_1$ similarly).

If ω_2 is also a firing weight, so must ω_3 be. Thus, the proposition holds as before.

If ω_2 is not a firing weight, neither is ω_3. In this case, some other weight ω_k^x determines the (greater) weights of the parent nodes of f_2 and f_3. These effectively overrun the effect of f_1 in the canonised evaluation.

Thus, we get $M^H(P_1) = M^H(P_2)$. □.

Intuitively, the linear aggregation lemma states that the canonised meaning of a logic program is not determined by the organisation of its individual, semantically equivalent facts.

The main result of this section is the following *Linear Aggregation Theorem* which generalises the above observation to clauses.

Proposition 5 (Linear Aggregation Theorem) *Let P denote a finite type-1 fuzzy logic program. Let $R_k, k = 1, 2, 3$ denote three rules that are identical expect for the models of the rule head M_k:*

$$R_k : cl_0(\alpha_0) \sim M_k \text{ :- } cl_1(\alpha_1) \sim r_1, cl_2(\alpha_2) \sim r_2, ..., cl_m(\alpha_m) \sim r_m.$$

Define logic programs P_1 and P_2 as follows: $P_1 = P \cup R_1 \cup R_2, P_2 = P \cup R_3$ and assume $M_3 = M_1 \vee M_2$. It follows that $M^H(P_2) = M^H(P_1)$.

Proof. The proposition follows from the lemma by noticing that the proof does not change if we allow the evaluations originate from the rules R_k. □.

2.5 Type-1 Fuzzy LPs in Prolog

The Fuzzy Ask Query Machine (Algorithm 1) is closely related to Prolog. As suspected, it is possible to implement a type-1 fuzzy logic program interpreter in Prolog. This also removes the urgent need of implementing a yet another logic programming framework, and most importantly, proving its flawlessness.

Even if the proof trees of the answers can be read from the program trace, it is not necessary. Without going into the details of Prolog programming, let us next consider a simple approach of explicitly constructing proof trees from which fuzzy evaluations can be easily computed.

Consider the program P_2:

```
classicalScuplture(theThinker)~m1.
rich(paul)~m2.
rich(john)~m3.
likes(paul,theThinker)~m4.
likes(john,theThinker)~m5.
likes(susan, theThinker)~m6.
likes(susan,X)~r0 :- rich(X)~r1,likes(X,Y)~r2,
                     classicalScuplture(Y)~r3.
```

We may encode the program P_2 as a standard (crisp) Prolog program P_2' as follows:

```
c_classicalScuplture(theThinker,m1).
c_rich(paul,m2).
c_rich(john,m3).
c_likes(paul,theThinker,m4).
c_likes(john,theThinker,m5).
c_likes(susan,theThinker,m6).
c_likes(susan,X,E0) :-
  c_rich(X,E1),
  c_likes(X,Y,E2),
  c_classicalScuplture(Y,E3),
  E0 = [r0,[[E1,r1],[E2,r2],[E3,r3]]].
```

We say that P_2' is a *(Prolog) implementation* of P_2. Other equivalent implementations exist.

The program P_2' follows the overall structure of the crisp reduction of P_2 and associates clauses with fuzzy models in terms of constants, variables, and list structures. In short, the models associated with the goals of the clauses

have been identified with names `m1, m2, m3, m4, m5, m6` and `r0, r1, r2, r3`, and the fuzzy predicate `likes/2` has been represented as a crisp predicate `c_likes/3`, with respective variables.

In Prolog, the question

```
?-c_likes(susan,X,E).
```

now returns (output abbreviated)

```
X = theThinker E = m6 ;
X = paul       E = [r0,[[m2, r1], [m4, r2], [m1, r3]]] ;
X = john       E = [r0,[[m3, r1], [m5, r2], [m1, r3]]] ; No
```

Each of the pairs (e.g. $(X = theThinker, E = m_6)$) represents a fuzzy evaluation (e.g. $(X = theThinker \sim m_6)$). The list structures denote the parse trees for computing the deduced fuzzy models.

For instance, the list `[r0, [[m2, r1], [m4, r2], [m1, r3]]]` identifies the formula

$$(\omega(m_2, r_1) \wedge \omega(m_4, r_2) \wedge \omega(m_1, r_3)) \otimes r_0. \tag{2.4}$$

By choosing appropriate models and fixing the operator $\otimes$ we may thus evaluate the query in a variety of ways.

Following the above approach, we may implement a generic interpreter of fuzzy logic programs in Prolog. We may either simply instruct writing programs as above, or implement a general-purpose interpreter interface which appropriately encapsulates the implementation, e.g., annotating appropriate Prolog programs with lists of fuzzy models.

From the algebraic point of view, implementing procedures related to working with models (e.g., computing the firing strengths of fuzzy sets) is not necessary. An implementation may simply return evaluations as lists as above, denoting the parse trees of the formulas of the related fuzzy models.

Finally, it is interesting to consider logic programs with overlapping facts. Let P_3 denote a logic program which is otherwise identical to P_2, expect that the fact

```
rich(paul)~m2.
```

has been replaced with following two facts:

```
rich(paul)~m2a.
rich(paul)~m2b.
```

Encode the respected Prolog program P_3' as above, and execute the (standard) query:

```
?-c_likes(susan,X,E).
```

We get

```
X = theThinker  E = m6 ;
X = paul        E = [r0, [[m2a, r1], [m4, r2], [m1, r3]]] ;
X = paul        E = [r0, [[m2b, r1], [m4, r2], [m1, r3]]] ;
X = john        E = [r0, [[m3, r1], [m5, r2], [m1, r3]]]  ; No
```

In other words, we get an additional evaluation for the added fact, as expected. The canonised model m_2 of the related deduced fuzzy fact

```
likes(susan,paul)~m2.
```

is now:

$$(\omega(m_{2a}, r_1) \wedge \omega(m_4, r_2) \wedge \omega(m_1, r_3))r_0 \vee (\omega(m_{2b}, r_1) \wedge \omega(m_4, r_2) \wedge \omega(m_1, r_3)) \otimes r_0 =$$

$$((\omega(m_{2a}, r_1) \vee \omega(m_{2b}, r_1)) \wedge \omega(m_4, r_2) \wedge \omega(m_1, r_3)) \otimes r_0 =$$

$$(\omega(m_{2a} \vee m_{2b}, r_1) \wedge \omega(m_4, r_2) \wedge \omega(m_1, r_3)) \otimes r_0.$$

Thus, when the canonised meaning is considered, the programs P_2 and P_3 are semantically equivalent whenever $m_2 = m_{2a} \vee m_{2b}$.

2.6 Applications

Type-1 fuzzy logic programs follow the control-theoretic approach of designing knowledge bases and establish a system of programming with fuzzy clauses of their own right. It is, however, instructional to consider the process of modelling crisp systems in terms of type-1 fuzzy logic programs.

2.6.1 Basic Crisp Reductions

Anticipating the following discussion, we begin by establishing some useful definitions.

Definition 15 (Neutral Fuzzy Set) *Let A denote a fuzzy set. If $A = \{(x, \mu_A(x)|x \in [0,1], \mu_A : [0,1] \rightarrow [0,1]\}$ we say that A is a neutral fuzzy set.*

In addition, certain trapezoidial neutral fuzzy sets deserve given names:

Definition 16 (Singleton, False, True, and Empty Models) *Let* $\tau_S = \tau_S(x;\, a, b) = \tau(x; a, b, a, b)$, $\tau_F = \tau(x; 0, 1, 0, 1)$, *and* $\tau_T = \tau(x; 1, 1, 1, 1)$. *We say that* τ_S, τ_F, *and* τ_T *are called singleton, false, and true fuzzy models, respectively. Finally, we say that* $\tau_\emptyset = \tau(x; a, 0, b, 0)$ *is called an empty fuzzy model.*

Let A denote a fuzzy model. If $A = \tau_S(x; a, 1)$ we say that A is *precise*. If $A = \tau_S(x; a, b), b < 1$ we say that A is *subnormally precise*. If $A = \tau_F$ or $A = \tau_T$, we say that A is *crisp*.

Let us then consider the following crisp logic program P_4:

```
classicalScuplture(theThinker)~m1.
rich(paul)~m2.
likes(susan,theThinker)~m3.
likes(paul,theThinker)~m4.
likes(john,theThinker)~m5.
likes(susan,X)~r0 :- rich(X)~r1,likes(X,Y)~r2,
                     classicalScuplture(Y)~r3.
```

The introduction of the fuzzy models adds certain contextual freedom to modelling. Intuitively, we might use P_4 to record a fact that Paul is *rather* rich, Paul likes Rodin's work only a *a little*, and so on.

As an attempt to represent the underlying crisp program P_4', let us choose the following neutral fuzzy models m_1, m_2, m_3, m_4, m_5 and r_0, r_1, r_2, r_3 as follows: $m_i = r_j = \tau_T$.

In this case, the fuzzy ask query

```
?-likes(susan,X).
```

returns $\langle X = theThinker \sim m_3, X = paul \sim e_1 \rangle$ where (compare with (2.4))

$$e_1 = (\omega(m_2, r_1) \wedge \omega(m_4, r_2) \wedge \omega(m_1, r_3)) \otimes r_0 = 1 \otimes r_0 = \tau_T.$$

Then, consider the fuzzy ask query

```
?-likes(susan,john).
```

We get an empty answer $R_1 = []$ and the related Prolog implementation would answer `No`.

Following the logic programming tradition, we take the stance that *an empty answer means that nothing matches the query* i.e. the question is *not provable.*

In particular, an empty answer does not imply *is false*. Thus, strictly speaking, the program P_4 simply does not reveal wheter Susan likes John or not.

Fuzzy logic programs, however, enable encoding more information than their crisp counterparts. Consider the above program with a small change: set $m_4 = \tau_F$. Intuitively, this means that John does *not* like the sculpture in question. (Notice that the crisp reduction of the logic program remains unchanged.)

The fuzzy ask query

```
?-likes(susan,X).
```

will return $R_2 = \langle X = theThinker \sim m_3, X = paul \sim e_2 \rangle$ where $e_1 = \tau_\emptyset$.

In other words, because Paul does not effectively like *The Thinker*, we can not conclude that Susan likes Paul.

Notice that we would get the same answer by choosing $m_4 = \tau_\emptyset$ (or in this case, any model that does not include τ_T).

Again, the process of drawing conclusions must be proceed with caution. In particular, a non-existing answer and an evaluation with an empty model are not the same thing. Further, we define that in applications, $\tau_\emptyset$ does not in general mean the same as *is false*. (Certain applications may define the meaning of $\tau_\emptyset$ otherwise, just as it makes sense to assert the closed world assumption in some logic programming applications.)

We interpret the above answer so that *a fuzzy logic program represents the partial information that is known about a certain domain.* In other words, a "negative" conclusion may only follow from a proof that reports explicitly or implicitly asserted "negative" information.

2.6.2 Modelling "Not" By Adding Rules

The concrete lesson of the above exercise is that if we want to be able to derive "negative" conclusions, we must add new rules to the database, essentially desribing new *cases* of the existing procedures.

Consider adding a rule:

```
likes(susan,X)~s0 :- rich(X)~s1.
```

which is supposed to capture the idea that "If X is *not* rich, then Susan does *not* like X."

Now it must be decided how "not" is modelled in the program. One approach is to think an empty set as the model of "not". In some applications, however, this approach would (in error) equate "not having information" with "being false".

A better strategy is modelling "not" with the explicitly false model, by choosing

$$s_0 = \tau_F \text{ and } s_1 = \tau_F. \tag{2.5}$$

This is in line with the idea of thinking deductions as proofs, and escapes the problem of multiple interpretations in applications. It again also genuinely *adds information* to the logic program, by explicitly enabling negative deductions.

Introduce a yet another fact `rich(john)~m6.` and consider the resulting program P_5:

```
classicalScuplture(theThinker)~m1.
rich(paul)~m2.
rich(john)~m3.
likes(susan,theThinker)~m4.
likes(paul,theThinker)~m5.
likes(john,theThinker)~m6.
likes(susan,X)~r0 :- rich(X)~r1,likes(X,Y)~r2,
                     classicalScuplture(Y)~r3.
likes(susan,X)~s0 :- rich(X)~s1.
```

Choose models $m_1 = m_2 = m_3 = m_5 = r_0 = r_1 = r_2 = r_3 = \tau_T$ and $m_4 = m_6 = s_0 = s_1 = \tau_F$. In other words, we have explicitly recorded more information about John (John is not rich), and stated that Paul does not really like *The Thinker*. Further, we have said that Susan does not like the things explicitly reported non-rich.

The fuzzy ask query `?-likes(susan,X).` now provides an answer $\langle X = theThinker \sim m_3, X = paul \sim p_1, X = john \sim j_1, X = paul \sim p_2, X = john \sim j_2 \rangle$ where $m_4 = \tau_T, p_1 = \tau_\emptyset, j_1 = \tau_\emptyset, p_2 = \tau_\emptyset$, and $j_2 = \tau_F$. Considering the canonised meaning of the query, we may thus deduce facts `likes(susan,paul)~p.` and `likes(susan,john)~j.` where $p = p_1 \vee p_2 = \tau_\emptyset$ and $j = j_1 \vee j_2 = \tau_F$. Because John is not rich, we can now conclude that Susan does not like him.

If we want to be able to conclude that Susan does not like Paul either (he does not like *The Thinker*), we might add a rule that states that Susan does not like things that do not like (a) classical sculpture. It is important to notice that adding individual disjunctive "negative" rules one by one (which now gives the intended application semantics) is not the same thing as adding a single combined rule with "negative" goals.

2.6.3 Systems with Subnormal Models

Using interval-valued fuzzy models enables encoding intervals in which clauses apply. For instance, choosing $r_1 = \tau(x; 0.5, 1, 1, 1)$ in program P_5 might intuitively be interpreted so that the related rule accepts imprecise classifications "50% or more rich". In addition, it is also possible to model point-like fuzzy models with singleton fuzzy sets.

Further, the design of type-1 fuzzy rule systems enables also applications with subnormal models. Intuitively, this allows making statements whose *significance*, *trust*, or *certainty* may vary[2]. However, engagement with subnormal models comes with a price: more rules might be required and the challenge of adopting meaningful application semantics may become noteworthy.

Technically, the generalisation is nevertheless subtle. Recall the logic program P_4:

```
classicalScuplture(theThinker)~m1.
rich(paul)~m2.
likes(susan,theThinker)~m3.
likes(paul,theThinker)~m4.
likes(john,theThinker)~m5.
likes(susan,X)~r0 :- rich(X)~r1,likes(X,Y)~r2,
                     classicalScuplture(Y)~r3.
```

For instance, we may set $m_1 = m_2 = m_3 = m_4 = m_5 = \tau_T$ and $r_0 = r_1 = \tau(x; 0.7, 0, 0.8, 1) \vee \tau(x; 0.8, 1, 1, 1), r_2 = r_3 = \tau_T$.

In practise, subnormal models are most useful when applied with systems of overlapping rules. For instance, we might wish to add more rules of the kind `likes(susan, X)...`, capturing more cases, and consider modelling m_3, m_4, and m_5 with models similar to r_0.

If procedures are *exhaustive* or *dense* enough (the models of the overlapping rules overlap appropriately; see, e.g., Figure 3.2), the canonised meaning of logic program does not necessarily have to include subnormal models.

It is easy to see that we may implement variants of, e.g. *Mamdani fuzzy models* [38] with type-1 fuzzy logic programs. In addition to the predefined operators, the chief distinction is allowing the use of variables for connecting various procedures (we shall return to this topic shortly).

Finally, note that choosing $m_2 = \tau_T$ would probably make an overly precise statement. For instance, the classification might originate from a statistical analysis or an expert review. If we choose to model the related

[2]Note that the introduction of probabilistic interpretations must also be taken into account in the design of the rules (and the assigned models).

imprecision, we may simply associate m_2 with a more complex fuzzy set, perhaps taking the error or the variance of the classification process explicitly into account. Further, there is no requirement of using neutral fuzzy models. In other words, predicates may be associated with different types of fuzzy sets, taking the particular application semantics more closely into account.

Chapter 3

Linguistic Reasoning and Context

The rules and the procedures of type-1 fuzzy logic programs enable the definition of many kinds of useful fuzzy systems. However, while the structure of logic programs is to some extent independent to the particular design of fuzzy models, choosing appropriate models is of course pivotal in successful applications.

We will next consider mapping fuzzy models onto intuitive labels which provides a basis for formalising certain forms of linguistic reasoning and inference. The following treatment also illustrates the application of type-1 fuzzy logic programs in the context of stereotypical fuzzy control applications that often appear in the literature [53, 38, 17].

3.1 Fuzzy If-Then Rules

Many fuzzy reasoning systems represent fuzzy rules in *if-then* format. Representing certain fuzzy system becomes more intuitive when models are represented using linguistic expressions.

3.1.1 Type-1 Fuzzy LPs and Fuzzy Control Systems

Consider the following procedure R with n rules and 2 conjuncted body goals (where `t` denotes temperature and `p` pressure):

```
If t is very_cold   and p is weak then throttle is positive_large.
If t is rather_cold and p is weak then throttle is positive_small.
...
If t is normal      and p is ok   then throttle is zero.
```

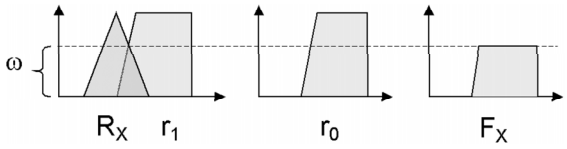

Figure 3.1: Fuzzy matching and evaluation illustrated

The essential information of the rules can be compiled into a $n \times m, m = 3$ matrix

$$(a_{ij}), i = 1, 2, ..., n, j = 1, 2, ..., m$$

that is sometimes called a *fuzzy associative memory* (FAM):

```
#t           * p         -> throttle
 very_cold,    weak,        positive_large
 rather_cold,  weak,        positive_small
 ...
 normal,       ok,          zero
```

It is easy to see that we may implement R with a type-1 fuzzy logic program P_6 :

```
throttle(X)~r10 :- t(X)~r11,p(X)~r12.
throttle(X)~r20 :- t(X)~r21,p(X)~r22.
...
throttle(X)~rn0 :- t(X)~rn1,p(X)~rn2.
```

In other words, the procedure of n rules is defined by a matrix of $n \times 3$ fuzzy models:

$$(r_{ij}), i = 1, 2, ..., n, j = 0, 1, 2.$$

According to the formulation of type-1 fuzzy logic programs, also the throttle, the temperature, and the pressure may be modelled with arbitrary type-1 fuzzy sets (including precise models). Of course, defuzzification might be required in the application.

The relationship between type-1 fuzzy logic programs and control-oriented fuzzy reasoning systems is apparent. For instance, consider the following rule:

```
famous(X)~r0 :- rich(X)~r1.
```

Let r_1 denote the model of the body goal, r_0 the model of the head goal, R_X a particular fuzzy evaluation of the resolvent clause (i.e. X being rich),

and choose $\otimes = \wedge$. Let F_X denote the model of the induced fuzzy property (i.e. X being famous). We get

$$F_X = \omega(r_1, R_X) \otimes r_0.$$

Figure 3.1 illustrates computing the fuzzy evaluation with an example (compare, e.g., with Mamdani fuzzy systems [38]).

3.1.2 Linguistic Variables

The keywords `very_cold`, `weak`, ... that appear in the procedure R above, denote *linguistic terms* that identify the related fuzzy sets, defined using an appropriate *linguistic variable.*

Definition 17 (Linguistic Variable) *A linguistic variable is a quintuple* $(v,\ T(v), X, G, M)$ *where* v *denotes the name of the variable,* $T(v)$ *is the term set,* X *is the base set,* G *is the syntactic rule, and* M *is the semantic rule.*

In brief, the syntactic rule establishes the terms, while the semantic rule associates linguistic values with their meanings, represented as fuzzy sets.

We may thus define a linguistic variable p that denotes the pressure and associate it with the terms $t_1 =$ `very_cold`, $t_2 =$ `rather_cold`,..., each of which denotes a particular fuzzy set A_i, i.e a fuzzy model.

3.2 Linguistic Construction of Fuzzy Models

In some cases, definition of the fuzzy models benefits from a more structured approach. Instead of simply enumerating all the terms, we may establish a distinguished set of the *basic terms* from which the fuzzy models are constructed with general-purpose operators.

Technically, we may achieve this with either using special kinds of linguistic variables called property constructors, or using hedges.

3.2.1 Property Constructors

Assume using type-1 fuzzy logic programs to generalise crisp reasoning. For practical reasons, it is useful to establish definitions of general-purpose linguistic terms that can be used as a basis for defining other terms.

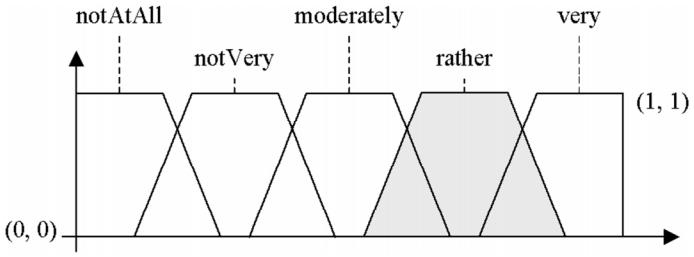

Figure 3.2: Property constructor linguistic variable. (The fuzzy set associated with the term *rather* is highlighted.)

Definition 18 (Property Constructor) *Let v_a denote a linguistic variable associated with the following terms: $\langle t_i \rangle = \langle notAtAll, notVery, moderately, rather, very \rangle$ and the following fuzzy sets*
$\langle A_i \rangle = \langle \tau(x; 0, 1, 0.16, 1) \vee \tau(x; 0.16, 1, 0.26, 0),$
$\tau(x; 0.11, 0, 0.21, 1) \vee \tau(x; 0.21, 1, 0.37, 1) \vee \tau(x; 0.37, 1, 0.47, 0),$
$\tau(x; 0.32, 0, 0.42, 1) \vee \tau(x; 0.42, 1, 0.58, 1) \vee \tau(x; 0.58, 1, 0.68, 0),$
$\tau(x; 0.53, 0, 0.63, 1) \vee \tau(x; 0.63, 1, 0.79, 1) \vee \tau(x; 0.79, 1, 0.89, 0),$
$\tau(x; 0.74, 0, 0.84, 1) \vee \tau(x; 0.84, 1, 1, 1) \rangle$.

We say that v_a denotes a property constructor.

Intuitively, the terms of v_a thus resemble adverbs that denote quantities in natural clauses (*"The temperature is r a t h e r cold."*). Figure 3.2 depicts the terms and the fuzzy sets defined by the linguistic variable v_a. In practise, the set of predefined terms may change upon application, or may be adapted according to a particular context.

The variable v_a is special because it is neutral and may be associated with other terms. In particular, it may be used as a *concept constructor*. When associated with an appropriate *abstract concept*, it may be used to construct a fuzzy model, type of which is established by the related abstract concept.

Definition 19 (Concept Constructor) *Let C denote the symbol of an abstract concept. Let o denote the term of a property constructor.*

We say that $o(C)$ denotes a concept constructed by applying the property constructor o to the abstract concept C.

In some applications, names of the predicates suffice in identifying the abstract concepts. (Concept constructors are not to be confused with hedges discussed shortly.)

As an example, consider the linguistic term `rather_cold`. We may think that the term is composed using the property constructor `rather`, concatenated to the abstract concept `cold`.

It is easy to see that we may establish similar concept constructors for other kinds of syntactic expressions as well, e.g., related to quantities (*verySmall, small, neutral, large, veryLarge*) and relations between quantities (*muchLessThan, littleLessThan, around, littleOverThan, muchOverThan*). Note that the choice of the constructors depends upon whether the modelling is based on *single bipolar concepts* (e.g. *red* or *circular*) or, e.g., *concept pairs* (e.g. *cold-hot* or *negative-positive*).

In brief, the property constructors allow capturing rich fuzzy models with few basic terms. Further, with certain assumptions, it is possible to systematically modify a fuzzy system by re-evaluating the concept constructors.

3.2.2 Hedges

Derived fuzzy models may also be constructed by applying functional operators to the existing basic models.

Definition 20 (Hedge) *Let A denote a fuzzy set and o a function $o : \mathbb{M} \rightarrow \mathbb{M}$ where $\mathbb{M}$ denotes an appropriate set of fuzzy models. We say that the (operator) o denotes a hedge.*

Given a fuzzy model A, we may thus define new fuzzy models with appropriate hedge operators by writing $o_i(...o_j(A)$. Note that according to this definition, we may consider property constructors as *hedge constants.*

The commonly defined hedges include, e.g., *negation*, $\mathrm{NOT}(A) = 1 - A$, for *not*, *concentration*, $\mathrm{CON}(A) = A^2$, for *very*, and *dilatation*, $\mathrm{DIL}(A) = A^{0.5}$, for *more or less*. Some hedges may pose restrictions to the subset of fuzzy models they apply to.

Intuitively, we may thus write `CON(cold)` instead of `very_cold`. Since it is possible to define a meaningful set of general-purpose hedges, this again simplifies the technical language (e.g., CON(hot)).

As with the case of property constructors, we agree upon a notation that enables encoding modified models without touching the underlying syntax of logic programs. Let o denote a modifier and A an applicable fuzzy model. For $o(A)$ we write `o_a`. Further, we may abbreviate $\mathrm{CON}(A)$ as $\mathrm{very}(A)$.

Depending whether we use property constructors or hedges as concept constructors, we may thus write the first rule in the program P_6 either as:

```
throttle(X)~positive_large :- t(X)~very_cold,p(X)~weak.
```

or:

```
throttle(X)~positive_large :- t(X)~CON(cold),p(X)~weak.
```

Due to the differences of the property constructors and (functional) hedges, these two approaches might not yield exactly the same models.

Developing type-1 fuzzy logic programs that exploit the property constructors benefits from the definition of the following general purpose hedges:

Definition 21 (At Most and At Least Hedges) *Let A denote a fuzzy set. Let*

$$x_1 = min(\{x|(x,y) \in (x, A(x)), y = max(A(x))\})$$

and

$$x_2 = max(\{x|(x,y) \in (x, A(x)), y = max(A(x))\}).$$

Define

$$AM(A) = \begin{cases} A(x_1), & x \leq x_1 \\ A(x), & x > x_1. \end{cases}$$

$$AL(A) = \begin{cases} A(x), & x \leq x_2 \\ A(x_2), & x > x_2. \end{cases}$$

We say that AM is the at most hedge and AL the at least hedge.

In applications, the models operated with hedges AM and AL appear as match patterns that establish the acceptable ranges of goals.

3.3 Capturing Fuzziness via Linguistic Description

Using linguistic variables provides a way to capture (expert) knowledge in applications.

The linguistic mapping of models associates clauses with intuitive interpretations. Two main use cases may be differentiated: capturing fuzziness in rules and capturing fuzziness in facts.

For instance, recall the fuzzy logic program P_1:

```
rich(paul)~a.
likes(paul,theThinker)~b.
classicalScuplture(theThinker)~c.
likes(susan,X)~r0 :- rich(X)~r1,likes(X,Y)~r2,
                     classicalScuplture(Y)~r3.
```

Assuming the appropriate basic terms are available, we may describe the related models using intuitive linguistic expressions according to property constructors.

For instance, select a = very, b = moderately, c = very, r_0 = very, r_1 = AL(rather), r_2 = AL(moderately), and r_3 = very. Intuitively, we may now thus read the program P_1 as follows:

- Paul is very rich.
- Paul likes *The Thinker* moderately.
- *The Thinker* is a very classical sculpture.
- If someone (presumably a person) is at least rather rich and likes classical sculpture at least moderately, then Susan likes that person very much.

This modelling allows deducing a fuzzy fact from the program P_1 which can be read as follows:

- Susan likes Paul very much.

Clearly, as a procedure, the rule `likes` should be completed with rules that describe the other cases. For instance, such rules might include the clause according to which Susan does not like persons who are not at least rather rich.

The second use case is about capturing fuzziness with linguistic facts. This enables recording intuitive clauses like

```
rich(john)~notVery.
```

The main benefit is the ability to record vague but useful classifications that describe the identified domain objects. When associated with the assertion context (described briefly), this use case provides a method for capturing potentially very large quantities of practically applicable information.

Finally, it is worth recalling the there exists methods for inducing rule systems semi-automatically (e.g. ANFIS [38]). In principle, any rule induction approach suitable for Mamdani fuzzy systems may be adapted to suit the need of type-1 fuzzy LPs. Semiautomatic processes, however, may lead into less transparent and less intuitive design.

3.4 Context-Aware Logic Programs

The paradox of global fuzzy programs is that fuzziness is often due *subjective or context-dependent evaluation.* However, it can be useful to include modules of logic programs with varying degrees or significance, according to their contextual utility.

In applications, logic programs are typically composed from several physical units for maintenance and re-use. In addition to locally maintained units, distributed systems may include references to third-party units, acceptance of which might vary upon context.

When fuzzy models are considered, it is generally accepted that fuzziness may manifest itself through context-dependent design choices. For instance, a given model behind the *fuzzy concept* `classicalSculpture` is not necessarily global. In addition, it may be useful to emphasise the facts asserted by known authors, or the people who share one's understanding of things. In particular, many fuzzy models tend to rely on *subjective*, *local*, or *contextual* design and choices of modelling.

In other words, it should be possible to include modules of logic programs with varying degrees, according to their utility in an application. For instance, consider the case of finding art books from an Internet database, classified by users using the predicate `classical`. Intuitively, it makes sense to favour the classifications made by users whose assertion context matches the interpretation context of the query. Intuitively, the usage of the term might be related to one's interests and level of professionalism in a particular context.

These considerations raise issues related to the integration and aggregation of various sources of clauses, and evaluating the applicability of the third-party units within a particular context. We will next extend the architecture of type-1 fuzzy logic programs to acknowledge the definition of context-aware units.

3.4.1 Modular Logic Programs

A large logic program consists of one ore more physical units called *modules.* A module is simply a logic program, which may include a *module declaration.* In typical logic programming applications, modules are used to hide local predicates, document the public predicates, and organise logic programs into meaningful and manageable units. We will next introduce an additional use case, by considering the weighted interpretation of modules.

Let $\{P_i\}$ denote a set of type-1 fuzzy logic programs. When programs are perceived as sets of clauses, we may construct new logic programs simply by

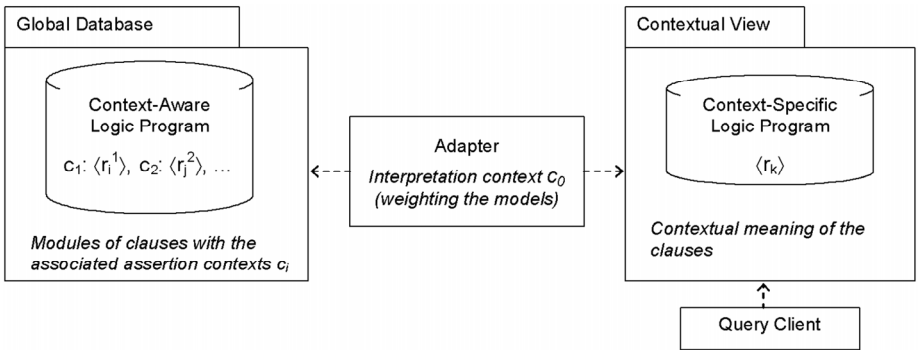

Figure 3.3: Architecture of context-aware logic programs

aggregating the existing logic programs:

$$P = \bigcup_i P_i.$$

Let P_0 denote the (usually local) *main module* and $P_i, i = 1, 2, ..., n$ the *external modules*. Associate each module a weight $w_i \in [0, 1]$. Let $w_i P_i$ denote a type-1 fuzzy logic program which include the rules of P_i, models of which are multiplied by w_i.

We may define a *weighted modular logic program* P as:

$$P = \bigcup_i w_i P_i. \tag{3.1}$$

Intuitively, we may perceive this approach as a mechanism of declaring the evaluated significance (or even *trust*) of the various sources of information.

Figure 3.3 demonstrates the basic architecture of context-aware logic programs. In brief, the program consists of modules of rules ($\langle r_k^i \rangle$) associated with an explicitly encoded *assertion context* ($c_i, i = 1, 2, ...$), meaning of which is determined upon the *interpretation context* (c_0). We may think that the selection (processing) of the interpretation context identifies (computes) a *contextual view* which effectively decides the logic program to which the subsequent queries apply.

Note that it is ultimately the viewer who may decide which clauses to consider as a part of the (assertion and interpretation) contexts. This means that while in many applications, the predicates to be included to the assertion contexts must be designed in advance (so that relevant information will be faithfully recorded), the interpreter is basically free to ignore any predefined contexts and choose, e.g., clauses that the information producers consider "content" to the "context".

Note also that the strategy of assigning each module a weight w_i enables constructing context-aware programs even when the weighted aggregation of the models can not be defined. A viable strategy might simply be *selecting* the modules assertion context of which is sufficiently close to the interpretation context, thus effectively filtering the context-aware logic program appropriately.

3.4.2 Weighting by Contextual Distances

In addition to the direct definition of the weights, the weighted modular logic program may also be defined in terms of contextual distances.

Associate each module $P_i, i = 0, 1, ..., n$ with a *context declaration* $c_i \in \mathbb{K}$ where $\mathbb{K}$ denotes the set of appropriate contexts (see Figure 3.3). Let $\delta(i, j) : \mathbb{K} \times \mathbb{K} \rightarrow [0, 1]$ denote a *distance function* (of an appropriate (semi)metric space) that denotes the distance between two contexts.

Measuring contextual distances is basically an application of (profile) matching algorithms (see, e.g., [95]). A context declaration serves as an identifier to a recorded structure (basically a list) of clauses that describes the context. Agreeing upon a recursive process of measuring the distance between facts that belong to the assertion and interpretation contexts (e.g. `age(45)~m1.` as a part of a particular assertion context and `age(18)~m2.` as a part of the interpretation context) provides a method for computing distances between contexts.

We may now alternatively define (3.1) based on the indirectly defined contextual distance as:

$$P = \bigcup_i (1 - \delta(c_0, c_i))P_i. \tag{3.2}$$

Since δ denotes a distance function, we get $\delta(c_0, c_0) = 0$ and may write (3.2) as:

$$P = \bigcup_{i=1}^{n} (1 - \delta(c_0, c_i))P_i \cup P_0.$$

As noted above, a simple way of encoding context declarations is modelling contexts in terms of clauses found in the modules. For instance, a module may assert the context for which the fuzzy models apply:

```
authorBackground(classicalArts)~T.
...
classicalSculpture(TheThinker)~m1.
...
```

A contextual distance function may then evaluate the distances between different contexts (perhaps using a taxonomy of `authorBackground`s) and weight the modules accordingly. A context may include more than a single fact, and may be due rule evaluation.

Finally, it is worth observing that the process of weighting by contextual distances works best when there are large modules to choose from, with different contexts. Thus, if none of the assertion contexts match with the interpretation context, the resulting clauses will be associated with highly subnormal models, providing little basis for sensible rules and queries.

3.5 Fuzzy Models as Query Interfaces

The type-1 fuzzy logic programs we have considered so far might be called *pure* in the sense that they include only structures with a direct interpretation related to fuzzy interpretation. However, perceiving type-1 fuzzy logic programs in the context of the related (Prolog) implementations enables conceptualising fuzzy programs as *fuzzy (query) interfaces*, hiding, encapsulating, or (linguistically) decorating the details of the underlying data model. In turn, this illustrates Zadeh's *principle of incompatibility* [106] by considering fuzziness as an interface to (intuitively more complex) crisp logic programs.

For instance, consider the following fuzzy rule:

```
likes(susan,X)~r0 :- rich(X)~r1,likes(X,Y)~r2,
                     classicalSculpture(Y)~r3.
```

Assuming $r_2 = \tau_T$, we may write a Prolog implementation of the rule as follows:

```
c_likes(susan,X,E) :-
  c_rich(X,E1),
  likes(X,Y),
  c_classicalSculpture(Y,E3),
  E = [r0,[[E1,r1],[E3,r3]]].
```

This rule is semantically equivalent to another rule (see Subsection 2.6):

```
c_likes(susan,X,E) :-
  c_rich(X,E1),
  c_likes(X,Y,E2),
  c_classicalSculpture(Y,E3),
  E = [r0,[[E1,r1],[E2,r2],[E3,r3]]].
```

The reason is that considering the interpretation, it does not make any difference if we get an empty fuzzy model as an answer or no answer at all.

The implementation of a type-1 fuzzy logic program may thus include definitions and programmatic structures that are *invisible* to the fuzzy semantics. In other words, we may use non-pure implementations for setting up useful fuzzy models. This approach enables exploiting the commonly useful features of Prolog implementations, most notably including structures (compound terms denoted by structural functors) and lists. With certain assumptions, we may thus relax the grammar of simple fuzzy logic programs in applications (see Definition 6).

Assume a module includes the crisp facts:

```
year(2005).
annualSalary(paul,100000,euros).
country(paul,finland).
country(susan,uk).
```

Clearly, we may use these facts to programmatically assert a fuzzy fact associating Paul with a fuzzy property rich:

```
rich(paul)~m2.
```

where it might be decided that $m_2 = \text{very}$.

We say that the clause has been *induced* or *bootstrapped* using the underlying clauses perhaps part of the information database. More general bootstrapping mechanisms might be established based on machine learning, data mining or other induction algorithms, or with arbitrary computer programs.

The benefit of fuzzy linguistic decoration is most evident in cases where multiple crisp properties can be meaningfully described with a single fuzzy label. For instance, consider searching for used cars or residences which are "at least in rather good condition". Thus, in some applications we may consider fuzziness as the property of the *end-user (knowledge) interface*, used to decorate the encapsulated data model (consider Figure 1.1).

The assertion `rich(paul)~m2.` may thus have a procedural origin, computing the model m_2 according to the given contextual clauses. For example, compering Paul's annual salary with the salaries of the other people identified by the logic program, with the average salary in Finland (in 2005), with the prices in Finland (in 2005), or with the people all over the world (e.g. in the UK).

Looking from the other way around, this approach also provides the basic mechanism for working with *defaults*. Thus, when signalled within a particular context that Paul is very rich, we might conclude that Paul earns around

100000 euros per year. When the linguistic models are due property constructors, the respected default may be read from the best matching model (consider Figure 3.2).

3.6 Challenge of Global Interpretation

While the notion of contextual interpretation points out mechanisms for dealing with contextual fuzzy models, it does not escape the problem of global interpretation. In particularly, one does not only have to identify both the assertion context of the modules and the interpretation context of the main module(s), but also the *intended interpretation of the logic program.*

For instance, considering our example with Susan, Paul, John, and the sculpture *The Thinker*, two fundamental issues remain. First, addressing classical sculptures, does Susan appreciate a classification of her own, or a classification by some acknowledged author?

Second, considering rich things (presumably people), does Susan like "rich people" in a particular country (e.g. in UK), or globally, i.e. the "rich people" around the world? Would Susan consider a rich Eskimo (or Inuit[1]) a rich person? This problem is actually a reformulation of the first problem. The main difference is that finding a "standard definition" might be impossible[2].

The problem of concept definition is thus very difficult and manifests the exercise of power in deciding whose concepts we are willing or obliged to use. Further, it is questionable whether a global, widely accepted definition of "rich" exists in the first place. This seems to imply that Susan (or the people in UK) must to some extent write a definition of her own. However, once a definition of a "rich" is settled, say, based on the belonging and the annual salary in pounds (and not, say, the number of the close relatives), it can be applied to all people. Nevertheless, not everyone might agree upon the fundamentals of the definition which implies that several different predicates are required for capturing the property "rich".

This also raises the discussion of names, i.e. which names should be assigned to the different predicates capturing the different notions of being rich[3]. The related discussion points out the scope of applicable global fuzzy predicates and the fundamental limits of meaningful interpretation and assertion contexts. In other words, the data published locally may thus be used

[1]The English word "Eskimo" is considered offensive by many Inuit.

[2]In principle, one might find (and accept) a definition of classical sculptures from a textbook. This clearly is not the case with the concept of being rich.

[3]This actually implies that a collision-free namespace mechanism is required for the predicate names.

simply as *evidence* (or as a sort of "raw data") for inducing descriptional properties suitable for the purposes of others, perhaps without consulting the original publisher.

Chapter 4

Inductive Models and Implementations

We shall next consider the process of inducing fuzzy sets and describe an algorithm for inducing fuzzy membership values based on decision tree learning. In addition, we outline two approaches for implementing type-1 fuzzy logic programs with two different kinds of tools, CWM and SWI Prolog, and briefly consider the architecture of fuzzy knowledge agents based on SW technologies.

4.1 Inducing Fuzzy Models

A fuzzy model is simply a fuzzy set associated with an appropriate logical term and an optional type definition. While designing fuzzy models by hand may be an option, it might not always be possible, it might not provide the best results, or the manual work might become too expensive to be of any real use in applications.

4.1.1 Inducing Fuzzy sets

Assume a finite set of objects $X = \{x_k\}, k = 1, 2, ..., n$ described with m attributes, $a_i, i = 1, 2, ..., m$, each associated with a finite list of alternative values.

Suppose we wish to divide the objects into two sets, $C \subset X$, and $C^C = X \setminus C$. We say that C denotes a *concept*. The concept C may be recorded as a subset of elements in X, or we described using an appropriate *characteristic function* γ that identifies the elements of C:

$$C = \{x \in X | \gamma(x) = 1\}.$$

When the function γ can not be directly defined, we may try to define it indirectly, e.g., using examples, with respect to some concept induction algorithm. In particular, we may identify two sets of examples, the positive and the negative examples that establish some of the elements in C, and some of the elements in C^C. In short, this is the key problem of *Machine Learning* (ML) [54, 71]

In general, the challenge lies in generalising the hypothesis (characteristic function) beyond the immediate examples, i.e., learning to *recognise* or *predict* the target concept. Further, the data may include errors, and a selection must be made between different plausible hypothesis. In the worst case, the intended target concept can not be reasonably induced from the data at all. This requires selecting a learning algorithm with an appropriate bias, carefully testing that learning indeed occurs, and verifying that the learnt aspect of the data is the intended one.

The process of crisp concept induction may be used as a basis for a process of fuzzy concept induction. A fuzzy concept is simply a fuzzy set, generalising the definition of a characteristic function $\gamma : X \to \{0, 1\}$ using a membership function $\mu : X \to [0, 1]$.

The most simple strategy is to induce fuzzy membership functions based on a direct distance.

Definition 22 (Direct Distance) *Let x denote a domain object and D^+ denote the set of positive examples, each represented as a vector of m attributes.*

We define the direct distance between $x \in X$ and D^+ as

$$\Delta(x, D^+) = min\{\Sigma_i \delta_{x_i,d_i} | d \in D^+\}.$$

In practice, the direct distance is the number of the unequal attribute values between an object x and the most similar instance in D. As suspected, the distance may be used for defining a fuzzy membership degrees.

Note that this definition does not take advantage of the negative examples. Further, the distance strongly depends on the encoding of the attribute values. The more structured profiling approaches may yield more general definitions (see, e.g., [95]).

Definition 23 (Fuzzy Membership Degree by a Direct Distance) *Let D^+ denote the set of positive examples for capturing the target concept C. Let m denote number of the attributes, i.e. the dimension of X and D.*

We say that

$$\mu_C^{D^+}(x) = 1 - \frac{1}{m}\Delta(x, D+)$$

denotes the fuzzy membership function of C.

Note that we are assuming an *a priori* identification of the domain objects $x \in X$. In some applications (related to recognition problems), this is a very strong assumption.

4.1.2 Inducing MFs with Decision Trees

Decision Tree (DT) learning algorithms provide a basis for constructing and representing fuzzy concepts in terms of decision trees of Boolean functions [66].

Algorithm 2 (Redundant Decision Tree Learner (RDTL))

We call a *Redundant Decision Tree Learner* any entropy-based decision tree learning algorithm DTL (that induces a decision tree from the training data of positive and negative examples, outputting 1 for Yes and 0 for No) with the following modifications:

1. Assume an inductive DT learning algorithm DTL that uses information gain (entropy) as a measure for choosing attribute tests to the decision tree.

2. Modify DTL so that instead of choosing a single attribute a with the best information gain, it chooses a set of attributes $\{a_i\}$, including all of the attributes with the best information gain.

3. Modify DTL so that instead of (recursively) adding a single subtree to the decision tree by the attribute a, it adds one subtree for each attribute a_i.

It is easy to see that the RDTL and DTL are identical in terms of input-output behaviour. In other words, a redundant decision tree simply records the alternative branches. The structure of decision tree may depend upon a particular DTL implementation (e.g. due different ordering of the branches).

Proposition 6 (Unique Redundancy Property) *Let* T_1 *be a decision tree induced by an entropy-based decision tree and* T_2 *a related redundant decision tree. Let* $T(x)$ *denote the classification of* x *according to the decision tree* T. *Trivially,* $\mathtt{Dom}(T_1) = \mathtt{Dom}(T_2)$*; denote this set by* X. *Then*

1. *For each* $x \in X, T_1(x) = T_2(x)$.

2. *As a tree,* T_2 *is uniquely defined by* T_1*: for each* T_1 *there is exactly one* T_2.

Proof. The proposition follows directly from Algorithm 2. The second part is true effectively because there are no choices left to the algorithm. □.

The rationale behind the Algorithm 2 becomes clear in the context of inducing fuzzy membership degrees.

Definition 24 (Fuzzy Membership Degree by RDTL) *Let* T *be a redundant decision tree induced by the Algorithm 2 and* x *an input attribute vector,* $x \in X = A_1 \times A_2 \times ... \times A_m$. *Let* $\{\Gamma_k\} = \{\Gamma_k \subset T : \Gamma_k(x) = 0\}$ *be a finite set that denotes all the branches of* T *that output a negative classification. Let* $p > 0$ *be the length of the longest branch of* $\{\Gamma_k\}$. *If* $T(x) = 0$, *let* s *be the length of the shortest branch* $\Gamma \in \{\Gamma_k\}$ *so that* $\Gamma(x) = 0$.

Define the membership function induced by $T, \mu_T : X \to L, L \subset [0,1]$, *as follows: If* $T(x) = 1$, *define* $\mu_T(x) = 1$, *otherwise define* $\mu_T(x) = \frac{(s-1)}{p}$.

If the decision tree is unable to output a prediction (due encountering a previously unseen attribute value), we simply set $\mu_T(x) = 0.5$. (This is not the best strategy, however.) The added redundancy of Algorithm 2 ensures that Definition 24 is well-defined, i.e. the membership degree is unambiguous (Proposition 6).

It is easy to see that the definition genuinely generalises the crisp concept in terms of Classical reasoning:

Proposition 7 (Subsumption Property) *Let* $\mu_T(x)$ *be a membership degree induced by a redundant decision tree* T. *Then* $\mu_T(x) = 1$ *if and only if* $T(x) = 1$. *Otherwise* $0 \leq \mu_T(x) < 1$.

Proof. Follows straightforwardly from Definition 24. □.

4.1.3 A Case Study

To demonstrate the generalisation property of RDTL based fuzzy models, we will next briefly discuss a case study (for details, please see Publication 4 [66]).

As a part of an evaluation project of Web-based learning systems, a student survey was compiled at the Tampere University of Technology, Hypermedia laboratory. After attending to various university courses including

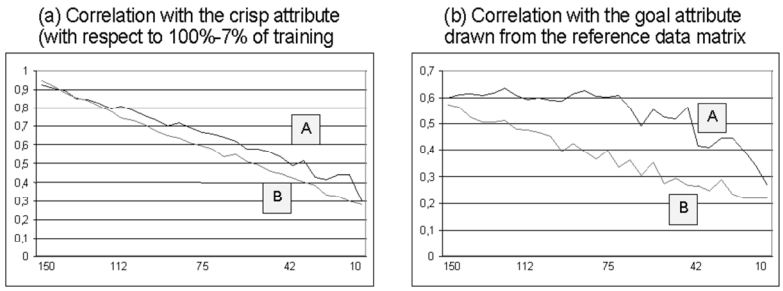

Figure 4.1: Correlation graphs demonstrating the generalisation performance with the validation data set, using the RDTL algorithm (A) and the direct distance (B)

Web-based modules, 150 students answered to a set of (multiple choice) questions. In the experiment, the descriptive attributes included 14 values from 11 questionnaire questions and three summation variables, resulting from the pre-processing of 17 (semantically overlapping) background questions with Principle Component Analysis (PCA) with manual selection[1].

Figure 4.1 depicts four Pearson's correlation curves that show the generalisation performance in terms of the training data and an *a priori known* reference data (compiled heuristically). The X axis shows the training set size, starting from the whole set of the available training data ($N = 150$). The values of the Y axis consist of the arithmetic means of the correlation coefficients after 10 trials with the validation data set, using random training data sets consisting of X instances.

The RDTL algorithm seems to outperform the direct distance with the reference data (curve A in Figure 4.1 (b)). The performance stays maximal until using only half of the available training data. The reason for this seems to be the generalisation performance of the underlying DT learner, which in this case seemingly captures the crisp concept with some 75 training instances.

In certain applications, a significant benefit from using ML learning methods originates from the fact that explicit modelling is not always required. In this case, the DT reasoner compiles a decision tree without expert help, i.e. as a model-free or a black-box learning algorithm. If statistical accuracy suffices as a design criteria, this provides more general and cheaper tools when compared to manual case-specific modelling (see Section 5.2).

[1]The data was compiled by the so-called EVA group. Principle component analysis was in detail carried out by E. Kalliomäki.

4.1.4 Statistics and Fuzziness in Modelling

The relationship between probabilities and vagueness (i.e. fuzziness) has been much debated in the literature (see Section 1.3). In brief, the general understanding seems to be that the concept of vagueness genuinely complements the concept of randomness in modelling indeterminacy, establishing the two facets of the phenomenon [61].

For purposes of this treatment, we will adopt a very practical design stance, according to which *the design of fuzzy models may be guided by statistic or probabilistic arguments.* This also seems to correspond to the underlying ideology behind type-2 fuzzy models [53]. A concrete example of this stance is established by the conception of the fuzzy sets induced with the RDTL algorithm (Definition 24).

Philosophical argument aside, the adopted strategy becomes concrete when the computation costs are considered. From a practical point of view, a probabilistic models need to be updated when new evidence is achieved. Monotonic reasoning systems, on the other hand, aim for a structure that preserves the already deduced facts (For all B, If $A \models \alpha$ then $(A \wedge B) \models \alpha$.) This is particularly true in the case of assertional languages which knowingly escape the problems related to assumption-related contradictions (which enable deducing "anything") by not introducing a standard negation operator into the language.

Considering the processing requirements, type-1 fuzzy systems are clearly more demanding than simple assertional languages. In practice this means at least re-computing the models when new information is received. In practice, however, the Fact Canonisation Lemma (Proposition 3) allows to some extent optimising the way models are in practice computed.

Strictly speaking, some of this discussion is left unanswered. From this perspective, type-1 fuzzy logic programs are simply technical means to model and implement various applications, interpretation of which is agreed upon the domain characteristics.

4.2 Implementations

We shall next outline two implementations of approximate reasoning systems based on type-1 fuzzy sets. The first implementation is based on experimental SW technologies, aiming to demonstrate the power of a *de facto* standardised data model in applications. The second implementation is based on the well-established Prolog programming paradigm, yielding more or less production-quality applications.

4.2.1 Case CWM

CWM rule system [9] provides a framework for filtering, pretty-printing, querying, and performing inferences based on RDF data. Providing a well-known general-purpose data processor for the Semantic Web, CWM establishes a relatively well-established test bed for studying the potential of existing and emerging SW rule applications. CWM is distributed under the W3C Software License (basically freely but without any warranty).

From the research perspective, a significant characteristic of the rule system is the possibility to add new software components. Technically, this enables the implementation of new namespace-specific functions, which in practice may be used as new primitives of inference. Written in Python, implementing new components is relatively easy, even if the architecture of CWM is sometimes criticised for its prototype flavour. Existing additional CWM modules include, e.g., cryptographic, time, and mathematical functions.

CWM rules extend the N3 syntax for RDF by providing a format for rules. For instance, consider the following rule (prefix definitions omitted for brevity):

```
this log:forAll v:p, v:c .
{ v:c ex:childOf v:p . v:p rdf:type ex:Man . }
log:implies
{ v:p rdf:type ex:Father . }.
```

The rule states the definition of a father. In Prolog, this could be expressed as:

```
father(P) :- child(P,_),man(P).
```

The CWM rule framework is essentially crisp. However, as in the case of many crisp reasoning systems (see, e.g., [39], [22]), it is possible to implement fuzzy rules on top of the crisp processing via programming. In practice, a viable approach is to first implement linguistic variables with the CWM rules, from which the fuzzy rules are then compiled as a separate set of CWM rules (see Publication 5 [67]).

The main objective of the case study was to experiment using the basic enabling technologies in order to realise the expert and the end-user interfaces of a fuzzy knowledge agent (see Figure 1.1). Figure 4.2 outlines the component layers of a knowledge agent, founded to a broad range of enabling technologies. The overall architecture consists of three abstract layers

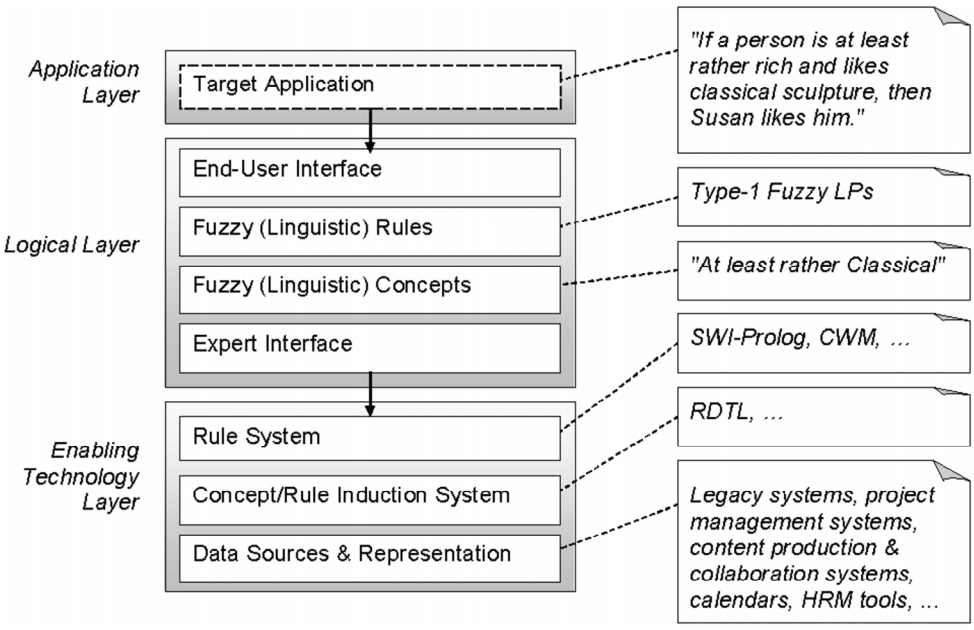

Figure 4.2: Layers of a fuzzy (linguistic) knowledge agent architecture

including the target application, the logical layer, and the layer of the enabling technologies (perhaps invisible to the end-user).

When perceived as an interface, fuzzy description provides an intuitive method for describing the underlying data model using a small, appropriately selected set of linguistic terms with clear semantics. In short, the expert user interface provides an access to the bootstrapping mechanism, providing a programmable method for setting up the linguistic concepts that the end-user interface perceives through the fuzzy logic programming view, or the end-user (knowledge) interface to the fuzzy knowledge agent. As suspected, the chief benefit from using SW technologies originates from the possibility of retrieving and merging data from radically different but uniformly modelled systems and data sources with standard query and processor interfaces.

Further, to try out the potential of mixing statistical data with logical rules, we implemented a Decision Tree (DT) module for the CWM rule system (see Publication 3 [64]). The DT module defines a set of functions, integrated with the CWM rule system, for training, serialising, and publishing decision trees, and using them for making induced classifications and predictions. This provided the basic functionality for the experiments anticipated in the early studies (see Publication 6 [68]).

However, while it indeed is possible to implement CWM applications that

benefit from the induced fuzzy models, the author decided that from the logical point of view, a clearer separation of the model induction phase, the reasoning process, and the application layer was in order. Thus, even if the prototyping nature of the CWM was appreciated and the interoperability and the vision of the Semantic Web were considered very important, the emphasis was shifted from technology-specific tools to (more constrained) general-purpose logic programming tools, focusing the logical aspects of fuzzy knowledge agents.

From the applications' perspective, it would seem that the first large-scale semantic applications might be built on top of semantic extensions of the existing popular collaborative authoring tools, such as the Semantic Wikipedia [97]. This provides the option of implementing fuzzy knowledge agents as auxiliary services, providing an alternative access to the triplestore(s) operated by the Wiki platform(s). Considering Figure 4.2, this simply means appropriately including the Semantic Wikipedia triplestores to the bootstrapping processes of the knowledge agents.

4.2.2 Case SWI-Prolog

SWI-Prolog [100] offers a comprehensive and a widely used Free Software Prolog environment, licensed under the Lesser GNU Public License. In addition to the interpreter software, SWI-Prolog includes analyser and graphics toolkits, and well-established accessories for working with other programming environments and applications.

SWI-Prolog includes a reasonable support for the RDF data model and the related RDF/XML serialisation syntax. For instance, selected RDF graphs may be uploaded and asserted as RDF triples `rdf(X,Y,Z).`, to be accessed from other procedures. This enables implementing applications founded to the well-established syntax and semantics of Prolog programs, taking advantage of the interoperability of the RDF data model (perhaps after pre-processing).

From the perspective of implementing logic programs with fuzzy models, the selection is by default suitable, due to the close connection of type-1 fuzzy logic programs and Prolog. In short, type-1 fuzzy logic program simply annotates a Prolog implementation with the respected fuzzy models.

Mapping type-1 fuzzy logic programs onto Prolog programs is straightforward (consider Section 2.5). Let

$$\mathtt{cl}_0(\alpha_0) \sim r_0 \mathtt{:-cl}_1(\alpha_1) \sim r_1, ..., \mathtt{cl}_n(\alpha_n) \sim r_n.$$

denote a fuzzy rule where cl denotes a clause and α_i denotes an argument configuration.

We implement it as follows:

$$\texttt{c\_cl}_0(\alpha_0, E_0)\texttt{:-}$$

$$\texttt{c\_cl}_1(\alpha_1, E_1), ..., \texttt{c\_cl}_n(\alpha_n, E_n), E_0 = [r_0, [E_1, r_1], ..., [E_n, r_n]].$$

If the fuzzy rule includes goals without an associated fuzzy model ($cl_i(\alpha_i)$ without the $\sim r_i$ part), those terms are simply copied to the implementation unchanged. Note that we assume that the rule system does not include clauses for other purposes with the prefix `c_` and that the (local) variables E_i do not previously appear in the clauses (i.e. to be instantiated by accident).

To abbreviate the list syntax, we may introduce a special case for implementing facts. Let

$$\texttt{cl}_0(\alpha_0) \sim r_0.$$

denote a fact. We implement it as follows:

$$\texttt{c\_cl}_0(\alpha_0, r_0).$$

In many cases, logic programs include at least two modules: the general-purpose program and data from a particular application. In practice, procedures of the general-purpose program assume a certain encoding for the data.

According to the definition of type-1 fuzzy logic programs, the answers to the ask queries change when the clauses of the program are changed, or when the fuzzy models are modified. In the latter case, re-evaluating queries simply means re-computing the parse trees of the already evaluated answers. Thus, when appropriately designed, the Prolog implementation (of certain kinds of logic programs) need to be run only once, outputting the formulas for computing certain interesting queries. Effectively, this allows recording application-specific reasoners in terms of parse trees, evaluated according to the observed models (for reuse, due computational efficiency and smaller memory requirements).

However, even if not necessary, parametrised fuzzy models and the related algebraic operations may also be implemented in Prolog. For instance, consider the fact

```
rich(paul)~r.
```

By introducing a library of parametrised MFs, we may assert that the name of the model r denotes, e.g., a trapezoidial MF $\tau(x; 0.6, 0.7, 0.8, 0.9)$ (see Definition 5):

```
trapezoidMF(r, 0.6, 0.7, 0.8, 0.9).
```

Again, restricting to only trapezoidial functions is not a necessity.

We may implement a general-purpose procedure `evalFM(+elist, -value)` which evaluates fuzzy models (outputting fuzzy sets represented, e.g., with a finite trapezoidial base; see Formula (2.2)) based on the registered names (the output argument `value`), when given the parse tree of a formula (the input argument `elist`). If the processing related to the computing with fuzzy models is separated from the processing related to computing answers to the queries, the reasoning and the evaluating components may again be separated in applications.

Chapter 5

Case Studies

In order to review and illustrate the developed methods and techniques, we will next briefly consider two educational applications as illustrative use cases. The first use case illustrates implementing decision-support systems for teachers. The second use case outlines a personalisation application that helps students to choose appropriate learning material. Of course, the applications of the developed methods are not restricted to educational use cases.

5.1 Introduction

Consider organising a large academic course as a teacher and studying it as a student. Teachers face the problem of knowing their students well enough to be able to adapt the teaching accordingly, while students face the problem of self-evaluating their progress, and choosing the appropriate (extra) learning material for self-study.

However, given a sufficiently large and a settled (basic) course, there readily exists potentially applicable historical data for decision support: points from the exercises, and the mid-term exams of the past course(s), to the least. Associated with the actual grades, these establish classified time series that can be used for predicting the progress of the current students.

An example of this kind of a data structure is presented in Figure 5.1. In brief, the data structure, organised simply as a matrix, includes the information of n students with m recorded achievements, associated with their final grades, resulting n vectors, each with $m + 1$ co-ordinates (attributes). Obviously, the semantic structure of the example matrix originates from the structure of m exercises and mid-term exams, each exam consisting of two assignments.

Student	Exercises				Exam 1	Exercises			Exam 2	Grade
s_1	x_{11}	x_{12}	...	x_{1p-1}	x_{1p}, x_{1p+1}	x_{1p+2}	...	x_{1m-2}	x_{1m-1},x_{1m}	g_1
s_2	x_{21}	x_{22}	...	x_{2p-1}	x_{2p}, x_{2p+1}	x_{2p+2}	...	x_{2m-2}	x_{2m-1},x_{2m}	g_2
...										
s_n	x_{n1}	x_{n2}	...	x_{np-1}	x_{np}, x_{np+1}	x_{np+2}	...	x_{nm-2}	x_{nm-1},x_{nm}	g_n

Figure 5.1: Matrix of student credits classified with the final grades

With certain assumptions, it should be possible to use these kinds of archives for teaching, organising, and studying courses with predictive models, and verifying the accuracy of the predictions based on the past experiences. Further, in addition to the classifications based on the course credits, the students and the domain objects might of course be associated with other semantic annotations as well. For instance, the exercises might be associated with students' explicit annotations, establishing a basis for the future adaptation in personalisation applications.

5.2 Decision-Support for Teaching

We first illustrate a case study focusing on the teachers' perspective. In short, we wish to model the course information logically, to provide decision-support information in terms of simple linguistic queries based on type-1 fuzzy logic programs.

5.2.1 Inducing Models for Student Classification

Let us consider the most difficult case first; trying to classify students by predicting their exact grades (for details, please see Publication 1 [62]). To simplify the discussion, we establish an inductive model following a very straightforward heuristic reasoning process.

The following experiments were conducted using the accumulated data sets from the basic course *Engineering Mathematics I*, during two successive years 2003 and 2004. In brief, the data describes the performance of the individual students in terms of 22 attributes, pre-classified with the final grades. The data includes 358 student instances considering the year 2003, and 311 instances considering the year 2004. Compiled into chronological order of assignments and four mid-terms exams, this approach provides a vector of 22 attributes describing the progress of each student (compare with Figure 5.1) Considering ongoing courses, the attribute vectors of course become available only gradually, as the courses proceed.

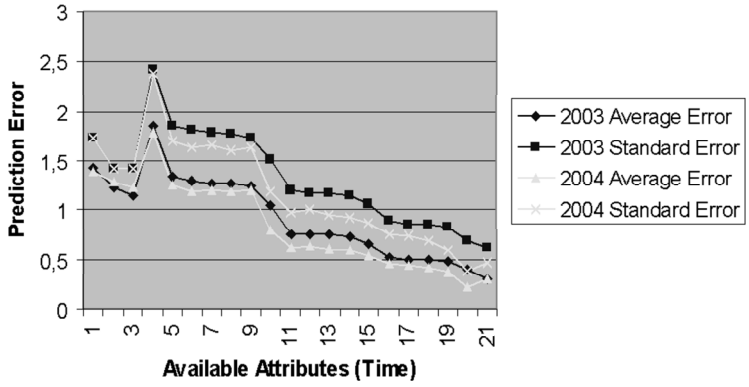

Figure 5.2: The error statistics of predicting the exact grades based on the 2003 and 2004 student archives

While the data provides a basis for applying machine learning algorithms to predict the future performance of the students, deploying a model-free method is not necessary in this particular application. Since the grades are now basically determined by the attribute values in terms of an explicit grade formula, a viable strategy is to predict the goal attribute g (grade) simply by *assuming average performance in the future*, i.e. calculating the grade based on the averages of the already seen attribute values. When studies in mathematics are considered, this is a reasonable assumption. This might, however, not be the case with less structured disciplines.

Figure 5.2 demonstrates the basic statistics related to an attempt to predict the exact grades based on the year 2003 ($n = 358$) and the year 2004 ($n = 311$) records by assuming average performance. Predictions provide two series of seemingly converging parameters for modelling: the predicted grades (not shown in the figure), the average error (the arithmetic mean) of the predicted grades (for k attributes), and the standard error of the predictions (i.e. the standard deviation of the mean errors for k attributes). (For discussion about using statistics in fuzzy modelling, see, e.g., [42, 53].)

Around mid-course, the exact grades $(0 - 5)$ can be thus predicted with the average error around 0.8 (with a standard error around 1.3). This clearly yields rather imprecise models. However, better models may be constructed by simplifying the prediction task. Figure 5.3 depicts the average errors when trying to predict only the failing students (total 90 students in 2003 and total 101 students in 2004). Around mid-course, the failing students may be predicted with the average prediction accuracy of 0.8. This information still suffices for capturing a critical student group at risk, thus helping teachers to focus their efforts even if the resulting model is less descriptive than the

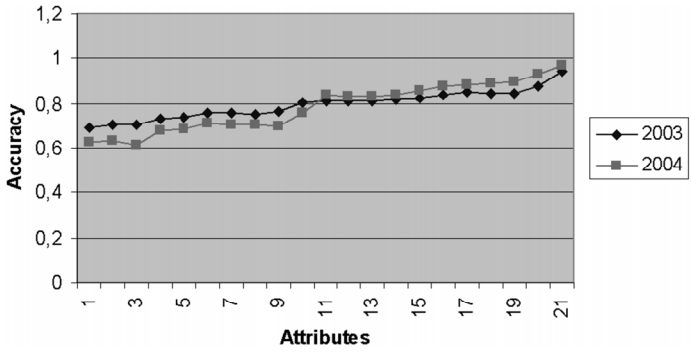

Figure 5.3: The accuracy of predicting the failing students based on the 2003 and 2004 student archives

previous one.

Finally, as we have already pointed out (see Chapter 4), under certain circumstances, better prediction and thus fuzzy modelling results may be achieved using predictive models based on machine learning algorithms. This approach, however, was not studied in depth in this experiment since the simple heuristic modelling was sufficient for purposes of the example.

5.2.2 Logical Formulation of the Application Domain

While improving the prediction system is an interesting domain-specific research question of its own right, let us next consider the architecture of decision-support systems that could benefit from the induced classifications. In particular, let us next consider modelling students using a predicate `poor`: the poorer the performance, the poorer the student. (Alternatively, we might equivalently model students using an appropriately defined predicate `good`. The predicate `poor` was largely chosen for historical reasons, emphasising the attempts to prevent students from dropping out from the basic courses.)

If we equate the predicted grade with the concept of being poor (or not being poor), we may assign each student a fuzzy predicate `poor`, model of which is computed from the predicted grade:

```
poor(s1)~m1.
poor(s2)~m2.
...
poor(sN)~mN.
```

Alternatively, the database might be used for modelling time explicitly, i.e. adding k facts for each student (e.g. `poor(s1,1)~m1t1.`).

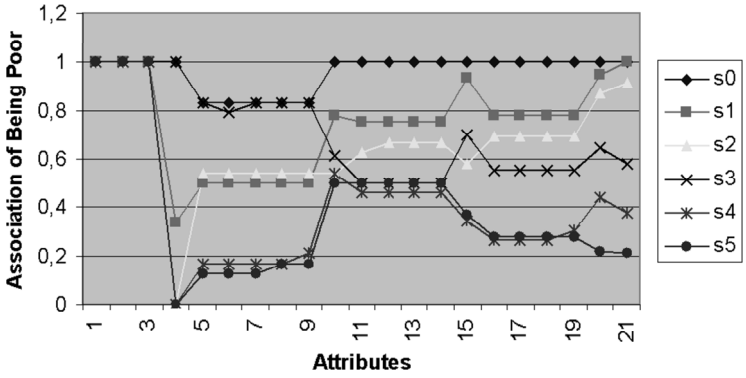

Figure 5.4: The classification trajectories of five particular students

In turn, this setting provides a basis for analysing the development of students in time. Figure 5.4 depicts the *classification trajectories* describing six different types of students that took the year 2004 course with the grades 0, 1, 2, 3, 4, and 5.

The imprecision associated with the fuzzy models may be read from the Figure 5.2, using the errors of the year 2003. In other words, the classifications get rather precise around mid-course. Intuitively, the analysis reveals that the student s_3 seems to improve her performance during the course while the student s_1 is doing worse and worse as the course proceeds. Of course, more information is needed for explaining *why* the behaviour occurs.

It is easy to see that we may straightforwardly implement the models m_i as neutral type-1 fuzzy sets, e.g., as normal interval-based MFs, or trapezoidial MFs, based on the predicted final grade and the error of prediction (see Figure 5.2). For instance, we might assign a student s_1 a predicate `poor(s1)~m1.` where $m_1 = \tau(x; 0.33, 0, 0.5, 1) \vee \tau(x; 0.5, 1, 0.67, 0)$ when $k = 12$. If we choose the models with respect to some appropriate property constructors (see Definition 18), we may in addition establish rules and queries based on rather intuitive semantic labels.

5.2.3 Rules and Queries

Assuming that procedures similar to `poor/1` are available, we can implement logical rules and queries based on the clauses with empirical models.

For instance, consider the following query: *"Find students that are at least rather poor and very passive"*. This rule can be implemented as a verify query:

```
?-poor(X)~AL_rather,passive(X)~very.
```

outputting a sequence of answers associated with firing strengths. Now, choosing `min` for the argument configuration, rejecting the answers with firing strengths under a certain threshold, and sorting the answers gives a list of students matching the query.

However, the chief utility of the approach lies within the potential of asserting logical rules. For instance, consider the following rule that intuitively states that poor and passive students are problematic:

```
problematic(X)~very :- poor(X)~very,passive(X)~very.
```

To establish an intuitive use case, assume students do their weekly assignments by posting their answers and the related questions to the course newsgroup. Further, suppose we can encourage the problematic students by giving them a larger share of the teachers' attention[1]. We may then utilise the piece of knowledge expressed by `problematic/1` by assigning teachers a higher priority for processing the messages from the problematic students:

```
highPriority(teacherTom,M)~very :-
            recentMessage(S,M)~very,problematic(S)~very.
...
highPriority(teacherTom,M)~notAtAll :-
            recentMessage(S,M)~very,problematic(S)~notAtAll.
```

Establishing similar procedures for the teaching assistants, we may thus try to optimise the workload of the teaching staff while trying to pay special attention to the students at risk. The most simple application is to check the high priorities of the teaching staff via ask queries:

```
?-highPriority(S,M).
```

The results may be used as a basis of visualisations, or as triggers of action rules, perhaps sending notification messages to the teaching staff. (In a more general setting, defuzzification might be required to trigger or fire the consequent action rules or actuators etc.)

5.2.4 Components of Predictive Fuzzy Systems

Figure 5.5 depicts the basic components of a predictive fuzzy system. At the heart of the system lies the predictive model which assigns the domain objects

[1]Note that this is a pedagogic decision, validity of which is beyond the scope of this treatment.

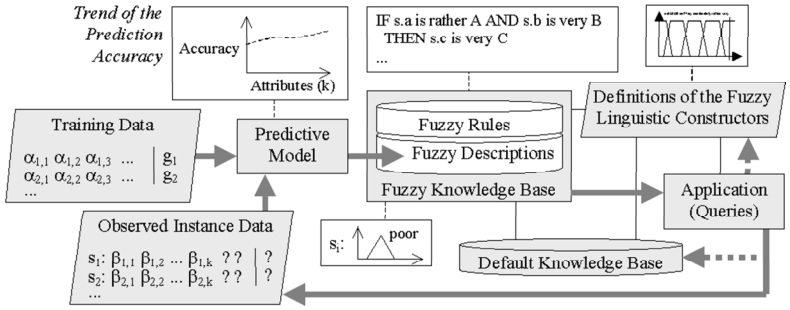

Figure 5.5: The basic components of a predictive fuzzy system

(e.g. students s_i) fuzzy descriptions, based on the available training data and the observed instance data from the application. The fuzzy descriptions add to the default knowledge base, providing a basis for interpreting the fuzzy rules and queries. The linguistic component is defined by identifying the linguistic concepts to which the descriptions apply (e.g. `poor`), and by establishing the fuzzy linguistic constructors (e.g. `rather`), to be used in the rules and queries. In brief, applications use the predictive fuzzy system for performing fuzzy inference and making fuzzy queries based on empirical evidence.

More complex applications might in addition explicitly control the default knowledge base and the definitions of the fuzzy linguistic constructors, in terms of bootstrapping the predictive fuzzy system with application-specific defaults (perhaps also modifying the fuzzy linguistic constructors upon context).

Finally, considering our modest educational decision-support system, few general observations are in order:

1. The models m_i include *fuzziness due to lack of information* about students' skills and development in the course[2]. We would like to minimise this source of imprecision but we really can not. Following the above line of modelling, the best we can do is to estimate the imprecision in terms of statistical arguments considering the past experiences. Alternatively, we could assign the models according to subjective evaluations, perhaps based on the assistants' evaluations.

[2]Note that to some extent, this is a modelling issue. After all, we *could* say that the immediately observable performance of a student, e.g. the outcome of the latest assignment, determines his or her skills and the level of understanding without any reference to the past records.

2. According to the probabilistic arguments (see Figure 5.2), the *fuzziness due lack of information decreases during the course.* The models thus get more precise as more information becomes available.

3. The query expressions include *fuzziness due to the use of linguistic expressions* (e.g. `AL_rather`). While we could minimise this source of imprecision, we don't really want to. Rather, we appreciate the intuitive imprecision that is associated to the names of the linguistic terms. Removing this imprecision would require discarding the fuzzy query interface, and thus force teachers to directly operate with the underlying technical domain concepts (e.g. sequences of credits of a particular week).

4. Even if a restricted set of linguistic expressions is used in the rules and the queries, the approach does not reduce into a keyword based system. This is due to the richness and variance of the inductive models.

It is easy to see that these observations apply to most inductive fuzzy systems with a linguistic interface. Further, this setting highlights the role of the enabling technologies as the *necessary* components of successful applications. The remaining *sufficient* components largely depends upon the success of the appropriate fuzzy models, the design of the procedures, and the planned interaction via queries.

5.3 Towards Personalised Studying

Let us then briefly consider the second educational use case, knowledge personalisation. Assume a logic program including a large set of contextual facts about exercises, asserted by the students. Further, assume that the assertion and the interpretation contexts include three properties, `mainSubject`, `year`, and `skillLevel`. Intuitively, these encode the main subject of the students, the year they started their studies, and their estimated skills[3].

Consider the following type-1 fuzzy logic program:

```
% global names
title(e1,'The length of a pipe network.').
title(topic1,'geometric reasoning').
title(topic2,'definite integral').
...
```

[3]Note that the important question of which properties should be included in the context is an empirical research problem which is beyond the scope of this treatment.

```
% annotations for the exercise e1, asserted by the student s2
level(e1, s2)~rather_difficult.
mainLessons(e1,topic1,s2)~very_significant.
mainLessons(e1,topic2,s2)~notAtAll_significant.

% the identified assertion context
Subject(s2,cs)~very_strong.
Subject(s2,bs)~moderately_strong.
Subject(s2,ss)~notAtAll_strong.
year(s2,2002).
skillLevel(X) ...# assertion about s2 similar to "poor"

% annotations by other students, with the related contexts
...
```

The example includes three annotations of the exercise e_1, the estimated level of difficulty, and the two identified main points to be learnt by the completion of the exercise. In addition, the program records the assertion context (of the student s_2) using five clauses. (During the interpretation phase, it needs to be defined which clauses establish the particular interpretation context.) In brief, the student s_1 argues that the exercise e_1 is rather difficult and is more about geometric reasoning than calculus with definite integrals.

The context thus identifies the student behind the assertions via other clauses. For simplicity, the property `mainSubject` is modelled using three predicates `Subject/2` instead of a genuine taxonomy property. Intuitively, these predicates classify the students background with respect to three identified areas of study: `cs` (computer science), `bs` (business studies), and `ss` (social science).

Note that the formulation of the `skillLevel/1` raises the point of negation on the level of the syntax of logic programming, similar to `not/1` in crisp Prolog.

Assuming a sufficiently large database of such contextual clauses, we may now compute and compare the contextual distances between various assertion and interpretation contexts. In other words, we may implement a student's view to the course data in terms of a context-aware logic program, weighting the annotation modules accordingly. In principle, it should thus be possible, e.g., to select additional exercises according to students' particular needs.

However, while the implementation is technically relatively straightforward, several questions related to application-specific design remain, including:

1. Should the poor students be guided by the vision of the good students or the vision of the (other) poor students?

2. Assume that, say, the students of computer science and the students of business studies use the domain glossary in a slightly different fashion. Should the use of mismatching (but systematic, within one's field of expertise) parallel vocabularies be encouraged, or does it make sense to aim for a global model terminology?

3. How should we interpret the annotations made by very poor or ill-behaving students? In particular, can we trust the annotations of students who do not seem know the topics in the first place?

These questions in turn highlight the three issues related to fuzzy knowledge agents: domain-specific modelling, language considerations, and trust. Note how the issues are related since all reasoning takes place within the formal language (syntax and models).

Finally, in this case it would seem to make sense to treat the predicate `skillLevel` not only as a context-defining predicate, but also as a source of *credibility information*, for explicitly weighting the clauses within the related context-specific unit, as an indicator of trust. However, this in turn suggests differentiating the clauses associated with context from the clauses associated with trust.

This discussion indicates that as a natural concept, *context bears internal structure that needs to be analysed before deploying context-aware logic programs.* In particular, different applications may choose to identify different contextual structures. As a practical consequence, the acceptable contextual structures may vary from application to application, thus posing significant changes to the organisation and the consequences of context-aware fuzzy logic programs upon context.

Chapter 6

Summary of Publications

The contribution of this thesis comprises the six included publications [68, 67, 66, 64, 63, 62] and the previously unpublished parts of the main text.

In short, the publications include explorative work, case and implementation studies, and reported generalisations for the applied techniques. Roughly speaking, the publications 6 and 5 provide the application framework and the motivation for the work. The publication 4 establishes an algorithm for bridging fuzzy and statistical classifications, and the publication 3 considers a case of implementing rule systems with predictive components. The publication 2 generalises the logical approach by proposing a logic programming system with type-1 fuzzy models with few use cases, and the publication 1 describes a case study of a decision-support system using the developed predictive type-1 fuzzy models and methods, demonstrating the basic characteristics of the approach. Finally, the introduction aims providing a synthesis including the so far missing details related to the results and some illustrative applications of type-1 fuzzy logic programs.

The selected publications are presented in the reversed chronological order (1, 2, 3, 4, 5, and 6), starting from the most latest (and to some extent the most important) publications. To some extent, the chronological order (6, 5, 4, 3, 2, and 1) reflects the historical organisation of the studies.

Main points of the selected publications may be summarised as follows:

I. *Publication 1* presents a stereotypic application of fuzzy logic programs exploiting inductive fuzzy models. The article outlines the basic components of a predictive fuzzy system. It demonstrates the approach with a case study by inducing fuzzy classifications of the students of a university course, based on student data during two successive years 2003 and 2004.

   The basic result is that it is possible to induce predictive fuzzy models

successfully using a simple naive learner. Further, type-1 fuzzy sets provide a convenient way to model both the imprecision related to the statistical error of the predictions, and the imprecision related to the use of an intuitive linguistic query interface. Finally, besides fuzzy rules and queries, fuzzy models are applicable in visualising the dynamics of processes in terms of rudimentary decision support systems.

II. *Publication 2* formalises the approach of implementing fuzzy systems in terms of logic programming. The article proposes an approach to fuzzy logic programming via type-1 fuzzy models, effectively generalising the crisp interpretation of standard Prolog programs.

    The study establishes a system for fuzzy logic programming with mainstream Prolog tools and provides a bridge between logic-oriented (e.g. interval-based fuzzy expert systems) and control-oriented fuzzy systems (e.g. Mamdani fuzzy reasoning systems). Further, the applicability of massively distributed fuzzy systems is evaluated, and the notion of context-aware fuzzy logic programs is developed. Publication 2 provides a formal basis for implementing the kinds of fuzzy systems studied throughout the thesis, including the applications discussed, e.g., in Publications 1, 4, and 6.

III. *Publication 3* describes a case study and an implementation of applying Decision Tree (DT) learning algorithms in the context of the CWM rule system. In short, the CWM rule systems provides a technical framework for implementing Semantic Web (SW) rule systems. The publication describes the case of extending the CWM system with a DT learning module that enables learning and using predictive predicates based on empirical learning data. This allows CWM programs to adapt their behaviour based on empirical data.

    The study provides a reference for integrating the logical and the statistical aspects of rule systems, by considering predicates and functions defined indirectly, in terms of empirical learning data. Further, the application demonstrates the process of assigning predicates (probabilistic) credibility information according to the observed prediction accuracy, to be modelled with type-1 fuzzy sets in the later studies.

IV. *Publication 4* establishes an algorithm for inducing fuzzy membership functions upon learning data, by considering a redundant generalisation of decision trees and the corresponding learning algorithms. In short, the study provides a way to describe instance data with fuzzy concepts

based on training data with crisp classifications, intuitively fuzzifying the classifications.

The empirical case studies show that with certain assumptions, the approach outperforms the simple direct distance in terms of assigning instance objects fuzzy membership values. It is assumed that this is due to the generalisation property of the underlying decision tree learning algorithm, managing to learn and represent the presented phenomenon in terms of a decision tree.

V. *Publication 5* explores a strategy of implementing fuzzy applications with the CWM rule system in the domain of Semantic Web applications. The article demonstrates the structure and the application of fuzzy action rules, and establishes a mapping between fuzzy rule systems and the crisp CWM rules.

The study provides the initial framework for experimenting and integrating the existing rule systems and describes the concepts of fuzzy reasoning systems and their relationship with empirical data. The possibility of contextual clauses is also discussed in the publication.

VI. *Publication 6* describes an approach of integrating statistical decision-support information with fuzzy action rule systems. The article outlines and analyses the basic components of fuzzy systems with predictive models, establishing the basic rationale and much of the motivation for the thesis.

In brief, the publication outlines the basic characteristics and the domain for working with predictive fuzzy models, anticipating the other publications 5, 4, 3, 2, and 1. Of the two illustrated use cases, calendar agents and educational decision-support applications, the latter was chosen to provide the application framework for the thesis.

Finally, the introduction and the synthesis of the thesis in addition includes previously unpublished results (see Section 1.4).

Chapter 7

Conclusion

Implementing knowledge-intensive applications requires a logical formulation of the problem domain. However, while a suitable logical formulation provides the necessary technical framework of applications, it alone is not sufficient; significant application-specific design efforts and decisions are in addition required. Designing practical applications thus always includes a heuristic aspect that can not be thoroughly formalised.

However, the practical challenges related to application design in turn help pointing out the role of the formal methods and the enabling technologies in the process. In particular, relying upon relatively simple representational systems may simplify the technical design, but as a consequence, may also favour making unjust design decisions. In the context of this study, relying upon crisp representation and reasoning languages easily means neglecting the vague aspects of the application domain.

Unfortunately, improving the descriptional property of the formal systems typically means an increase in system complexity [45]. In addition to the apparent changes in the logical consequences and the behaviour, this means two things. First, analysing the behaviour of systems (consequences of assertions) becomes more difficult or even impossible (consider, e.g., tractability and decision problems). Second, using the system as a tool becomes more difficult, since understanding its syntax and behaviour requires more expertise (consider, e.g., understanding logical models expressed in propositional logic and in predicate logic).

However, the practical usefulness of systems can also increased simply by transforming some existing well-known logical framework into a new intuitive form, suitable for certain applications. This is essentially the stance of description logics [6]. Indeed, from this perspective, type-1 fuzzy logic programs do not essentially add anything to the descriptional power of the underlying crisp logical language since pure Prolog implementations exist.

When vagueness can be captured in modelling, two complementary technical aspects of designing applications may be distinguished. The first and perhaps the more apparent aspect of the work is capturing the logical vocabulary and the formal relationships sufficient deducing and querying a large number of interesting consequences from a relatively small set of axiomatic clauses. This work involves the introduction of the logical language, and organisation of the universe of discourse into *identifiable objects* (essentially: recognising and naming objects). It is important to note that the introduction of predicates also means naming objects; attributes, roles, and other relationships as *identifiable concepts.* The study of (statistical) machine learning and data mining algorithms suggests that concept construction is an *active* process, in which the (human) designers play a central role [71, 81].

The second aspect of the work involves assigning models to clauses that describe their position in relationship with the reasoning architecture. This aspect can be easily overlooked when considering only crisp logic programs. In real-world applications, however, classifications are often vague, or the interaction via the end-user interface includes the use of imprecise terms.

The process of designing fuzzy models thus opens a "new" area for the logical formulation of application domains. A significant design challenge lies in initialising the fuzzy models that yield an appropriate reasoning system. We believe that introducing large-scale reasoning systems has a lot to learn from fuzzy reasoning systems. However, since information is not usually directly available in a (fuzzy) logical form, a suitable bootstrapping process is required, transforming information into a logical form[1].

Having many degrees of freedom in design can make it challenging to balance between different design choices and objectives. This is where formal systems should help, providing a language of making descriptions and an inference system for pinpointing the consequences of a particular design [32].

Analysing the model induction process reveals that the probabilistic and the fuzzy (or vague) arguments are typically mixed in applications. Indeed, it is sometimes claimed that vagueness genuinely complements randomness in modelling indeterminacy [61]. It is fair to say that fuzzy models are often due statistical arguments, even if this aspect of modelling is sometimes ignored. (For instance, consider the often cited examples of fuzzy sets: "the set of old people" or "the set of tall people".)

However, the diffusion of the different concepts in modelling is not necessary a bad thing: inducing fuzzy models upon empirical data enables capturing fuzzy logic programs, accuracy of which can be measured. In other

[1]Note that this process simply makes the information more *usable* from the viewpoint of the reasoning system, perhaps adding context-specific heuristics.

words, the subjectivity of the modelling may be backed up with objective arguments.

Problems may arise if reasoning processes are interpreted in terms of, e.g., probabilistic models instead of truth-functional ones since any probabilistic reasoning process must follow the axioms of probabilities. Fuzzy systems, on the other hand, include less built-in semantics which gives more freedom in application design. The price is that it is easy to associate intuitive semantics to fuzzy systems which in fact do not follow the rules of the intended interpretation. For instance, consider associating the concept of *trust* with fuzzy models. Clearly, this intuitive interpretation is justified only if we agree (verify) that the property of "trust" proceeds in the reasoning as intended. (Considering type-1 fuzzy logic programs, this indeed might be the case when selecting $\otimes = \wedge$.)

Considering the subjective aspect of fuzzy modelling, the fact that people seem to be able to understand each other suggests that, being individual or not, people think essentially in the same way. In turn, this also suggests that human communication relies heavily upon the idea of a *shared context* and *tacit communication.* Implementing knowledge agents thus requires encoding tacit information into explicit form, through *externalisation* [60]. Acknowledging the idea of *context* in knowledge modelling is thus fundamental in large-scale fuzzy applications. Technically speaking, this means that the notion of context needs to formalised, and that the process of aggregating overlapping clauses must smoothly support the in-between area of overlapping contexts.

When simplifying the issue, the idea of context-aware logic programs suggests that there is a natural or even universal way to define certain concepts (predicates). This design stance, however, is problematic at best. In fact, the design of applications is usually heavily constrained by the particular design tradition, legacy systems, the costs of application development and production, and the available means for making measurements. For instance, consider our educational use case and the idea of organising student groups based on students' observed performance during the course. This setting has very a strong institutional bias which is easily overlooked when analysing the performance of applications. Thus, trying to overcome the built-in limitations of the information model (in particular, the amount of available externalised information) might require rethinking the entire application. In this thesis we shift this (essentially unsolved) issue to the domain of bootstrapping systems, acknowledging that sharing certain information simply does not work without sharing and accepting substantial portions of third-party ontologies or other reasoning frameworks as well. Unfortunately, the built-in commitment to rigid ontologies also reveals that the design of the

knowledge agents is vulnerable to the changes on the schema level. This pitfall is common to most knowledge models and it seems that there is not an easy way out from the problem.

As suspected, the tacit aspects of modelling and design in turn make it impossible to externalise everything. In other words, while many of the application problems can not be adequately solved through the formal means alone, it would seem that some of the problems can not be solved at all. Indeed, it is amazing how people manage these issues[2].

Increased externalisation and the added inference capabilities come with a cognitive penalty. The more details we want to capture with the representation and reasoning system, the more complex it will become. At certain point, *the complexity of a system increases so that our ability to make precise and yet significant statements about its behaviour diminishes until precision and significance become almost mutually exclusive characteristics.* This the Zadeh's famous *principle of incompatibility.*

Indeed, it has been considered that fuzzy (linguistic) systems as such might provide a way out from the fundamental complexity related to domain modelling [38]. However, this only seems to be the case when acknowledging the two roles of the *end user* and the *expert user* when manipulating and accessing knowledge (see Figure 1.1). In other words, when the complexity of the underlying information database is mapped onto a suitable (linguistic) logical form of knowledge, such simplifying transition may take place. However, expert insight and help is still needed in the transformation process.

As a result, end-users may find it difficult to appreciate the decisions made by the reasoning system, behaviour of which they do not fully understand. This raises the very difficult issue of *giving understandable explanations*, which in turn bridges the work to the bigger picture of *acceptable* human-computer interaction [33]. It would seem that while fuzzy linguistic procedures may provide a viable approach to this problem, some deductions are counter-intuitive to the least, i.e. without the theoretical understanding of the reasoning system and a direct access to the expert user interface and the underlying information database.

This is to say that while fuzzy knowledge agents indeed provide means to encapsulate the fundamental complexity of various application domains, the fuzzy knowledge interface does not as such enclose the explanations of the deductions in a form suitable for the end-user. To some extent, this problem might be solved with explicit meta-reasoning and question answering proce-

[2] And perhaps we just *can not* manage these issue either: since some of the issues can not be reasonably defined and measured, direct feedback of potential failure(s) is seldom available.

dures associated with the typical deductions. However, further explanations would then again require access to the underlying, more complex system.

Finally, it is worth emphasising that the meaning of words becomes apparent only when paralleled with actions. Strictly speaking, formal clauses bear little meaning when detached from an appropriate reasoning framework. However, the practical impact of decision-support applications is not solely determined by their formal consequences. For instance, it is relatively easy to come up with a concept "poor student" but what does it exactly mean? As such, it does not mean anything more than a predicted model of an associated predicate within an institutional context. Its formal meaning is determined through the associated procedures within, say, an educational decision-support application including procedures using that predicate. This meaning is then interpreted within a particular informal context (i.e. *pragmatics*). However, it is important to observe that other applications of the predicate (e.g. as simple classifying labels) may in practice add a significant informal semantic layer to the formal interpretation, which, as intuitive decision-support data, may turn out superior to the formal deductions. For instance, consider the visualisation of the classification trajectories of "poor students" in educational decision support (see Figure 5.4).

In this thesis we have pointed out a path of designing and implementing fuzzy knowledge agents. The work has essentially included the development of a type-1 fuzzy logic programming framework and an approach of inducing fuzzy models, with use cases and demonstrations drawn from the field of educational applications. In short, this work aims establishing a relatively straightforward foundation of knowledge-intensive modelling in real-world applications.

Considering the local context of the work, the interesting areas of future work include:

- Knowledge modelling based on the so-called *folksonomies*, i.e., logical description of application domains without a clear, centralised design.
- Modelling and implementing various *systems of (linguistic) business rules.*
- Fuzzy *reactive systems.*
- *Context-aware reasoning* in the Semantic Web (with massively distributed logic programs).
- *Visualisation of knowledge models.*

Indeed, several related research activities have been already launched at the Institute of Mathematics, Hypermedia laboratory, Tampere University of Technology.

Formalisation of knowledge and application-specific reasoning is challenging art that combines many of the fields related to applied computing; modelling and logic programming, information retrieval, statistics, machine learning, data mining, and domain-specific reasoning, to name a few. As with so many applied domains, the success of implementing fuzzy knowledge agents is ultimately determined by the weakest component in the design process. A common nominator of the above research topics is the eventual engagement with the details of *domain-specific modelling.* While a working theory helps clarifying many of the strategic design decisions involved in the applications, significant challenges lie in obtaining case-specific data with acceptable costs, establishing appropriate long-term practices, and keeping the induction and the reasoning processes transparent to be of any practical use.

And, unless we are very careful, the very language we use deceives us. With the words of Lao Tzu from some 1500 years ago [44]:

> We look at it, and we do not see it, and we name it 'the Equable.' We listen to it, and we do not hear it, and we name it 'the Inaudible.' We try to grasp it, and do not get hold of it, and we name it 'the Subtle.' With these three qualities, it cannot be made the subject of description; and hence we blend them together and obtain The One.

Bibliography

[1] Abreu, S. & Diaz, D. 2003. Objective: In Minimum Context. *Lecture Notes in Computer Science*, Springer-Verlag GmbH, 2916 / 2003, pp. 128-147.

[2] Adomavicius, G. & Tuzhilin, A. 2005. Personalization technologies: a process-oriented perspective, *Communications of the ACM,* October 2005, 48 (10), pp. 83-90.

[3] Aduna 2005. *openRDF.org ...home of Sesame.* Aduna, Available at http://www.openrdf.org/

[4] Agarwal, S. & Hitzler, P. Modeling Fuzzy Rules with Description Logics. In *Workshop of OWL Experiences and Directions*, November 11-12, Galway, Ireland.

[5] Arriaga, F.de, Alami, M.El. & Arriaga, A. 2005. Evaluation of Fuzzy Intelligent Learning Systems. In *Proceedings of m-ICTE2005*, June 7-10, Spain.

[6] Baader, F., Calvanese, D., McGuiness, D. L. Nardi, D., Patel-Schneider, P. F. (2002). *Description Logic Handbook*, Cambridge University Press

[7] Berners-Lee, T., Hendler, J. & Lassila, O. 2001. The Semantic Web. *Scientific American*, May 2001.

[8] Berners-Lee, T. 2004. *Cwm - a general purpose data processor for the semantic web*, W3C, Available at http://www.w3.org/2000/10/swap/doc/cwm

[9] Berners-Lee, T. 2004. Home page of the cwm - a general purpose data processor for the semantic web. World Wide Web Consortium 1994-2004. Available at http://www.w3.org/2000/10/swap/doc/cwm.

[10] Bossi A., Bugliesi, M., Gabbrielli, M. Levi, G. & Meo, M. C. 1993. Differential Logic Programming. In *Proceedings of the 20th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, USA: ACM Press, pp. 359-370.

[11] Bratko, I. 2001. Prolog Programming for Artificial Intelligence (3rd Edition). UK: Addison-Wesley.

[12] Bray T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., & Yergeau, F. 2006. Extensible Markup Language (XML) 1.0 (Fourth Edition), *W3C Recommendation,* 16 August 2006, W3C, Available at http://www.w3.org/TR/2006/REC-xml-20060816/.

[13] Breese, J. S., Heckerman, D. & Kadie., C. 1998. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Annual Conference on Uncertainty in Artificial Intelligence*, July 1998, pp. 43-52.

[14] Bugliesi, M., Lamma, E. & Mello, P. 1994. Modularity in Logic Programming, *Journal of Logic Programming*, 19/20, pp. 443-502.

[15] Chen, J., Zhou, B., Shi, J., Zhang, H., & Fengwu, Q. 2001. Function-based object model towards website adaptation. In *Proceedings of the 10th international conference on World Wide Web*, Hong Kong, Hong Kong, pp. 587-596.

[16] Clocksin, W.F. & Mellish, C.S. (1984) *Programming in Prolog*, USA: Springer-Verlag.

[17] Cox, E. 2005. *Fuzzy Modeling and Genetic Algorithms for Data Mining and Exploration.* USA: Elsevier.

[18] CSAIL 2005. *Haystack Project*, MIT/CSAIL, Available at http://haystack.lcs.mit.edu/

[19] Cuzzocrea, A.& Bellas, F. 2004. Distributed knowledge networks: towards a new paradigm for delivering knowledge. *Proceedings of Information Technology: Coding and Computing*, 2004, Vol. 1, pp. 118-123.

[20] Dignum, V. 2004. Personalized support for knowledge sharing. *Proceedings of the conference on Dutch directions in HCI* , Amsterdam, Holland.

[21] Fiala, Z., Hinz, M., Houben, G-J. & Frasincar, F. 2004. Design and Implementation of Component-Based Adataptive Web Presentations. In *Proceedings of SAC'04*, March 14-17, Nicosia, Cyprus, pp. 1698-1704.

[22] FLINT, 2005. Home page of the FLINT toolkit. Logic Programming Associates (LPA), Available at http://www.lpa.co.uk/fln_det.htm.

[23] Fowler, M. & Scott, K. (1999) *UML Distilled, Second Edition: A Brief Guide to the Standard Object Modelling Language*, USA: Addison-Wesley.

[24] Ganter, B. & Wille, R. (1996) *Formal Concept Analysis: Mathematical Foundations*, USA: Springer-Verlag.

[25] Gebhardt, J. & Kruse, R. 1992. A possibilistic interpretation of fuzzy sets by the context model. In *Proceedings of IEEE International Conference on Fuzzy Systems*, 8-12 March 1992, pp. 1089-1096.

[26] Gimson, R., Finkelstein, S., R., Maes, S. & Suryanarayana, L. (Eds.) 2003. Device Independence Principles. *W3C Working Group Note* , Available at http://www.w3.org/TR/2003/NOTE-di-princ-20030901/

[27] Diaz, D. 2007. The GNU Prolog web site, Available at http://www.gprolog.org/

[28] Guttenplan, S. (Ed.) (1994) *A Companion to the Philosophy of Mind*, USA: Blackwell Publishers Ltd.

[29] Hawke, S., Tabet, S., & Sainte Marie, C. de 2005. Rule Language Standardization, Report from the W3C Workshop on Rule Languages for Interoperability, *W3C Workshop on Rule Languages for Interoperability*, April 27-28, 2005, Washington, D.C., USA, W3C, Available at http://www.w3.org/2004/12/rules-ws/report/.

[30] Hawke, S., Tabet, S., Sainte Marie, C. de, & Grosof, B. 2005. *W3C Workshop on Rule Languages for Interoperability*, W3C, Available at http://www.w3.org/2004/12/rules-ws/cfp

[31] Hayes, P. 2004. RDF Semantics, *W3C Recommendation,* 10 February 2004, W3C, Available at http://www.w3.org/TR/2004/REC-rdf-mt-20040210/.

[32] Hintikka, J. 1996. What Do Logic and Computer Science to Do with Each Other?, *Logic, Mathematics and the Computer*, Publications of the Finnish Artificial Intelligence Society, No. 14, pp. 85-88.

[33] Hoffmann, A. 1998. *Paradigms of Artificial Intelligence: A Methodological & Computational Analysis*, Singapore: Springer.

[34] Hullermeier, E. 2005. Fuzzy methods in machine learning and data mining: Status and prospects. *Fuzzy Sets and Systems*, Volume 156 (3), pp. 387-406.

[35] Hyvönen, E. 2004. *MuseumFinland – Finnish Museums on the Semantic Web.* University of Helsinki, Available at http://www.cs.helsinki.fi/group/seco/museums

[36] Ibrahim, A.M. 2001. Assessment of Distance Education Quality Using Fuzzy Sets Model. In *Proceedings of the International Conference on Engineering Education*, August 6-10, 2001 Oslo, Norway.

[37] Jameson, A. 1995. Numerical Uncertainty Management in User and Student Modeling: An Overview of Systems and Issues. *User Modeling and User-Adapted Interaction*, 5 (3-4), pp. 193-251.

[38] Jang, J-S.R. & Sun, E.M. 1997. *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*, USA: Prentice Hall.

[39] Kaminski, D.M. 1992. Introducing the Fuzzy Paradigm Using Prolog. *Proceedings of the twenty-third SIGCSE technical symposium on Computer science education*, Kansas City, Missouri, USA: ACM Press, pp. 202-206.

[40] Kavcic, A., Pedraza-Jimenez, R., Molina-Bulla, H., Valverde-Albacete, F. J., Cid-Sueiro, J. & Navia-Vazquez, A. 2003. Student Modelling Based on Fuzzy Inference Mechanisms. In *Proceedings of the EUROCON 2003*, Ljubljana, Slovenia.

[41] Knowledge Web 2006. Knowledge Web home page, Available at http://knowledgeweb.semanticweb.org/

[42] Kosko, B. 1990. Fuzziness vs. probability, *International Journal of General Systems.* Vol. 17. pp. 211-240.

[43] KTWeb 2004. KTWeb home page, Available at http://www.ktweb.org/

[44] Legge, J. 1891 (Trans.). *Lao Tzu: The Tao The Ching.* UK: Oxford University Press.

[45] Levesque, H.J. & Brachman, R.J. 1987. Expressiveness and Tractability in Knowledge Representation and Reasoning. *Computational Intelligence*, 3, pp. 78-93.

[46] Liu, H. & Maes, P. 2004. What would they think?: a computational model of attitudes. In *Proceedings of the 9th international conference on Intelligent user interfaces*, Funchal, Madeira, Portugal, USA: ACM Press, pp. 38-45.

[47] Loke, S. W. & Davison, A. 1996. Logic Programming with the World-Wide Web. In *Proceedings of the Hypertext'96*, Washington DC, USA, pp. 235-245.

[48] Lopes, A. L. M. & Lanzer, E. A. 2002. Data envelopment analysis - DEA and fuzzy sets to assess the performance of academic departments: a case study at Federal University of Santa Catarina - UFSC. *Pesquisa Operacional*, 22 (2), pp. 217-230.

[49] Lum, W. Y. & Lau, F.C.M. 2002. A Context-Aware Decision Engine for Content Adaptation. *Pervasive Computing.* IEEE, 1 (3), pp. 41-49.

[50] Ma. J. & Zhou, D. 2000. Fuzzy Set Approach to the Assessment of Student-Centered Learning. *IEEE Transactions on Education*, 43 (2), May 2000, pp. 237-241.

[51] Manola, F., Miller, E. & McBride, B. (Eds.) 2004. RDF Primer, *W3C Recommendation*, 10 February 2004, Available at http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

[52] Mazzieri, M. 2004. A Fuzzy RDF Semantics to Represent Trust Metadata, In *Proceedings of Semantic Web Applications and Perspectives (SWAP), 1st Italian Semantic Web Workshop*, 10 December 2004, Ancona, Italy, Available at http://semanticweb.deit.univpm.it/swap2004/cameraready/mazzieri.pdf.

[53] Mendel, J. M. 2001. *Uncertain Rule-Based Logic Systems: Introduction and New Directions.* USA: Prentice Hall.

[54] Mitchell, T. 1997. *Machine Learning.* USA: McGraw-Hill.

[55] Monteiro, L. & Porto, A. 1989. Contextual Logic Programming. In *Proceedings of the 6th International Conference on Logic Programming*, USA: The MIT Press, pp. 284-302.

[56] Negroponte, N. 1997. Agents: From Direct Manipulation to Delegation. In Bradshaw, J. M. (Ed.) *Software Agents*, London, UK: AAAI Press/ MIT Press.

[57] Newton, G., Pollock, J. & McGuinness, D., L. 2004. *Submission request to the World Wide Web Consortium: Semantic Web Rule Language (SWRL)*, W3C, Available at http://www.w3.org/Submission/2004/03/

[58] Nogueira, V., Abrey, S. & David, G. 2004. Towards Temporal Reasoning in Constraint Contextual Logic Programming. In *Proceedings of 3rd International Workshop on Multiparadigm Constraint Programming Languages*, Saint-Malo, France, 6-10 September, 2004, pp. 119-131.

[59] Nokelainen, P., Silander, T., Tirri, H., Nevgi, A. & Tirri, K. 2001. Modeling Students' Views on the Advantages of Web-Based Learning with Bayesian Networks. In *Proceedings of the Tenth International PEG Conference*, June 23-26, 2001, Tampere, Finland.

[60] Nonaka, I. & Takeuchi, H. 1995. *The Knowledge-Creating Company.* USA: Oxford University Press.

[61] Novak, V. 2005. Are fuzzy sets a reasonable tool for modeling vague phenomena? *Fuzzy Sets and Systems*, 156, pp. 341-348.

[62] Nykänen, O. 2006. Inducing Fuzzy Models for Student Classification. *Educational Technology & Society Journal.* Volume 9, Issue 2, pp. 223-234.

[63] Nykänen, O. 2006. An Approach to Logic Programming with Type-1 Fuzzy Models Using Prolog. *Proceedings of the IADIS International Conference of Applied Computing.* 25-28 February, 2006 San Sebastian, Spain.

[64] Nykänen, O. 2005. Implementing Semantic Web Agents That Learn Upon Experience: A Machine Learning Module for the CWM Rule System. *Proceedings of the IADIS International Conference of Applied Computing.* 22-25 February, 2005 Algarve, Portugal, Vol. 1. pp. 409-416.

[65] Nykänen, O. 2005. On the development of XML based specifications at the W3C: A brief review of the future road (Towards the semantic web). Keynote in *Proceedings of the 9th XML Finland Conference: XML - the Enabling Technology for Integrating Business Processes.* March 8-9, 2005, Pori, Finland, pp. 4-19.

[66] Nykänen, O. 2004. Fuzziness as a Recognition Problem: Using Decision Tree Learning Algorithms for Inducing Fuzzy Membership Functions. *Data Mining V: Data Mining, Text Mining, And Their Business Applications.* WIT Press, UK, pp. 143-153.

[67] Nykänen, O. 2004. On Implementing Fuzzy Semantic Web Agents with the CWM Rule System. *Proceedings of the IADIS WWW/Internet 2004 Conference.* 6-9 October 2004, Madrid, Spain, pp. 1075-1078.

[68] Nykänen, O. 2004. An Approach for Integrating Statistical Decision-Support Data with Fuzzy Action Rules. *Proceedings of IEEE International Conference on Advanced Learning Technologies.* 30 August - 1 September 2004, Joensuu, Finland, pp. 156-160.

[69] Nykänen, O. 2004. On Managing and Visualizing the Semantic Web: Using Queries for Identifying and Labeling the Metastructures of RDF Data. *Proceedings of the IADIS WWW/Internet 2004 Conference.* 6-9 October 2004, Madrid, Spain, pp. 366-373.

[70] Nykänen, O. 2004. Metadata for Learning Resources: Technologies and Directions of the Semantic Web - A Brief Review. *Proceedings of IEEE International Conference on Advanced Learning Technologies.* 30 August - 1 September 2004, Joensuu, Finland. p. 896-897.

[71] Nykänen, O. 2001. *Foundations of Machine Learning.* Licentiate Thesis, Tampere University of Technology, Department of Mathematics, 162 pages.

[72] Nykänen, O. 2000. A Framework for Generating Non-trivial Interactive Mathematical Exercises in the Web: Dynamic Exercises. *ED-MEDIA 2000, World Conference on Educational Multimedia, Hypermedia & Telecommunications.* June 26 - July 1, 2000. Montral, Canada. pp. 1438-1439.

[73] Nykänen, O. 1999. Visualizing Navigation in Educational WWW Hypertext by Introducing Partial Order and Hierarchy. *ED-MEDIA 1999, World Conference on Educational Multimedia, Hypermedia & Telecommunications.* Seattle, Washington, USA, 19.-24.6.1999, 2, pp. 951-957.

[74] Nykänen, O. & Ala-Rantala, M. 1999. A design for a hypermedia-based learning environment. *Education and Information Technologies*, 3, pp. 277-290.

[75] On-To-Knowledge 2002. On-To-Knowledge project home page, Available at http://www.ontoknowledge.org/

[76] OntoWeb 2003. OntoWeb Portal, AIFB Karlsruhe, Germany & VUB STAR Lab, Belgium, Available at http://www.ontoweb.org/

[77] Pan, J.Z., Stoilos, G., Stamou, G., Tzouvaras, V. & Horrocks, I. (To appear). f-SWRL: A Fuzzy Extension of SWRL (Extended version). In *Journal of Data Semantic, special issue on Emergent Semantics.*

[78] Poole, D., Mackworth, A. & Goebel, R. 1998. *Computational Intelligence: A Logical Approach*, USA: Oxford University Press.

[79] Prud'hommeaux, E. & Grosof, B. 2004. *RDF Query Survey*, W3C, Available at http://www.w3.org/2001/11/13-RDF-Query-Rules/

[80] Russel, S. & Norvig, P. 1995. *Artificial Intelligence; A Modern Approach.* USA: Prentice Hall.

[81] Schaffer, C.A., 1994. Conservation Law for Generalization Performance. In *Proceedings of the Int. Conf. on ML*, San Mateo, Ca., Morgan Kaufmann.

[82] Spriestersbach, A., Ziegert, T., Grassel, G., Wasmund, M. & Dermler, G. 2003.) Flexible pagination and layouting for device independent authoring. In *WWW2003 Emerging Applications for Wireless and Mobile access Workshop*, AT&T Labs Research, 2003.

[83] Stamou, G., Pan, J.Z., Tzouvaras, V., & Horrocks, I. 2005. A Fuzzy Extension of SWRL. In *W3C Workshop on Rule Languages for Interoperability*, April 27-28, 2005, Washington, D.C., USA, W3C, Available at http://www.w3.org/2004/12/rules-ws/.

[84] Stathacopoulou R., Magoulas G.D. & Grigoriadou M. (1999) Neural network-based fuzzy modeling of the student in intelligent tutoring systems. In *Proceedings of the INNS-IEEE International Joint Conference on Neural Networks*, Washington, USA, July 1999.

[85] Stathacopoulou, R., Magoulas, G. D., Grigoriadou, M. & Samarakou, M. 2005. Neuro-fuzzy knowledge processing in intelligent learning environments for improved student diagnosis. *Information Sciences*, 170 (2-4), pp. 273-307.

[86] Sterling, L. & Shapiro, E. 1994. *The Art of Prolog: Advanced Programming Techniques*, USA: MIT Press.

[87] Straccia, U. 2005. Description Logics with Fuzzy Concrete Domains. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI-05)*, Edinburgh, Scotland, UK, 26-29 July 2005, pp. 559-567.

[88] Suarez, J. 2003. Student evaluation through membership functions in CAT systems. *Revista Mexicana De Fisica*, 49 (4), pp. 371-378.

[89] Swartz, A. (Ed.) 2000. *RDF Site Summary (RSS) 1.0*, Available at http://web.resource.org/rss/1.0/

[90] Sycara, K. P., Klusch, M., Widoff, S. & Lu. J. 1999. Dynamic Service Matchmaking Among Agents in Open Information Environments. *ACM SIGMOD Record*, 28 (1), pp. 47-53.

[91] Trask, R. L. 1999. *Language: The Basics (Second Edition).* USA: Routledge.

[92] Turban, E., Zhou, D. & Ma. J. 2000. A Methodology for Evaluating Grades of Journals: A Fuzzy Set-based Group Decision Support System. In *Proceedings of the 33rd Hawaii International Conference on System Sciences*, January 4-7, 2000, Maui, Hawaii.

[93] Turunen, E. 1999. *Mathematics Behind Fuzzy Logic.* Germany: Physica-Verlag Heidelberg.

[94] Vaucheret, C., Guadarrama, S. & Muoz. S. 2002 Fuzzy Prolog: A Simple General Implementation using CLP(R). In *Proceedings of the 9th International Conference on Logic for Programming Artificial Intelligence and Reasoning*, Tbilisi, Georgia, October 14-18, October 2002.

[95] Veit, D. 2002. *Matchmaking in Electronic Markets: An Agent-Based Approach towards Matchmaking in Electronic Negotiations.* Germany: Springer-Verlag.

[96] Vidyasagar, M. 1997. *A Theory of Learning and Generalization.* UK: Springer-Verlag.

[97] Völkel, M., Krötzsch, M., Vrandecic, D, Haller, H., & Studet, R. Semantic Wikipedia. In *Proceedings of WWW 2006*, May 23-26, 2006, Edinburgh, Scotland.

[98] W3C 2004. *Semantic Web*, World Wide Web Consortium, 2004, Available at http://www.w3.org/2001/sw/.

[99] Weon, S. & Kim. J. 2001. Learning Achievement Evaluation Strategy Using Fuzzy Membership Function. In *Proceedings of the 31st ASEE/IEEE Frontiers in Education Conference*, October 10-13, 2002, Reno, NV.

[100] Wielemaker, J. 2005. SWI-Prolog's home. HCS, Informatics Institute of the University of Amsterdam. Available at http://www.swi-prolog.org/.

[101] Wiener, N. 1948. *Cybernetics.* USA: Wiley.

[102] Wilbanks, J. (Ed.) 2005. *W3C Workshop on Semantic Web for Life Sciences*, W3C, Available at http://www.w3.org/2004/07/swls-ws.html.

[103] Woods, S. 2000. Dreams of a Unified Information Space. *IEEE Internet Computing*, July-August 2000.

[104] Xenos, M. 2004. Prediction and Assessment of Student Behaviour in Open and Distance education in Computers Using Bayesian Networks. *Computers & Education*, 43 (4), pp. 345-359.

[105] Zadeh, L. 2004. Web Intelligence, World Knowledge and Fuzzy Logic - The Concept of Web IQ (WIQ). In *Proceedigs of the KES 2004*, Wellington, New Zealand, September 20-25, 2004, pp. 1-5.

[106] Zadeh, L.A. 1973. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transactions on Systems, Man, and Cybernetics*, 3 (1), January 1973, pp. 28-44.

Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O. Box 527
FIN-33101 Tampere, Finland