Linköping Studies in Science and Technology
Licentiate Thesis No. 1144

A Framework for Reasoning with Rough Sets

*A*ida da Graça Cleto da Silva Vitória

Linköpings universitet
INSTITUTE OF TECHNOLOGY

LiU-TEK-LIC- 2004:73

Department of Science and Technology
Linköpings Universitet
SE-601 74 Norrköping, Sweden

Norrköping, January, 2005

A Framework for Reasoning with Rough Sets

LiU-TEK-LIC- 2004:73

Department of Science and Technology
Linköpings Universitet
SE-601 74 Norrköping, Sweden

Linköpings universitet
INSTITUTE OF TECHNOLOGY

ISBN 91-85297-15-1 ISSN 0280-7971

Printed in Sweden by UniTryck, Linköping, 2004

Abstract

Rough sets framework has two appealing aspects. First, it is a mathematical approach to deal with vague concepts. Second, rough set techniques can be used in data analysis to find patterns hidden in the data. The number of applications of rough sets to practical problems in different fields demonstrates the increasing interest in this framework and its applicability.

Most of the current rough sets techniques and software systems based on them only consider rough sets defined explicitly by concrete examples given in tabular form. The previous research mostly disregards the following two problems. The first problem is related with how to define rough sets in terms of other rough sets. The second problem is related with how to incorporate domain or expert knowledge.

This thesis[1] proposes a language that caters for implicit definitions of rough sets obtained by combining different regions of other rough sets. In this way, concept approximations can be derived by taking into account domain knowledge. A declarative semantics for the language is also discussed. It is then shown that programs in the proposed language can be compiled to extended logic programs under the paraconsistent stable model semantics. The equivalence between the declarative semantics of the language and the declarative semantics of the compiled programs is proved. This transformation provides the computational basis for implementing our ideas. A query language for retrieving information about the concepts represented through the defined rough sets is also defined. Several motivating applications are described. Finally, an extension of the proposed language with numerical measures is discussed. This extension is motivated by the fact that numerical measures are an important aspect in data mining applications.

[1]This work has been partially supported by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (http://rewerse.net).

Acknowledgements

I would like to express my sincere gratitude towards my supervisor, Jan Małuszynski, for his invaluable guidance and constant support.

At the department of Computer Science, at New University of Lisbon, my thanks goes to Carlos Viegas Damásio for his suggestions and the time taken to discuss many aspects of this work with me.

I also would like to thank the following people for their comments and support: Andrzej Szałas, Jan Komorowski, Wojciech Ziarko, and Andrzej Skowron.

Many thanks to Pierangelo Dell'Acqua, Stan Miklavcic, Igor Zozoulenko, Ivan Rankin, Zhuangwei Liu, Bo Zhu, Sorin Manolache, and Fatima Bilajbegovic for their friendship and for having created an enjoyable working environment.

Um agradecimento muito especial para a minha família em Portugal que apesar da distância me deu sempre todo o apoio.

Aida Vitória
Norrköping, December 2004

To Pierangelo Dell'Acqua

Contents

1 Introduction **7**
- 1.1 Motivation . . . 7
- 1.2 Problem Formulation . . . 8
- 1.3 Contributions . . . 9
- 1.4 Structure of the Thesis . . . 10

2 Basics of Rough Set Theory **13**
- 2.1 Rough Sets: The Main Idea . . . 14
- 2.2 Approximations and Rough Sets . . . 19
- 2.3 Decision Rules . . . 22
- 2.4 Numerical Measures . . . 27
  - 2.4.1 Measuring Quality of Decision Rules . . . 27
  - 2.4.2 Reducts . . . 29
- 2.5 Prediction . . . 32
- 2.6 The Variable Precision Rough Set Model . . . 33

3 Logic Programming Framework **35**
- 3.1 The Main Idea: Definite Logic Programs . . . 35
- 3.2 Extended Logic Programs . . . 38
  - 3.2.1 Declarative Semantics of Extended Logic Programs . . . 40
  - 3.2.2 Queries . . . 43

4 A Language for Defining Rough Relations **45**
- 4.1 The Syntax . . . 46
- 4.2 The Declarative Semantics . . . 50
- 4.3 Computing the Semantics of Rough Programs . . . 57
  - 4.3.1 Compiling Rough Programs into Extended Logic Programs . . . 58

4.3.2 Correctness of the Compilation Procedure 63
4.4 Queries 72

5 Application Examples **81**
5.1 Hierarchy-Structured Decision Tables 81
5.2 Avoiding Expensive Tests 87
5.3 Representing Default Knowledge 93

6 The Rough Knowledge Base System **99**
6.1 Rough Sets Revisited 101
6.2 A Language with Numerical Measures 103
6.3 Compilation of *RKBS* Programs 108
6.3.1 All Solutions Predicates in *Prolog* 109
6.3.2 The Compilation 111
6.4 The Query Language of *RKBS* 119
6.5 Application Examples 121
6.5.1 Variable Precision Rough Relations 121
6.5.2 Avoiding Expensive Tests Revisited 122

7 Conclusions and Future Work **129**
7.1 Concluding Remarks 129
7.2 Future Work 132

A Notation Summary **135**

Bibliography **139**

List of Figures

2.1 The four regions generated by a rough set S. 21

6.1 Compilation procedure of *RKBS*. 111
6.2 The Rough Knowledge Base System. 125
6.3 The *RKBS* showing query $\mathcal{Q}_1$ and its answer. 126
6.4 The *RKBS* showing query $\mathcal{Q}_2$ and its answer. 127

List of Tables

2.1 Table of people with sight problems 15
2.2 Decision table derived from Table 2.1 17

5.1 Table of patients with heart problems. 83
5.2 Decison table associated with class E_1. 84
5.3 Decison table associated with class E_2. 84
5.4 Decision table obtained by integrating tables 5.2 and 5.3 with table 5.1. 84
5.5 Decison table associated with the boundary region of table 5.1. 86
5.6 Decison table associated with the boundary of table 5.5. . . . 86
5.7 Decision table of patients with heart problems. 90
5.8 Decision table classifying vehicle distances. 94
5.9 Decision table classifying the danger of several traffic situations. 95

Chapter 1

Introduction

This thesis addresses the problem of using rough sets for knowledge representation. We propose an extension of the basic rough set formalism [Paw82] that caters for the representation of vague concepts and reasoning about those concepts.

We present a language that allows the user to define rough sets implicitly. This contrasts with most of the currently existing systems based on rough set techniques where rough sets can only be defined explicitly by a set of examples.

We also introduce a query language to retrieve non-trivial knowledge implied by the defined rough sets.

This introductory chapter starts by presenting the motivation that drove us into this research (section 1.1), followed by the formulation of the concrete problem addressed in this thesis (section 1.2), and then we highlight the main contributions of our work (section 1.3). Finally, we give an overview of the structure of this thesis (section 1.4).

1.1 Motivation

Rough sets techniques [Paw82, NNSS96] can be used to discover new interesting data patterns (or knowledge) hidden in large tables with many lines and several columns. In more concrete terms, the main aim of rough set techniques is to synthesize descriptions of concepts from the data in the tables. The concept descriptions consist of a set of decision rules. These decision rules can be used in two different perspectives. First, rules can be

used for building predictive models (i.e. classifiers) from the data. Second, rules can reveal interesting relationships in the data, i.e. each rule may represent an interesting pattern hidden in the data. These techniques have been successfully applied to many real problems in different areas like medicine [KØ99, LR04], economy [TS02], and bioinformatics [MK02]. Therefore, it is not a surprise that rough set methods are enjoying an increasing popularity in the data mining field.

The rough set framework has two major appealing aspects. First, it proposes a method to handle inconsistencies due to imprecise or noisy data. Approximate concept descriptions can then be induced. This point is particularly relevant from the point of view of knowledge representation. As a consequence of using approximations, the derived decision rules describing concepts are categorized into *certain* and *possible* rules. In addition, rough set techniques have a clear mathematical foundation.

Several other important problems are also tackled in the context of rough sets:

- reducing the number of relevant attributes;
- measuring the significance of attributes;
- discovering the degree of dependency between attributes;
- generating classifiers with the possibility to predict more than one class for an object.

What we wish to emphasize here is the relevance of the rough set framework from the knowledge representation and data mining perspectives. The capability to handle vague and contradictory knowledge makes rough sets an important technique that can be incorporated in knowledge base systems. On the other hand, rough set methods can also be used to perform data exploration what makes them relevant from a data mining point of view. These two aspects account for the motivation that drove us in this research.

1.2 Problem Formulation

Most of the research in the rough sets field has been focused on the following issues: algebraic characterization and interpretation of rough sets [Pag97]; relations of rough set theory with other theories to represent knowledge, like modal logics [YL96]; integration of rough sets with other techniques like inductive logic programming [MK00] or fuzzy sets [Wyg89, DP92]; extensions

to the basic rough set formalism using different types of indiscernibility relations and more general definitions of upper and lower approximations [YL96, SS96, Zia93]; construction of software tools for data mining based on rough sets methods [ØK97]; application of rough set techniques to real problems [KØ99, MK02, ZF02].

Most of the current rough sets techniques and software systems based on them only consider rough sets defined explicitly by concrete examples given in tabular form. The previous research mostly disregards the following problems.

- How to define rough sets in terms of other rough sets. For instance, we may wish to express that a rough set is obtained as a projection of another rough set over a subset of its attributes.
- How to incorporate domain or expert knowledge. An example of domain knowledge could be *"if a gene participates in cytoplasmic transport then it may also participate in the transport process"*. A question arises of how concept approximations can be derived by taking into account not only the examples provided explicitly by one or more tables but also the domain knowledge.

The problems described above are in the focus of this thesis. They are also addressed in [DŁS02] presenting the Computer Aided Knowledge Engineering technique supported by system *CAKE*. However, several important differences exist between this system and our framework. This issue is further discussed in chapter 7.

1.3 Contributions

The main contributions of this thesis are as follows.

- Definition of a language [MV02, VM02, VDM03b] that caters for implicit definitions of rough sets obtained by combining different regions of other rough sets (e.g. lower approximations, upper approximations, and boundaries). The language also allows defining rough sets in terms of explicit examples, as in most currently available systems. A declarative semantics for the language is also proposed.
- Definition of a query language [VDM03b] for retrieving information about the concepts represented through the defined rough sets.

- Definition of a computational engine for the proposed language. This engine is obtained by a translation of the proposed language to the language of extended logic programs, under the paraconsistent stable model semantics. We also prove the correctness of the proposed translation with respect to the declarative semantics of the language. In this way we establish a link between two important fields, rough set theory and paraconsistent logic programming [DP98].
- Several motivating applications are discussed [VDM03a]. These examples show that several useful techniques and extensions to rough sets, reported in the literature, and implemented in an "ad hoc" way can be naturally expressed in our language.
- Extension of the proposed language with numerical measures [VDM04] is presented. This extension is motivated by the fact that numerical measures are an important aspect in data mining applications.
- A software system based on the proposed language. The system was developed by R. Andersson [And04b] under joint supervision of the author and J. Małuszyński.

Although the major ideas presented in this thesis have been previously published in conference and journal papers, there are are some new notions that are not addressed in those previous publications. First, we present a declarative semantics for the language that caters for implicit definition of rough sets, without considering quantitative measures. Second, the declarative semantics of the query language is also formalized. Third, the correctness of the transformation of rough programs (without quantitative measures) into extended logic programs is only proved in this thesis. Finally, the correctness of the query answering algorithm has not been previously published.

This thesis also gives a more comprehensive introduction to both rough sets and logic programming main notions than the previous publications. Hence, readers acquainted with none or just one of the fields can easily read this work.

1.4 Structure of the Thesis

The rest of this thesis is organized as follows.

- Chapter 2 gives an introduction to rough sets and a brief overview of how several main problems are addressed in this framework.

- Chapter 3 surveys some important notions of logic programming and paraconsistent stable model semantics. These topics help the reader to understand the transformation technique applied to the proposed language.

- Chapter 4 introduces formally a language that caters for implicit definitions of rough sets in terms of other rough sets. We present the declarative semantics of the language. We also show a transformation of programs in this language to paraconsistent logic programs. Moreover, we prove that this transformation is correct with respect to the declarative semantics of the language. In addition, a query language is also defined and an algorithm to obtain answers to the queries is discussed.

- Chapter 5 demonstrates the feasibility of our approach on practical applications by formulating in our language several problems, presented in the rough set literature.

- Chapter 6 proposes an extension of the language with numerical measures. We also give an overview of a software system, available through a Web page, based on these ideas.

- Chapter 7 concludes this thesis and points to several problems that deserve further research.

Chapter 2

Basics of Rough Set Theory

This chapter provides a brief overview of rough set theory. Rough set theory was introduced by Z. Pawlak [Paw82, Paw91] in the early eighties as a methodology for handling uncertainty in data.

The underlying idea of rough set theory is that several objects may look similar due to the limitations in our knowledge. Hence, it is only possible to distinguish classes of objects rather than individual objects. Consequently, only approximate descriptions of concepts (sets of objects) can be constructed. Section 2.1 discusses this idea while section 2.2 presents the notion of concept approximations and formalizes the notion of rough set.

One of the important problems addressed in the rough set framework is the generation of decision rules from which classifiers are then built. Section 2.3 is devoted to this topic.

Numerical measures are another central problem in rough set theory and this is the issue addressed in section 2.4. We first survey some basic numerical measures that can be used to analyze the quality of the derived decision rules. We then discuss how to measure the degree of dependency between attributes and significance of attributes.

Classifiers obtained by the rough set techniques may predict more than one class for an object. The problem of how multiple class prediction can be handled is briefly surveyed in section 2.5.

We conclude this chapter by presenting, in section 2.6, an extension to the basic rough set formalism, called Variable Precision Rough Set Model, that is widely used in practical applications.

Most of the contents of this chapter are based on the ideas presented in the tutorials [KPPS99, Zia02b].

2.1 Rough Sets: The Main Idea

Datasets in many practical problems are presented as a single database relation or table. For instance, entries in the table may correspond to persons with sight problems and they record for each person whether he has astigmatism, the person age, whether the tear production is normal or reduced, and whether the person is currently using spectacles. Assume that all these persons have experimented the use of contact lenses. The table also records for each person whether he has experienced any major problem, related to the use of contact lenses, that led him to stop using contact lenses. A concrete example of such table is given.

Example 2.1 *In table 2.1 the column headings (or attributes) have the following meaning:* `Ast` *stands for astigmatism and can have the value* 0 *(no astigmatism) or* 1 *(with astigmatism);* `Age` *can have the values* 0 *(not more than* 20 *years old),* 1 *(more than* 20 *years old but not more than* 50 *years old), and* 2 *(more than* 50 *years old).* `TearP` *stands for tear production and can have the value* 1 *(reduced tear production) or* 2 *(normal tear production);* `Spec` *stands for spectacles and can have the value* 0 *(using spectacles) or* 1 *(not using spectacles); and* `Lenses` *stands for contact lenses and can have the value* 0 *(stopped using contact lenses) or* 1 *(did not stop using contact lenses).*

□

As the example above shows, objects of a given universe U (e.g. people with sight problems) are described in terms of certain chosen attributes (e.g. tear production). An attribute a can be seen as a total function $a : U \rightarrow V_a$, where V_a is called the *value domain* of a. In this thesis, we assume that we do not have missing (unknown) attribute values. Thus, every object is associated with a tuple of attributes.

The special constant `null` may belong to the value domain V_a off an attribute a. If for an object $o \in U$, $a(o) =$ `null`, then this means that for this particular object o the value of attribute a is not defined (alternatively, attribute a could be seen as a partial function). Note that `null` does not

	Ast	Age	TearP	Spec	**Lenses**
o_1	0	0	1	0	0
o_2	1	1	1	0	0
o_3	1	0	2	1	1
o_4	1	1	1	0	0
o_5	1	0	1	0	0
o_6	0	0	1	0	0
o_7	0	2	1	1	1
o_8	0	2	1	1	1
o_9	1	2	1	0	1
o_{10}	0	0	2	1	0
o_{11}	1	0	1	0	0
o_{12}	1	0	1	0	0
o_{13}	1	0	2	1	1
o_{14}	1	2	2	1	1
o_{15}	0	2	1	1	1
o_{16}	0	0	2	1	1
o_{17}	1	0	1	0	0
o_{18}	1	0	2	1	0

Table 2.1: Table of people with sight problems

denote a missing value. In the latter case a value exists but it is not known, while in our framework `null` means that the attribute value is not relevant. For example, if patient's age is '`<2`' then the value of the attribute `employer` should be `null`.

Objects of the universe are often classified as belonging or not to a predefined class determined by one of the attributes, often called *decision attribute*. Consider again the example above. We may consider two classes of persons: those who had to abandon the use of contact lenses and those who had not. Each person $o \in U$ is then classified as belonging to the former class ($\texttt{Lenses}(o) = 0$) or to the latter one ($\texttt{Lenses}(o) = 1$). Hence, `Lenses` is in this case the decision attribute.

The notion of decision table is formalized to capture these ideas.

Definition 2.1 *A* decision table $\mathcal{D}$ *is a triple* (U, A, d)*, where* U *is a set of objects,* A *is a set of condition attributes, and* d *is a (often binary) decision attribute such that* $\texttt{null} \notin V_d$ *(i.e. for each object* $o \in U$*,* $d(o)$ *is defined).*

The table of example 2.1 can be seen as the decision table $\mathcal{L}enses = (U, A, \texttt{Lenses})$, where $U = \{o_1, \ldots, o_{18}\}$ and $A = \{\texttt{Ast}, \texttt{Age}, \texttt{TearP}, \texttt{Spec}\}$. Moreover, using the attributes from A, persons o_3 and o_{18} are indiscernible from each other because they are represented by the same tuple of condition attributes $\langle 1, 0, 2, 1\rangle$. However, they have different outcomes for the decision attribute: o_3 did not have to stop using contact lenses, while o_{18} did have. This points out that the information available in a decision table may be contradictory.

Definition 2.2 *Given a decision table $\mathcal{D} = (U, A, d)$, an object $o_i \in U$ is* indiscernible from *object $o_j \in U$ if and only if, for all condition attributes $a \in A$, $a(o_i) = a(o_j)$.*

Hence, a decision table $\mathcal{D} = (U, A, d)$ induces an *indiscernibility relation* R_A

$$R_A = \{(o_i, o_j) \in U^2 \mid o_i \text{ is indiscernible from } o_j\} .$$

The indiscernibility relation is an equivalence relation and it induces a partition of the universe U into equivalence classes. These equivalence classes are also known in the rough set literature as *indiscernibility classes* or *elementary sets*. The pair (U, R_A) is called *approximation space* and R^*_A denotes the set of its equivalence classes.

Note that if $a(o) = \texttt{null}$, for some object o of an indiscernibility class, then $a(o') = \texttt{null}$, for every other object o' in the same equivalence class.

To simplify the presentation of several central ideas in later sections of this chapter (e.g. decision rules), we introduce informally the notion of derived decision table.

Example 2.2 *The decision table 2.2 $\mathcal{L}enses' = (U', A, \texttt{Lenses'})$ is derived from table 2.1. Column* Class *designates an indiscernibility class of table 2.1. Hence, the universe U' of the derived table is composed of indiscernibility classes. The condition attributes remain the same as in table 2.1, i.e. $A = \{\texttt{Ast}, \texttt{Age}, \texttt{TearP}, \texttt{Spec}\}$. The values for the decision attribute* Lenses' *are now non-empty subsets of $\{0, 1\}$. If an indiscernibility class E only contains objects whose outcome for the decision attribute in table 2.1 is 0 (1) then the decision attribute in this derived table has value $\{0\}$ ($\{1\}$). However, if some objects in an indiscernibility class E have outcome 0 for decision attribute* Lenses *while other objects belonging to E have outcome 1, then the decision attribute in this derived table has value $\{0, 1\}$ (i.e.* Lenses'$= \{0, 1\}$*).*

There are 8 *indiscernibility classes:*

$$\begin{aligned} E_1 &= \{o_1, o_6\}, \\ E_2 &= \{o_3, o_{13}, o_{18}\}, \\ E_3 &= \{o_2, o_4\}, \\ E_4 &= \{o_7, o_8, o_{15}\}, \\ E_5 &= \{o_9\}, \\ E_6 &= \{o_5, o_{11}, o_{12}, o_{17}\}, \\ E_7 &= \{o_{14}\}, \\ E_8 &= \{o_{10}, o_{16}\}. \end{aligned}$$

From the second line of table 2.2, we can read that indiscernibility class E_2 *contains some objects whose outcome for the decision attribute* `Lenses` *is* 0 *(*o_{18}*) while other objects in* E_2 *have outcome* 1 *(*o_3 *and* o_{13}*).*

`Class`	`Ast`	`Age`	`TearP`	`Spec`	**Lenses'**
E_1	0	0	1	0	$\{0\}$
E_2	1	0	2	1	$\{0,1\}$
E_3	1	1	1	0	$\{0\}$
E_4	0	2	1	1	$\{1\}$
E_5	1	2	1	0	$\{1\}$
E_6	1	0	1	0	$\{0\}$
E_7	1	2	2	1	$\{1\}$
E_8	0	0	2	1	$\{0,1\}$

Table 2.2: Decision table derived from Table 2.1

Each indiscernibility class can be described by a boolean formula. For instance, consider E_1*. It can be described by*

$$(\mathit{Ast} = 0 \wedge \mathit{Age} = 0 \wedge \mathit{TearP} = 1 \wedge \mathit{Spec} = 0) \ .$$

Alternatively, we can simply use the tuple $\langle 0, 0, 1, 0 \rangle$ *to describe the same indiscernibility class. This is the approach followed in this thesis.*

□

As shown in the example 2.2, indiscernibility classes can be described by a unique tuple of $\prod_{a_i \in A} V_{a_i}$[1]. These elementary sets are like atomic

[1]The expression $\prod_{a_i \in A} V_{a_i}$ denotes the cartesian product $V_{a_1} \times \cdots \times V_{a_k}$, where $A = \{a_1, \cdots, a_k\}$.

information granules that are used to define concepts, i.e. subsets of the universe. The next example illustrates this point.

Example 2.3 *Consider the decision table 2.2. This decision table is associated with two concepts: those people who stopped using contact lenses and those who did not. The former concept is denoted by the subset*

$$WithoutLenses = \{o_1, o_2, o_4, o_5, o_6, o_{10}, o_{11}, o_{12}, o_{17}, o_{18}\} ,$$

while the latter is denoted by the subset

$$WithLenses = \{o_3, o_7, o_8, o_9, o_{13}, o_{14}, o_{15}, o_{16}\} .$$

An important question is

- *When can we say that a person could be recommended the use of contact lenses, based on the attributes* `Ast`*,* `Age`*,* `TearP`*, and* `Spec` *and their values shown on the decision table 2.2?*

To answer the above question we need to build a discriminating description of concept $WithLenses$. This description can be obtained in terms of the description of the indiscernibility classes. We remind the reader that each indiscernibility class can be described by a tuple of (condition) attribute values. For instance, $E_4 \subseteq WithLenses$. Hence, we can conclude that for people satisfying the conditions

$$(\mathit{Ast} = 0 \wedge \mathit{Age} = 2 \wedge \mathit{TearP} = 1 \wedge \mathit{Spec} = 1) ,$$

the use of contact lenses should be recommended. However, indiscernibility class E_2 seems to raise a problem in this context because E_2 is neither contained in $WithLenses$ nor it is disjoint from $WithLenses$. This indicates that concept $WithLenses$ cannot be defined precisely using the available information.

□

We can conclude from the example above that it is not always possible to learn precise descriptions of concepts. This is due to the fact that we may have contradictory knowledge that leads to vague concepts. It is in this context that the notion of rough set emerges naturally, since it introduces the idea of set (or concept) approximations.

2.2 Approximations and Rough Sets

Let (U, R_A) be an approximation space. The tuple describing each indiscernibility class $E \in R_A^*$ is denoted by $\overrightarrow{E^A}$. For example, in table 2.2, $\overrightarrow{E_1^A} = \langle 0, 0, 1, 0 \rangle$.

Definition 2.3 *Let $\mathcal{D} = (U, A, d)$ be a decision table and $X \subseteq U$. Rough set theory introduces two types of approximations of concept X in the approximation space (U, R_A).*

- Lower approximation *of X, denoted by $\underline{X}$,*

$$\underline{X} = \{\overrightarrow{E^A} \mid E \in R_A^* \text{ and } E \subseteq X\} .$$

- Upper approximation *of X, denoted by $\overline{X}$,*

$$\overline{X} = \{\overrightarrow{E^A} \mid E \in R_A^* \text{ and } E \cap X \neq \emptyset\} .$$

Let $\neg X = U \setminus X$, where U is the set of objects under consideration. The upper approximation $\overline{X}$ can informally be interpreted as a description of the objects that *possibly* belong to a given concept X. Notice that some doubt exists in this description because there may exist a tuple $t \in \overline{X} \cap \overline{\neg X}$. The lower approximation of X should informally be viewed as a description of those objects that *definitely* belong to the concept. The set $\underline{\overline{X}} = \overline{X} \cap \overline{\neg X}$ is called the *boundary* and it corresponds to the conflicting cases (i.e. the doubtful ones).

Example 2.4 *Consider once more the decision table 2.2 and the concepts*

$$WithLenses = \{o_3, o_7, o_8, o_9, o_{13}, o_{14}, o_{15}, o_{16}\} ,$$

$$\neg WithLenses = \{o_1, o_2, o_4, o_5, o_6, o_{10}, o_{11}, o_{12}, o_{17}, o_{18}\} .$$

Note that $\neg WithLenses$ represents the same concept as $WithoutLenses$, introduced in example 2.3. Their upper and lower approximations are given.

$$\begin{aligned}
\overline{WithLenses} &= \{\langle 1,0,2,1\rangle, \langle 0,2,1,1\rangle, \langle 1,2,1,0\rangle, \\
&\quad \langle 1,2,2,1\rangle, \langle 0,0,2,1\rangle\} , \\
\underline{WithLenses} &= \{\langle 0,2,1,1\rangle, \langle 1,2,1,0\rangle, \langle 1,2,2,1\rangle\} , \\
\overline{\neg WithLenses} &= \{\langle 0,0,1,0\rangle, \langle 1,0,2,1\rangle, \langle 1,1,1,0\rangle, \\
&\quad \langle 1,0,1,0\rangle, \langle 0,0,2,1\rangle\} , \\
\underline{\neg WithLenses} &= \{\langle 0,0,1,0\rangle, \langle 1,1,1,0\rangle, \langle 1,0,1,0\rangle\} .
\end{aligned}$$

The upper approximation of $WithLenses$ describes those persons who possibly will not have problems in using contact lenses while the lower approximation of $\neg WithLenses$ describes those person who certainly will have problems due to the use of contact lenses. Notice that the tuple $\langle 1,0,2,1\rangle$ belongs to both $\overline{WithLenses}$ and $\overline{\neg WithLenses}$, i.e. $\langle 1,0,2,1\rangle \in \underline{WithLenses}$. This fact indicates that for persons satisfying the condition

$$(\texttt{Ast} = 1 \wedge \texttt{Age} = 0 \wedge \texttt{TearP} = 2 \wedge \texttt{Spec} = 1)$$

there exists contradictory evidence and, therefore, it is not possible to state with certainty whether they should be recommended to use contact lenses.

□

Let (U, R_A) be an approximation space and $X, Y \subseteq U$. It has been proved that set approximations have several important properties [Paw91]. We list some of them.

(1) $\underline{X} \subseteq X \subseteq \overline{X}$.

(2) $\overline{(X \cup Y)} = \overline{X} \cup \overline{Y}$.

(3) $\overline{(X \cap Y)} \subseteq \overline{X} \cap \overline{Y}$.

(4) $\underline{(X \cup Y)} \supseteq \underline{X} \cup \underline{Y}$.

(5) $\underline{(X \cap Y)} = \underline{X} \cap \underline{Y}$.

(6) $\overline{\neg X} = \neg \underline{X}$.

(7) $\underline{\neg X} = \neg \overline{X}$.

(8) If $X \subseteq Y$ then $\underline{X} \subseteq \underline{Y}$ and $\overline{X} \subseteq \overline{Y}$.

A concept that cannot be defined precisely is represented by a pair of sets of tuples, to be called rough set.

Definition 2.4 *A* rough set *(or* rough relation*) S is a pair $(\overline{S}, \overline{\neg S})$ such that $\overline{S}, \overline{\neg S} \subseteq \prod_{a_i \in A} V_{a_i}$, for some non empty set of attributes A. The* rough complement *of a rough set $S = (\overline{S}, \overline{\neg S})$ is the rough set $\neg S = (\overline{\neg S}, \overline{S})$.*

Example 2.5 *Consider again example 2.4. Since concept $WithLenses$ cannot be defined precisely in terms of the elementary sets belonging to R^*_A, where $A = \{\texttt{Ast}, \texttt{Age}, \texttt{TearP}, \texttt{Spec}\}$, this concept is then represented by the rough set $Lenses = (\overline{WithLenses}, \overline{\neg WithLenses})$.*

□

We stress that there is not a unique way to define a rough set. Different definitions for the concept of rough set have been proposed in the literature [Pag97]. For instance, a rough set could be defined as the pair $S = (\underline{S}, \overline{S})$. All these definitions formalize the idea of vague sets due to the existence of a boundary region. The reason for preferring one definition over another is related to the concrete application the definition's author has in mind. We have chosen to define a rough set as $S = (\overline{S}, \overline{\neg S})$ rather than $S = (\underline{S}, \overline{S})$ because the former definition gives information about all negative examples while the latter only indicates those negative examples in the boundary region.

The notion of rough set used in our framework and the one usually presented in rough set literature differs in a number of respects. First in our framework, lower and upper approximations are sets of tuples while these approximations are usually defined as subsets of the universe U. Second, we assume that a rough set may partition the set of all possible tuples $\mathcal{W} = \prod_{a_i \in A} V_{a_i}$, where A is a set of condition attributes, into four regions. Figure 2.1 illustrates this point.

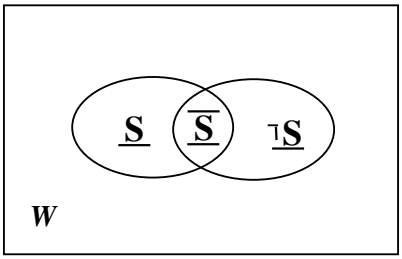


Figure 2.1: The four regions generated by a rough set S.

Given a rough set S, these regions correspond to $\underline{S}$, $\underline{\neg S}$, $\overline{\underline{S}}$, and the set of tuples about which there is no information, $\mathcal{W} \setminus (\underline{S} \cup \underline{\neg S} \cup \overline{\underline{S}})$. Most literature considers that a rough set divides the universe U in three regions: lower approximation of a rough set and its complement, and boundary.

Let (U, R_A) be an approximation space, $[x]_A$ be the equivalence class of object $x \in U$ in that approximation space, and S be a rough set. Then, $\varepsilon(x, S)$ denotes whether x belongs to the concept represented by rough set

S and it is defined as follows.

$$\varepsilon(x,S) = \begin{cases} \texttt{true} & \text{if } \overrightarrow{[x]_A} \in \underline{S} \\ \texttt{false} & \text{if } \overrightarrow{[x]_A} \in \underline{\neg S} \\ \top & \text{if } \overrightarrow{[x]_A} \in \overline{S} \\ \bot & \text{otherwise .} \end{cases}$$

$\varepsilon(x,S) = \top$ indicates the existence of contradictory information about object x and $\varepsilon(x,S) = \bot$ shows lack of information about x.

Let us assume, without loss of generality, that d is a binary decision attribute of a decision table $\mathcal{D} = (U, A, d)$. It is easy to see that we can associate a rough set $D = (\overline{D}, \overline{\neg D})$ with $\mathcal{D}$, where $\overline{D}$ is the set of tuples with positive outcome for the decision attribute d and $\overline{\neg D}$ is the set of tuples with negative outcome. Our definition of rough set cannot be seen as an alternative representation for a decision table. This can be explained by the fact that, for the former case, there is no information associated with each tuple of how many objects belong to the corresponding indiscernibility class or how many objects in an indiscernibility class have positive (negative) outcome for the decision attribute. From a formal point of view, this problem can be easily addressed and we will discuss it in section 2.4.

We have adopted the convention to give the same name to a decision table, to its decision attribute, and to the rough relation it defines, since all these concepts are very associated with each other. However, the appearance of the printed names is different for each case. For instance, if a decision table is called "$\mathcal{L}$enses", a name usually starting with calligraphic letter, then its decision attribute is called "`Lenses`" and it defines the rough relation "*Lenses*". We use interchangeably the expression "objects" and "individuals" to refer to the elements of the universe U under consideration.

We stress that, in this work, we only consider approximation spaces (U, R_A) where R_A is an equivalence relation. The literature also discusses its generalizations to tolerance approximation spaces [SS96] and similarity approximation spaces [SV97].

2.3 Decision Rules

In the context of supervised learning, an important task is the discovery of classification rules from the data provided in the decision tables. The decision rules not only capture patterns hidden in the data as they can also be used to classify new unseen objects.

Definition 2.5 *Let $\mathcal{D} = (U, A, d)$ be a decision table and $\{a_1, \ldots, a_n\} \subseteq A$. Assume also that $\{v_{d1}, \ldots, v_{dk}\} \subseteq V_d$ and $v_i \in V_{a_i}$, for $1 \leq k$ and $1 \leq i \leq n$. A* decision rule *is an expression of the form*

$$(a_1 = v_i) \wedge \ldots \wedge (a_n = v_n) \longrightarrow (d = v_{d1}) \vee \ldots \vee (d = v_{dk}) \; .$$

If $k = 1$ then the decision rule is called a determinstc rule. *Otherwise ($k > 1$), the decision rule is called* non-deterministic rule.

A decision rules can informally be understood as an implication. Symbols "$\wedge$", "$\vee$" can be read as "and" and "or", respectively.

Given a decision rule r, $cond(r)$ denotes the expression on the left hand side of symbol "$\rightarrow$" and $dec(r)$ denotes the expression on the right hand side.

Example 2.6 *Consider again table 2.2. Then, r_1 is a deterministic decision rule obtained from this table*

$$r_1 \equiv (\mathit{Age} = 0) \wedge (\mathit{TearP} = 2) \wedge (\mathit{Spec} = 1) \longrightarrow (\mathit{Lenses'} = \{0, 1\}) \quad .$$

It states that if a person's age is not more than 20 years, and his tear production is normal, and he does not use spectacles then the outcome for the decision attribute Lenses *is 0 or 1, i.e. the optician cannot with certainty recommend contact lenses.*

The above rule corresponds to the following non-deterministic rule for decision table 2.1.

$$(\mathit{Age} = 0) \wedge (\mathit{TearP} = 2) \wedge (\mathit{Spec} = 1) \longrightarrow (\mathit{Lenses} = 0) \vee (\mathit{Lenses} = 1)$$

Hence, this rule describes some of the objects belonging to the upper approximation of rough set Lenses.

Consider now the decision rule

$$r_2 \equiv (\mathit{Age} = 0) \wedge (\mathit{Spec} = 1) \longrightarrow (\mathit{Lenses'} = \{0, 1\}) \; .$$

Both decision rules r_1 and r_2 above can be used to identify some of the objects belonging to $\overline{Lenses}$. However, an important difference should be pointed. The latter rule is more general than the former in the sense that it states a smaller number of conditions in cond(r). Therefore, r_2 is more likely to be applied to a larger number of new objects to predict the outcome for the decision attribute.

□

Let us call *decision classes* to the partitions of the universe generated by the decision attribute, i.e. elements of $R^*_{\{d\}}$. The preceding example points that rules induced from a decision table with a minimal number of conditions (i.e. attribute-value pairs, $(a = v)$) are the most useful because those rules are more general. In the context of data mining, one of the main aims of rough-set based algorithms is to either find a set of minimal rules that covers a given decision class or to compute all minimal rules for the chosen decision class.

As shown in the previous sections, the concept associated with a given decision class may not be definable in terms of the elementary sets, i.e. the concept is rough. Decision rules can then be computed either with respect to the lower approximation or upper approximation of the target concept. An important difference between these two cases is that rules generated from the lower approximation are deterministic while rules generated from the upper approximation may be non-deterministic.

We turn now to the formalization of minimal decision rules.

Let $\mathcal{D} = (U, A, d)$ be a decision table, $c_1 \equiv (a_1 = v_1) \wedge \ldots \wedge (a_n = v_n)$ and $c_2 \equiv (a_1 = v_1) \vee \ldots \vee (a_n = v_n)$ be two conditions, with $\{a_1, \ldots, a_n\} \subseteq A$ and $1 \leq n$. $Cover(c_1)$ and $Cover(c_2)$ denote the following subsets of U.

$$\begin{array}{rcl} Cover(c_1) & = & \bigcap_{1 \leq i \leq n} \{o \in U \mid a_i(o) = v_i\} \ , \\ & \text{and} & \\ Cover(c_2) & = & \bigcup_{1 \leq i \leq n} \{o \in U \mid a_i(o) = v_i\} \ . \end{array}$$

A rule r *covers* all objects that match the condition on its left-hand side, denoted as $Cover(r)$, i.e. $Cover(r) = Cover(cond(r))$. This definition can be extended to a set of rules S:

$$Cover(S) = \bigcup_{r \in S} Cover(r) \ .$$

Let $c_1 \equiv (a_{11} = v_{11}) \wedge \ldots \wedge (a_{1n} = v_{1n})$ and $c_2 \equiv (a_{21} = v_{21}) \wedge \ldots \wedge (a_{2k} = v_{2k})$, with $n, k \geq 1$, be two conditions. We write $c_1 \preceq c_2$ to denote that every attribute-value pair $(a_{1i} = v_{1i})$ occurring in c_1, with $1 \leq i \leq n$, also occurs in c_2. The expression $c_1 \prec c_2$ means that $c_1 \preceq c_2$ and $c_1 \neq c_2$. For instance, $(\texttt{Age} = 2) \preceq (\texttt{Age} = 2) \wedge (\texttt{TearP} = 1)$.

Definition 2.6 ([Zia02b]) *Let r be a decision rule. A* value reduct *for r, denoted as $red(r)$, is a condition satisfying the following properties.*

(1) $red(r) \preceq cond(r)$.

(2) $Cover(red(r)) \subseteq Cover(dec(r))$. *i.e. value reduct preserves the inclusion relation of the set of objects covered by the rule in* $Cover(dec(r))$*;*

(3) *For every condition* $c \prec red(r)$, $Cover(c) \not\subseteq Cover(dec(r))$, *i.e. the value reduct is a minimal condition, with respect to the partial order* $\prec$*, satisfying properties* (1) *and* (2).

The definition of minimal rules is based on the notion of value reduct.

Definition 2.7 *Let* r *be a decision rule.* r *is a* minimal rule *if and only if* $red(r) = cond(r)$.

Given a decision rule r, a single value reduct can be computed in linear time with respect to the number of attributes that appear in $cond(r)$.

Example 2.7 *Consider example 2.6 and table 2.2. Decision rule*

$$(\texttt{Age} = 0) \wedge (\texttt{TearP} = 2) \wedge (\texttt{Spec} = 1) \longrightarrow (\texttt{Lenses'} = \{0, 1\})$$

is not minimal while

$$(\texttt{Age} = 0) \wedge (\texttt{Spec} = 1) \longrightarrow (\texttt{Lenses'} = \{0, 1\}) \quad \textit{and}$$
$$(\texttt{Age} = 0) \wedge (\texttt{TearP} = 2) \longrightarrow (\texttt{Lenses'} = \{0, 1\})$$

are minimal decision rules. Note that

$$Cover((\texttt{Age} = 0) \wedge (\texttt{Spec} = 1)) = E_2 \cup E_8 = Cover(\texttt{Lenses'} = \{0, 1\}) \ .$$

□

Let $\mathcal{D} = (U, A, d)$ be a decision table, $A = \{a_1, \ldots, a_n\}$, and assume that $v \in V_d$. An algorithm based on value reducts that computes a set of minimal rules covering the decision class $D_v = \{o \in U \mid d(o) = v\}$ can be easily devised. Assume that we are interested in a set of deterministic rules.

- **Computation of Minimal Rules** [Zia02b]:

(i) Compute $\underline{D_v}$. Let L be a set of decision rules initialized to the empty set.

(ii) Consider the decision rule

$$r \equiv (a_1 = v_1) \wedge \ldots \wedge (a_n = v_n) \rightarrow (d = v)$$

associated with each tuple $\langle v_1, \ldots, v_n \rangle \in \underline{D_v}$.

(ii.1) Compute a value reduct for rule r, i.e. red(r).

(ii.2) If the $Cover(red(r)) \not\subseteq \bigcup_{r_i \in L} Cover(r_i)$ then

$$L = L \cup \{red(r) \rightarrow (d = v)\} .$$

(iii) Output each rule in L.

If a set of non-deterministic rules is sought instead then $\underline{D_v}$ should be replaced by $\overline{D_v}$ in the algorithm above. The algorithm can be computed in polynomial time. Its time complexity is proportional to the number of condition attributes and number of tuples in $\underline{D_v}$ ($\overline{D_v}$).

Example 2.8 *Consider again table 2.2 and let us compute the minimal deterministic decision rules for* $\underline{Lenses'_{\{0,1\}}}$*, corresponding to the boundary region of table* $\mathcal{L}enses$*. We have that*

$$\underline{Lenses'_{\{0,1\}}} = \{\langle 1,0,2,1\rangle , \langle 0,0,2,1\rangle\} .$$

It can be easily verified that for the decision rules associated with tuples $\langle 1,0,2,1\rangle$ *and* $\langle 0,0,2,1\rangle$ *there are two possible value reducts*

$$\begin{array}{ll} (\mathtt{Age} = 0) \wedge (\mathtt{Spec} = 1) & \textit{and} \\ (\mathtt{Age} = 0) \wedge (\mathtt{TearP} = 2) & \end{array}$$

It can be also easily verified that $(\mathtt{Age} = 2)$ *is a value reduct for each of the decision rules obtained from the tuples belonging to the set*

$$\underline{Lenses'_{\{1\}}} = \{\langle 0,2,1,1\rangle , \langle 1,2,1,0\rangle , \langle 1,2,2,1\rangle\} \ .$$

Note that $\underline{Lenses'_{\{1\}}}$ *corresponds to the lower approximation of rough set Lenses.*

Since

$$\begin{array}{ll} Cover(\mathtt{Age} = 2) \not\subseteq Cover((\mathtt{Age} = 0) \wedge (\mathtt{Spec} = 1)) & \textit{and} \\ Cover((\mathtt{Age} = 0) \wedge (\mathtt{Spec} = 1)) \not\subseteq Cover(\mathtt{Age} = 2) \ , & \end{array}$$

the following two minimal decision rules describe the set of objects $Lenses'_{\{0,1\}} \cup Lenses'_{\{1\}}$*.*

$$\begin{array}{rcl} (\mathtt{Age} = 0) \wedge (\mathtt{Spec} = 1) & \longrightarrow & (\mathtt{Lenses'} = \{0,1\}) , \\ (\mathtt{Age} = 2) & \longrightarrow & (\mathtt{Lenses'} = \{1\}) . \end{array}$$

If we consider table 2.1 instead then, the deterministic decision rules above correspond to a non-deterministic rule and to a deterministic rule, respectively.

$$\begin{aligned} (\mathit{Age} = 0) \wedge (\mathit{Spec} = 1) &\longrightarrow (\mathit{Lenses} = 0) \vee (\mathit{Lenses} = 1)\ , \\ (\mathit{Age} = 2) &\longrightarrow (\mathit{Lenses} = 1)\ . \end{aligned}$$

These two rules cover all objects described by tuples in the upper approximation of rough set Lenses (obtained from table 2.1).

□

In step (iii) above, L is a non-empty set of minimal rules such that $Cover(L) = \underline{D_v}$. Hence, these set of rules form a discriminating description of the approximated concept. However, from the point of view of knowledge discovery, it is more interesting to find all possible minimal rules than just one set of minimal rules. In contrast to the above algorithm that is polynomial, computing all possible minimal rules is NP-hard.

All minimal rules can be computed by first creating the decision-relative discernibility matrix. A Boolean expression can then be constructed from this matrix. This expression is simplified (absorption law of Boolean algebra can be applied) and prime implicants of the simplified expression are computed. Each prime implicant can finally be translated to a decision rule. More detailed descriptions of this algorithm can be obtained from [KPPS99, SP97]. A survey of the main algorithms for inducing decision rules using rough set theory is presented in [Baz98, Ste98].

2.4 Numerical Measures

The first part of this section is devoted to the discussion of several numerical measures that can be associated with a decision rule for measuring its quality. In the second part, we introduce the notion of reduct and we briefly mention an algorithm to compute decision rules based on reducts.

2.4.1 Measuring Quality of Decision Rules

Quality measures associated with decision rules can be used to eliminate some of the decision rules. We list below some of these quality measures [KPPS99].

Given a set S, the expression $|S|$ denotes the number of elements in S.

Definition 2.8 (Support) *Let r be a decision rule induced from a decision table $\mathcal{D} = (U, A, d)$. The* support *of r, denoted as $Supp(r)$, is defined as*

$$Supp(r) = |Cover(cond(r)) \cap Cover(dec(r))| \ .$$

The support of a rule represents the number of objects of the universe that match both conditions $cond(r)$ and $dec(r)$.

Definition 2.9 (Strength) *Let r be a decision rule induced from a decision table $\mathcal{D} = (U, A, d)$. The* strength *of r, denoted as $Strength(r)$, is defined as*

$$Strength(r) = \frac{|Supp(r)|}{|U|} \ .$$

The strength of a rule indicates the proportion of objects in the universe that match both $cond(r)$ and $dec(r)$; i.e. the percentage of objects for which the pattern expressed by the rule is true. Hence, $Strength(r)$ can be seen as an estimate of the probability $Pr(cond(r) \wedge dec(r))$.

Definition 2.10 (Accuracy) *Let r be a decision rule induced from a decision table $\mathcal{D} = (U, A, d)$. The* accuracy *of r, denoted as $Acc(r)$, is defined as*

$$Acc(r) = \frac{Supp(r)}{|Cover(cond(r))|} \ .$$

The accuracy of a rule corresponds to the conditional probability $Pr(o \in Cover(dec(r)) \mid o \in Cover(cond(r)))$. By other words, $Acc(r)$ expresses how trustworthy is the rule is drawing the conclusion $dec(r)$ for an object matching the condition on the left-hand side of the rule.

Definition 2.11 (Coverage) *Let r be a decision rule induced from a decision table $\mathcal{D} = (U, A, d)$. The* coverage *of r, denoted as $Cov(r)$, is defined as*

$$Cov(r) = \frac{Supp(r)}{|Cover(dec(r))|} \ .$$

The coverage of a rule corresponds to the conditional probability $Pr(o \in Cover(cond(r)) \mid o \in Cover(dec(r)))$. Hence, $Cov(r)$ quantifies how well the rule left-hand side, $cond(r)$, describes the set of objects covered by its right-hand side, $dec(r)$.

2.4.2 Reducts

The derived decision table shown in example 2.2 contains nearly the same information as in table 2.1. However in the derived table, if `Lenses'`$(E) = \{0, 1\}$, for some indiscernibility class E, then we cannot know how many objects in E have outcome 0 for decision attribute `Lenses` and how many have outcome 1. To overcome this problem, we define a family of functions. Let $\mathcal{D} = (U, A, d)$ be a decision table and $V_d = \{v_1, \ldots, v_n\}$. We have then that, for each indiscernibility class $E \in R^*_A$,

$$\lambda^i_{\mathcal{D}}(\overrightarrow{E^A}) = |\{o \in E \mid d(o) = v_i\}| \ ,$$

with $1 \leq i \leq n$. Moreover, $card(\overrightarrow{E^A}) = |E| = \sum_{1 \leq i \leq n} \lambda^i_{\mathcal{D}}(\overrightarrow{E^A})$, i.e. $card(\overrightarrow{E^A})$ denotes the number of objects in the indiscernibility class described by tuple $\overrightarrow{E^A}$. Function *card* can be extended to a set of tuples T.

$$card(T) = \sum_{t \in T} card(t) \ .$$

Let (U, R_{B_1}) and (U, R_{B_2}) be two approximation spaces. We now define those objects of the universe for which knowing the values of attributes B_1 is sufficient for determining the values of attributes B_2, denoted as $Pos_{B_1}(B_2)$.

$$Pos_{B_1}(B_2) = \bigcup_{X \in R^*_{B_2}} \underline{X} \ ,$$

where $\underline{X}$ is the lower approximation of X in the approximation space (U, R_{B_1}).

An interesting numerical measure associated with a decision table is the degree of functional dependency [KPPS99, Zia02b] between two subsets of attributes of the table.

Definition 2.12 *Let $\mathcal{D} = (U, A, d)$ be a decision table and $B_1, B_2 \subseteq A \cup \{d\}$. The* degree of functional dependency *in the relationship between attribute sets B_1 and B_2, denoted as $\kappa(B_1, B_2)$, is defined as*

$$\kappa(B_1, B_2) = \frac{card(Pos_{B_1}(B_2))}{|U|} \ .$$

Function $\kappa(B_1, B_2)$ can be understood as the proportion of objects of the universe U for which knowing the values of attributes B_1 is enough to determine the values of attributes B_2. Obviously, $0 \leq \kappa(B_1, B_2) \leq 1$. If

$\kappa(B_1, B_2) = 1$ then functional dependency $B_1 \rightarrow B_2$ exists in the table and it can be easily shown that $R_{B_1} \subseteq R_{B_2}$ (i.e. the partition generated by attributes B_1 is finer than the partition generated by B_2). If $\kappa(B_1, B_2) = 0$ then no values of attributes B_2 can be determined by values of attributes B_1. If $0 < \kappa(B_1, B_2) < 1$ then the values of attributes B_2 can be determined by values of attributes B_1, only for some objects (but not all).

Example 2.9 *Consider decision tables of example 2.1 and 2.2. For the first table,* $\kappa(\{\texttt{Ast}, \texttt{Age}, \texttt{TearP}, \texttt{Spec}\}, \{\texttt{Lenses}\}) \simeq 0.72$ *indicating that the decision attribute is not functionally determined by the condition attributes. For the second table* $\kappa(\{\texttt{Ast}, \texttt{Age}, \texttt{TearP}, \texttt{Spec}\}, \{\texttt{Lenses'}\}) = 1$. *Hence, we can conclude that the functional dependency*

$$\{\texttt{Ast}, \texttt{Age}, \texttt{TearP}, \texttt{Spec}\} \rightarrow \{\texttt{Lenses'}\}$$

holds for this table.

□

If we consider decision table 2.2, we can conclude that

$$\kappa(\{\texttt{Age}, \texttt{TearP}\}, \{\texttt{Lenses'}\}) = 1 \ .$$

Hence, we only need condition attributes `Age` and `TearP` in order to be able to determine the decision class of an object, i.e. to determine $\texttt{Lenses'}(E_i)$ $(1 \leq i \leq 8)$. This makes possible savings in the amount of information that needs to be represented and it may also lead to a table where regularities are more easy to find. Relative reducts formalize this idea.

Definition 2.13 *Let A and B be two sets of attributes. A* relative reduct *of A with respect to $\kappa(A, B)$, denoted as $red(A, B)$, is a subset of A having the following properties.*

(1) $\kappa(A, B) = \kappa(red(A, B), B)$.

(2) *For all $a \in red(A, B)$,*

$$\kappa(red(A, B) \setminus \{a\}, B) \neq \kappa(A, B) \ ,$$

i.e. $red(A, B)$ is a minimal subset of A satisfying (1).

From a practical point of view, we are often interested in discovering relative reducts with respect to $\kappa(A, \{d\})$, for a decision table $\mathcal{D} = (U, A, d)$.

A single relative reduct can be computed in linear time. However, computing all relative reducts or a minimal reduct is NP-hard. Most of the algorithms for determining all reducts are based on a decision-relative discernibility matrix [KPPS99, SP97].

In section 2.3, we have presented an algorithm to compute a set S of (minimal) decision rules forming a discriminating description for a given decision class D_v. This algorithm is based on a covering approach since $\overline{D_v} = Cover(S)$ or $\underline{D_v} = Cover(S)$. A set of non-deterministic rules is obtained in the former case, while only deterministic rules are computed for the latter. However, rules can be computed in a different way: find first relative reducts and then create decision rules by overlaying the reducts over objects in the table. We illustrate this second approach with the next example.

Example 2.10 *Consider decision table 2.2. A relative reduct of* $\{\texttt{Ast}, \texttt{Age}, \texttt{TearP}, \texttt{Spec}\}$*, with respect to* $\kappa(\{\texttt{Ast}, \texttt{Age}, \texttt{TearP}, \texttt{Spec}\}, \texttt{Lenses'})$*, is* $\{\texttt{Age}, \texttt{TearP}\}$*. The following decision rules would then be obtained*

$$\begin{array}{lcl}
(\texttt{Age} = 0) \wedge (\texttt{TearP} = 1) & \longrightarrow & (\texttt{Lenses} = 0)\ , \\
(\texttt{Age} = 0) \wedge (\texttt{TearP} = 2) & \longrightarrow & (\texttt{Lenses} = 0) \vee (\texttt{Lenses} = 1)\ , \\
(\texttt{Age} = 1) \wedge (\texttt{TearP} = 1) & \longrightarrow & (\texttt{Lenses} = 0)\ , \\
(\texttt{Age} = 2) \wedge (\texttt{TearP} = 1) & \longrightarrow & (\texttt{Lenses} = 1)\ , \\
(\texttt{Age} = 2) \wedge (\texttt{TearP} = 2) & \longrightarrow & (\texttt{Lenses} = 1)\ .
\end{array}$$

Notice that the above decision rules are not minimal. For instance, $(\texttt{Age} = 2)$ *is a value reduct of the last two rules above.*

□

It is also possible to use approximations of a relative reduct $red(A, B)$, with respect to $\kappa(A, B)$. These approximations are subsets of A that "almost" preserve the same capability as the attribute set A in determining the values of attributes B. An advantage of using approximations of reducts is that decision rules synthesized from them are less sensitive to noise in the data and, therefore, the quality of classification of new objects tends to increase. We briefly mention two approaches: dynamic reducts [Baz96] and s-reducts [SP97].

Let $\mathcal{D} = (U, A, d)$ be a decison table and $red(A, B)$ be a relative reduct, with respect to $\kappa(A, B)$, where $B \subseteq A \cup \{d\}$. A *dynamic reduct* is a subset of A appearing "sufficiently often" as a relative reduct in random sample

subtables obtained from $\mathcal{D}$. Notice that "sufficiently often" should be understood as a parameter that is tuned according to the data in the table.

We can measure how the dropping of a number of attributes from attribute set A_1 changes the coefficient $\kappa(A_1, B)$. Let $A_2 \subseteq A_1$.

$$\alpha_{(A_1,B)}(A_2) = 1 - \frac{\kappa(A_2, B)}{\kappa(A_1, B)} \ .$$

It is worth noting that if A_2 is the minimal subset of A_1 such that $\kappa(A_1, B) = \kappa(A_2, B)$ (i.e. A_2 is a relative reduct of A_1 with respect to $\kappa(A_1, B)$) then $\alpha_{(A_1,B)}(A_2) = 0$. The attribute set A_2 is an *s-reduct* if $\alpha_{(A_1,B)}(A_2)$ is not larger than a given threshold called error level. This error level should be tuned for the table being considered.

2.5 Prediction

In sections 2.3 and 2.4.2, we have introduced decision rules and gave an idea about how they can be computed. These decision rules can then be used to make predictions for unseen objects. For instance, the decision rules obtained from one relative reduct form a classifier. Classifiers with better prediction capabilities are usually obtained by combining rules obtained from several reducts.

Classifiers may be non-deterministic because

- the new object matches non-deterministic rules, or
- the object matches several (deterministic) rules that lead to different decisions.

An obvious question is how to solve the problem of conflicting decisions. This issue can be addressed by *voting* [KPPS99]. We describe below this strategy.

Without loss generality, we assume that a classifier only contains deterministic decision rules. A non-deterministic rule $(a_1 = v_1) \wedge \ldots \wedge (a_n = v_n) \rightarrow (d = v_d) \vee (d = v_{d'})$ can always be replaced by two deterministic rules, $(a_1 = v_1) \wedge \ldots \wedge (a_n = v_n) \rightarrow (d = v_d)$ and $(a_1 = v_1) \wedge \ldots \wedge (a_n = v_n) \rightarrow (d = v_{d'})$.

Let $\mathcal{C}$ be a classifier, i.e. a set of (deterministic) decision rules. Moreover, $Rul_{\mathcal{C}}(o)$ denotes the set of decision rules of $\mathcal{C}$ that match object o and it is formally defined as

$$Rul_{\mathcal{C}}(o) = \{r \in \mathcal{C} \mid o \in Cover(r)\} \ .$$

- **Voting Algorithm:**

(1) If $Rul_{\mathcal{C}}(o) = \emptyset$ then classification is not possible. Otherwise, proceed with step (2).

(2) For each possible decision $v \in V_d$ compute the number of votes casted by each rule.

$$votes(v) = \sum_{r \in Rul_{\mathcal{C}}(o)} \nu_v(r) \ ,$$

where

$$\nu_v(r) = \begin{cases} 0 & \text{if } dec(r) \neq v \\ Supp(r) & \text{otherwise} \end{cases}$$

(3) Compute $\delta = \sum_{v' \in V_d} votes(v')$, i.e. the total number of casted votes.

(4) For each possible decision $v \in V_d$ compute the certainty factor

$$certainty(o, v) = \frac{votes(v)}{\delta} \ .$$

(5) Output the decision with the largest certainty.

The voting algorithm described above can be modified in a number of ways. For instance, the number of votes casted by each rule fired can be based on some other measured instead of rules's support.

2.6 The Variable Precision Rough Set Model

In section 2.2, we introduced the notions of concept approximations, lower and upper approximations. These ideas have been further generalized by W. Ziarko, see [Zia93], who introduced the *variable precision rough set model* (VPRSM).

We start by discussing informally the VPRSM. Consider two parameters l and u, called *precision control parameters*, such that $0 \leq l < u \leq 1$. Generalization of lower (upper) approximation and boundary region of a rough relation can be obtained as follows. The lower approximation of a concept X ($\neg X$) is obtained from those indiscernibility classes E such that its degree of overlapping with the set X ($\neg X$) is larger or equal than u ($1-l$). Those indiscernibility classes E such that their degree of overlapping with X is between l and u remain in the boundary region. This technique can also be seen as a way to "thin" the boundary region and it has the advantage of

making concept approximations less sensitive to possible noise contained in the data.

To formalize this idea, we need to introduce a function assigning to each indiscernibility class E a measure of the degree of overlapping of set X with E. This function corresponds to the conditional probability

$$Pr(o \in X \mid o \in E) = \frac{|(X \cap E)|}{|E|} \ .$$

Let (U, R_A) be an approximation space. Concept approximations can then be defined as

$$\begin{aligned} \underline{X} &= \{\overrightarrow{E^A} \mid E \in R_A^* \text{ and } Pr(o \in X \mid o \in E) \geq u\} \ , \\ \underline{\neg X} &= \{\overrightarrow{E^A} \mid E \in R_A^* \text{ and } Pr(o \in X \mid o \in E) \geq (1-l)\} \ , \\ \overline{X} &= \{\overrightarrow{E^A} \mid E \in R_A^* \text{ and } u < Pr(o \in X \mid o \in E) < l\} \ . \end{aligned}$$

It is worth to note that if $u = 1$ and $l = 0$ then the above definitions of lower (upper) approximations and boundary are equivalent to the ones presented in section 2.2.

Let $X \subseteq U$ and $Pr(X) = \frac{|X|}{|U|}$. To obtain some gain in the predictive capability, it is required that $u > Pr(X)$ and $l < Pr(X)$. Requiring that $u > Pr(X)$ ($l < Pr(X)$) will enable us to predict that an object $o \in X$ ($o \in \neg X$) more accurately than random guess.

Chapter 3

Logic Programming Framework

This chapter surveys the logic programming concepts needed in the sequel and it is self-contained. It gives to the reader the essential notions to understand the compilation technique discussed in chapters 4 and 6.

We start by introducing definite logic programs, in section 3.1, and then present in section 3.2 a more general class of logic programs, called extended logic programs. We also discuss the declarative semantics of extended logic programs (section 3.2.1) and queries (section 3.2.2).

3.1 The Main Idea: Definite Logic Programs

Logic programming [Llo87, NM95, Bar03, BL04] is a computational formalism that uses logic (e.g. first-order logic) to express knowledge and inference to manipulate the knowledge in order to be able to extract new knowledge.

In this work, the syntax of a logic program makes use of three disjoint alphabets: an alphabet of variable symbols Var, an alphabet of constant symbols $Const$, and an alphabet of predicate symbols $Pred$. Moreover, the set of symbols $\{\neg, not\} \nsubseteq Pred$. A *term* t is either a constant symbol or a variable, i.e $t \in Var \cup Const$. To distinguish between constants and variable symbols, we follow the usual convention: variables start with upper case letter (e.g. $X, Dist \in Var$), while using names beginning with lower case letters for constants (e.g. $small, c \in Const$). An *atom* is an expression of the form $p(t_1, \ldots, t_n)$, where p is an n-ary predicate symbol ($p \in Pred$)

and each $t_1, \ldots, t_n$ is a term. We write p/n, with $n \geq 0$, to express that p is an n-ary predicate symbol. An atom with zero arguments is simply written as p .

Intuitively, predicates denote n-ary relations and atoms can be seen as statements saying that an n-ary tuple belongs to an n-ary relation. For instance, the atom `office`(`xana`, `spetsen`, `7`) expresses that `xana`'s office is on `7`th floor of building `spetsen`, i.e. the tuple $\langle$`xana`, `spetsen`, `7`$\rangle$ belongs to the relation denoted by predicate `office`.

In the logic programming framework, knowledge is represented through clauses.

Definition 3.1 *A* definite clause *is an expression of the form*

$$H \texttt{ :- } A_1, \ldots, A_n. \ ,$$

where H and each A_i ($0 \leq i \leq n$) is an atom.

The left side of a (definite) clause (with respect to `:-`) is called the *head* and the right side is designated as *body* of the clause. A *fact* is a clause with empty body (i.e. $n = 0$), succinctly represented by $H.$. When no confusion arises, we will refer to "clauses" instead of "definite clauses".

Clauses can informally be understood as implications: if every atom in the body is true then the head must also be true. Therefore, the comma symbol "," is interpreted as conjunction.

Example 3.1 *We give an example of a definite clause and a fact.*

(1) `fly(tom) :- bird(tom).` *"If tom is a bird then tom flies.*
(2) `bird(tom).` *"Tom is a bird."*

□

The order by which atoms appear in the body of a clause is irrelevant. Thus, both clauses `fly(tom) :- bird(tom), healthy(tom).` and `fly(tom) :- healthy(tom), bird(tom).` have the same meaning.

If no variables occur in the atoms of a clause (atom), then the clause (atom) is *ground*. For instance, clause (1) of the example above is ground.

Definition 3.2 *Let $X_1, \ldots, X_n$ be variables occurring in some atom $q(t_1, \ldots, t_m)$, with $1 \leq n \leq m$. A* grounding substitution *θ is a set of bindings*

$$\theta = \{X_1/c_1, \ldots, X_n/c_n\}$$

(including the empty set) of variables X_i ($1 \leq i \leq n$) to constants $c_i \in Const$.

A substitution $\theta = \{X_1/c_1, \ldots, X_n/c_n\}$ can be applied to a clause C (atom A), written as $C\theta$ ($A\theta$), and it represents the clause (atom) obtained from C (A) by substituting each variable X_i for c_i ($1 \leq i \leq n$). For instance, if $\theta = \{X_1/c_1, X_2/c_2\}$ then $p(X_1, X_2)\theta$ is the atom $p(c_1, c_2)$. We can also say that variable X_1 (X_2) is *instantiated* with constant c_1 (c_2).

A *ground instance of a clause* C is obtained by applying a grounding substitution θ to clause C and $C\theta$ is ground. A non-ground clause stands for all its ground instances. Therefore, variables are implicitly universally quantified.

Example 3.2 *We give an example of a non-ground definite clause.*

(3) `fly(X) :- bird(X).` *"All birds fly".*

More formally, clause (3) *represents the following implication*

$$\forall X(\mathit{bird}(X) \Rightarrow \mathit{flies}(X)) .$$

Clause (1)*, in example 3.1, can be obtained by applying substitution* $\{X/tom\}$ *to clause* (3)*.*

□

Definition 3.3 *A* definite logic program *is a set of definite clauses.*

Given a definite logic program $\mathcal{P}$, $ground(\mathcal{P})$ represents the set of all ground instances of any clause $C \in \mathcal{P}$. This notation will also be used for sequence of atoms $A_1, \ldots, A_n$ ($n \geq 1$), i.e. $ground(A_1, \ldots, A_n)$.

Example 3.3 *Consider the following definite logic program.*

```
P  =  {fly(X) :- bird(X). ,
       bird(piu). ,
       bird(tom).}.
```

Note that for this logic program, $Const = \{\mathit{piu}, \mathit{tom}\}$*. The non-ground clause* `fly(X) :- bird(X).` *stands for the two ground clauses below, corresponding to its ground instances.*

```
fly(piu) :- bird(piu).          fly(tom) :- bird(tom).
```

The atom `fly(piu)` *is a ground atom, whereas* `fly(X)` *is not. We can also say that the atom* `fly(piu)` *is a ground instance of* `fly(X)` *(i.e.* $\mathit{fly(piu)} \in ground(\mathit{fly(X)})$*).*

□

An interesting aspect of the logic programming framework is the possibility to query the knowledge encoded in logic programs. By querying a logic program one may retrieve interesting non-trivial knowledge. For instance, given the program of example 3.3, we may ask whether `tom` flies or which birds fly. In the former case, we expect to obtain a positive answer. In the latter case, we should get as answer `tom` and `piu`. This point is particularly relevant for the operational semantic (i.e. implementation) of the query languages proposed in later chapters.

We discuss in more detail queries in next section. However, we first introduce a more general class of logic programs that includes definite logic programs.

3.2 Extended Logic Programs

Extended Logic Programming (ELP) is the target language of the transformations discussed in chapter 4, and we resort only to the disjunctive free fragment of the languages described in [Pea93, SI95], generalizing Answer Set Semantics [GL90] to the paraconsistent case.

In contrast to ELP, it is not possible to represent negative information in a definite logic program. The main distinctive feature of ELP is that it allows to express two forms of negation, explicit and default, allowing both open-world and closed-world reasoning. Explicit negation describes negative evidence, e.g. "*Tom does not fly.*", while default negation allows reasoning with lack of information, e.g. "*There is no evidence that tom flies.*".

The language in which extended logic programs are expressed is also based in an alphabet of variable, constant, and predicate symbols, i.e. $Var \cup Const \cup Pred$. Let At denote the set of all atoms built with alphabet symbols. An *objective literal* L is either an atom $A \in At$ or its explicit negation $\neg A$. The set of all objective literals is $OLit = At \cup \neg At$, where $\neg At = \{\neg A : A \in At\}$. The default negation of a literal L is represented by $not\ L$, also called default negated literal. A *literal* is either an objective literal L or its default negation $not\ L$, and the set of all literals is

$$Lit = OLit \cup not\ OLit = \{A, \neg A, not\ A, not\ \neg A : A \in At\} .$$

Intuitively, an objective literal represents a (positive or negative) evidence, while the default negated literal represents lack of (respectively, positive or negative) evidence. This makes it possible, for example, to represent differently the information that a flight departed without delay obtained from the flight control, from lack of the delay announcement.

Similar to definite logic programs, knowledge is encoded as sets of clauses in ELP. However, clauses of an extended logic program may include explicit and default negation.

Definition 3.4 *A* clause *is an expression*

$$L_0 \texttt{ :- } L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n.$$

where each L_i *is an objective literal and* $0 \leq m \leq n$.

An *integrity constraint* has the form

$$\texttt{:- } L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n. \ ,$$

with $n \geq m \geq 1$ and it can be seen as a clause with the head being the atom *false* (or $\bot$) representing falsehood. For instance, the integrity constraint

$$\texttt{:- human}(X), \texttt{male}(X), \texttt{female}(X).$$

expresses that no human can be male and female, simultaneously.

Definition 3.5 *An* extended logic program *is a finite set of clauses and integrity constraints.*

The notions of ground atom previously introduced can be easily extended to literals and sequences of literals. Moreover, the definitions of ground program and substitution presented for definite logic programs apply also to extended logic programs.

Example 3.4 *Assume that the following clauses belong to the extended logic program* $\mathcal{P}$.

"Someone is guilty if he is guilty"

(1) `guilty(X) :- guilty(X).`

"Someone is innocent if we cannot prove he is guilty."

(2) `innocent(X) :- not guilty(X).`

"Someone is not guilty if he is innocent."

(3) ¬`guilty(X) :- innocent(X).`

"A person is a male if we cannot prove he is a female."

(4) `male(X) :- person(X), not female(X).`

"A person is a female if we cannot prove she is a male."

(5) `female(X) :- person(X), not male(X).`

"A person cannot be guilty and non-guilty, simultaneously."

(6) `:- person(X), guilty(X), ¬guilty(X).`

"Tommy is a person."

(7) `person(tommy)`

Clauses (4) *and* (5) *together express the idea that a person must be either a male or a female.*

3.2.1 Declarative Semantics of Extended Logic Programs

The declarative semantics of a program captures its meaning. The declarative semantics of (extended) logic programs is based on the notion of interpretation. An interpretation is simply a subset of the ground objective literals, also known as the *extended Herbrand base*.

Definition 3.6 *An* interpretation $\mathcal{I}$ *of an extended logic program* $\mathcal{P}$ *is any subset of* $ground(OLit) = ground(At) \cup \neg ground(At)$.

As usual, an interpretation settles the set of *true* literals. If $L \in \mathcal{I}$ then the objective literal L has the truth value *true*, and if $L \notin \mathcal{I}$ then the objective literal L is *false*. Clearly, if an objective literal L is *false* then *not* L is *true*.

An interpretation induces the following consequence relation:

$$\begin{array}{lll} \mathcal{I} \models L & \text{if an only if} & L \in \mathcal{I}\ , \\ \mathcal{I} \models \text{not } L & \text{if and only if} & L \notin \mathcal{I}\ , \\ \mathcal{I} \models L_1, \ldots, L_n & \text{if and only if} & \mathcal{I} \models L_1 \text{ and } \ldots \text{ and } \mathcal{I} \models L_n\ , \end{array}$$

where L is an arbitrary ground objective literal and each L_i $(1 \leq i \leq n)$ is an arbitrary ground literal.

Example 3.5 *Consider the extended logic program* $\mathcal{P}$ *of example 3.4. A possible interpretation for* $\mathcal{P}$ *is*

$\mathcal{I} = \{innocent(tommy), \neg guilty(tommy), male(tommy), person(tommy)\}$.

In this interpretation the literals `guilty(tommy)`, `female(tommy)`, *not* `male(tommy)` *are false, i.e.* $\mathcal{I} \not\models$ `guilty(tommy)` , $\mathcal{I} \not\models$ `female(tommy)` , $\mathcal{I} \not\models$ *not* `male(tommy)` . *Obviously,*

$$\mathcal{I} \models \neg \texttt{guilty(tommy)}, \textit{not } \texttt{female(tommy)} \ .$$

□

An interpretation $\mathcal{I}$ *satisfies* a program clause if the corresponding implication holds in $\mathcal{I}$, and *satisfies* an integrity constraint if at least one literal in its body is *false*. Next definition formalizes this idea.

Definition 3.7 *A* model $\mathcal{M}_\mathcal{P}$ *of an extended logic program* $\mathcal{P}$ *is any interpretation that satisfies every clause and integrity constraint of* $ground(\mathcal{P})$, *i.e.* $(0 \leq m \leq n)$

1. *For every* $L_0 \texttt{ :- } L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n \in ground(\mathcal{P})$, *if* $\mathcal{I} \models L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n$ *then* $\mathcal{I} \models L_0$.
2. *For every* $\texttt{ :- } L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n \in ground(\mathcal{P})$, *then* $\mathcal{I} \not\models L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n$.

For instance, the interpretation $\mathcal{I}$ in example 3.5 is also a model of the program. Intuitively, an integrity constraint discards all model candidates that make every literal in its body *true*.

An extended logic program may have zero, one, or more models. Moreover, set inclusion is a partial order for the set of models of an extended logic program. Since we want to consider only the models such that each objective literal can be justified by some evidence in the program, only (some of) the minimal models are of interest. To capture formally this intuition, we need to recall an important property of definite logic programs (theorem 3.1) and introduce the notion of reduct [1] of an extended logic program [GL88].

Note that an extended logic program may have several minimal models while a definite logic program has always a least model (unique minimal model).

Theorem 3.1 ([Llo87]) *Let* $\mathcal{P}$ *be a definite logic program. Then,* $\mathcal{P}$ *has a least model.*

[1] Reduct of a logic program and reduct of a decision table (commonly used in the rough set framework) are two independent notions.

Definition 3.8 ([GL88]) *Let $\mathcal{P}$ be an extended logic program and $\mathcal{I}$ an interpretation. The reduct of $\mathcal{P}$ with respect to $\mathcal{I}$ is the definite logic program $\psi_{\mathcal{I}}(\mathcal{P})$ such that L_0 :- $L_1, \ldots, L_m. \in \psi_{\mathcal{I}}(\mathcal{P})$ if and only if there is a program clause of the form L_0 :- $L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n. \in ground(\mathcal{P})$ such that $\mathcal{I} \models not\ L_{m+1}, \ldots, not\ L_n$, where $0 \leq n \leq m$.*

It should be stressed that $\psi_{\mathcal{I}}(\mathcal{P})$ is always a definite logic program. Hence by theorem 3.1, it must have a least model.

Definition 3.9 *Let $\mathcal{P}$ be an extended logic program. An interpretation $\mathcal{I}$ is a* paraconsistent stable model *of $\mathcal{P}$ if and only $\mathcal{I}$ is the least model of $\psi_{\mathcal{I}}(\mathcal{P})$ and $\mathcal{I}$ satisfies all integrity constraints of $\mathcal{P}$.*

The semantics of extended logic programs is captured by those minimal models that are also paraconsistent stable models. The semantics is paraconsistent because a piece of information and its explicit negation can simultaneously hold. Note that an extended logic program may have no paraconsistent stable models, although it can have several minimal models. Intuitively, these programs are meaningless. The paraconsistent stable model semantics coincides with the stable model semantics [GL88, IS98, ALP$^+$00] whenever explicit negated literals do not occur in the program.

Example 3.6 *Consider once more the extended logic program $\mathcal{P}$ of example 3.4 and assume that $\mathcal{P}_1 = \mathcal{P} \cup \{$:- female(X). $\}$. Although,*

$$\begin{aligned} \mathcal{I}_1 \quad = \quad & \{innocent(tommy), \neg guilty(tommy), \\ & female(tommy), person(tommy)\} \end{aligned}$$

is a model of $\mathcal{P}$, it cannot be a model of $\mathcal{P}_1$ because the new integrity constraint rejects those interpretations containing any ground instance of literal `female(X).`

The following interpretations

$$\begin{aligned} \mathcal{M}_1 \quad = \quad & \{guilty(tommy), male(tommy), person(tommy)\} \quad and \\ \mathcal{M}_2 \quad = \quad & \{innocent(tommy), \neg guilty(tommy), \\ & male(tommy), person(tommy)\} \end{aligned}$$

are minimal models of $\mathcal{P}_1$. However, only $\mathcal{M}_2$ is a paraconsistent stable model, whereas $\mathcal{M}_1$ is not.

Intuitively, the reason for $\mathcal{M}_1$ not being a paraconsistent stable model is that `guilty(tommy)` *is a justification to itself (see clause* (1) *of example 3.4). However, we can justify having* `innocent(tommy)` *in $\mathcal{M}_2$ because there*

is no evidence that `guilty(tommy)` *is true (i.e. not* `guilty(tommy)` *is true) and then, by clause* (2)*, we must have that* `innocent(tommy)` *is true.*

Formally, taking definition 3.9, the reduct of $\mathcal{P}_1$ *with respect to* $\mathcal{M}_1$ *is the definite logic program* $\varphi_{\mathcal{M}_1}(\mathcal{P}_1)$ *consisting of the following clauses.*

```
guilty(tommy) :- guilty(tommy).
¬guilty(tommy) :- innocent(tommy).
male(tommy) :- person(tommy).
person(tommy).
```

The least model of $\varphi_{\mathcal{M}_1}(\mathcal{P}_1)$ *is* $\mathcal{M} = \{$`male(tommy)`, `person(tommy)`$\}$. *Since* $\mathcal{M}_1 \neq \mathcal{M}$, *we conclude that* $\mathcal{M}_1$ *is not a paraconsistent stable model. It can be easily checked that* $\mathcal{M}_2$ *is in this case a paraconsistent stable model.* □

We introduce now the notion of a ground literal l to be implied by an extended logic program $\mathcal{P}$, denoted as $\mathcal{P} \models l$.

Definition 3.10 *Let* $\mathcal{P}$ *be an extended logic program and* L *be a ground literal. There is a paraconsistent stable model* $\mathcal{M}$ *of* $\mathcal{P}$ *such that* $\mathcal{M} \models L$ *if and only if* $\mathcal{P} \models L$.

We conclude this section with a final remark. The semantics presented in this section is non-monotonic because by adding a new statement to a program the set of literals implied by the program may decrease. For instance consider again the program of example 3.4. If the fact `guilty(tommy).` would be added to $\mathcal{P}$ then $\mathcal{P} \not\models$`innocent(tommy)`.

3.2.2 Queries

As we mentioned in the end of the section 3.1, one of the main aims of the ELP framework is to extract information from extended logic programs by querying them. Let us introduce the notion of query.

Definition 3.11 *A* query *is a pair* $(L_1, \ldots, L_n\ ,\ \mathcal{P})$, *with* $n \geq 1$, *where* $\mathcal{P}$ *is an extended logic program and each* L_i *is a literal.*

We need now to define the notion of answer.

Definition 3.12 *Let* $(\mathcal{Q},\ \mathcal{P})$ *be a query. An* answer *to the query is the set of ground substitutions*

$$\{\theta \mid Q\theta \in ground(\mathcal{Q}) \text{ and } \mathcal{M} \models Q\theta\}\ ,$$

for some paraconsistent stable model $\mathcal{M}$ of $\mathcal{P}$.

Example 3.7 *Consider the extended logic program $\mathcal{P}$ of example 3.4 and the queries*

($\texttt{innocent(X)}$, $\mathcal{P}$)	*"Who is innocent?"*
($\texttt{guilty(tommy)}$, $\mathcal{P}$)	*"Is Tommy guilty?"*

The answer to the first query is $\{\{X/\texttt{tommy}\}\}$, since `innocent(tommy)` *belongs to a paraconsistent stable model of $\mathcal{P}$ (see $\mathcal{M}_2$ in example 3.6). However, the answer to the second question is $\emptyset$ because the literal* `guilty(tommy)` *does not belong to any paraconsistent stable model of $\mathcal{P}$.* □

Finally, we refer that *Smodels* [NS96, Sim] and *dlv* [ELM$^+$98, Pro] are currently available systems for computing stable models of programs (often with tens of thousands of clauses). Both systems can also handle integrity constraints, and can be used in practice to determine paraconsistent stable models of extended logic programs. Moreover, any standard *Prolog* system [DEBC96] can be used to compute answers to queries ($\mathcal{Q}$, $\mathcal{P}$), when $\mathcal{P}$ is a definite logic program.

Chapter 4

A Language for Defining Rough Relations

This chapter presents a new language [VDM03b, VDM03a] for defining and querying rough relations, based on logic programming.

The main intuitive idea underlying this language is as follows. A rough relation S divides the universe in four regions: those examples that definitely belong to the concept represented by S (to be denoted $\underline{S}$); those examples that definitely do not belong to the concept (to be denoted $\underline{\neg S}$), those examples for which there is contradictory evidence (to be denoted $\overline{\underline{S}}$); and those examples for which there is not any information of whether they belong to the concept (i.e. the remaining part of the universe not contained in $\underline{S} \cup \underline{\neg S} \cup \overline{\underline{S}}$). Using clauses we can then combine regions of different rough relations to define implicitly a new rough relation. The language introduced in this chapter does not take into account quantitative measures. This extension is discussed in chapter 6.

The declarative semantics of the language (discussed in section 4.2) associates a rough set S with each predicate symbol s of the language. Hence, we give indirectly a four-valued interpretation to each predicate: if tuple $t \in \underline{S}$ then $s(t)$ has the logic value `true`; if tuple $t \in \underline{\neg S}$ then $s(t)$ has the logic value `false`; if tuple $t \in \overline{\underline{S}}$ then $s(t)$ has the logic value $\top$ (denoting contradictory evidence); otherwise, $s(t)$ has the logic value $\bot$ (denoting lack of information). Four-valued logics have been studied by other authors, of which the most well-known is Belnap's four-valued logic [Bel77b, Bel77a]. However, statements in our language make explicit reference to one of the

three regions of a rough set (lower approximation, upper approximation, or boundary region) or the remaining part of the world not belonging to any of these three regions. In other words, the statements contain explicit tests of whether a tuple t belongs to one of those four regions (as shown in the next section, predicate symbols can only occur in rough and testing literals). Since each of those four regions is a crisp set, we do not need to resort to four-valued logical operations (e.g. disjunction). We use instead two-valued logic. The main advantage of using two-valued semantics is that there are several systems [DEBC96, Sim] readily available that can be used for making computations and answering queries. This aspect is particularly relevant from the point of view of implementing a system that can answer queries about knowledge bases encoded in our language. Formulating a language with a four-valued semantics that caters for representing and reasoning with rough concepts is out of the scope of this thesis, although this could be an issue for future work.

Section 4.1 introduces the language and its declarative semantics is then formalized in section 4.2. A transformation of the proposed language into the language of extended logic programs and a proof of its correctness is presented in section 4.3. Finally, section 4.4 puts forward a query language to extract information from rough relations defined in a program.

4.1 The Syntax

The language we are going to introduce uses three disjoint alphabets: an alphabet of variable symbols Var, an alphabet of constant symbols $Const$, and an alphabet of predicate symbols $Pred$. The notions of term and atom are similar to the ones introduced for logic programs. A *term* t is any symbol belonging to $Var \cup Const$. We follow the usual convention that variables start with upper case letter (e.g. $X, Dist \in Var$) and constants begin with lower case (e.g. $small, c \in Const$). Moreover, an n-ary predicate p is often denoted $p/n \in Pred$ and the set of symbols $\{\neg, not\} \nsubseteq Pred$. An *atom* A is an expression of the form $p(t_1, \ldots, t_n)$, where p is an n-ary predicate symbol and each $t_1, \ldots, t_n$ is a term.

Given a predicate p/n of arity $n > 0$, formulas of the form $\underline{l}(t_1, \ldots, t_n)$, $\overline{l}(t_1, \ldots, t_n)$, or $\underline{\overline{l}}(t_1, \ldots, t_n)$, where l is either p or $\neg p$, are called *rough literals*. Moreover, the expression $p?(t_1, \ldots, t_n)$ represents a *testing literal*.

Definition 4.1 *A* rough clause *is any expression of the form*

$$H \text{ :- } B_1, \ldots, B_n, T_1, \ldots, T_m.$$

where H and every B_i $(0 \leq i \leq n)$ is a rough literal, and each T_j $(0 \leq j \leq m)$ is a testing literal such that all variables occurring in a testing literal also occur in some B_i.

Rough clauses with an empty body (i.e. $n = 0$ and $m = 0$) are called *rough facts*.

The order by which rough (testing) literals occur in a rough clause is irrelevant. Thus, both rough clauses

$$\overline{p}(X,Y) \texttt{:-}\ \underline{q_1}(X,Y), \underline{\neg q_2}(X,Y), r_1?(X), r_2?(Y).$$
$$\overline{p}(X,Y) \texttt{:-}\ \underline{\neg q_2}(X,Y), \underline{q_1}(X,Y), r_2?(X), r_1?(Y).$$

have the same meaning.

We can now define the notion of rough program.

Definition 4.2 *A* rough program $\mathcal{P}$ *is a finite set of rough clauses.*

Intuitively, each predicate p denotes a rough relation P and we use rough literals to represent evidence about tuples. The lower (upper) approximation of P is represented by $\underline{p}(t_1, \ldots, t_n)$ ($\overline{p}(t_1, \ldots, t_n)$). The boundary of P is denoted by $\underline{\overline{p}}(t_1, \ldots, t_n)$. Obviously, the rough literals $\underline{\overline{p}}(t_1, \ldots, t_n)$ and $\underline{\overline{\neg p}}(t_1, \ldots, t_n)$ have the same meaning. For instance, the rough facts[1]

$$\underline{\overline{\texttt{recommendLenses}}}(\texttt{young}, \texttt{myope}, \texttt{yes}).$$

and

$$\underline{\overline{\neg\texttt{recommendLenses}}}(\texttt{young}, \texttt{myope}, \texttt{yes}).$$

express the information that the tuple $\langle \texttt{young}, \texttt{myope}, \texttt{yes} \rangle$ belongs to both $\underline{\overline{\mathit{RecommendLenses}}}$ and to $\underline{\overline{\neg \mathit{RecommendLenses}}}$ (thus, to the boundary of *RecommendLenses*). Informally, the first fact states that young myope people with astigmatism should use contact lenses while the second asserts exactly the opposite (perhaps, because different opticians have different opinions for these customers). The rough fact

$$\underline{\texttt{recommendLenses}}(\texttt{young}, \texttt{myope}, \texttt{no}).$$

states that the tuple $\langle \texttt{young}, \texttt{myope}, \texttt{no} \rangle$ is a positive example of rough relation *RecommendLenses* but cannot be a negative example of it (i.e. $\langle \texttt{young}, \texttt{myope}, \texttt{no} \rangle \in \underline{\mathit{RecommendLenses}}$). Notice that rough literals of the form $\underline{\overline{\neg p}}(t_1, \ldots, t_n)$ or $\underline{\neg p}(t_1, \ldots, t_n)$ express negative evidence.

[1] The third condition attribute refers to whether the person suffers from astigmatism.

A testing literal $p?(t_1, \ldots, t_n)$ asserts that there is no information whether tuple $\langle t_1, \ldots, t_n \rangle$ describes a positive and/or a negative example of the concept represented by rough relation P.

A decision table $\mathcal{D} = (U, A, d)$ can be easily represented in our language, if quantitative measures are ignored. A row $\langle c_1, \ldots, c_n \rangle$ of $\mathcal{D}$ corresponding to a positive (negative) example, where each $c_i \in V_{a_i}$ is the value of a condition attribute $a_i \in A$, is represented as the fact $\overline{d}(c_1, \ldots, c_n)$ ($\overline{\neg d}(c_1, \ldots, c_n)$). An important aspect to bear in mind is that the proposed language does not represent the individuals in U. Rough relations are represented as sets of tuples of attribute values, not as sets of individuals.

We stress that condition attributes are not referred by their name (e.g. `Age`) in the rough literals. They are instead identified by their position in the argument list of the rough literal. For instance, we use the convention that the first argument of the predicate `recommendLenses` represents the condition attribute `Age`. The condition attribute associated with a term t_i in a rough literal

$$\begin{array}{llll} \overline{q}(t_1, \ldots, t_n) & \text{or} & \overline{\neg q}(t_1, \ldots, t_n) & \text{or} \\ \underline{q}(t_1, \ldots, t_n) & \text{or} & \underline{\neg q}(t_1, \ldots, t_n) & \text{or} \\ \underline{\overline{q}}(t_1, \ldots, t_n) & \text{or} & \underline{\overline{\neg q}}(t_1, \ldots, t_n), & \end{array}$$

is represented as $att_Q(i)$. Each term t_i can only represent values belonging to $V_{att_Q(i)}$.

A rough clause represents an implication, as in the context of logic programs. The use of variables in a rough literal of a rough clause indicate that the underlying implication is valid for each possible value of the corresponding condition attribute. Since rough clauses allow lower and upper approximations of a relation as well as boundaries to occur both in the body and in the head of a clause, it is possible to define separately each of the regions (i.e. lower and upper approximations and boundary) of a rough relation in terms of regions of other rough relations. For instance, we can state that the boundary of a rough relation Q is contained in the lower approximation of another rough relation P. If predicates $q/3$ and $p/3$ denote the rough relations Q and P, respectively, then the rough clause

$$\underline{p}(X_1, X_2, X_3) \texttt{:-}\ \underline{\overline{q}}(X_1, X_2, X_3).$$

captures such information.

Given a rough relation P with n attributes, an n-ary tuple t is *undefined* with respect to P if and only if t is neither a positive nor a negative example

of the relation, i.e.

$$\langle t_1, \ldots, t_n \rangle \notin \overline{P} \text{ and } \langle t_1, \ldots, t_n \rangle \notin \overline{\neg P} \ ,$$

where each $t_i \in V_{att_P(i)}$. We can test in the body of a rough clause whether a tuple $\langle t_1, \ldots, t_n \rangle$ is undefined with respect to P, by using the testing literal $p?(t_1, \ldots, t_n)$.

The following rough clause

$$\underline{p}(X_1, X_2, X_3) \texttt{:-}\ \underline{\overline{q}}(X_1, X_2, X_3), r?(X_1, X_2, X_3).$$

asserts that if a tuple $t \in \overline{Q}$ and and t is undefined with respect to R (i.e. there is no information whether t describes a positive or a negative example of the concept represented by rough relation R) then t also belongs to $\underline{P}$.

The following two examples motivate the potential usefulness of our language. More examples are presented in the next chapter.

Example 4.1 *A relation Train has two arguments (condition attributes) representing time and location, respectively. Two (or more) sensors automatically detect presence/absence of an approaching train at a crossing, producing facts like* $\overline{\texttt{train(12:50,montijo).}}$ *automatically added to the knowledge base. A malfunction of a sensor may result in the contradictory fact* $\overline{\neg \texttt{train(12:50,montijo).}}$ *being added, too. Crossing is allowed if for sure no train approaches. This can be described by the following clause involving lower approximation in the body.*

$\overline{\texttt{cross}}\texttt{(X,Y) :-}\ \underline{\neg \texttt{train}}\texttt{(X,Y).}$

□

Example 4.2 *Statistical data on purchases of certain product during a calendar year is organized as a decision table with the following* 3 *condition attributes defining groups of customers:*

`Area` - *zip code of the area where the customer lives*

`Income` - *customer's income interval*

`Age` - *customer's age interval*

Notice that the decision table may define a rough relation: a `young` *customer living in* `Norrköping` *and having* `medium` *income may be considered* `inactive` *(perhaps, because he has not bought any product item during last*

year) while another young *customer, also living in* Norrköping *and with* medium *income is classified as active.*

The marketing department uses the activity tables act1 *and* act2 *from two consecutive years to identify the groups of* growing activity *(*ga*). The tables are represented as rough facts in our language. The activity of a group may be defined: (1) as* definitely growing*, if the group was possibly inactive in year 1 and definitely active in year 2; (2) as* definitely non growing*, if its activity changed from possibly active to definitely inactive; and (3) as a* boundary*, if the activity was boundary in both years. This can be described by the following rough clauses.*

(1) $\underline{ga}$(Area, Inc, Age) :- $\overline{\neg act1}$(Area, Inc, Age),
$\underline{act2}$(Area, Inc, Age).

(2) $\underline{\neg ga}$(Area, Inc, Age) :- $\overline{act1}$(Area, Inc, Age),
$\underline{\neg act2}$(Area, Inc, Age).

(3) $\overline{\underline{ga}}$(Area, Inc, Age) :- $\overline{\underline{act1}}$(Area, Inc, Age),
$\overline{\underline{act2}}$(Area, Inc, Age).

□

The language described in this section extends substantially the language presented in [MV02, VM02]. The language discussed in this previous work only allows the use of upper approximations in the definition of new rough relations.

4.2 The Declarative Semantics

The main idea underlying the proposed language is that each predicate symbol occurring in a rough program denotes a rough relation. We formalize this idea in this section.

Definition 4.3 *Let $\mathcal{P}$ be a rough program. A* rough interpretation *$\mathcal{I}$ of $\mathcal{P}$ is a function mapping each predicate symbol q/n occurring in P into a rough relation $Q^{\mathcal{I}} = (\overline{Q^{\mathcal{I}}}, \overline{\neg Q^{\mathcal{I}}})$ such that $\overline{Q^{\mathcal{I}}}, \overline{\neg Q^{\mathcal{I}}} \subseteq \prod_{1 \leq i \leq n} V_{att_Q(i)}$.*

If no variables occur in a rough (testing) literal then the rough (testing) literal is called a *ground rough (testing) literal*. A rough clause is ground, if all rough and testing literals occurring in it are also ground.

The notion of a rough literal L being true in a rough interpretation $\mathcal{I}$, denoted as $\mathcal{I} \models L$, is defined by statements (1) – (8) below.

(1) $\mathcal{I} \models \overline{q}(c_1, \ldots, c_n) \Leftrightarrow \langle c_1, \ldots, c_n \rangle \in \overline{Q^{\mathcal{I}}}$.

(2) $\mathcal{I} \models \underline{q}(c_1, \ldots, c_n) \Leftrightarrow (\langle c_1, \ldots, c_n \rangle \in \overline{Q^{\mathcal{I}}}$ and $\langle c_1, \ldots, c_n \rangle \notin \overline{\neg Q^{\mathcal{I}}})$.

(3) $\mathcal{I} \models \overline{\underline{q}}(c_1, \ldots, c_n) \Leftrightarrow (\langle c_1, \ldots, c_n \rangle \in \overline{Q^{\mathcal{I}}}$ and $\langle c_1, \ldots, c_n \rangle \in \overline{\neg Q^{\mathcal{I}}})$.

(4) $\mathcal{I} \models \overline{\neg q}(c_1, \ldots, c_n) \Leftrightarrow \langle c_1, \ldots, c_n \rangle \in \overline{\neg Q^{\mathcal{I}}}$.

(5) $\mathcal{I} \models \underline{\neg q}(c_1, \ldots, c_n) \Leftrightarrow (\langle c_1, \ldots, c_n \rangle \in \overline{\neg Q^{\mathcal{I}}}$ and $\langle c_1, \ldots, c_n \rangle \notin \overline{Q^{\mathcal{I}}})$.

(6) $\mathcal{I} \models \underline{\overline{\neg q}}(c_1, \ldots, c_n) \Leftrightarrow (\langle c_1, \ldots, c_n \rangle \in \overline{Q^{\mathcal{I}}}$ and $\langle c_1, \ldots, c_n \rangle \in \overline{\neg Q^{\mathcal{I}}})$.

(7) $\mathcal{I} \models q?(c_1, \ldots, c_n) \Leftrightarrow (\langle c_1, \ldots, c_n \rangle \notin \overline{Q^{\mathcal{I}}}$ and $\langle c_1, \ldots, c_n \rangle \notin \overline{\neg Q^{\mathcal{I}}})$.

(8) $\mathcal{I} \models B_1, \cdots, B_n, T_1, \cdots, T_m \Leftrightarrow (\mathcal{I} \models B_1, \cdots, \mathcal{I} \models B_n$, $\mathcal{I} \models T_1, \cdots, \mathcal{I} \models T_m)$, where each B_i $(0 \leq i \leq n)$ is a ground rough literal and each T_j $(0 \leq j \leq m)$ is a ground testing literal.

Note that for any rough interpretation $\mathcal{I}$, it is possible that

$$\mathcal{I} \models \overline{q}(t_1, \ldots, t_n), \overline{\neg q}(t_1, \ldots, t_n) \ ,$$

for two rough literals $\overline{q}(t_1, \ldots, t_n)$ and $\overline{\neg q}(t_1, \ldots, t_n)$. However, if we consider lower approximations instead then we must have

$$\mathcal{I} \not\models \underline{q}(t_1, \ldots, t_n), \underline{\neg q}(t_1, \ldots, t_n) \ .$$

Consider a rough literal $l(t_1, \ldots, t_n)$, with $1 \leq n$ and l is either $\overline{q}$, or $\underline{q}$, or $\underline{\overline{q}}$, or $\overline{\neg q}$, or $\underline{\neg q}$, or $\underline{\overline{\neg q}}$, for some predicate q/n. Recall that each term t_i is associated with a condition attribute of rough relation Q whose value domain is $V_{att_Q(i)}$.

Definition 4.4 *Let $X_{1j}, \ldots, X_{nk}$ $(1 \leq n$ and $1 \leq j \leq k)$ be variables occurring in some rough literal, where the second index indicates the variable's position in the argument list of the literal. Assume also that q is the predicate symbol occurring in the rough literal. A* grounding substitution *θ is a set of bindings $\{X_{1j}/c_1, \ldots, X_{nk}/c_n\}$ (including the empty set) of variables X_{im} $(1 \leq i \leq n$ and $j \leq m \leq k)$ to constants $c_i \in V_{att_Q(m)}$.*

The notion of ground rough program is similar to the notion of ground (extended) logic program, presented in the previous chapter. A *ground instance of a rough clause C*, denoted by $C\theta$, is obtained by applying a grounding substitution θ to rough clause C and $C\theta$ is ground. Given a

rough program $\mathcal{P}$, $ground(\mathcal{P})$ represents the set of all ground instances of any rough clause $C \in \mathcal{P}$.

A rough interpretation $\mathcal{I}$ of a rough program $\mathcal{P}$ *satisfies* a rough clause

$$H \texttt{:-} B_1, \ldots, B_n, T_1, \ldots, T_m. \in ground(\mathcal{P})$$

if and only if

- if $\mathcal{I} \models B_1, \ldots, B_n, T_1, \ldots, T_m$ then $\mathcal{I} \models H$.

Definition 4.5 *A* model *of a rough program $\mathcal{P}$ is a rough interpretation that satisfies each rough clause of $ground(\mathcal{P})$.*

A rough program may have several models or no models at all. For instance, the rough program

$$\mathcal{P} = \{\underline{p}(a) \texttt{:-}\ \overline{q}(b).\ , \overline{\neg p}(a).\ , \overline{q}(b).\}$$

has no models. Example 4.3 below shows a rough program with more than one model.

We turn now to the definition of a partial order between the models of a rough program.

Definition 4.6 *Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two models of a rough program $\mathcal{P}$. $\mathcal{M}_1 \preceq \mathcal{M}_2$ if and only if*

$$\overline{Q^{\mathcal{M}_1}} \subseteq \overline{Q^{\mathcal{M}_2}} \text{ and } \overline{\neg Q^{\mathcal{M}_1}} \subseteq \overline{\neg Q^{\mathcal{M}_2}}\ ,$$

for every predicate symbol q/n occurring in $\mathcal{P}$.

Based on the partial order $\preceq$ defined above, we introduce the notion of minimal models.

Definition 4.7 *A model $\mathcal{M}$ is a* minimal model *of a rough program $\mathcal{P}$ if and only if $\mathcal{M}' \not\preceq \mathcal{M}$, for every other model $\mathcal{M}'$ of $\mathcal{P}$.*

We give next an example illustrating a situation where a rough program has more than one model.

Example 4.3 *Consider that we want to represent some expert knowledge saying that*

> *"If someone possibly has an* `infection` *but his* `temperature` *is* `normal`*, then he either might suffer from* `diseaseA` *or from* `diseaseB` *(but never from both)."*

Moreover, two decision tables are given. Based on the existence of certain symptoms and results of some clinical tests, several experts decide independently whether a patient has `diseaseA` *or* `diseaseB`*. Symptoms and clinical test results form the condition attributes. For instance, a condition attribute is* `temperature` *that can have the values* `low`*,* `normal`*, or* `high`*.*

The expert knowledge can be represented as follows. Assume that predicates `diseaseA` *and* `diseaseB` *have arity three (i.e. the corresponding decision tables have three condition attributes).*

$\overline{diseaseA}$(infect, normal, Z) :-
$\underline{\neg diseaseB}$(infect, normal, Z).

$\overline{diseaseB}$(infect, normal, Z) :-
$\underline{\neg diseaseA}$(infect, normal, Z).

Note that rough relations $DiseaseA$ *and* $DiseaseB$*, denoted by predicates* `diseaseA` *and* `diseaseB` *respectively, are partially defined explicitly by decision tables. Some other tuples belonging to these relations are obtained from the clauses above. In reality experts may decide in different ways whether a person may have a certain disease.*

Consider the rough program $\mathcal{P}$ *consisting of the two clauses above and including also the following facts obtained from the decision tables.*

$\overline{\neg diseaseA}$(infect, normal, c). $\overline{\neg diseaseB}$(infect, normal, c).

Program $\mathcal{P}$ *has (at least) two models,* $\mathcal{M}_1$ *and* $\mathcal{M}_2$*, reflecting two possible situations according to the available knowledge.*

- $\langle infect, normal, c\rangle \in \overline{\underline{DiseaseA^{\mathcal{M}_1}}}$ *and* $\langle infect, normal, c\rangle \in \underline{\neg DiseaseB^{\mathcal{M}_1}}$.
- $\langle infect, normal, c\rangle \in \overline{\underline{DiseaseB^{\mathcal{M}_2}}}$ *and* $\langle infect, normal, c\rangle \in \underline{\neg DiseaseA^{\mathcal{M}_2}}$.

In practice it may be desirable to find preferred models or at least discard some models seen as not relevant. This issue has been studied in the context of logic programming and the proposed techniques may be applicable here. In the context of rough sets, we could address this problem by extending to clauses the quantitative measures associated with decision tables. In the example above, model $\mathcal{M}_1$ *is obtained by applying the first clause above, while model* $\mathcal{M}_2$ *is obtained by applying the second clause. If the tuple* $\langle$`infect`, `normal`, `c`$\rangle$ *appears as a negative example many more times in the*

decision table for `diseaseB` *than in the decision table for* `diseaseA` *then we may decide to discard the second model.*

□

Let $\mathcal{P}$ be a rough program and q be a relation symbol occurring in $\mathcal{P}$. We are obviously not interested in any model $\mathcal{M}$ of $\mathcal{P}$ that makes more tuples to belong to $\overline{Q^{\mathcal{M}}}$ or to $\overline{\neg Q^{\mathcal{M}}}$ than what is needed to satisfy the rough clauses of $\mathcal{P}$. Thus, minimal models (as defined in 4.7) seem good candidates to express the meaning of a rough program. However, not all minimal models may properly capture the semantics of a rough program. Next example tries to illustrate this point.

Example 4.4 *Consider the following rough program*

$$\mathcal{P} = \{\overline{r}(c) \text{ :- } \underline{\neg r}(c). \, , \overline{\neg r}(c).\} \, .$$

A minimal model $\mathcal{M}$ of $\mathcal{P}$ maps predicate r into the rough relation $R^{\mathcal{M}} = (\{\langle c\rangle\}, \{\langle c\rangle\})$ i.e. $\langle c\rangle$ belongs to the boundary of rough relation $R^{\mathcal{M}}$. However, no information encoded in the rough clauses of $\mathcal{P}$ leads to the conclusion that $\langle c\rangle \in \overline{R^{\mathcal{M}}}$. In order to be able to conclude that $\langle c\rangle \in \overline{R^{\mathcal{M}}}$, it would be needed that $\mathcal{M} \models \underline{\neg r}(c)$. By rough clause

$$\overline{r}(c) \text{ :- } \underline{\neg r}(c). \qquad (1)$$

we could then conclude that $\langle c\rangle \in \overline{R^{\mathcal{M}}}$. However, $\mathcal{M} \not\models \underline{\neg r}(c)$. Hence, it seems reasonable to reject model $\mathcal{M}$. Note that a rough interpretation $\mathcal{I}$ such that $R^{\mathcal{I}} = (\emptyset, \{\langle c\rangle\})$ is not a model of $\mathcal{P}$ because it does not satisfy rough clause (1)*. Thus, $\mathcal{P}$ seems to bear a contradiction. Rough clause* (1) *informally states that if there is evidence that $\langle c\rangle$ is only a negative example of the concept represented by rough relation R (i.e. there is no evidence that $\langle c\rangle$ is a positive example of the concept) then we can conclude that $\langle c\rangle$ is also a positive example of that concept.*

□

Let $\mathcal{I}$ be a rough interpretation of a rough program $\mathcal{P}$. In order to be able to define the declarative semantics of a rough program, we need to introduce the following function, $\Psi_{\mathcal{I}}$, transforming $\mathcal{P}$ into a ground rough program $\Psi_{\mathcal{I}}(\mathcal{P})$ such that neither lower approximations nor testing literals occur in the body of any of its rough clauses. This transformation can be informally described as follows. Assume that C is a rough clause of $ground(\mathcal{P})$. For every ground rough literal $\underline{q}(t_1, \ldots, t_n)$ referring to a lower approximation

and occurring in the body of C, if $\mathcal{I} \not\models \overline{\neg q}(t_1, \ldots, t_n)$ then $\underline{q}(t_1, \ldots, t_n)$ in the body of C is replaced by $\overline{q}(t_1, \ldots, t_n)$. Moreover, if a ground testing literal occurring in the body of C is true in $\mathcal{I}$, then it is removed from the body of the clause. The underlying idea behind this transformation is that if $\mathcal{I}$ is a model of $\mathcal{P}$ then it should also be a model of the transformed rough program. We give a simple example of this transformation.

Example 4.5 *Consider the following ground rough clause*

$$\underline{p}(a,b) \text{ :- } \underline{q}(a,b), r?(b,c). \in \mathit{ground}(\mathcal{P})$$

and an interpretation $\mathcal{I}$ of $\mathcal{P}$ such that $\overline{Q^{\mathcal{I}}} = \overline{\neg Q^{\mathcal{I}}} = \overline{R^{\mathcal{I}}} = \overline{\neg R^{\mathcal{I}}} = \emptyset$. Since $\mathcal{I} \not\models \overline{\neg q}(a,b)$ and $\mathcal{I} \models r?(b,c)$, we have that

$$\underline{p}(a,b) \text{ :- } \overline{q}(a,b). \in \Psi_{\mathcal{I}}(\mathcal{P}) .$$

□

We present below the formal definition of function Ψ. This definition extends the notion of reduct proposed in [GL88] to rough programs.

Consider that $\neg\neg q \equiv q$, for any predicate symbol q.

Definition 4.8 *Let $\mathcal{P}$ be a rough program and $\mathcal{I}$ be a rough interpretation of $\mathcal{P}$. Assume also that each l_j $(1 \leq j \leq k)$ in the expression below is either q_i or $\neg q_i$, for some predicate q_i. Then $\Psi_{\mathcal{I}}(\mathcal{P})$ maps $\mathcal{P}$ into a ground rough program satisfying the following condition ($n, i, k \geq 0$ and $m_1, \ldots, m_k \geq 0$):*

$$H \text{ :- } B_1, \ldots, B_n, \overline{l_1}(t_{11}, \ldots t_{1m_1}), \ldots, \overline{l_k}(t_{k1}, \ldots t_{km_k}). \in \Psi_{\mathcal{I}}(\mathcal{P})$$

if and only if there is a rough clause

$$\begin{array}{ll} H \text{ :- } & B_1, \ldots, B_n, \\ & \underline{l_1}(t_{11}, \ldots t_{1m_1}), \ldots, \underline{l_k}(t_{k1}, \ldots t_{km_k}), \\ & \overline{T_1}, \ldots, T_i. \in \mathit{ground}(\mathcal{P}) \end{array}$$

such that

$$\begin{array}{lll} \mathcal{I} & \not\models & \overline{\neg l_1}(t_{11}, \ldots t_{1m_1}), \ldots, \overline{\neg l_k}(t_{k1}, \ldots t_{km_k}) \text{ and} \\ \mathcal{I} & \models & T_1, \ldots, T_i , \end{array}$$

where each B_j $(0 \leq j \leq n)$ is a rough literal not referring to a lower approximation and each T_l $(0 \leq l \leq i)$ is a testing literal.

An important property of a rough program $\Psi_{\mathcal{I}}(\mathcal{P})$ is that it either has a least model (a unique minimal model) or no model at all.

Lemma 4.1 *Let $\mathcal{P}$ be a rough program and $\mathcal{I}$ be a rough interpretation of $\mathcal{P}$. If $\Psi_{\mathcal{I}}(\mathcal{P})$ has a model then it has a least model, with respect to partial relation $\preceq$.*

Proof: Let us assume that $\Psi_{\mathcal{I}}(\mathcal{P})$ has a model. To prove that $\Psi_{\mathcal{I}}(\mathcal{P})$ has a least model, we show that any rough interpretation $\mathcal{M}$ defined as below is also a model of $\Psi_{\mathcal{I}}(\mathcal{P})$. Let V be any non-empty set of models of $\Psi_{\mathcal{I}}(\mathcal{P})$.

$$Q^{\mathcal{M}} = (\bigcap_{\mathcal{M}' \in V} \overline{Q^{\mathcal{M}'}}, \bigcap_{\mathcal{M}' \in V} \overline{\neg Q^{\mathcal{M}'}}) ,$$

for each rough relation Q.

Assume that $H \texttt{:-} B. \in \Psi_{\mathcal{I}}(\mathcal{P})$ and that $\mathcal{M} \models B$. Then, $W \models B$, for every $W \in V$. The key to understanding this point is that the body B can only contain rough literals referring to upper approximations and to boundaries. Moreover, upper approximations and boundaries can be seen as monotonic operators.

We conclude that $W \models H$, for every model $W \in V$, and consequently, $\mathcal{M} \models H$.

It is obvious that $\mathcal{M} \preceq W$, for all $W \in V$.

□

The meaning of a rough program is captured by those minimal models that satisfy the condition described in the following definition.

Definition 4.9 *Let $\mathcal{P}$ be a rough program and $min(\mathcal{P})$ be the set of minimal models of $\mathcal{P}$. The semantics of $\mathcal{P}$, denoted as $sem(\mathcal{P})$, is defined as*

$$sem(\mathcal{P}) = \{\mathcal{M} \in min(\mathcal{P}) \mid \mathcal{M} \text{ is the least model of } \Psi_{\mathcal{M}}(\mathcal{P})\} .$$

Example 4.6 *Consider a rough program $\mathcal{P}$ containing only the rough clauses and facts of example 4.3. Then, $sem(\mathcal{P}) = \{\mathcal{M}_1, \mathcal{M}_2\}$.*

$$\begin{aligned} DiseaseA^{\mathcal{M}_1} &= (\{\langle \texttt{infect, normal, c}\rangle\}, \{\langle \texttt{infect, normal, c}\rangle\}) , \\ DiseaseB^{\mathcal{M}_1} &= (\emptyset, \{\langle \texttt{infect, normal, c}\rangle\}) , \end{aligned}$$

$$\begin{aligned} DiseaseA^{\mathcal{M}_2} &= (\emptyset, \{\langle \texttt{infect, normal, c}\rangle\}) \\ DiseaseB^{\mathcal{M}_2} &= (\{\langle \texttt{infect, normal, c}\rangle\}, \{\langle \texttt{infect, normal, c}\rangle\}) . \end{aligned}$$

Note that $\Psi_{\mathcal{M}_1}(\mathcal{P})$ has only the following rough clauses

$\overline{diseaseA}$(infect, normal, c) :-
    $\overline{\neg diseaseB}$(infect, normal, c).

$\overline{\neg diseaseA}$(infect, normal, c).

$\overline{\neg diseaseB}$(infect, normal, c).

It is easy to see that $\mathcal{M}_1$ ($\mathcal{M}_2$) is the least model of $\Psi_{\mathcal{M}_1}(\mathcal{P})$ ($\Psi_{\mathcal{M}_2}(\mathcal{P})$).

□

Different minimal models $\mathcal{M} \in sem(\mathcal{P})$ can be informally understood as different alternative scenarios implied by the knowledge encoded in $\mathcal{P}$.

Example 4.7 *Consider again the rough program presented in example 4.4,*

$$\mathcal{P} = \{\overline{r}(c) :\text{-}\ \underline{\neg r}(c).\ , \overline{\neg r}(c).\}\ ,$$

and the minimal model $\mathcal{M}$ such that $R^{\mathcal{M}} = (\{\langle c\rangle\}, \{\langle c\rangle\})$. Note that $\Psi_{\mathcal{M}}(\mathcal{P}) = \{\overline{\neg r}(c).\}$. Obviously, $\mathcal{M}$ is not the least model of $\Psi_{\mathcal{M}}(\mathcal{P})$. Hence, $\mathcal{M} \notin sem(\mathcal{P})$.

We can easily see that $sem(\mathcal{P}) = \emptyset$.

□

Finally, we introduce the notion of a rough (testing) literal l to be implied by a rough program $\mathcal{P}$, denoted as $\mathcal{P} \models l$.

Definition 4.10 *Let $\mathcal{P}$ be a rough program and l be a rough or testing literal. There is a model $\mathcal{M} \in sem(\mathcal{P})$ such that $\mathcal{M} \models l$ if and only if $\mathcal{P} \models l$.*

4.3 Computing the Semantics of Rough Programs

In the previous section, we define the declarative semantics of a rough program $\mathcal{P}$ as a subset of its minimal models. An obvious question is how such models, belonging to $sem(\mathcal{P})$, can be computed. This section addresses this problem.

Each rough program is compiled to an extended logic program. As we show in section 4.3.2, each paraconsistent stable model, of the extended logic program obtained by compiling a rough program $\mathcal{P}$, is isomorphic to a model belonging to $sem(\mathcal{P})$, and vice-versa. The operational semantics of extended logic programs is well studied [NS96, Bar03] and there are several

systems, like *dlv* [ELM$^+$98, Pro] and *Smodels* [NS97, Sim], that can be used to compute paraconsistent stable models of extended logic programs.

Some rough programs have at most one model. Absence of recursion is a sufficient condition for a rough program to have either a least model or no models. These rough programs can be compiled to a non-recursive extended logic program. Rough programs can be queried (queries are discussed in section 4.4). Queries to a rough program are also transformed into queries to the compiled program. Given a non-recursive extended logic program, any standard *Prolog* system [DEBC96] can be used to determine whether this program has a paraconsistent stable model and answer queries. Hence for non-recursive rough programs, we could implement a system based on our ideas in *Prolog*. An easy way to verify whether a rough program is recursive consists in checking whether the ground compiled extended logic program is recursive.

We give below some examples of recursive and non-recursive rough programs.

Example 4.8 *Consider the following rough programs.*

$$\begin{array}{lcl} \mathcal{P}_1 & = & \{\underline{q}(a,b) \text{ :- } \overline{\underline{r}}(a,b).\ , \overline{r}(a,b) \text{ :- } \overline{q}(a,b).\}\ , \\ \mathcal{P}_2 & = & \{\underline{q}(a,b) \text{ :- } \overline{\underline{r}}(a,b).\ , \overline{r}(a,b) \text{ :- } \overline{q}(c,b).\}\ , \\ \mathcal{P}_3 & = & \{\underline{q}(a,b) \text{ :- } \overline{\neg q}(a,b).\}\ . \end{array}$$

Rough program $\mathcal{P}_1$ is recursive while rough programs $\mathcal{P}_2$ and $\mathcal{P}_3$ are not. □

Generally speaking, the application problems discussed in rough set literature that can be formulated in our language do not seem to require recursive rough programs and, therefore, they are not compiled to recursive extended logic programs. Some of these applications are discussed in the next chapter. Moreover, we have also implemented in *Prolog* a system that is able to reason about rough relations defined in a non-recursive rough program. This system is the topic of chapter 6.

4.3.1 Compiling Rough Programs into Extended Logic Programs

In this section, we discuss in detail how rough programs can be transformed into extended logic programs.

The transformation to be presented has the following property. A model $\mathcal{M}$ of a rough program $\mathcal{P}$ belongs to $sem(\mathcal{P})$ if and only if there is a paraconsistent stable model $\mathcal{M}'$ of the transformed program $\mathcal{P}'$ such that each predicate symbol q/n occurring in $\mathcal{P}$ denotes the rough relation

$$\begin{array}{rcl} Q^{\mathcal{M}} & = & (\{\langle c_1,\ldots,c_n\rangle \mid q(c_1,\ldots,c_n) \in \mathcal{M}'\}, \\ & & \{\langle c_1,\ldots,c_n\rangle \mid \neg q(c_1,\ldots,c_n) \in \mathcal{M}'\}) \ . \end{array}$$

In section 4.3.2, we prove that this property is in fact guaranteed by the proposed compilation.

The intuition underlying the compilation procedure is as follows. Assume that P and Q are the rough relations denoted by predicates p and q occurring in a rough program, respectively. Then, the literal $p(t_1,\ldots,t_n)$ states that the tuple $\langle t_1,\ldots,t_n\rangle$ belongs to P and the literal $\neg p(t_1,\ldots,t_n)$ indicates that tuple $\langle t_1,\ldots,t_n\rangle$ is not in P. (i.e. belongs to $\neg P$). The default negated literal $not\ p(t_1,\ldots,t_n)$ ($not\ \neg p(t_1,\ldots,t_n)$) states that there is no evidence that the tuple $\langle t_1,\ldots,t_n\rangle$ is a positive (negative) example of P. Now the notions of approximations and boundary reflected by rough literals can be equivalently expressed by conjunctions of literals of extended logic programs, as formalized by the following transformation τ_2. This transformation can be used to compile rough literals in the bodies of rough clauses.

$$\begin{array}{rcl}
\tau_2(p?(t_1,\ldots,t_n)) & = & not\ p(t_1,\ldots,t_n), not\ \neg p(t_1,\ldots,t_n) \ , \\
\tau_2(\underline{p}(t_1,\ldots,t_n)) & = & p(t_1,\ldots,t_n), not\ \neg p(t_1,\ldots,t_n) \ , \\
\tau_2(\underline{\neg p}(t_1,\ldots,t_n)) & = & \neg p(t_1,\ldots,t_n), not\ p(t_1,\ldots,t_n) \ , \\
\tau_2(\overline{p}(t_1,\ldots,t_n)) & = & p(t_1,\ldots,t_n) \ , \\
\tau_2(\overline{\neg p}(t_1,\ldots,t_n)) & = & \neg p(t_1,\ldots,t_n) \ , \\
\tau_2(\overline{p}(t_1,\ldots,t_n)) & = & p(t_1,\ldots,t_n), \neg p(t_1,\ldots,t_n) \ , \\
\tau_2(\overline{\neg p}(t_1,\ldots,t_n)) & = & \tau_2(\overline{p}(t_1,\ldots,t_n)) \ , \\
\tau_2((B_1,\ldots,B_n)) & = & \tau_2(B_1),\ldots,\tau_2(B_n) \ .
\end{array}$$

The translation above is not directly applicable to the heads of rough clauses, since the heads in the target programs can contain neither conjunctions of literals nor default negated literals. In order to address this problem, rough clauses in the source program are compiled into a clause and an integrity constraint of the target program, as described below. For example, consider the rough clause

$$\underline{p}(X_1,X_2,X_3) \text{:-}\ \overline{q}(X_1,X_2,X_n).$$

stating that the boundary of Q (rough relation denoted by predicate $q/3$) is contained in the lower approximation of P (rough relation denoted by

predicate $p/3$). Any element in the boundary of Q should be also considered a positive example of P but it should be excluded that those tuples are examples of $\neg P$. Moreover, a tuple t belongs to the boundary of Q if and only if it represents both positive and negative evidence of it. Thus,

$$p(X_1, X_2, X_3) \texttt{:-}\ q(X_1, X_2, X_3), \neg q(X_1, X_2, X_3). \quad (1)$$

and

$$\texttt{:-}\ \neg p(X_1, X_2, X_3), q(X_1, X_2, X_3), \neg q(X_1, X_2, X_3). \quad (2)$$

capture the same information as the rough clause above. Clause (1) states that tuples belonging to both Q and $\neg Q$ also belong to P, while the integrity constraint (2) does not allow those tuples to belong to $\neg P$.

The discussion above gives a motivation for the formalization of the translation of rough clauses into clauses of an extended logic program. This formalization is defined as the following function τ_1 which refers to the above defined function τ_2. Note that $\neg p$ in an extended logic program should essentially be viewed as a new predicate symbol representing explicit negation.

$$\begin{array}{rcl}
\tau_1(\underline{p}(t_1,\ldots,t_n)\texttt{:-}\ B.) & = & \{p(t_1,\ldots,t_n)\texttt{:-}\ \tau_2(B).\ , \\
 & & \ \ \texttt{:-}\ \neg p(t_1,\ldots,t_n)\ , \tau_2(B).\}\ , \\
\tau_1(\overline{p}(t_1,\ldots,t_n)\texttt{:-}\ B.) & = & \{p(t_1,\ldots,t_n)\texttt{:-}\ \tau_2(B).\}\ , \\
\tau_1(\underline{\neg p}(t_1,\ldots,t_n)\texttt{:-}\ B.) & = & \{\neg p(t_1,\ldots,t_n)\texttt{:-}\ \tau_2(B).\ , \\
 & & \ \ \texttt{:-}\ p(t_1,\ldots,t_n)\ , \tau_2(B).\}\ , \\
\tau_1(\overline{\neg p}(t_1,\ldots,t_n)\texttt{:-}\ B.) & = & \{\neg p(t_1,\ldots,t_n)\texttt{:-}\ \tau_2(B).\}\ , \\
\tau_1(\overline{\underline{p}}(t_1,\ldots,t_n)\texttt{:-}\ B.) & = & \{\neg p(t_1,\ldots,t_n)\texttt{:-}\ \tau_2(B).\ , \\
 & & \ \ p(t_1,\ldots,t_n)\texttt{:-}\ \tau_2(B).\}\ , \\
\tau_1(\underline{\overline{\neg p}}(t_1,\ldots,t_n)\texttt{:-}\ B.) & = & \tau_1(\overline{\underline{p}}(t_1,\ldots,t_n)\texttt{:-}\ B.)\ .
\end{array}$$

A rough program $\mathcal{P}$ will be transformed into an extended logic program by compiling each rough clause. Thus, $\tau_1(\mathcal{P}) = \bigcup_{C\in\mathcal{P}} \tau_1(C)$.

Next example illustrates the proposed encoding of rough programs.

Example 4.9 *Assume that we have two similar decision tables*

$$\begin{array}{l}
\mathcal{D}eathmi_1{=}(U_1, \{\texttt{Age}, \texttt{Hypert}, \texttt{Scanabn}\}, \texttt{Deathmi}_1)\ , \\
\mathcal{D}eathmi_2{=}(U_2, \{\texttt{Age}, \texttt{Hypert}, \texttt{Scanabn}\}, \texttt{Deathmi}_2)\ ,
\end{array}$$

referring to different periods of time (e.g. year 1 *and year* 2*, respectively). These tables record for several patients their age group (*Age*), whether they have hypertension (*Hypert*), and the result of a medical test to the heart*

(`Scanabn`*). The decision attribute indicates whether the patient had a major heart problem during the follow up period. Both tables are represented as a set of facts in our language.*

Our aim is to monitor changes in the boundary region from one period of time to another. For instance, this can give us an idea whether there are groups of patients for who the risk of having a serious cardiac problem has increased, decreased, or remained stable from the first period of time to the second period. Thus, if an indiscernibility class described by a tuple t belongs to the boundary of table $\mathcal{D}eathmi_1$ and the indiscernibility class corresponding to the same tuple t is contained in the lower approximation of the rough relation represented by table $\mathcal{D}eathmi_2$ then, we may interpret this fact as an increase of risk for those patients having the symptoms and test results indicated by t.

These ideas can be expressed by the following rough clauses defining a new rough relation, denoted by predicate `risk`*, in such a way that $\underline{Risk}$, $\underline{\neg Risk}$, and $\overline{\underline{Risk}}$ correspond to an increase, decrease, and stability of the risk of a cardiac event, respectively.*

(1) $\underline{risk}$`(Age, Hypert, Scanabn) :-`
$\overline{\underline{deathmi}}_1$`(Age, Hypert, Scanabn),`
$\underline{deathmi}_2$`(Age, Hypert, Scanabn).`

(2) $\underline{\neg risk}$`(Age, Hypert, Scanabn):-`
$\overline{\underline{deathmi}}_1$`(Age, Hypert, Scanabn),`
$\underline{\neg deathmi}_2$`(Age, Hypert, Scanabn).`

(3) $\overline{\underline{risk}}$`(Age, Hypert, Scanabn) :-`
$\overline{\underline{deathmi}}_1$`(Age, Hypert, Scanabn),`
$\overline{\underline{deathmi}}_2$`(Age, Hypert, Scanabn).`

Next, we show the result of compiling (i.e. applying function τ_1 to) each rough clause above.

- *Compilation of rough clause* (1).

```
risk(Age, Hypert, Scanabn) :-
       deathmi1(Age, Hypert, Scanabn),
       ¬deathmi1(Age, Hypert, Scanabn),
        deathmi2(Age, Hypert, Scanabn),
        not ¬deathmi2(Age, Hypert, Scanabn).

:- ¬risk(Age, Hypert, Scanabn),
     deathmi1(Age, Hypert, Scanabn),
     ¬deathmi1(Age, Hypert, Scanabn),
      deathmi2(Age, Hypert, Scanabn),
      not ¬deathmi2(Age, Hypert, Scanabn).
```

- *Compilation of rough clause* (2).

```
¬risk(Age, Hypert, Scanabn) :-
        deathmi1(Age, Hypert, Scanabn),
        ¬deathmi1(Age, Hypert, Scanabn),
        ¬deathmi2(Age, Hypert, Scanabn),
        not deathmi2(Age, Hypert, Scanabn).


:- risk(Age, Hypert, Scanabn),
    deathmi1(Age, Hypert, Scanabn),
    ¬deathmi1(Age, Hypert, Scanabn),
     ¬deathmi2(Age, Hypert, Scanabn),
     not deathmi2(Age, Hypert, Scanabn).
```

- *Compilation of rough clause* (3).

```
risk(Age, Hypert, Scanabn) :-
        deathmi1(Age, Hypert, Scanabn),
        ¬deathmi1(Age, Hypert, Scanabn),
        deathmi2(Age, Hypert, Scanabn),
        ¬deathmi2(Age, Hypert, Scanabn).


¬risk(Age, Hypert, Scanabn) :-
        deathmi1(Age, Hypert, Scanabn),
        ¬deathmi1(Age, Hypert, Scanabn),
        deathmi2(Age, Hypert, Scanabn),
        ¬deathmi2(Age, Hypert, Scanabn).
```

□

Recall that, for each rough interpretation $\mathcal{I}$ of a rough program $\mathcal{P}$, a predicate q occurring in $\mathcal{P}$ may denote a different rough relation, represented as $Q^{\mathcal{I}}$. Consequently, the denotation of a predicate is always with respect to a rough interpretation.

4.3.2 Correctness of the Compilation Procedure

In order to be able to prove that the compilation function τ_1 is correct, we first show that each model of a rough program $\mathcal{P}$ corresponds to a model of $\tau_1(\mathcal{P})$, and vice-versa.

We start by defining a bijective function that maps each model of a rough program into a model of an extended logic program.

Definition 4.11 *Let $\mathcal{I}$ be a rough interpretation of a rough program. Then, $\varphi(\mathcal{I})$ is a interpretation of an extended logic program defined as follows.*

(i) If $\langle c_1, \ldots, c_n \rangle \in \overline{Q^{\mathcal{I}}}$ then $q(c_1, \ldots, c_n) \in \varphi(\mathcal{I})$.

(ii) If $\langle c_1, \ldots, c_n \rangle \in \overline{\neg Q^{\mathcal{I}}}$ then $\neg q(c_1, \ldots, c_n) \in \varphi(\mathcal{I})$.

(iii) $\varphi(\mathcal{I})$ is the smallest set (with respect to set inclusion) satisfying both conditions(i) and (ii).

Lemma 4.2 *Let $\mathcal{I}$ be a rough interpretation of a rough program. Then, $\varphi^{-1}(\varphi(\mathcal{I})) = \mathcal{I}$.*

Proof: Note that φ is a bijection, i.e. it is a surjection and an injection. □

Lemma 4.3 *Let $\mathcal{I}_1$ and $\mathcal{I}_2$ be two rough interpretations of a rough program. Then, $\mathcal{I}_1 \preceq \mathcal{I}_2$ if and only if $\varphi(\mathcal{I}_1) \subseteq \varphi(\mathcal{I}_2)$.*

Proof: The statement $\mathcal{I}_1 \preceq \mathcal{I}_2 \Leftrightarrow \varphi(\mathcal{I}_1) \subseteq \varphi(\mathcal{I}_2)$ can be easily proved by taking into account the definition of function φ and the definition of partial relation $\preceq$.

□

Lemma 4.4 *Let $B_1, \ldots, B_n$ $(n \geq 1)$ be rough or testing literals. Assume also that $\mathcal{I}$ is a rough interpretation.*

- *$\mathcal{I} \models B_1, \ldots, B_n$ if and only if $\varphi(\mathcal{I}) \models \tau_2(B_1, \ldots, B_n)$.*

Proof: The prove can be simply done by structural induction. We start with the base case.

(i.1) First, we show that if $\mathcal{I} \models B$ then $\varphi(\mathcal{I}) \models \tau_2(B)$. Assume that $\mathcal{I} \models B$ and let us consider the different possibilities for B.

- $B \equiv \overline{q}(c_1, \ldots, c_n)$. Thus, $\langle c_1, \ldots, c_n \rangle \in \overline{Q^{\mathcal{I}}}$ and, by definition 4.11, $q(c_1, \ldots, c_n) \in \varphi(\mathcal{I})$. Since $\tau_2(\overline{q}(c_1, \ldots, c_n)) = q(c_1, \ldots, c_n)$, we conclude that $\varphi(\mathcal{I}) \models \tau_2(B)$.
- $B \equiv \overline{\neg q}(c_1, \ldots, c_n)$. The argument is similar to the previous case.
- $B \equiv \underline{q}(c_1, \ldots, c_n)$. Thus, $\langle c_1, \ldots, c_n \rangle \in \overline{Q^{\mathcal{I}}}$ and $\langle c_1, \ldots, c_n \rangle \notin \overline{\neg Q^{\mathcal{I}}}$. By definition 4.11, $q(c_1, \ldots, c_n) \in \varphi(\mathcal{I})$ and $\neg q(c_1, \ldots, c_n) \notin \varphi(\mathcal{I})$. We have then that

  $$\varphi(\mathcal{I}) \models q(c_1, \ldots, c_n), not\ \neg q(c_1, \ldots, c_n) .$$

  Since

  $$\tau_2(\underline{q}(c_1, \ldots, c_n)) = q(c_1, \ldots, c_n), not\ \neg q(c_1, \ldots, c_n) ,$$

  we conclude that $\varphi(\mathcal{I}) \models \tau_2(B)$.
- $B \equiv \underline{\neg q}(c_1, \ldots, c_n)$. The argument is similar to the previous case.
- $B \equiv \underline{\overline{q}}(c_1, \ldots, c_n)$. Thus, $\langle c_1, \ldots, c_n \rangle \in \overline{Q^{\mathcal{I}}}$ and $\langle c_1, \ldots, c_n \rangle \in \overline{\neg Q^{\mathcal{I}}}$. By definition 4.11, $q(c_1, \ldots, c_n) \in \varphi(\mathcal{I})$ and $\neg q(c_1, \ldots, c_n) \in \varphi(\mathcal{I})$. We have then that

  $$\varphi(\mathcal{I}) \models q(c_1, \ldots, c_n), \neg q(c_1, \ldots, c_n) .$$

  Since

  $$\tau_2(\underline{\overline{q}}(c_1, \ldots, c_n)) = q(c_1, \ldots, c_n), \neg q(c_1, \ldots, c_n) ,$$

  we conclude that $\varphi(\mathcal{I}) \models \tau_2(B)$.
- $B \equiv \underline{\overline{\neg q}}(c_1, \ldots, c_n)$. The argument of the previous case applies to this case, too.
- $B \equiv q?(c_1, \ldots, c_n)$. Thus, $\langle c_1, \ldots, c_n \rangle \notin \overline{Q^{\mathcal{I}}}$ and $\langle c_1, \ldots, c_n \rangle \notin \overline{\neg Q^{\mathcal{I}}}$. By definition 4.11, $q(c_1, \ldots, c_n) \notin \varphi(\mathcal{I})$ and $\neg q(c_1, \ldots, c_n) \notin \varphi(\mathcal{I})$. We have then that

  $$\varphi(\mathcal{I}) \models not\ q(c_1, \ldots, c_n), not\ \neg q(c_1, \ldots, c_n) .$$

  Since $\tau_2(\underline{q}(c_1, \ldots, c_n)) = not\ q(c_1, \ldots, c_n), not\ \neg q(c_1, \ldots, c_n)$, we conclude that $\varphi(\mathcal{I}) \models \tau_2(B)$.

(i.2) Second, we show that if $\varphi(\mathcal{I}) \models \tau_2(B)$ then $\mathcal{I} \models B$. Assume that $\varphi(\mathcal{I}) \models \tau_2(B)$ and let us consider the different possibilities for B.

- $B = \overline{q}(c_1, \ldots, c_n)$. Since $\tau_2(\overline{q}(c_1, \ldots, c_n)) = q(c_1, \ldots, c_n)$ and $\varphi(\mathcal{I}) \models q(c_1, \ldots, c_n)$, we conclude that $q(c_1, \ldots, c_n) \in \varphi(\mathcal{I})$. By definition 4.11, we have that $\langle c_1, \ldots, c_n \rangle \in \overline{Q^{\mathcal{I}}}$. Hence, $\mathcal{I} \models B$.
- $B = \overline{\neg q}(c_1, \ldots, c_n)$. The argument is similar to the previous case.
- $B = \underline{q}(c_1, \ldots, c_n)$. Since

$$\tau_2(\overline{q}(c_1, \ldots, c_n)) = q(c_1, \ldots, c_n), not\ \neg q(c_1, \ldots, c_n)$$

  and $\varphi(\mathcal{I}) \models q(c_1, \ldots, c_n), not\ \neg q(c_1, \ldots, c_n)$, we conclude that $q(c_1, \ldots, c_n) \in \varphi(\mathcal{I})$ and $\neg q(c_1, \ldots, c_n) \notin \varphi(\mathcal{I})$. By definition 4.11, we have that $\langle c_1, \ldots, c_n \rangle \in \overline{Q^{\mathcal{I}}}$ and $\langle c_1, \ldots, c_n \rangle \notin \overline{\neg Q^{\mathcal{I}}}$. Hence, $\mathcal{I} \models B$.
- $B = \underline{\neg q}(c_1, \ldots, c_n)$. The argument is similar to the previous case.
- $B \equiv \overline{q}(c_1, \ldots, c_n)$. Then by definition of τ_2, we have that $\varphi(\mathcal{I}) \models q(c_1, \ldots, c_n), \neg q(c_1, \ldots, c_n)$. Moreover by definition 4.11, we also have that $\langle c_1, \ldots, c_n \rangle \in \overline{Q^{\mathcal{I}}}$ and $\langle c_1, \ldots, c_n \rangle \in \overline{\neg Q^{\mathcal{I}}}$. We can then conclude that $\mathcal{I} \models B$.
- $B \equiv \overline{\neg q}(c_1, \ldots, c_n)$. This case is equal to the previous one.
- $B \equiv q?(c_1, \ldots, c_n)$. Then by definition of τ_2, we have that $\varphi(\mathcal{I}) \not\models q(c_1, \ldots, c_n)$ and $\varphi(\mathcal{I}) \not\models \neg q(c_1, \ldots, c_n)$. Moreover by definition 4.11, we also have that $\langle c_1, \ldots, c_n \rangle \notin \overline{Q^{\mathcal{I}}}$ and $\langle c_1, \ldots, c_n \rangle \notin \overline{\neg Q^{\mathcal{I}}}$. We can then conclude that $\mathcal{I} \models B$.

We proceed now to the inductive step.

(ii.1) First, we prove that if $\mathcal{I} \models B_1, \ldots, B_n$ then $\varphi(\mathcal{I}) \models \tau_2(B_1, \ldots, B_n)$. Note that the base case ($n = 1$) has been proved in (i.1). If $\mathcal{I} \models B_1, \ldots, B_n$ then $\mathcal{I} \models B_1, \ldots, \mathcal{I} \models B_n$. Consequently, by inductive hypothesis, $\varphi(\mathcal{I}) \models \tau_2(B_1), \ldots, \varphi(\mathcal{I}) \models \tau_2(B_n)$. The same is to say that $\varphi(\mathcal{I}) \models \tau_2(B_1), \ldots, \tau_2(B_n)$. By definition of τ_2, $\varphi(\mathcal{I}) \models \tau_2(B_1, \ldots, B_n)$.

(ii.2) Second, we show that if $\varphi(\mathcal{I}) \models \tau_2(B_1, \ldots, B_n)$ then $\mathcal{I} \models B_1, \ldots, B_n$. Note that the base case ($n = 1$) has been proved in (i.2). If $\varphi(\mathcal{I}) \models \tau_2(B_1, \ldots, B_n)$ then, by definition of τ_2, $\varphi(\mathcal{I}) \models \tau_2(B_1), \ldots, \tau_2(B_n)$. This is the same as $\varphi(\mathcal{I}) \models \tau_2(B_1), \ldots, \varphi(\mathcal{I}) \models \tau_2(B_n)$. By inductive hypothesis, $\mathcal{I} \models B_1, \ldots, \mathcal{I} \models B_n$ and, consequently, $\mathcal{I} \models B_1, \ldots, B_n$.

Lemma 4.5 *Let $\mathcal{P}$ be a rough program and $\mathcal{P}' = \tau_1(\mathcal{P})$. If $\mathcal{M}_{\mathcal{P}'}$ is a model of $\mathcal{P}'$ then $\varphi^{-1}(\mathcal{M}_{\mathcal{P}'})$ is a model of $\mathcal{P}$.*

Proof: Assume that $\mathcal{M}_{\mathcal{P}} = \varphi^{-1}(\mathcal{M}_{\mathcal{P}'})$ and let us prove that $\mathcal{M}_{\mathcal{P}}$ is a model of $\mathcal{P}$. Hence, we need to show that $\mathcal{M}_{\mathcal{P}}$ satisfies each rough clause H :- B. $\in$ *ground*($\mathcal{P}$). If $\mathcal{M}_{\mathcal{P}} \not\models B$ then $\mathcal{M}_{\mathcal{P}}$ trivially satisfies H :- B. . Otherwise, let us assume that $\mathcal{M}_{\mathcal{P}} \models B$. The head H of the rough clause can be one of the rough literals:

(i) $H \equiv \overline{q}(c_1, \ldots, c_n)$. Then, the compilation function τ_1 ensures that $q(c_1, \ldots, c_n)$:- $\tau_2(B)$. $\in \mathcal{P}'$. By lemma 4.4, $\mathcal{M}_{\mathcal{P}'} \models \tau_2(B)$. Consequently, $\mathcal{M}_{\mathcal{P}'} \models q(c_1, \ldots, c_n)$ because $\mathcal{M}_{\mathcal{P}'}$ is a model of $\mathcal{P}'$. By definition of function φ, we can conclude that $\langle c_1, \ldots, c_n \rangle \in \overline{Q^{\mathcal{M}_{\mathcal{P}}}}$. Hence, $\mathcal{M}_{\mathcal{P}} \models \overline{q}(c_1, \ldots, c_n)$.

(ii) $H \equiv \overline{\neg q}(c_1, \ldots, c_n)$. The argument is similar to case (i).

(iii) $H \equiv \underline{q}(c_1, \ldots, c_n)$. Then, the compilation function τ_1 ensures that

$$\{q(c_1, \ldots, c_n) \text{ :- } \tau_2(B). \ , \ \text{ :- } \neg q(c_1, \ldots, c_n), \tau_2(B).\} \subseteq \mathcal{P}' \ .$$

By lemma 4.4, $\mathcal{M}_{\mathcal{P}'} \models \tau_2(B)$. Consequently, $\mathcal{M}_{\mathcal{P}'} \models q(c_1, \ldots, c_n)$ and $\mathcal{M}_{\mathcal{P}'} \not\models \neg q(c_1, \ldots, c_n)$ because $\mathcal{M}_{\mathcal{P}'}$ is a model of $\mathcal{P}'$. By definition of function φ, we can conclude that $\langle c_1, \ldots, c_n \rangle \in \overline{Q^{\mathcal{M}_{\mathcal{P}}}}$ but $\langle c_1, \ldots, c_n \rangle \notin \overline{\neg Q^{\mathcal{M}_{\mathcal{P}}}}$. Hence, $\mathcal{M}_{\mathcal{P}} \models \underline{q}(c_1, \ldots, c_n)$.

(iv) $H \equiv \underline{\neg q}(c_1, \ldots, c_n)$. The argument is similar to case (iii).

(v) $H \equiv \underline{\overline{q}}(c_1, \ldots, c_n)$. Then, the compilation function τ_1 ensures that

$$\{q(c_1, \ldots, c_n) \text{ :- } \tau_2(B). \ , \ \neg q(c_1, \ldots, c_n) \text{ :- } \tau_2(B).\} \subseteq \mathcal{P}' \ .$$

By lemma 4.4, $\mathcal{M}_{\mathcal{P}'} \models \tau_2(B)$. Consequently, $\mathcal{M}_{\mathcal{P}'} \models q(c_1, \ldots, c_n)$ and $\mathcal{M}_{\mathcal{P}'} \models \neg q(c_1, \ldots, c_n)$ because $\mathcal{M}_{\mathcal{P}'}$ is a model of $\mathcal{P}'$. By definition of function φ, we can conclude that $\langle c_1, \ldots, c_n \rangle \in \overline{Q^{\mathcal{M}_{\mathcal{P}}}}$ and $\langle c_1, \ldots, c_n \rangle \in \overline{\neg Q^{\mathcal{M}_{\mathcal{P}}}}$. Hence, $\mathcal{M}_{\mathcal{P}} \models \underline{\overline{q}}(c_1, \ldots, c_n)$.

(vi) $H \equiv \underline{\overline{\neg q}}(c_1, \ldots, c_n)$. This case is equivalent to case (v).

□

Lemma 4.6 *Let $\mathcal{P}$ be a rough program and $\mathcal{P}' = \tau_1(\mathcal{P})$. If $\mathcal{M}_{\mathcal{P}}$ is a model of $\mathcal{P}$ then $\varphi(\mathcal{M}_{\mathcal{P}})$ is a model of $\mathcal{P}'$.*

Proof: Assume that $\mathcal{M}_\mathcal{P}$ is a model of $\mathcal{P}$. Hence, we need to show that $\varphi(\mathcal{M}_\mathcal{P})$ satisfies each clause H :- B_1. $\in ground(\mathcal{P}')$ and each integrity constraint :- B_2. $\in ground(\mathcal{P}')$. Note that clauses and integrity constraints belonging to $ground(\mathcal{P}')$ can only have some particular forms determined by the compilation function τ_1. Let us then consider each possible case of function τ_1.

(i) Assume that the rough clause $\overline{q}(c_1,\ldots,c_n)$:- B. $\in \mathcal{P}$. Since $\mathcal{M}_\mathcal{P}$ satisfies $\overline{q}(c_1,\ldots,c_n)$:- B. , we have either that $\mathcal{M}_\mathcal{P} \not\models B$ or $\mathcal{M}_\mathcal{P} \models B, \overline{q}(c_1,\ldots,c_n)$. If $\mathcal{M}_\mathcal{P} \not\models B$ then, by lemma 4.4, $\varphi(\mathcal{M}_\mathcal{P}) \not\models \tau_2(B)$, and consequently, $\varphi(\mathcal{M}_\mathcal{P})$ trivially satisfies the clause in $\tau_1(\overline{q}(c_1,\ldots,c_n)$:- $B.) \subseteq \mathcal{P}'$. Otherwise, $\mathcal{M}_\mathcal{P} \models B, \overline{q}(c_1,\ldots,c_n)$ and, by lemma 4.4, $\varphi(\mathcal{M}_\mathcal{P}) \models \tau_2(B)$. Moreover by definition 4.11, $q(c_1,\ldots,c_n) \in \varphi(\mathcal{M}_\mathcal{P})$.
Hence, $\varphi(\mathcal{M}_\mathcal{P})$ satisfies the clause in $\tau_1(\overline{q}(c_1,\ldots,c_n)$:- $B.)$.

(ii) Assume that the rough clause $\overline{\neg q}(c_1,\ldots,c_n)$:- B. $\in \mathcal{P}$. This case can be justified in a way similar to the previous one.

(iii) Assume that the rough clause $\underline{q}(c_1,\ldots,c_n)$:- B. $\in \mathcal{P}$. Since $\mathcal{M}_\mathcal{P}$ satisfies $\underline{q}(c_1,\ldots,c_n)$:- B. , we have either that $\mathcal{M}_\mathcal{P} \not\models B$ or $\mathcal{M}_\mathcal{P} \models B, \underline{q}(c_1,\ldots,c_n)$. If $\mathcal{M}_\mathcal{P} \not\models B$ then, by lemma 4.4, $\varphi(\mathcal{M}_\mathcal{P}) \not\models \tau_2(B)$, and consequently, $\varphi(\mathcal{M}_\mathcal{P})$ satisfies the clause and the integrity constraint in $\tau_1(\underline{q}(c_1,\ldots,c_n)$:- $B.) \subseteq \mathcal{P}'$. Otherwise, $\mathcal{M}_\mathcal{P} \models B$ and $\mathcal{M}_\mathcal{P} \models \underline{q}(c_1,\ldots,c_n)$. By lemma 4.4, $\varphi(\mathcal{M}_\mathcal{P}) \models \tau_2(B)$ and by definition of $\models$, $\langle c_1,\ldots,c_n\rangle \in \overline{Q^{\mathcal{M}_\mathcal{P}}}$ and $\langle c_1,\ldots,c_n\rangle \notin \overline{\neg Q^{\mathcal{M}_\mathcal{P}}}$. By definition 4.11, we have that $q(c_1,\ldots,c_n) \in \varphi(\mathcal{M}_\mathcal{P})$ but $\neg q(c_1,\ldots,c_n) \notin \varphi(\mathcal{M}_\mathcal{P})$. Hence, $\varphi(\mathcal{M}_\mathcal{P})$ satisfies the clause $q(c_1,\ldots,c_n)$:- $\tau_2(B)$. and the integrity constraint
:- $\neg q(c_1,\ldots,c_n), \tau_2(B)$. obtained by compiling the rough clause.

(iv) Assume that the rough clause $\underline{\neg q}(c_1,\ldots,c_n)$:- B. $\in \mathcal{P}$. An argument similar to the previous case can be also used here.

(v) Assume that the rough clause $\overline{\underline{q}}(c_1,\ldots,c_n)$:- B. $\in \mathcal{P}$. Since $\mathcal{M}_\mathcal{P}$ satisfies $\overline{\underline{q}}(c_1,\ldots,c_n)$:- B. , we have either that $\mathcal{M}_\mathcal{P} \not\models B$ or $\mathcal{M}_\mathcal{P} \models B, \overline{\underline{q}}(c_1,\ldots,c_n)$. If $\mathcal{M}_\mathcal{P} \not\models B$ then, by lemma 4.4, $\varphi(\mathcal{M}_\mathcal{P}) \not\models \tau_2(B)$, and consequently, $\varphi(\mathcal{M}_\mathcal{P})$ satisfies both clauses in $\tau_1(\overline{\underline{q}}(c_1,\ldots,c_n)$:- $B.) \subseteq \mathcal{P}'$. Otherwise, $\mathcal{M}_\mathcal{P} \models B$ and $\mathcal{M}_\mathcal{P} \models \overline{\underline{q}}(c_1,\ldots,c_n)$. By lemma 4.4, $\varphi(\mathcal{M}_\mathcal{P}) \models \tau_2(B)$ and by definition of $\models$, $\langle c_1,\ldots,c_n\rangle \in \overline{Q^{\mathcal{M}_\mathcal{P}}}$ and $\langle c_1,\ldots,c_n\rangle \in \overline{\neg Q^{\mathcal{M}_\mathcal{P}}}$. By definition 4.11, we have that $q(c_1,\ldots,c_n) \in \varphi(\mathcal{M}_\mathcal{P})$ and $\neg q(c_1,\ldots,c_n) \in$

$\varphi(\mathcal{M}_\mathcal{P})$. Thus, $\varphi(\mathcal{M}_\mathcal{P})$ satisfies both clauses $q(c_1,\ldots,c_n)\texttt{:-}\ \tau_2(B).$ and $\neg q(c_1,\ldots,c_n)\texttt{:-}\ \tau_2(B).$ obtained by compiling the rough clause.

(vi) Assume that the rough clause $\overline{\neg \underline{q}}(c_1,\ldots,c_n)\texttt{:-}\ B. \in \mathcal{P}$. This case is equivalent to the previous one.

□

Lemma 4.7 *Let $\mathcal{P}$ be a rough program. $\mathcal{M}$ is the least model (with respect to $\preceq$) of $\mathcal{P}$ if and only if $\varphi(\mathcal{M})$ is the least model (with respect to $\subseteq$) of $\tau_1(\mathcal{P})$.*

Proof: This lemma is direct consequence of lemmas 4.3, 4.5, and 4.6.

□

Lemma 4.8 *Let $\mathcal{P}$ be a rough program and $\mathcal{M}$ be one of its models. Then,*

$$\psi_{\varphi(\mathcal{M})}(\tau_1(\mathcal{P})) = \{H\texttt{:-}\ B. \in \tau_1(\Psi_\mathcal{M}(\mathcal{P}))\}\ .$$

Proof: To simplify the presentation of this proof, we represent (rough) predicate argument tuples as $\overrightarrow{t}$, i.e. $q(t_1,\ldots,t_n)$ is represented as $q(\overrightarrow{t})$.

(i) We assume that

$$q(\overrightarrow{t})\ \texttt{:-}\ B. \in \tau_1(\Psi_\mathcal{M}(\mathcal{P}))\ .$$

and then show that $q(\overrightarrow{t})\ \texttt{:-}\ B. \in \psi_{\varphi(\mathcal{M})}(\tau_1(\mathcal{P}))$.

From the hypotheses, it follows that one of the rough clauses below belongs to $\Psi_\mathcal{M}(\mathcal{P})$.

$$\begin{array}{l} \overline{q}(\overrightarrow{t})\ \texttt{:-}\ B'. \\ \underline{q}(\overrightarrow{t})\ \texttt{:-}\ B'. \\ \overline{\underline{q}}(\overrightarrow{t})\ \texttt{:-}\ B'.\ , \end{array}$$

where B' is such that $\tau_2(B') = B$.

Assume that $\overline{q}(\overrightarrow{t})\ \texttt{:-}\ B'. \in \Psi_\mathcal{M}(\mathcal{P})$ (the other two cases follow a similar reasoning). By definition of $\Psi_\mathcal{M}$, there is a rough clause

$$\overline{q}(\overrightarrow{t})\ \texttt{:-}\ B_1, \underline{r_1}(\overrightarrow{t_1}),\ldots,\underline{r_k}(\overrightarrow{t_k}), s_1?(\overrightarrow{t'_1}),\ldots,s_m?(\overrightarrow{t'_m}). \in \mathit{ground}(\mathcal{P})\ ,$$

with $k, m \geq 0$,

$$\begin{array}{ll} \mathcal{M} \not\models \overline{\neg r_1}(\overrightarrow{t_1}),\ldots,\overline{\neg r_k}(\overrightarrow{t_k}) & (1) \\ \mathcal{M} \models s_1?(\overrightarrow{t'_1}),\ldots,r_k?(\overrightarrow{t'_k}) & (2) \end{array}$$

and $B' \equiv B_1, \overline{r_1}(\overrightarrow{t_1}), \ldots, \overline{r_k}(\overrightarrow{t_k})$. Consequently,

$$q(\overrightarrow{t}) \text{ :- } \tau_2(B_1), r_1(\overrightarrow{t_1}),\ not\ \neg r_1(\overrightarrow{t_1}) \ldots, r_k(\overrightarrow{t_k}),\ not\ r_k(\overrightarrow{t_1 k}),$$
$$not\ s_1(\overrightarrow{t'_1}),\ not\ \neg s_1(\overrightarrow{t'_1}) \ldots,\ not\ \neg s_m(\overrightarrow{t'_m}), \neg s_m(\overrightarrow{t'_m}).$$

belongs to $\tau_1(ground(\mathcal{P}))$. Moreover, we conclude from (1) that

$$\varphi(\mathcal{M}) \models\ not\ \neg r_1(\overrightarrow{t_1}), \ldots,\ not\ \neg r_k(\overrightarrow{t_k})$$

and (2) implies that

$$\varphi(\mathcal{M}) \models\ not\ s_1(\overrightarrow{t'_1}),\ not\ \neg s_1(\overrightarrow{t'_1}) \ldots,\ not\ s_m(\overrightarrow{t'_m}),\ not\ \neg s_m(\overrightarrow{t'_m})\ .$$

But then,

$$q(\overrightarrow{t}) \text{ :- } \tau_2(B_1), r_1(\overrightarrow{t_1}), \ldots, r_k(\overrightarrow{t_k}). \in \psi_{\varphi(\mathcal{M})}(\tau_1(\mathcal{P}))\ .$$

Note that $\psi_{\varphi(\mathcal{M})}(\tau_1(ground(\mathcal{P}))) = \psi_{\varphi(\mathcal{M})}(\tau_1(\mathcal{P}))$.
Since $(\tau_2(B_1), r_1(\overrightarrow{t_1}), \ldots, r_k(\overrightarrow{t_k})) = \tau_2(B') = B$, we can conclude that $q(\overrightarrow{t}) \text{ :- } B. \in \psi_{\varphi(\mathcal{M})}(\tau_1(\mathcal{P}))$.

(ii) We now assume

$$q(\overrightarrow{t}) \text{:- } B. \in \psi_{\varphi(\mathcal{M})}(\tau_1(\mathcal{P}))$$

and then show that $q(\overrightarrow{t}) \text{:- } B. \in \tau_1(\Psi_{\mathcal{M}}(\mathcal{P}))$.
From the hypotheses, it follows that there is a clause

$$q(\overrightarrow{t}) \text{ :- } B,\ not\ \neg r_1(\overrightarrow{t_1}), \ldots,\ not\ \neg r_k(\overrightarrow{t_k}),$$
$$not\ s_1(\overrightarrow{t'_1}),\ not\ \neg s_1(\overrightarrow{t'_1}), \ldots,\ not\ s_m(\overrightarrow{t'_m}),\ not\ \neg s_m(\overrightarrow{t'_m}).$$

belonging to $\tau_1(ground(\mathcal{P}))$, with $m, k \geq 0$, and

$$\varphi(\mathcal{M}) \models\ not\ \neg r_1(\overrightarrow{t_1}), \ldots,\ not\ \neg r_k(\overrightarrow{t_k}) \quad (3)$$
$$\varphi(\mathcal{M}) \models\ not\ s_1(\overrightarrow{t'_1}),\ not\ \neg s_1(\overrightarrow{t'_1}), \ldots,$$
$$not\ s_m(\overrightarrow{t'_m}),\ not\ \neg s_m(\overrightarrow{t'_m}) \quad (4)$$

Moreover, $B \equiv B', r_1(\overrightarrow{t_1}), \ldots, r_k(\overrightarrow{t_k})$. One of the following rough clauses has then to belong to $ground(\mathcal{P})$.

$$\overline{q}(\overrightarrow{t}) \text{ :- } B_1, \underline{r_1}(\overrightarrow{t_1}), \ldots, \underline{r_k}(\overrightarrow{t_k}), s_1?(\overrightarrow{t'_1}), \ldots, s_m?(\overrightarrow{t'_m}).$$
$$\underline{q}(\overrightarrow{t}) \text{ :- } B_1, \underline{r_1}(\overrightarrow{t_1}), \ldots, \underline{r_k}(\overrightarrow{t_k}), s_1?(\overrightarrow{t'_1}), \ldots, s_m?(\overrightarrow{t'_m}).$$
$$\overline{\underline{q}}(\overrightarrow{t}) \text{ :- } B_1, \underline{r_1}(\overrightarrow{t_1}), \ldots, \underline{r_k}(\overrightarrow{t_k}), s_1?(\overrightarrow{t'_1}), \ldots, s_m?(\overrightarrow{t'_m}).\ ,$$

where $\tau_2(B_1) = B'$.

Assume that

$$\overline{q}(\overrightarrow{t}) \text{ :- } B_1, \underline{r_1}(\overrightarrow{t_1}), \ldots, \underline{r_k}(\overrightarrow{t_k}), s_1?(\overrightarrow{t'_1}), \ldots, s_m?(\overrightarrow{t'_m}). \in ground(\mathcal{P})$$

(the other two cases follow a similar reasoning). We can conclude from (3) that

$$\mathcal{M} \not\models \underline{\neg r_1}(\overrightarrow{t_1}), \ldots, \underline{\neg r_k}(\overrightarrow{t_k})$$

and (4) implies that

$$\mathcal{M} \models s_1?(\overrightarrow{t'_1}), \ldots, s_m?(\overrightarrow{t'_m}) \ .$$

We have then that $\overline{q}(\overrightarrow{t}) \text{ :- } B_1, \overline{r_1}(\overrightarrow{t_1}), \ldots, \overline{r_k}(\overrightarrow{t_k}). \in \Psi_{\mathcal{M}}(\mathcal{P})$. Consequently,

$$q(\overrightarrow{t}) \text{ :- } B', r_1(\overrightarrow{t_1}), \ldots, r_k(\overrightarrow{t_k}). \in \tau_1(\Psi_{\mathcal{M}}(\mathcal{P})) \ .$$

Hence, $q(\overrightarrow{t}) \text{ :- } B. \in \tau_1(\Psi_{\mathcal{M}}(\mathcal{P}))$.

□

Lemma 4.9 *Let $\mathcal{P}$ be a rough program. If $\mathcal{M}$ is the least model of $\Psi_{\mathcal{M}}(\mathcal{P})$ then $\varphi(\mathcal{M})$ is the least model of $\{H \text{ :- } B. \in \tau_1(\Psi_{\mathcal{M}}(\mathcal{P}))\}$.*

Proof: Let $\mathcal{P}' = \{H \text{ :- } B. \in \tau_1(\Psi_{\mathcal{M}}(\mathcal{P}))\}$. If $\mathcal{M}$ is the least model of $\Psi_{\mathcal{M}}(\mathcal{P})$ then, by lemma 4.7, $\varphi(\mathcal{M})$ is the least model of $\tau_1(\Psi_{\mathcal{M}}(\mathcal{P}))$ and, therefore, it is a model of $\mathcal{P}'$.

Note that default negated literals cannot occur in any clause or integrity constraint of $\tau_1(\Psi_{\mathcal{M}}(\mathcal{P}))$. The reason is that neither lower approximations nor testing literals occur in the body of any rough clause in $\Psi_{\mathcal{M}}(\mathcal{P})$. In addition, $\tau_1(\Psi_{\mathcal{M}}(\mathcal{P}))$ is definite logic program with integrity constraints.

Assume that $\varphi(\mathcal{M})$ is not the least model of $\mathcal{P}'$. Then, there is a model (e.g. the least model) $\mathcal{M}'$ of $\mathcal{P}'$ such that $\mathcal{M}' \prec \varphi(\mathcal{M})$. Therefore, there is one atom $q(t_1, \ldots, t_n)$ (or $\neg q(t_1, \ldots, t_n)$) such that $q(t_1, \ldots, t_n) \in \varphi(\mathcal{M})$ but $q(t_1, \ldots, t_n) \notin \mathcal{M}'$. Only the occurrence of a default negated literal, e.g. *not* $q(t_1, \ldots, t_n)$, in an integrity constraint belonging to $\tau_1(\Psi_{\mathcal{M}}(\mathcal{P}))$ could force the entrance of an atom in the least model of $\mathcal{P}'$ in order to be also able to satisfy the integrity constraints. However, as we pointed out above, this cannot be the case.

□

Lemma 4.10 *Let $\mathcal{P}$ be a rough program and $\mathcal{I}$ be a rough interpretation of $\mathcal{P}$. If $\varphi(\mathcal{I})$ satisfies each integrity constraint in $\tau_1(\mathcal{P})$ then $\varphi(\mathcal{I})$ satisfies each integrity constraint in $\tau_1(\Psi_{\mathcal{I}}(\mathcal{P}))$.*

Proof: To simplify the presentation of this proof, we represent predicate argument tuples as $\overrightarrow{t}$, i.e. $q(t_1, \ldots, t_n)$ is represented as $q(\overrightarrow{t})$.

Assume that $\varphi(\mathcal{I})$ satisfies each integrity constraint in $\tau_1(\mathcal{P})$. Taking into account the definition of τ_1, we conclude that integrity constraints originate from compilation of rough clauses with a lower approximation in their head. Hence, suppose that

$$\underline{q}(\overrightarrow{t}) \text{ :- } B_1, \underline{r_1}(\overrightarrow{t_1}), \ldots, \underline{r_k}(\overrightarrow{t_k}), s_1?(\overrightarrow{t'_1}), \ldots, s_m?(\overrightarrow{t'_m}). \in \mathcal{P} \ ,$$

where $k, m \geq 0$ and no lower approximations or testing literals occur in B_1. Let i_c be an integrity constraint defined as follows.

$$\begin{aligned} i_c = \text{:- } & \neg q(\overrightarrow{t}), \tau_2(B_1), r_1(\overrightarrow{t_1}),\ not\ \neg r_1(\overrightarrow{t_1}), \ldots, r_k(\overrightarrow{t_k}),\ not\ \neg r_k(\overrightarrow{t_k}), \\ & not\ s_1(\overrightarrow{t'_1}),\ not\ \neg s_1(\overrightarrow{t'_1}), \ldots, not\ s_m(\overrightarrow{t'_m}),\ not\ \neg s_m(\overrightarrow{t'_m}). \end{aligned}$$

Then, $i_c \in \tau_1(\mathcal{P})$ and $\varphi(\mathcal{I}) \models i_c$.

Assume also that

(a) $\mathcal{I} \not\models \overline{\neg r_1}(\overrightarrow{t_1}), \ldots, \overline{\neg r_k}(\overrightarrow{t_k})$ and

(b) $\mathcal{I} \models s_1?(\overrightarrow{t'_1}), \ldots, s_m?(\overrightarrow{t'_m})$.

From (a) and (b) and by definition of $\Psi_{\mathcal{I}}$,

$$\underline{q}(\overrightarrow{t}) \text{ :- } B_1, \overline{r_1}(\overrightarrow{t_1}), \ldots, \overline{r_k}(\overrightarrow{t_k}). \in \Psi_{\mathcal{M}}(\mathcal{P}) \ .$$

Thus, $i'_c \in \tau_1(\Psi_{\mathcal{I}}(\mathcal{P}))$, with

$$i'_c = \neg q(\overrightarrow{t}), \tau_2(B_1), r_1(\overrightarrow{t_1}), \ldots, r_k(\overrightarrow{t_k}). \in \tau_1(\Psi_{\mathcal{I}}(\mathcal{P})) \ .$$

Moreover,

- from (a), we have that $\varphi(\mathcal{I}) \models\ not\ \neg r_1(\overrightarrow{t_1}), \ldots\ not\ \neg r_k(\overrightarrow{t_k})$ and
- from (b), we have that

$$\varphi(\mathcal{I}) \models not\ s_1(\overrightarrow{t'_1}),\ not\ \neg s_1(\overrightarrow{t'_1}), \ldots, not\ s_m(\overrightarrow{t'_m}),\ not\ \neg s_m(\overrightarrow{t'_m}) \ .$$

Since $\varphi(\mathcal{I}) \models i_c$, we can conclude that $\varphi(\mathcal{I}) \models i'_c$.

□

Theorem 4.1 *Let $\mathcal{P}$ be a rough program and $\mathcal{P}' = \tau_1(\mathcal{P})$. Then, $\mathcal{M} \in sem(\mathcal{P})$ if and only if $\varphi(\mathcal{M})$ is a paraconsistent stable model of $\mathcal{P}'$.*

Proof:

(i) First, we prove that if $\mathcal{M} \in sem(\mathcal{P})$ then $\varphi(\mathcal{M})$ is a paraconsistent stable model of $\mathcal{P}'$.

Assume that $\mathcal{M} \in sem(\mathcal{P})$. Then, $\mathcal{M}$ is the least model (with respect to $\preceq$) of $\Psi_{\mathcal{M}}(\mathcal{P})$ and, by lemma 4.9, $\varphi(\mathcal{M})$ is the least (with respect to $\subseteq$) model of $\{H \texttt{ :- } B. \in \tau_1(\Psi_{\mathcal{M}}(\mathcal{P}))\}$. Consequently, by lemma 4.8, $\varphi(\mathcal{M})$ is the least model of $\psi_{\varphi(\mathcal{M})}(\tau_1(\mathcal{P}))$. Moreover, $\varphi(\mathcal{M})$ satisfies each integrity constraint belonging to $\tau_1(\mathcal{P})$ because $\mathcal{M}$ is a model of $\mathcal{P}$ and, by lemma 4.6, $\varphi(\mathcal{M})$ is a model of $\tau_1(\mathcal{P})$. We can then conclude that $\varphi(\mathcal{M})$ is a paraconsistent stable model of $\tau_1(\mathcal{P})$.

(ii) Second, we show that if $\varphi(\mathcal{M})$ is a paraconsistent stable model of $\mathcal{P}'$ then $\mathcal{M} \in sem(\mathcal{P})$.

Assume that $\varphi(\mathcal{M})$ is a paraconsistent stable model of $\mathcal{P}'$. Then,

(a) $\varphi(\mathcal{M})$ satisfies all integrity constraints belonging to $\tau_1(\mathcal{P})$ and

(b) $\varphi(\mathcal{M})$ is the least model of $\psi_{\varphi(\mathcal{M})}(\tau_1(\mathcal{P}))$.

From (a) and lemma 4.10, we conclude that $\varphi(\mathcal{M})$ satisfies all integrity constraints in $\tau_1(\Psi_{\mathcal{M}}(\mathcal{P}))\}$. From (b) and lemma 4.8, we have that $\varphi(\mathcal{M})$ is the least model of $\{H \texttt{ :- } B. \in \tau_1(\Psi_{\mathcal{M}}(\mathcal{P}))\}$. These two facts lead us to the conclusion that $\varphi(\mathcal{M})$ has to be the least model of $\tau_1(\Psi_{\mathcal{M}}(\mathcal{P}))\}$. Then, by lemma 4.7, $\mathcal{M}$ is the least model of $\Psi_{\mathcal{M}}(\mathcal{P})$ and, therefore, $\mathcal{M} \in sem(\mathcal{P})$.

□

Theorem 4.2 (Correctness) *Let $\mathcal{P}$ be a rough program and $\mathcal{P}' = \tau_1(\mathcal{P})$ and l be a rough or testing literal. Then, $\mathcal{P} \models l$ if and only if $\mathcal{P}' \models \tau_2(l)$.*

Proof: This theorem is a direct consequence of theorem 4.1, of lemma 4.4, and of definitions 3.10 and 4.10.

4.4 Queries

This section proposes a query language for querying rough programs. This can be achieved by adapting existing systems based on the stable model

semantics [ELM+98, Sim]. Here, we only present queries and their expected answers. Since there might exist more than one model for a rough program $\mathcal{P}$, answers are computed with respect to one model of $sem(\mathcal{P})$. If a rough program has a unique model, which may often be the case[2], the answers will refer to this model.

Definition 4.12 *A* rough query *is a pair $(\mathcal{Q}, \mathcal{P})$, where $\mathcal{P}$ is a rough program and $\mathcal{Q}$ is defined by the following abstract syntax rules*

$$\begin{array}{lcl} \mathcal{Q}_1 & \longrightarrow & A? \mid A?, \mathcal{Q}_1 \,. \\ \mathcal{Q}_2 & \longrightarrow & L_1 \mid L_1, \mathcal{Q}_2 \mid \mathcal{Q}_2, \mathcal{Q}_1 \,. \\ \mathcal{Q}_3 & \longrightarrow & L_1 \subseteq L_2 \mid L_1 \subseteq L_2, \mathcal{Q}_3 \,. \\ \mathcal{Q} & \longrightarrow & \mathcal{Q}_1 \mid \mathcal{Q}_2 \mid \mathcal{Q}_3 \,. \end{array}$$

where $A?$ is a testing literal and each L_i $(i = 1, 2)$ is a rough literal. Moreover, a rough query is well-formed *if the following conditions are satisfied.*

(i) Any testing literal $A?$ is ground (i.e. it does not contain any variables) or its variables occur also in some rough literal of the query.

(ii) For an expression of the form $L_1 \subseteq L_2$ occurring in the query, any variable occurring in L_1 should also occur in L_2, and vice-versa.

For example, rough queries $(\underline{p}(X1, X2), q?(X2, X3), \mathcal{P})$ and $(\underline{p}(X1, X2, X3) \subseteq \overline{q}(X1, X2), \mathcal{P})$ are not well-formed because the former does not satisfy condition (i) and the latter violates condition (ii). In what follows, we always assume that rough queries are well-formed.

Consider the rough query $(\overline{q}(c_1, c_2), \mathcal{P})$. Before presenting the notion of answer, we explain informally what is being queried and the corresponding answer. With that query we want to know whether the tuple $\langle c_1, c_2 \rangle$ belongs to the boundary region of the rough relation denoted by q, in some model of $\mathcal{P}$ belonging to $sem(\mathcal{P})$. If the atom occurring in the query is not ground then, as answer, we may obtain a list of examples valid in a certain model. For example, the query $(\overline{q}(X, Y), \mathcal{P})$ requests a list of pairs that belong to $\overline{Q^{\mathcal{M}}}$, for some some model $\mathcal{M} \in sem(\mathcal{P})$.

We formalize now the notion of answer to a rough query.

Definition 4.13 *Let $(\mathcal{Q}, \mathcal{P})$ be a rough query.*

[2]For instance, any rough program whose rough clauses do not contain lower approximations or testing literals in their bodies either has a least model or no model at all.

(i) If $\mathcal{Q}$ is of the form $\mathcal{Q}_1$ or $\mathcal{Q}_2$ (see definition 4.12) then an answer *to the rough query is the set of ground substitutions*

$$\{\theta \mid \mathcal{Q}\theta \in ground(\mathcal{Q}) \text{ and } \mathcal{M} \models \mathcal{Q}\theta\} ,$$

for some model $\mathcal{M} \in sem(\mathcal{P})$.

(ii) If $\mathcal{Q}$ is of the form $L_{11} \subseteq L_{21}, \ldots, L_{1n} \subseteq L_{2n}$, where each L_{1i} and L_{2i} $(1 \leq i \leq n)$ are rough literals, then the answer *to the rough query is*

(ii.1) `yes` *, if there is a model $\mathcal{M} \in sem(\mathcal{P})$ such that*

$$\begin{array}{rcl} \mathcal{M} \models L_{11}\theta & \Rightarrow & \mathcal{M} \models L_{21}\theta \ , \\ & \vdots & \\ \mathcal{M} \models L_{1n}\theta & \Rightarrow & \mathcal{M} \models L_{2n}\theta \ , \end{array}$$

for every ground substitution θ;

(ii.2) `no` *, otherwise.*

Note that $\{\emptyset\}$ is a possible answer to a rough query. This answer can be obtained in case (i) when the query is ground and it should essentially be viewed as an affirmative answer. This contrasts with the empty set answer that should be interpreted as a negative answer. For instance, if the answer to the rough query $(\overline{q}(c_1, c_2), \mathcal{P})$ is $\emptyset$ then this means that $\langle c_1, c_2 \rangle$ does not belong to the upper approximation of rough relation Q (whatever is the model $\mathcal{M} \in sem(\mathcal{P})$ that is considered).

The notion of answer to a rough query introduced above is declarative. Hence, we need to discuss how such answers can be computed. A rough query $(\mathcal{Q}, \mathcal{P})$, where $\mathcal{Q}$ is of the form $\mathcal{Q}_1$ or $\mathcal{Q}_2$ (see definition 4.12), is translated into a query to the extended logic program $\tau_1(\mathcal{P})$

$$(\tau_2(\mathcal{Q}), \tau_1(\mathcal{P})) ,$$

and each set of substitutions obtained as answer to this query is also an answer to the rough query.

We now discuss how a query of the form $(L_1 \subseteq L_2, \mathcal{P})$, where L_1 and L_2 are rough literals, could be answered. The idea is to translate it to a set of integrity constraints that are added to the compiled program $(\tau_1(\mathcal{P}))$. Hence, a new extended logic program $\mathcal{P}'$ is obtained in this way. Then, the answer to the query is `yes` (i.e. the test succeeds) if $\mathcal{P}'$ has at least one paraconsistent stable model. Otherwise, the answer is `no` (i.e. the test

fails). Thus, we reduce the answering problem for this kind of queries to the problem of checking the existence of paraconsistent stable models of an extended logic program where certain properties, expressed by the integrity constraints, hold.

Given an objective literal L, we assume that $\neg\neg L$ and L have the same meaning. Moreover, consider the rough query $(L_1 \subseteq L_2, \mathcal{P})$, where L_1 and L_2 are rough literals. We define a function τ_3 that transforms these queries into an extended logic program with integrity constraints, for each possible case of L_2 (i.e. $\overline{L}, \underline{L}, \underline{\overline{L}}$).

$$\begin{array}{lcl}
\tau_3((L_1 \subseteq \overline{L}, \mathcal{P})) & = & \tau_1(\mathcal{P}) \cup \{\texttt{:-}\ \tau_2(L_1), not\ L.\}\ , \\
\tau_3((L_1 \subseteq \underline{L}, \mathcal{P})) & = & \tau_1(\mathcal{P}) \cup \{\texttt{:-}\ \tau_2(L_1), not\ L.\ , \texttt{:-}\ \tau_2(L_1), \neg L.\}\ , \\
\tau_3((L_1 \subseteq \underline{\overline{L}}, \mathcal{P})) & = & \tau_1(\mathcal{P}) \cup \{\texttt{:-}\ \tau_2(L_1), not\ L.\ , \texttt{:-}\ \tau_2(L_1), not\ \neg L.\}\ .
\end{array}$$

It is trivial to extend function τ_3 for compiling queries of the form $(\mathcal{Q}_3, \mathcal{P})$. Assume that $\mathcal{P}$ is a rough program, L_{1i} and L_{2i} are rough literals, with $1 \leq i \leq n$, then

$$\tau_3((L_{11} \subseteq L_{21}, \cdots, L_{1n} \subseteq L_{2n}, \mathcal{P})) = \bigcup_{1 \leq i \leq n} \tau_3((L_{1i} \subseteq L_{2i}, \mathcal{P}))\ .$$

Thus, given a rough program $\mathcal{P}$, we have that the answer to the query $(\mathcal{Q}_3, \mathcal{P})$ is `yes`, if the extended logic program $\tau_3((\mathcal{Q}_3, \mathcal{P}))$ has a paraconsistent stable model. Otherwise, the answer is `no`.

Example 4.10 *Consider again the rough program*

$$\begin{array}{lcl}
\mathcal{P} & = & \{\overline{diseaseA}(infect,\ normal,\ Z)\ \texttt{:-} \\
& & \qquad \underline{\neg diseaseB}(infect,\ normal,\ Z).\ , \\
& & \overline{diseaseB}(infect,\ normal,\ Z)\ \texttt{:-} \\
& & \qquad \underline{\neg diseaseA}(infect,\ normal,\ Z).\ , \\
& & \overline{\neg diseaseA}(infect,\ normal,\ Z).\ , \\
& & \overline{\neg diseaseB}(infect,\ normal,\ Z).\}\ .
\end{array}$$

and the queries

(i) $(\underline{\overline{diseaseA}}(X, Y, Z),\ \mathcal{P})$,

(ii) $(\overline{diseaseA}(X, Y, Z) \subseteq \underline{\neg diseaseB}(X, Y, Z),\ \mathcal{P})$.

Recall that there are two models that belong to sem($\mathcal{P}$), see example 4.6. The answer to query (i) is the set of substitutions

$$\{\{X/\texttt{infect}, Y/\texttt{normal}, Z/\texttt{c}\}\}$$

because in one of the models, $\mathcal{M}_1$, $\langle \texttt{infect}, \texttt{normal}, \texttt{c}\rangle \in \overline{DiseaseA^{\mathcal{M}_1}}$.

Note that

$$\tau_3(\overline{\texttt{diseaseA}}(X,Y,Z) \subseteq \underline{\texttt{diseaseB}}(X,Y,Z),\ \mathcal{P}) \ = \ \tau_1(\mathcal{P}) \cup$$
$$\{\texttt{:-}\ \tau_2(\overline{\texttt{diseaseA}}(X,Y,Z)), not\ diseaseB(X,Y,Z).\ ,$$
$$\texttt{:-}\ \tau_2(\overline{\texttt{diseaseA}}(X,Y,Z)), \neg diseaseB(X,Y,Z).\}$$

Since $\mathcal{M}_1$ *is a paraconsistent stable model of this extended logic program, obtained after compiling rough query (ii), we conclude that the answer to*

$$(\overline{\texttt{diseaseA}}(X,Y,Z) \subseteq \underline{\neg\texttt{diseaseB}}(X,Y,Z),\ \mathcal{P})$$

is `yes`.

□

The query language proposed here is slightly more general than the one presented in [VDM03b], since now we allow for testing arbitrary inclusions between lower and upper approximations. For instance in [VDM03b], we could not test whether the lower approximation of one rough relation R_1, denoted by predicate r_1/n, was included in the upper approximation of another rough relation R_2, denoted by another predicate r_2/n. With the rough query language discussed in this chapter, this can be achieved through the query $(\underline{r_1}(X_1,\ldots,X_n) \subseteq \overline{r_2}(X_1,\ldots,X_n)\ ,\ \mathcal{P})$.

In some applications it is necessary to check rough inclusion or rough equality of given rough relations. We recall the notions of rough inclusion and rough equality [Paw91].

Definition 4.14 *Rough relation* Q_1 *is* roughly included *in rough relation* Q_2*, denoted as* $Q_1 \overline{\underline{\sqsubseteq}} Q_2$*, if and only if* $\underline{Q_1} \subseteq \underline{Q_2}$ *and* $\overline{Q_1} \subseteq \overline{Q_2}$.

Definition 4.15 *The rough sets* Q_1 *and* Q_2 *are* roughly equal*, denoted as* $Q_1 \approx Q_2$*, if and only if* $\overline{Q_1} = \overline{Q_2}$ *and* $\underline{Q_1} = \underline{Q_2}$.

Given a rough program $\mathcal{P}$ and two predicates q_1/n and q_2/n denoting rough relations Q_1 and Q_2, respectively, we can easily test whether $Q_1 \overline{\underline{\sqsubseteq}} Q_2$ or $Q_1 \approx Q_2$. The rough query

$$(\overline{q_1}(X_1,\ldots,X_n) \subseteq \overline{q_2}(X_1,\ldots,X_n), \underline{q_1}(X_1,\ldots,X_n) \subseteq \underline{q_2}(X_1,\ldots,X_n)\ ,\ \mathcal{P})$$

tests for rough inclusion. Rough equality can be tested through the rough query

$$\begin{array}{l}(\overline{q_1}(X_1,\ldots,X_n) \subseteq \overline{q_2}(X_1,\ldots,X_n),\\ \overline{q_2}(X_1,\ldots,X_n) \subseteq \overline{q_1}(X_1,\ldots,X_n),\\ \underline{q_1}(X_1,\ldots,X_n) \subseteq \underline{q_2}(X_1,\ldots,X_n),\\ \underline{q_2}(X_1,\ldots,X_n) \subseteq \underline{q_1}(X_1,\ldots,X_n) \ \ , \mathcal{P})\ .\end{array}$$

Finally, we show the equivalence between the proposed technique to compute answers of a rough query and its declarative semantics. To that end, we need to prove the following two lemmas.

Lemma 4.11 *Let $(\mathcal{Q},\mathcal{P})$ be a rough query.*

(i) Assume that $\mathcal{Q}$ is of the form $\mathcal{Q}_1$ or $\mathcal{Q}_2$ (see definition 4.12). If θ is a ground substitution belonging to an answer of $(\tau_2(\mathcal{Q}),\tau_1(\mathcal{P}))$ then θ also belongs to an answer of $(\mathcal{Q},\mathcal{P})$.

(ii) Assume that $\mathcal{Q}$ is of the form $\mathcal{Q}_3$. If the extended logic program $\tau_3((\mathcal{Q},\mathcal{P}))$ has a paraconsistent stable model then the answer to the rough query $(\mathcal{Q},\mathcal{P})$ is `yes`*. Otherwise, the answer is* `no`*.*

Proof:

(i) Assume that θ is a ground substitution belonging to an answer of $(\tau_2(\mathcal{Q}),\tau_1(\mathcal{P}))$. Hence by definition 3.10, we have that

$$\tau_1(\mathcal{P}) \models \tau_2(\mathcal{Q})\theta\ .$$

It easy to see that $\tau_2(\mathcal{Q})\theta = \tau_2(\mathcal{Q}\theta)$. By theorem 4.2, we have then that $\mathcal{P} \models \mathcal{Q}\theta$.

(ii) Assume that the extended logic program $\tau_3((\mathcal{Q},\mathcal{P}))$ has a paraconsistent stable model. Let $\mathcal{Q} \equiv L_{11} \subseteq L_{12},\ldots,L_{n1} \subseteq L_{n2}$.

(ii.1) If some $L_{i1} \subseteq L_{i2} \equiv L_{i1} \subseteq \overline{L}$ $(1 \leq i \leq n)$ then for all paraconsistent stable models $\mathcal{M}$ of $\tau_3((\mathcal{Q},\mathcal{P}))$ and ground substitutions θ we have that

$$\mathcal{M} \not\models \tau_2(L_{i1})\theta,\ \textit{not}\ L\theta\ . \qquad (1)$$

Consider a paraconsistent stable model $\mathcal{M}$ of $\tau_3((\mathcal{Q},\mathcal{P}))$ and a ground substitution θ such that $\mathcal{M} \models \tau_2(L_{i1})\theta$. By (1), we have then that $\mathcal{M} \not\models \textit{not}\ L\theta$. Therefore, $L\theta \in \mathcal{M}$ and, consequently, $\mathcal{M} \models L\theta$.

If $\mathcal{M} \models \tau_2(L_{i1}\theta), L\theta$ then, by lemma 4.4 and definition of function τ_2, $\varphi^{-1}(\mathcal{M}) \models L_{i1}\theta, \overline{L}\theta$. Since $\mathcal{M}$ is a paraconsistent stable model of $\tau_3((\mathcal{Q},\mathcal{P}))$, $\mathcal{M}$ is also a paraconsistent stable model of $\tau_1(\mathcal{P})$. Hence by lemma 4.1, $\varphi^{-1}(\mathcal{M}) \in sem(\mathcal{P})$.

We can then conclude that there is a model $\mathcal{M}' \in sem(\mathcal{P})$ such that, if $\mathcal{M}' \models L_{i1}\theta$ then $\mathcal{M}' \models \overline{L}\theta$, for any ground substitution θ.

(ii.2) If some $L_{i1} \subseteq L_{i2} \equiv L_{i1} \subseteq \underline{L}$ $(1 \leq i \leq n)$ then for all paraconsistent stable models $\mathcal{M}$ of $\tau_3((\mathcal{Q},\mathcal{P}))$ and ground substitutions θ we have that

$$\mathcal{M} \not\models \tau_2(L_{i1})\theta,\ not\ L\theta\ . \qquad (2)$$

and

$$\mathcal{M} \not\models \tau_2(L_{i1})\theta, \neg L\theta\ . \qquad (3)$$

Consider a paraconsistent stable model $\mathcal{M}$ of $\tau_3((\mathcal{Q},\mathcal{P}))$ and a ground substitution θ such that $\mathcal{M} \models \tau_2(L_{i1})\theta$. By (2) and (3), we have then that $\mathcal{M} \models L\theta$ and $\mathcal{M} \models\ not\ \neg L\theta$, i.e. $\mathcal{M} \models (L,\ not\ \neg L)\theta \Leftrightarrow \mathcal{M} \models \tau_2(\underline{L}\theta)$.

If $\mathcal{M} \models \tau_2(L_{i1}\theta)$ and $\mathcal{M} \models \tau_2(\underline{L}\theta)$ then, by lemma 4.4, $\varphi^{-1}(\mathcal{M}) \models L_{i1}\theta$ and $\varphi^{-1}(\mathcal{M}) \models \underline{L}\theta$. Moreover, as it was shown in the previous case, $\varphi^{-1}(\mathcal{M}) \in sem(\mathcal{P})$.

We can then conclude that there is a model $\mathcal{M}' \in sem(\mathcal{P})$ such that, if $\mathcal{M}' \models L_{i1}\theta$ then $\mathcal{M}' \models \underline{L}\theta$, for any ground substitution θ.

(ii.3) If some $L_{i1} \subseteq L_{i2} \equiv L_{i1} \subseteq \overline{L}$ $(1 \leq i \leq n)$ then for all paraconsistent stable models $\mathcal{M}$ of $\tau_3((\mathcal{Q},\mathcal{P}))$ and ground substitutions θ we have that

$$\mathcal{M} \not\models \tau_2(L_{i1})\theta,\ not\ L\theta\ . \qquad (4)$$

and

$$\mathcal{M} \not\models \tau_2(L_{i1})\theta,\ not\ \neg L\theta\ . \qquad (5)$$

Consider a paraconsistent stable model $\mathcal{M}$ of $\tau_3((\mathcal{Q},\mathcal{P}))$ and a ground substitution θ such that $\mathcal{M} \models \tau_2(L_{i1})\theta$. By (4) and (5), we have then that $\mathcal{M} \models L\theta$ and $\mathcal{M} \models \neg L\theta$, i.e. $\mathcal{M} \models (L, \neg L)\theta \Leftrightarrow \mathcal{M} \models \tau_2(\overline{\underline{L}}\theta)$.

If $\mathcal{M} \models \tau_2(L_{i1}\theta)$ and $\mathcal{M} \models \tau_2(\overline{\underline{L}}\theta)$ then, by lemma 4.4, $\varphi^{-1}(\mathcal{M}) \models L_{i1}\theta$ and $\varphi^{-1}(\mathcal{M}) \models \overline{\underline{L}}\theta$. Moreover, as it was shown previously, $\varphi^{-1}(\mathcal{M}) \in sem(\mathcal{P})$.

We can then conclude that there is a model $\mathcal{M}' \in sem(\mathcal{P})$ such that, if $\mathcal{M}' \models L_{i1}\theta$ then $\mathcal{M}' \models \underline{\overline{L}}\theta$, for any ground substitution θ.

□

Lemma 4.12 *Let $(\mathcal{Q}, \mathcal{P})$ be a rough query.*

(i) Assume that $\mathcal{Q}$ is of the form $\mathcal{Q}_1$ or $\mathcal{Q}_2$ (see definition 4.12). If θ is a ground substitution belonging to an answer of $(\mathcal{Q}, \mathcal{P})$ then θ also belongs to an answer of $(\tau_2(\mathcal{Q}), \tau_1(\mathcal{P}))$.

(ii) Assume that $\mathcal{Q}$ is of the form $\mathcal{Q}_3$. If the answer to the rough query $(\mathcal{Q}, \mathcal{P})$ is `yes` *then the extended logic program $\tau_3((\mathcal{Q}, \mathcal{P}))$ has a paraconsistent stable model.*

Proof:

(i) The statement above is direct consequence of theorem 4.2 and of definition of answer of a query to an extended logic program.

(ii) Let $\mathcal{M} \in sem(\mathcal{P})$. Assume also that $\mathcal{Q} \equiv L_1 \subseteq L_2$ and that the answer to the rough query $(L_1 \subseteq L_2, \mathcal{P})$ is `yes` . Thus by definition of answer to a rough query,

$$\mathcal{M} \models L_1\theta \Rightarrow \mathcal{M} \models L_2\theta \,, \qquad (6)$$

for every ground substitution θ.

If $\mathcal{M} \in sem(\mathcal{P})$ then, by theorem 4.1, $\varphi(\mathcal{M})$ is a paraconsistent stable model of $\tau_1(\mathcal{P})$. Moreover, from (6) and by lemma 4.4,

$$\varphi(\mathcal{M}) \models \tau_2(L_1)\theta \Rightarrow \varphi(\mathcal{M}) \models \tau_2(L_2)\theta \qquad (7)$$

We need now to show that $\tau_3((L_1 \subseteq L_2, \mathcal{P}))$ has a paraconsistent stable model. We consider each possible case for the rough literal L_2.

(ii.1) Let $L_2 \equiv \overline{L}$. Recall that the integrity constraint

$$\texttt{:-}\ \tau_2(L_1),\ not\ L.$$

is added in this case to $\tau_1(\mathcal{P})$. From (7), we have that

$$\varphi(\mathcal{M}) \models \texttt{:-}\ \tau_2(L_1),\ not\ L.\ .$$

Since $\varphi(\mathcal{M})$ is a paraconsistent stable model of $\tau_1(\mathcal{P})$, we can conclude that $\varphi(\mathcal{M})$ is a paraconsistent stable model of $\tau_3((L_1 \subseteq L_2, \mathcal{P}))$.

(ii.2) Let $L_2 \equiv \underline{L}$. Recall that the set of integrity constraints

$$\{\texttt{:-}\ \tau_2(L_1),\ not\ L.\ ,\texttt{:-}\ \tau_2(L_1), \neg L.\}$$

is added in this case to $\tau_1(\mathcal{P})$. By reasoning in a way similar to case (i), we can conclude that $\varphi(\mathcal{M})$ is a paraconsistent stable model of $\tau_3((L_1 \subseteq L_2, \mathcal{P}))$.

(ii.3) Let $L_2 \equiv \overline{L}$. Recall that the set of integrity constraints

$$\{\texttt{:-}\ \tau_2(L_1),\ not\ L.\ ,\texttt{:-}\ \tau_2(L_1),\ not\ \neg L.\}$$

is added in this case to $\tau_1(\mathcal{P})$. By reasoning in a way similar to case (i), we can conclude that $\varphi(\mathcal{M})$ is a paraconsistent stable model of $\tau_3((L_1 \subseteq L_2, \mathcal{P}))$.

This proof generalizes easily to the case $\mathcal{Q} \equiv L_{11} \subseteq L_{12}, \ldots, L_{n1} \subseteq L_{n2}$, with $1 < n$.

□

Theorem 4.3 *Let $(\mathcal{Q}, \mathcal{P})$ be a rough query.*

(i) Assume that $\mathcal{Q}$ is of the form $\mathcal{Q}_1$ or $\mathcal{Q}_2$ (see definition 4.12). A ground substitution θ belongs to the answer of $(\tau_2(\mathcal{Q}), \tau_1(\mathcal{P}))$ if and only if θ also belongs to the answer of $(\mathcal{Q}, \mathcal{P})$.

(ii) Assume that $\mathcal{Q}$ is of the form $\mathcal{Q}_3$. The extended logic program $\tau_3((\mathcal{Q}, \mathcal{P}))$ has a paraconsistent stable model if and only if the answer to the rough query $(\mathcal{Q}, \mathcal{P})$ is `yes` .

Proof: This theorem is a direct consequence of both lemmas 4.11 and 4.12.

□

The theorem above shows that the problem of answering rough queries reduces to one of the two problems: to compute answer substitutions for a query to the compiled program; or to test whether the compiled program has a paraconsistent stable model. This provides a foundation for implementation of the language. Since the compilation procedure is polynomial with respect to the size of a rough program, the efficiency of the algorithm to answer rough queries is mainly determined by the system (e.g. *Prolog*, *dlv*, *Smodels*) used to compute answers to the queries for the compiled program. Howevere, deciding the existence of a (paraconsistent) stable model for an extended logic program is a NP-complete problem [DEGV01].

Chapter 5

Application Examples

This chapter presents several examples [VDM03a] that highlight the applicability of the language discussed in the previous chapter.

We have chosen three different relevant problems reported in the rough set literature and show how these problems can be encoded in the proposed language. In contrast to the specific-purpose solutions usually presented, our language offers a general framework where the solution to different types of problems can be declaratively expressed. Moreover, another particularly important aspect illustrated is the integration of rough sets with domain knowledge.

We start by presenting, in section 5.1, a technique to reduce the boundary region of a rough relation. Then in section 5.2, we show how to monitor changes in the boundary region of a rough relation, when some condition attributes are eliminated. Finally, section 5.3 illustrates the integration of expert knowledge through the use of default rules encoded in our language.

5.1 Hierarchy-Structured Decision Tables

A technique to reduce, and eventually eliminate, the boundary region of a rough relation R (or decision table) is introduced in [Zia02a], in the context of the variable precision rough set model [Zia93]. This is a relevant issue because any object belonging to the boundary cannot be classified with certainty as belonging to R or $\neg R$. If we interpret a rough relation as a classifier then a large boundary might imply that the classifier is of little value.

One way to cope with the above problem is, for instance, to add more condition attributes to the table. Alternatively, if some attributes have been subject to discretization then, we could increase the precision of the existing attributes by providing more cut-points (i.e. the number of attribute values would increase). However, the disadvantage of these ideas is the rapid growth in the number of decision rules, each of them with a smaller domain coverage, i.e. $cov(r)$ tends to decrease for each decision rule r.

The main idea described in [Zia02a] is to associate only with the boundary examples a new layer of decision tables. For instance, more cut-points could be introduced for discretization of attribute values of objects in the boundary region. This would lead to the thinning of the boundary region, in may cases. Note that this "refining" process is only applied to that part of the table corresponding to the boundary region, instead of considering the whole decision table. This idea can be concretized in two ways: by building a hierarchical tree structure of decision tables or by creating a hierarchical linear structure of tables. These techniques are described below.

Each indiscernibility class contained in the boundary region can be treated as new independent universe of objects by itself and a new decision table is associated with each class, forming a new layer of decision tables. The attributes of the decision tables in a new layer have to be "more precise" in the sense they split each indiscernibility class (of the previous layer) into several sub-equivalence classes. This process can be applied recursively yielding a *hierarchical tree structure of decision tables.*

Next example shows how a tree-structured hierarchy of decision tables could be easily encoded in our language. We remind the reader that the value of an attribute a for an object o is `null`, i.e. $a(o) =$ `null`, if the valued of a is not defined for that particular object o.

Example 5.1 *Consider the decision table shown in table 5.1.*

Each line of the decision table above can be encoded in our language as a fact. For instance, the first lines would be represented as

```
¬deathmi(<70, no, no).
deathmi(>70, yes, yes).
deathmi(>70, yes, no).
      ⋮
```

We stress that expressions like "`<70`*", used as arguments of a predicate, should be understood as constants.*

	Age	Hypert	Scanabn	**Deathmi**
o_1	< 70	no	no	no
o_2	> 70	no	no	no
o_3	> 70	yes	yes	yes
o_4	> 70	yes	no	yes
o_5	> 70	yes	no	no
o_6	> 70	yes	no	no
o_7	< 70	yes	yes	no
o_8	< 70	yes	yes	yes

Table 5.1: Table of patients with heart problems.

It is easy to see that the indiscernibility classes

$$\begin{aligned} E_1 &= \{o_4, o_5, o_6\} \ , \\ E_2 &= \{o_7, o_8\} \end{aligned}$$

are in the boundary region.

In order to reduce the boundary region, a different set of condition attributes can be considered for some of these indiscernibility classes (in the boundary area), i.e. the new set of attributes considered for one class may be different from the set of attributes considered for another indiscernibility class in the boundary. The new combination of attributes may have been defined by experts in the field of application. It could also be the case that for some other indiscernibility classes the same attributes as in the original table have been considered, but increased discretization precision of the condition attributes has been applied to each of these classes, possibly with different cut-points for each of them. Let us illustrate these ideas with the table above.

Suppose that experts decided to consider a different set of attributes for patients belonging to E_1: instead of the age, it was considered whether the patient was a smoker. For patients in E_2 only different discretization for the `Age` *attribute was applied.*

As the reader can see from tables 5.2 and 5.3, the boundary region has been reduced to one indiscernibility class with two patients only, $E_3 = \{o_4, o_6\}$.

The decision tables 5.2 and 5.3 can be represented by the following facts. Note that each decision table is recorded under a different predicate name (`deathmi`*,* `deathmi`$_1$*, and* `deathmi`$_2$*).*

	Hypert	Scanabn	Smoke	**Deathmi**
o_4	yes	no	yes	yes
o_5	yes	no	no	no
o_6	yes	no	yes	no

Table 5.2: Decison table associated with class E_1.

	Age	Hypert	Scanabn	**Deathmi**
o_7	< 40	yes	yes	no
o_8	> 40	yes	yes	yes

Table 5.3: Decison table associated with class E_2.

deathmi$_1$*(yes, no, yes).*
$\neg$*deathmi*$_1$*(yes, no, no).*
$\neg$*deathmi*$_1$*(yes, no, yes).*

$\neg$*deathmi*$_2$*(<40, yes, yes).*
deathmi$_2$*(>40, yes, yes).*

Putting together all the above decision tables, we create a new rough relation shown in table 5.4. It corresponds to the initial decision table $\mathcal{D}$eathmi with a reduced boundary, by integrating tables 5.2 and 5.3. Using rough clauses, the rough relation corresponding to this table can be easily encoded. Predicate *deathmi*$_3$ *denotes this rough relation.*

	Age	Hypert	Scanabn	Smoke	**Deathmi**
o_1	< 70	no	no	null	no
o_2	> 70	no	no	null	no
o_3	> 70	yes	yes	null	yes
o_4	null	yes	no	yes	yes
o_5	null	yes	no	no	no
o_6	null	yes	no	yes	no
o_7	< 40	yes	yes	null	no
o_8	> 40	yes	yes	null	yes

Table 5.4: Decision table obtained by integrating tables 5.2 and 5.3 with table 5.1.

(1) $\overline{deathmi_3}$(Age,Hypert,Scanabn,null) :-
$\underline{deathmi}$(Age,Hypert,Scanabn).

(2) $\overline{\neg deathmi_3}$(Age,Hypert,Scanabn,null) :-
$\underline{\neg deathmi}$(Age,Hypert,Scanabn).

(3) $\overline{deathmi_3}$(null,Hypert,Scanabn,Smoke) :-
$\overline{deathmi_1}$(Hypert,Scanabn,Smoke).

(4) $\overline{\neg deathmi_3}$(null,Hypert,Scanabn,Smoke) :-
$\overline{\neg deathmi_1}$(Hypert,Scanabn,Smoke).

(5) $\overline{deathmi_3}$(Age,Hypert,Scanabn,null) :-
$\overline{deathmi_2}$(Age,Hypert,Scanabn).

(6) $\overline{\neg deathmi_3}$(Age,Hypert,Scanabn,null) :-
$\overline{\neg deathmi_2}$(Age,Hypert,Scanabn).

As we can see in this example, it is possible that some indiscernibility classes in the boundary region have been split into new classes (for instance, $E_1 = \{o_4, o_5, o_6\}$ was split into $E_{11} = \{o_4, o_6\}$ and $E_{12} = \{o_5\}$) and that some of them are still in the boundary region ($E_{11} = \{o_4, o_6\}$). Then, the same idea could be applied once more generating another layer of decision tables.

□

A slightly different method (also proposed in [Zia02a]) for reducing the boundary region is obtained by treating the whole subset of the universe corresponding to the boundary as a new domain by itself. Thus, a new decision table is associated with this subset of the universe forming a new layer. However, in this case each new layer has one table only and, consequently, we get a *hierarchical linear structure of decision tables.*

The following example shows how a linear-structured hierarchy of decision tables could be encoded in our language.

Example 5.2 *Consider the decision table 5.1 of example 5.1. The whole boundary region is treated as new domain by itself and we associate with it a new decision table. In this case the hypertension attribute was replaced by the sex of the patient and one more condition attribute indicating whether the patient is a smoker was considered. The aim is that when considering these new set of attributes, the boundary region will be reduced (or even eliminated).*

Looking at the table 5.5, we see that the boundary has been reduced to two objects $E_3 = \{o_7, o_8\}$. Experts decided to try a different discretization for the age attribute, with the aim to eventually eliminate the boundary region.

	Age	Scanabn	Sex	Smoke	**Deathmi**
o_4	> 70	no	M	yes	yes
o_5	> 70	no	F	no	no
o_6	> 70	no	M	no	no
o_7	< 70	yes	M	yes	no
o_8	< 70	yes	M	yes	yes

Table 5.5: Decison table associated with the boundary region of table 5.1.

	Age	Scanabn	Sex	Smoke	**Deathmi**
o_7	< 40	yes	M	yes	no
o_8	> 40	yes	M	yes	yes

Table 5.6: Decison table associated with the boundary of table 5.5.

Thus, another decision table (see table 5.6) was associated with the boundary of table 5.5 .

Decision tables 5.5 and 5.6 are represented in our language as a set of facts under predicates $\texttt{deathmi}_1$ *and* $\texttt{deathmi}_2$*, respectively.*

$\texttt{deathmi}_1(>70, \texttt{no}, M, \texttt{yes})$.
$\neg\texttt{deathmi}_1(>70, \texttt{no}, F, \texttt{no})$.
$\neg\texttt{deathmi}_1(>70, \texttt{no}, M, \texttt{no})$.
$\neg\texttt{deathmi}_1(<70, \texttt{yes}, M, \texttt{yes})$.
$\texttt{deathmi}_1(<70, \texttt{yes}, M, \texttt{yes})$.

$\neg\texttt{deathmi}_2(<40, \texttt{yes}, M, \texttt{yes})$.
$\texttt{deathmi}_2(>40, \texttt{yes}, M, \texttt{yes})$.

The relation between the decision tables 5.1, 5.5, and 5.6, a hierarchical linear structure, can be encoded using rough clauses. All these decision tables are related to the same rough set (relation) that we designate by predicate $\texttt{deathmi}_3$.

(1) $\overline{deathmi_3}$(Age,Hypert,Scanabn,null,null) :-
$\underline{deathmi}$(Age,Hypert,Scanabn).

(2) $\overline{\neg deathmi_3}$(Age,Hypert,Scanabn,null,null) :-
$\underline{\neg deathmi}$(Age,Hypert,Scanabn).

(3) $\overline{deathmi_3}$(Age,null,Scanabn,Sex,Smoke) :-
$\underline{\neg deathmi_1}$(Age,Scanabn,Sex,Smoke).

(4) $\overline{\neg deathmi_3}$(Age,null,Scanabn,Sex,Smoke) :-
$\underline{\neg deathmi_1}$(Age,Scanabn,Sex,Smoke).

(5) $\overline{deathmi_3}$(Age,null,Scanabn,Sex,Smoke) :-
$\overline{deathmi_2}$(Age,Scanabn,Sex,Smoke).

(6) $\overline{\neg deathmi_3}$(Age,null,Scanabn,Sex,Smoke) :-
$\overline{\neg deathmi_2}$(Age,Scanabn,Sex,Smoke).

When considering a graphical interface, possibly showing the hierarchical structure of the tables, the clauses above could be generated automatically. Moreover, the graphical interface could also hide those predicate arguments corresponding to attributes with `null` *value.*

5.2 Avoiding Expensive Tests

In many practical applications, the attribute values correspond to the outcome of a certain test applied to objects of the universe (e.g. a medical test performed on patients). Thus, we may intuitively associate with each attribute a cost, corresponding to the cost of the test that must be performed. Obviously, some attributes may be more expensive than others. For instance, a medical test may be considered expensive because it requires the use of expensive equipment, or because it may cause a lot of discomfort to the patient, or because in general the underlying procedure is expensive.

Given an object o of the universe U and, based on the values of a set of attributes A, a certain decision d is taken (e.g. whether a patient suffers from a certain disease). This information, for each object, can be thought as recorded in the form of a decision table $\mathcal{D}_A = (U, A, d)$. Assume that a set of attributes $B \subset A$ has been identified as being expensive and, therefore, desirable to avoid. We would like to identify those objects o for which the knowledge about attributes B is absolutely necessary for making a decision, i.e. for determining $d(o)$.

The problem described has been studied in [KØ99] and following the approach suggested there, identification of the above mentioned set of objects requires monitoring changes in the boundary of D_A (the rough relation

defined by decision table $\mathcal{D}_A$) when considering only the set of attributes $A \setminus B$ (i.e. removing expensive tests B). Next, we summarize the main idea described in [KØ99].

Given two sets A and B, the expression $A \setminus B$ denotes set difference.

Let $\mathcal{D}_A = (U, A, d)$ be a decision table, with a binary decision attribute, and $[t]$ denote the set of objects belonging to the indiscernibility class described by tuple t. If β is a set of tuples then the set of objects described by the tuples belonging to β is given by

$$obj(\beta) = \bigcup_{t \in \beta} [t] \ .$$

Assume also that $\mathcal{D}_{A \setminus B} = (U, A \setminus B, d)$ corresponds to the decision table $\mathcal{D}_A$ without attributes B (i.e. it is a projection of table $\mathcal{D}_A$). When considering a subset $A \setminus B$ of attributes, we have that

$$R_A \subseteq R_{A \setminus B} \ ,$$

where R_A and $R_{A \setminus B}$ are the indiscernibility relations induced by decision tables $\mathcal{D}_A$ and $\mathcal{D}_{A \setminus B}$, respectively. Intuitively, this means that when considering the set of attributes $A \setminus B$, several indiscernibility classes may be merged into one single class. Thus, when considering less attributes, the approximation space may be formed by a smaller number of larger indiscernibility classes. Consequently, the number of objects belonging to $obj(\underline{D_A})$ tends to decrease while the number of objects in $obj(\overline{D_A})$ tends to increase. Hence, it can also be easily concluded that

$$\begin{array}{rcll} obj(\underline{D_{A \setminus B}}) & \subseteq & obj(\underline{D_A}) & \text{and} \\ obj(\underline{\neg D_{A \setminus B}}) & \subseteq & obj(\underline{\neg D_A}) \ . & \end{array}$$

When considering the reduced set of attributes $A \setminus B$, the number of objects in the boundary region of D_A also increases, since some indiscernibility classes previously belonging to $\underline{\neg D}_A$ or belonging to $\underline{D}_A$ have now migrated to the new boundary. Intuitively, characterization of these indiscernibility classes defines the set of objects for which knowledge about attributes B is crucial for making a decision. For all other objects, knowledge about B will not change the region, $\underline{\neg D_A}$, $\underline{D_A}$ or $\overline{D_A}$, where they already belong. The set of migrating objects can then be defined as

$$\begin{array}{rcl} Migrate(A, B, D) & = & (obj(\underline{\overline{D_{A \setminus B}}}) \cap obj(\underline{D_A})) \\ & & \cup \\ & & (obj(\underline{\overline{D_{A \setminus B}}}) \cap obj(\underline{\neg D_A})) \ . \end{array} \quad (1)$$

Although the definition above looks different from the one used in [KØ99], they are both equivalent. However, the formulation presented here is more suitable in the context of our framework, as the reader will see soon.

Obviously, the set on non-migrating objects is defined as

$$\neg Migrate(A, B, D) = obj(\underline{D_{A \setminus B}}) \cup obj(\underline{\neg D_{A \setminus B}}) \cup obj(\overline{\underline{D_A}}) \ . \quad (2)$$

It is important to note that the set of (non)migrating objects is rough, if only attributes $A \setminus B$ are considered. Otherwise, if all attributes A are considered then the set is crisp. The example 5.3 illustrates this point.

The migration set enables us to find those objects that require the results of the tests associated with attributes B to be known in order to be able to make a decision. Thus, we aim at finding a description of this set of objects using attributes $A \setminus B$. This description can then be applied to new (unseen) objects to decide whether tests B should be performed.

In practice, for all objects falling in the upper approximation of the migrate set (i.e. conforming to the description of the upper approximation), tests associated with attributes B could be requested. However, if the upper approximation gets very large when attributes B are removed, then not that much is gained. This points to the need of associating some numerical measures with the upper and lower approximations giving some information about the number of objects they might contain. This issue is discussed further in chapter 6.

Both expressions above, the set of migrating and non-migrating objects, can be translated to a set of rough clauses. These rough clauses permit the user to discover a set of tuples describing those objects belonging to $Migrate(A, B, D)$ ($\neg Migrate(A, B, D)$). The next example illustrates this application.

Example 5.3 *Consider the decision table* $\mathcal{D}eathmi = (U, \{\texttt{Age}, \texttt{Test A}_1, \texttt{Test A}_2\}, \texttt{Deathmi})$*, where* U *is a set of patients with heart problems. Assume that the condition attributes* $\texttt{A}_1$ *and* $\texttt{A}_2$ *represent two medical tests. Moreover, test* $\texttt{A}_2$ *is usually considered as being expensive, and therefore, desirable to avoid.*

From table 5.7, it is easy to see that

(i) $\{\langle \texttt{>70, b1, c1} \rangle, \langle \texttt{>40 <70, b2, c4} \rangle\} \subseteq \underline{\neg Deathmi}$;

(ii) $\{\langle \texttt{>40 <70, b2, c3} \rangle, \langle \texttt{<40, b4, c3} \rangle\} \subseteq \underline{Deathmi}$;

(iii) $\{\langle \texttt{>70, b1, c2} \rangle, \langle \texttt{<40, b3, c5} \rangle\} \subseteq \overline{\underline{Deathmi}}$;

Age	Test A_1	Test A_2	**Deathmi**
> 70	b_1	c_1	no
> 70	b_1	c_2	yes
> 70	b_1	c_2	no
$> 40 < 70$	b_2	c_3	yes
$> 40 < 70$	b_2	c_4	no
< 40	b_3	c_5	yes
< 40	b_3	c_5	no
< 40	b_3	c_5	no
< 40	b_4	c_3	yes

Table 5.7: Decision table of patients with heart problems.

The table is encoded as facts in our language.

```
¬deathmi(>70, b1, c1).          deathmi(>70, b1, c2).
¬deathmi(>70, b1, c2).          deathmi(>40 <70, b2, c3).
¬deathmi(>40 <70, b2, c4).      deathmi(<40, b3, c5).
¬deathmi(<40, b3, c5).          deathmi(<40, b4, c3).
```

Moreover, the following clauses monitor the impact in the boundary region of not considering test A_2*. Basically, these clauses translate the set of migrating and non-migrating patients represented by formulas* (1) *and* (2) *above. The predicate* d *denotes the rough relation* D *corresponding to the projection in the first two attributes of* $\mathcal{D}$*eathmi (table 5.7).*

(1) $\overline{d}$(Age,Test_A1) :- $\overline{deathmi}$(Age,Test_A1,Test_A2).

(2) $\overline{\neg d}$(Age,Test_A1) :- $\overline{\neg deathmi}$(Age,Test_A1,Test_A2).

(3) $\overline{migrate}$(Age,Test_A1) :-
$\underline{\overline{d}}$(Age,Test_A1),
$\underline{deathmi}$(Age,Test_A1,Test_A2).

(4) $\overline{migrate}$(Age,Test_A1) :-
$\underline{\overline{d}}$(Age,Test_A1),
$\underline{\neg deathmi}$(Age,Test_A1,Test_A2).

(5) $\overline{\neg migrate}$(Age,Test_A1) :- $\underline{\neg d}$(Age,Test_A1).

(6) $\overline{\neg migrate}$(Age,Test_A1) :- $\underline{d}$(Age,Test_A1).

(7) $\overline{\neg migrate}$(Age,Test_A1) :-
$\underline{\overline{deathmi}}$(Age,Test_A1,Test_A2).

Thus, by clauses (1) *and* (2)*, we have that* $\{\langle$>70, b1$\rangle$, $\langle$>40 <70, b2$\rangle$, $\langle$<40, b3$\rangle\} \subseteq \underline{\overline{D}}$. *By clause* (4), $\langle$>70, b1$\rangle \in \overline{Migrate}$ *and it corresponds to a (class of) patient(s) that migrated from the lower approximation of rough set* $\neg Deathmi$. *But by clause (7) and taking into account (iii),* $\langle$>70, b1$\rangle \in \overline{\neg Migrate}$. *Thus,* $\langle$>70, b1$\rangle$ *is in the boundary of relation* migrate*, consequently, showing that the set of migrating patients is rough in this case.*

By clause (3) *or* (4)*,* $\langle$>40 <70, b2$\rangle \in \overline{Migrate}$ *and it corresponds to the merging of two indiscernibility classes, one originating from* $\underline{Deathmi}$ *and the other from* $\underline{\neg Deathmi}$.

By clause (6)*,* $\langle$<40, b4$\rangle \in \overline{\neg Migrate}$. *This indiscernibility class remains in the lower approximation, even after dropping attribute* A_2. *Thus, nothing is gained in performing the expensive test for these patients.*

By clause (7)*,* $\langle$<40, b3$\rangle \in \overline{\neg Migrate}$ *and it corresponds to a non-migrating (class of) patient(s) that remained in the boundary after dropping the attribute corresponding to the* 3*rd argument of* deathmi*, i.e. the expensive medical test.*

Let $\mathcal{P}$ *be the rough program obtained from the set of facts encoding decision table 5.7 together with the rough clauses* (1) – (7)*. The query* ($\overline{migrate}$(Age,A1), $\mathcal{P}$) *requests a description of all patients that may migrate when the expensive test is dropped. The answer is the set of substitutions* θ*,*

$$\theta = \{\{Age/\text{>40 <70},\ A1/\text{b2}\}, \{Age/\text{>70},\ A1/\text{b1}\}\}\ ,$$

indicating that the tuples $\langle$`>40 <70, b2`$\rangle$ *and* $\langle$`>70, b1`$\rangle$ *belong to the upper approximation of rough relation denoted by* `migrate` *(i.e.* $\overline{Migrate}$*). We may then conclude that for a new patient whose age is between* 40 *and* 70 *and who obtained the result* `b2` *for the test* $\mathtt{A}_1$*, it is advisable to perform the medical test* $\mathtt{A}_2$*. We may also ask*

- *"For which patients more than* 70 *years old it is worth to perform test* A_2*?"*

This can be translated to the rough query ($\overline{\texttt{migrate}}$`(>70,A1)`, $\mathcal{P}$)*. As answer we get the singleton* $\{\{A1/\texttt{b1}\}\}$ *stating that only patients with outcome* `b1` *for test* $\mathtt{A}_1$ *should be submitted to test* $\mathtt{A}_2$*.*

Another relevant question is

- *"Which patients may not be submitted to test* A_2*?"*

It can be represented by the rough query ($\overline{\neg\texttt{migrate}}$`(Age,A1)`, $\mathcal{P}$)*. As answer we get the set of substitutions* θ'

$$\theta' = \{\{Age/\texttt{<40},\ A1/\texttt{b3}\}, \{Age/\texttt{<40},\ A1/\texttt{b4}\}, \{Age/\texttt{>70},\ A1/\texttt{b1}\}\} .$$

This answer can be interpreted as stating that if a patient conforms to the case

$$\begin{array}{ll} ((\texttt{Age} < 40) \wedge (\texttt{Test}\ A_1 = \texttt{b3})) & \vee \\ ((\texttt{Age} < 40) \wedge (\texttt{Test}\ A_1 = \texttt{b4})) & \vee \\ ((\texttt{Age} > 70) \wedge (\texttt{Test}\ A_1 = \texttt{b1})) & \end{array}$$

then test $\mathtt{A}_2$ *may be rather irrelevant.*

□

In [KØ99], the migration set is defined in a different way.

$$Migrate(A, B, D) = obj(\overline{\underline{D_{A\setminus B}}}) \ \cap \ (U \setminus obj(\overline{\underline{D_A}})) \ . \qquad (3)$$

Note that U is the set of all objects represented in the decision tables $\mathcal{D}_A$ and $\mathcal{D}_{A\setminus B}$. Thus,

$$U \setminus obj(\overline{\underline{D_A}}) = obj(\underline{D_A}) \ \cup \ obj(\underline{\neg D_A}) \ .$$

This shows that expressions (1) and (3) above are equivalent. Based on (3), a new decision table $\mathcal{D}_1 = (U, A \setminus B, d_1)$ is defined in [KØ99]:

$$d_1(o) = \begin{cases} \texttt{yes} & o \in Migrate(A, B, D) \\ \texttt{no} & \texttt{otherwise} \end{cases}$$

A specific program then takes as input the decision table $\mathcal{D}_A = (U, A, d)$ and creates as output table $\mathcal{D}_1 = (U, A \setminus B, d_1)$, as defined above.

What we wish to emphasize here is that the language we propose is a general framework to create new rough relations and to describe them declaratively in terms of other rough relations. This contrasts with the way the problem was tackled in [KØ99], since there a specific program to create a specific rough relation had to be built.

5.3 Representing Default Knowledge

In this section, we show through a couple of examples that we can also easily express default knowledge in our language and, as in system *CAKE* [DŁS02], define priorities between defaults.

Intuitively, default knowledge corresponds to conclusions assumed to be true in general (we may also call it common sense knowledge), even if we do not have a direct evidence of their truth. For example, we assume that

- *"If someone is driving a car then he has a driving licence."*

However, this does not always have to be true. We may have information that invalidates this conclusion by default (e.g. the person is less than 18 years old).

Representation of default knowledge has been addressed by Reiter who has proposed default logic [Rei80]. The following example shows how *normal default rules* of the default logic can be encoded in our formalism. A formal comparison of default logic with our formalism is out of the scope of this thesis.

Example 5.4 *Consider table 5.8*

$\mathcal{D}istance = (U, \{\texttt{Dif}, \texttt{Road Conditions}, \texttt{Physical Distance}\}, \texttt{Distance})$.

This table takes a set of traffic situations U characterized by

- *the difference between the actual speed of a vehicle and the speed limit at the road where the vehicle circulates (attribute* `Dif`*);*
- *road conditions (dry, wet, snow, or ice); and*
- *the distance between the vehicle and the one in front of it.*

Dif	Road Conditions	Physical Distance	**Distance**
10	dry	9	medium
10	ice	15	medium
10	ice	12	small
30	wet	30	large
−10	snow	9	medium
−10	snow	9	small

Table 5.8: Decision table classifying vehicle distances.

This data could have been acquired from a number of different sources. For instance, road conditions could have been obtained by sensors, a camera records traffic images, while speed limit in roads are obtained from a database. Then, one or more experts in traffic safety, decide for each situation whether the distance between vehicles is large, medium, or small. It is easy to accept that this classification depends on the attributes mentioned above. It may also happen that, given the same traffic situation, different experts classify differently the distance (i.e. one expert might say that the distance is small and another consider it as medium).

Note that the decision attribute `Distance` *is not binary in this case, since it may assume the values* `small`, `medium`, *or* `large`. *However, it is easy to see this table as three decision tables defining the (rough) concepts of small, medium, and large distance. Moreover, this idea can be easily expressed in our language as rough clauses* (1)−(9) *together with the facts under predicate p (see below). The whole table is encoded as a set of positive facts.*

```
p̄(10, dry, 9, medium).        p̄(10, ice, 15, medium).
p̄(10, ice, 12, small).        p̄(30, wet, 30, large).
p̄(-10, snow, 9, medium).      p̄(-10, snow, 9, small).
```

```
(1)  -large(X1,X2,X3):- -p(X1,X2,X3,large).
(2)  ¬large(X1,X2,X3):- -medium(X1,X2,X3).
(3)  ¬large(X1,X2,X3):- -small(X1,X2,X3).

(4)  -medium(X1,X2,X3):- -p(X1,X2,X3,medium).
(5)  ¬medium(X1,X2,X3) :- -small(X1,X2,X3).
(6)  ¬medium(X1,X2,X3) :- -large(X1,X2,X3).

(7)  -small(X1,X2,X3):- -p(X1,X2,X3,small).
(8)  ¬small(X1,X2,X3):- -medium(X1,X2,X3).
(9)  ¬small(X1,X2,X3):- -large(X1,X2,X3).
```

Another decision table, see table 5.9,

$$\mathcal{D}anger = (U, \{Dif, Road\ Conditions, Distance\}, Danger)\ ,$$

shows whether a number of traffic situations has been classified as dangerous by an expert. As usual, this table is represented as a set of facts.

Dif	Road Conditions	Distance	**Danger**
10	dry	large	no
20	ice	small	yes
0	wet	medium	no
10	wet	medium	yes
10	wet	medium	no
−10	snow	medium	yes

Table 5.9: Decision table classifying the danger of several traffic situations.

```
-danger(10, dry, large).        -danger(20, ice, small).
¬danger(0, wet, medium).        -danger(10, wet, medium).
¬danger(10, wet, medium).       -danger(-10, snow, medium).
```

We also would like to add some common sense knowledge to the set of facts above. For instance, consider the following statement expressing that usually people assume by default that small distances between two vehicles yield to a dangerous situation.

- *"If the distance between two vehicles is small, then we may assume that the situation is dangerous (unless there is evidence to the contrary)."*

This statement could be expressed by the following (normal) default rule, where the variables should be understood as universally quantified.

$$\frac{\mathit{small}(X1, X2, X3) : \mathit{danger}(X1, X2, \mathit{small})}{\mathit{danger}(X1, X2, \mathit{small})} \ . \tag{5.1}$$

The default rule (5.1) can be formally read as follows.
If in a certain situation $\mathit{small}(x1, x2, x3)$ *holds (i.e. it can be proved) and* $\mathit{danger}(x1, x2, \mathit{small})$ *is consistent with the current knowledge, then (by default) we assume that* $\mathit{danger}(x1, x2, \mathit{small})$ *holds, too. Note that* $\mathit{danger}(x1, x2, \mathit{small})$ *is consistent with the current knowledge, if we have no evidence that* $\neg\mathit{danger}(x1, x2, \mathit{small})$ *is true. Thus, no contradiction with the available knowledge arises by the fact that* $\mathit{danger}(x1, x2, \mathit{small})$ *is assumed.*

Moreover, consider that we also want to express the next common sense (default) idea.

- *"If the distance between two vehicles is not small, then we may assume that the situation is not dangerous (unless it can be proved otherwise)."*

This statement could be expressed in default logic by default rules (5.2) and (5.3).

$$\frac{\mathit{medium}(X1, X2, X3) : \neg\mathit{danger}(X1, X2, \mathit{medium})}{\neg\mathit{danger}(X1, X2, \mathit{medium})} \ , \tag{5.2}$$

$$\frac{\mathit{large}(X1, X2, X3) : \neg\mathit{danger}(X1, X2, \mathit{large})}{\neg\mathit{danger}(X1, X2, \mathit{large})} \ . \tag{5.3}$$

Our next step is to show that the above default rules can be expressed in the proposed language.

(10) $\overline{\mathit{danger}_1}$(X1,X2,small):-
$\overline{\mathit{small}}$(X1,X2,X3), $\mathit{danger}?$(X1,X2,small).

(11) $\overline{\neg\mathit{danger}_1}$(X1,X2,medium):-
$\overline{\mathit{medium}}$(X1,X2,X3), $\mathit{danger}?$(X1,X2,medium).

(12) $\overline{\neg\mathit{danger}_1}$(X1,X2,large):-
$\overline{\mathit{large}}$(X1,X2,X3), $\mathit{danger}?$(X1,X2,large).

Rough clauses (10) − (12) *express the default rules (5.1)−(5.3), respectively. For instance, consider rough clause* (10). *The testing literal*

$$\texttt{danger?(X1,X2,small)}$$

in its body allows to test whether a tuple $\langle c_1, c_2, \texttt{small}\rangle$ *is undefined, i.e.* $\langle c_1, c_2, \texttt{small}\rangle \notin \overline{Danger}$ *and* $\langle c_1, c_2, \texttt{small}\rangle \notin \overline{\neg Danger}$. *If* $\langle c_1, c_2, \texttt{small}\rangle \in \overline{Danger}$ *then rough clause* (10) *is not applicable because no new information would be obtained. If* $\langle c_1, c_2, \texttt{small}\rangle \in \overline{\neg Danger}$ *then rough clause* (10) *is not applicable because its application would lead to a conclusion that would not be consistent with the available knowledge.*

Finally, we put together the knowledge coming from the table $\mathcal{D}anger$ *with the default knowledge. To achieve this we define a new rough relation (clauses (13)−(16)) and use a new predicate name (*$\texttt{danger}_2$*).*

(13) $\overline{\texttt{danger}_2}\texttt{(X1,X2,X3):-}\ \overline{\texttt{danger}}\texttt{(X1,X2,X3).}$

(14) $\overline{\texttt{danger}_2}\texttt{(X1,X2,X3):-}\ \overline{\texttt{danger}_1}\texttt{(X1,X2,X3).}$

(15) $\overline{\neg\texttt{danger}_2}\texttt{(X1,X2,X3):-}\ \overline{\neg\texttt{danger}}\texttt{(X1,X2,X3).}$

(16) $\overline{\neg\texttt{danger}_2}\texttt{(X1,X2,X3):-}\ \overline{\neg\texttt{danger}_1}\texttt{(X1,X2,X3).}$

From the fifth row of the first table we see that $\texttt{medium(-10,snow,9)}$ *holds. But, the tuple* $\langle\texttt{-10,snow,medium}\rangle$ *corresponds to a dangerous traffic situation (see last line of the second table). Thus, rough clause* (11) *(encoding default rule (5.2)) cannot be applied because* $\texttt{danger(-10,snow,medium)}$ *holds and consequently* $\langle\texttt{-10,snow,medium}\rangle$ *is not undefined with respect to rough relation Danger.*

Note that $\texttt{small(-10,snow,9)}$ *holds (last line of table* $\mathcal{D}istance$*) but the tuple* $\langle\texttt{-10,snow,small}\rangle$ *does not exist in the table* $\mathcal{D}anger$ *(i.e. thus, it does not correspond to a traffic situation known as non-dangerous). Hence, from rough clause* (10) *(corresponding to default rule (5.1)), we conclude* $\texttt{danger(-10,snow,small)}$.

□

Next example, illustrates the use of priorities between default rules.

Example 5.5 *Consider once more the (decision) tables of the previous example and default rules (5.1) and (5.2). Moreover, we also assume that*

- *"If distance between vehicles is medium and the road conditions are icy, then we may conclude that the traffic situation is dangerous (unless it can be proved otherwise)."*

Using a default rule, we could express this statement as

$$\frac{\mathit{medium}(X1, \mathit{ice}, X3) : \mathit{danger}(X1, \mathit{ice}, \mathit{medium})}{\mathit{danger}(X1, \mathit{ice}, \mathit{medium})} \ . \tag{5.4}$$

Informally, looking at the second row of the first table, we conclude that $\mathit{medium}(10, \mathit{ice}, 15)$ *holds. Moreover, from the second table we can conclude that the tuple* $\langle 10, \mathit{ice}, \mathit{medium}\rangle$ *is not considered as a dangerous (or non-dangerous) situation (actually, there is no such tuple in the second table). Thus, default rule (5.2) can be applied and we conclude that* $\neg\mathit{danger}(10, \mathit{ice}, \mathit{medium})$ *also holds. Similarly, default rule (5.4) can be applied, to conclude that* $\mathit{danger}(10, \mathit{ice}, \mathit{medium})$ *holds, too.*

Hence, from this example, we conclude that by applying different default rules, we may obtain contradictory information. Although this may be acceptable in some situations (it is a case belonging to the boundary region), in other situations we may wish to express priorities between several applicable defaults. For instance, if both default rules (5.2) and (5.4) are applicable, then we may give priority to (5.4) and block application of default (5.2) for safety reasons. To achieve this idea we first define a new rough relation expressing the default rules (5.1) and (5.4).

(10) $\overline{\mathit{danger}_1}$(X1,X2,small):-
$\overline{\mathit{small}}$(X1,X2,X3), danger?(X1,X2,small).

(11) $\overline{\mathit{danger}_1}$(X1,ice,medium):-
$\overline{\mathit{medium}}$(X1,ice,X3), danger?(X1,ice,medium).

We put then together the knowledge coming from the table $\mathcal{D}anger$ *with the default knowledge. The last rough clause,* (15)*, encodes default rule (5.2) and gives it lower priority than rough clause* (11) *(encoding default rule (5.4)).*

(12) $\overline{\mathit{danger}_2}$(X1,X2,X3):- $\overline{\mathit{danger}}$(X1,X2,X3).

(13) $\overline{\neg\mathit{danger}_2}$(X1,X2,X3):- $\overline{\neg\mathit{danger}}$(X1,X2,X3).

(14) $\overline{\mathit{danger}_2}$(X1,X2,X3):- $\overline{\mathit{danger}_1}$(X1,X2,X3).

(15) $\overline{\neg\mathit{danger}_2}$(X1,X2,medium):- $\overline{\mathit{medium}}$(X1,X2,X3),
danger?(X1,X2,medium), danger_1?(X1,X2,medium).

Chapter 6

The Rough Knowledge Base System

We present the principles of a system, called *Rough Knowledge Base System* (*RKBS*). The system is available through a Web page

`http://www.ida.liu.se/rkbs` .

It can reason about rough relations defined in a rough program and answer queries. The implementation was done by R. Andersson as a master thesis [And04b, And04a]. The ideas on which this implementation is based were already previously explored in a prototype written by A. Vitória and C. V. Damásio.

The language supported by *RKBS* (to encode rough programs) extends that of chapter 4 by associating quantitative measures with each tuple of a rough relation [VDM04]. However, this system may not be able to compute answers for all queries to a recursive rough program.

We believe that several interesting data mining applications using rough sets could be encoded in our language. For instance, the example discussed in section 5.2 illustrates an application of our language to a problem in the data mining field. Note that quantitative measures are an important aspect in data mining applications while recursion does not seem to be required by many of them. For this reason, we have extended our language to support the former feature while the latter is not yet supported.

A distinction must also be made between the compilation technique presented in chapter 4 and the compilation of rough programs with quantitative measures. The latter generates extended logic programs which require

aggregate functions[1], while these functions are not needed by the former. *RKBS* compiles rough programs with quantitative measures to standard *Prolog* [NM95, DEBC96] programs. This opens for the use of *Prolog* built-in predicates and structured terms like lists. Note that each extended logic program, obtained by compiling a non-recursive rough program encoded in the language discussed in chapter 4, corresponds also to a *Prolog* program.

Another reason for considering only non-recursive rough programs is that the semantics of aggregates and stable models has been an open problem under investigation [Zan02, FLP04]. We avoid this problem because each non-recursive rough program with quantitative measures can be compiled to a logic program with at most one paraconsistent stable model.

The user interface of the system has been implemented in *Java* [SM]. We have chosen *XSB Prolog* system [Sys] to write the compiler and to reason with the compiled programs for the following reasons. First, *XSB Prolog* supports *definite clause grammars*. This fact simplifies the writing of a parser and compiler for a rough program. Second, *XSB* provides ISO-predicates such as `setof/3` and `findall/3`. These predicates can be used to implement aggregate functions, like `sum` and `count`. A third reason is that *XSB* allows the use of a technique called *tabling* when computing answers to queries for logic programs. Due to this fact, it is possible to obtain answers to queries for a large class of recursive logic programs, while more traditional *Prolog* systems based on SLDNF-resolution [Llo87] would simply loop forever. This class corresponds to non-floundering logic programs that enjoy the bounded term-depth property [CSW95, CW96, SS98, Swi99]. An important well-known subclass of this class of programs is *Datalog*. This aspect opens the future possibility of easily extending our system to applications that require a limited use of recursive rough programs. Fourth, version 2.6 of *XSB* has the *XASP* package that provides an efficient interface to *Smodels* [NS96, NS97, Sim] from *XSB* system. Note that *XSB* cannot be used to compute stable models, while *SModels* can do it. Thus, this connection between both systems could make possible to extend *RKBS* to support any recursive rough program. Finally, *XSB Prolog* is free and well-documented software.

Since we are presenting work in progress, we have restricted ourselves to the implementation issues of the language supported by *RKBS*. Hence, we do not formally present the declarative semantics for this language, as we did for the language introduced in chaper 4. A fuller description of the

[1] `sum` and `count` from SQL are examples of aggregate functions.

declarative semantics of rough programs with quantitative measures will be submitted for later publication.

As for the language discussed in chapter 4, predicates of a rough program supported by *RKBS* denote rough sets. However, the notion of rough set has been extended to account for quantitative measures. In section 6.1, we extend the notion of rough set and review some quantitative measures. We then present in section 6.2 the language supported by our system. Since the compilation of a rough program with quantitative measures generates a *Prolog* program where the special predicates `bagof/3` and `findall/3` occur, we informally introduce these standard *Prolog* predicates in section 6.3.1. Section 6.3.2 is devoted to the compilation of (non-recursive) rough programs with quantitative measures. The query language of the system is discussed in section 6.4. Finally, we describe some examples in section 6.5 .

6.1 Rough Sets Revisited

This section presents an extension of the rough set notion discussed previously in chapter 2 that explicitly takes into account quantitative measures. We then review some quantitative measures associated with rough sets in the context of our framework.

Recall that the set of values associated with an attribute a is denoted as V_a.

Definition 6.1 *Given a set of attributes* $A = \{a_1, \ldots, a_n\}$, *a* rough set *(or* rough relation*)* S *is a pair of sets* $(\overline{S}, \overline{\neg S})$ *satisfying conditions (i) and (ii).*

(i) The elements of sets $\overline{S}$ *and* $\overline{\neg S}$ *are expressions of the form*

$$\langle t_1, \ldots, t_n \rangle : k \ ,$$

where $\langle t_1, \ldots, t_n \rangle \in \prod_{a_i \in A} V_{a_i}$ *and* k *is an integer larger than zero.*

(ii) The following implications are true.

$$\begin{array}{lcl} \langle t_1, \ldots, t_n \rangle : k \in \overline{S} & \Rightarrow & \forall k' \neq k (\langle t_1, \ldots, t_n \rangle : k' \notin \overline{S}) \ , \\ \langle t_1, \ldots, t_n \rangle : k \in \overline{\neg S} & \Rightarrow & \forall k' \neq k (\langle t_1, \ldots, t_n \rangle : k' \notin \overline{\neg S}) \ . \end{array}$$

The rough complement *of a rough set* $S = (\overline{S}, \overline{\neg S})$ *is the rough set* $\neg S = (\overline{\neg S}, \overline{S})$.

For simplicity, we denote by t a general tuple $\langle t_1, \ldots, t_n \rangle$ and by $[t]$ the indiscernibility class described by tuple t. Moreover, we may also write $t \in \overline{S}$ ($t \in \underline{S}$ or $t \in \overline{\underline{S}}$ or $t \in \overline{\neg S}$ or $t \in \underline{\neg S}$ or $\overline{\underline{\neg S}}$), if the associated quantitative measure k is irrelevant.

Intuitively, an element $t : k \in \overline{S}$ ($t : k \in \overline{\neg S}$) indicates that the indiscernibility class described by the tuple t belongs to the upper approximation of the rough set S ($\neg S$) and that this class contains $k > 0$ individuals that are positive examples of the concept described by S ($\neg S$). *Lower approximation* of rough set S, represented $\underline{S}$, is then defined as

$$\underline{S} = \{t : k_1 \in \overline{S} \mid \forall k_2 > 0\, (t : k_2 \notin \overline{\neg S})\}$$

and the *boundary region*, represented $\overline{\underline{S}}$, is defined as

$$\overline{\underline{S}} \quad = \quad \{t : k_1 : k_2 \mid \exists k_1, k_2 > 0\, (t : k_1 \in \overline{S} \text{ and } t : k_2 \in \overline{\neg S})\}\ .$$

A rough set $D = (\overline{D}, \overline{\neg D})$, as defined above, can be seen as an alternative representation of a decision table $\mathcal{D} = (U, A, \mathtt{d})$. An expression $t : k_1 \in \overline{D}$ corresponds to $k_1 > 0$ lines t of the table with positive outcome for the decision attribute, while $t : k_2 \in \overline{\neg D}$ corresponds to $k_2 > 0$ lines t with negative outcome for the decision attribute. The fact that we consider only binary decision attributes is not a restriction in practice, as shown in chapter 5 (see example 5.4).

Recall that in our work a rough set is not defined in terms of individuals of the universe, but instead in terms of the tuples that describe each indiscernibility class to which the individuals belong.

Quantitative Measures

Let a tuple t be the description of an indiscernibility class $[t]$ of a decision table $\mathcal{D} = (U, A, \mathtt{d})$. Assume also that $|d|$ ($|\neg d|$) is the number of individuals (or lines of the table) that have positive (negative) outcome for the decision attribute $\mathtt{d}$. Thus, $|d| + |\neg d|$ is the number of objects (lines) in the corresponding table. The following quantitative measures are then defined.

- The *support* of $d(t)$, denoted $\mathtt{supp}(d(t))$, corresponds to the number of individuals in the indiscernibility class $[t]$ that are positive examples. Thus, if $t : k \in \overline{D}$ then $\mathtt{supp}(d(t)) = k$.
- The *strength* of $d(t)$, denoted $\mathtt{strength}(d(t))$, indicates how often individuals in the indiscernibility class $[t]$ have positive outcome for the decision attribute $\mathtt{d}$. Thus, if $t : k \in \overline{D}$ then $\mathtt{strength}(d(t)) = \frac{k}{|d|+|\neg d|}$.

- The *accuracy* of $d(t)$, denoted $\texttt{acc}(d(t))$, corresponds to the conditional probability $Pr(d(i) = \texttt{yes} \mid i \in [t])$. By other words, $\texttt{acc}(d(t))$ expresses how trustworthy the indiscernibility class described by t is in drawing the conclusion that the outcome for the decision attribute `d` is positive. Thus, if $t : k_1 \in \overline{D}$ and $t : k_2 \in \overline{\neg D}$ then, $\texttt{acc}(d(t)) = \frac{k_1}{k_1+k_2}$.
- The *coverage* of $d(t)$, denoted $\texttt{cov}(d(t))$, corresponds to the conditional probability $Pr(i \in [t] \mid d(i) = \texttt{yes})$. By other words, $\texttt{cov}(d(t))$ expresses how well the indiscernibility class $[t]$ describes the positive decision class. Thus, if $t : k \in \overline{D}$ then $\texttt{cov}(d(t)) = \frac{k}{|d|}$.

Obviously, the same measures can also be defined for $\neg d(t)$. For instance, if $t : k_1 \in \overline{D}$ and $t : k_2 \in \overline{\neg D}$ then $\texttt{acc}(\neg d(t)) = \frac{k_2}{k_1+k_2}$. Moreover, $\texttt{supp}(d(t)) + \texttt{supp}(\neg d(t)) = |[t]|$ and $\texttt{acc}(d(t)) + \texttt{acc}(\neg d(t)) = 1$.

In section 2.4, we introduce the notions of support, strength, accuracy, and coverage for decision rules. Let $\langle t_1, \ldots, t_n \rangle$ be a tuple describing an indiscernibility class of a rough relation D such that $A_i = att_D(i)$, with $1 \leq i \leq n$. If we interpret any statement

$$\begin{array}{ll} \langle t_1, \ldots, t_n \rangle \in \overline{D} & \text{or} \\ \langle t_1, \ldots, , t_n \rangle \in \overline{\neg D} & \end{array}$$

as a decision rule

$$\begin{array}{ll} (A_1 = t_1) \wedge \ldots \wedge (A_n = t_n) \longrightarrow (d = \texttt{yes}) & \text{or} \\ (A_1 = t_1) \wedge \ldots \wedge (A_n = t_n) \longrightarrow (d = \texttt{no}), & \end{array}$$

respectively, then we can easily see that the quantitative measures presented above correspond to the ones discussed in section 2.4.

6.2 A Language with Numerical Measures

In this section, we extend the language presented in chapter 4 with quantitative measures. We add now to the language the capability to keep track of the number of individuals that belong to each indiscernibility class of a rough relation. Moreover, quantitative measures such us support, strength, accuracy, and coverage, discussed in section 6.1, can be used to define a rough relation. We start by an informal introduction of the language.

A *rough program* is a set of *rough facts* and *rough clauses*. Rough facts encode rough relations defined explicitly by a decision table, while rough

clauses are used to define implicitly new rough relations obtained by combining different regions (e.g. lower approximation, upper approximation, and boundary) of other rough relations. For instance,

$$\overline{r}(c_1, c_2, c_3) : 5. \quad \text{and}$$
$$\overline{\neg r}(c_1, c_2, c_3) : 8.$$

are two rough facts. The first says that the indiscernibility class described by the tuple of attribute values $\langle c_1, c_2, c_3\rangle$ has 5 individuals. Moreover, these individuals are positive examples of the concept represented by the rough relation denoted by r, designated as R. The second rough fact states that the same indiscernibility class has 8 individuals that are negative examples of R (or positive examples of $\neg R$). Next, we give an example of a rough clause in the extended language.

$$\overline{p}(X_1, X_2) \ \texttt{:-}[\alpha, F] \ \underline{q}(X_1, X_2), \overline{\neg r}(X_1, X_2).$$

The expression to the right of $\texttt{:-}[\alpha, F]$ (i.e. $\underline{q}(X_1, X_2), \overline{\neg r}(X_1, X_2)$) is called the *body* and the expression to the left (i.e. $\overline{p}(X_1, X_2)$) is called the *head* of the rough clause. Moreover, α should be a rational number between 0 and 1 and F should be an associative and commutative binary function. (e.g. the minimum). If quantitative measures are ignored, the rough clause above can informally be interpreted in a way similar to the interpretation of rough clauses discussed in chapter 4. The expression $\underline{q}(X_1, X_2)$ can be seen as representing an indiscernibility class belonging to the lower approximation of rough relation Q, denoted by q. Note that X_1 and X_2 are variables that can be thought as representing any attribute value. Hence, the body of the rule above captures those indiscernibility classes $[\langle c_1, c_2\rangle]$ that are in the intersection of the lower approximation of the rough relation Q with the upper approximation of rough relation $\neg R$. Moreover, the rough rule above expresses that each of these indiscernibility classes belongs to the upper approximation of rough relation P, denoted by p.

Let us now intuitively explain how quantitative information is handled. Assume that there are two indiscernibility classes described by tuple $\langle c_1, c_2\rangle$: one indiscernibility class is part of $\underline{Q}$ and the other indiscernibility class belongs to $\overline{\neg R}$. Function F is then used to combine $\texttt{supp}(q(c_1, c_2))$ with the $\texttt{supp}(\neg r(c_1, c_2))$. Hence, the rough clause above states that, given a tuple $\langle c_1, c_2\rangle$ describing an indiscernibility class, if

$$\langle c_1, c_2\rangle : k_2 \in \underline{Q} \quad \text{and}$$
$$\langle c_1, c_2\rangle : k_3 \in \overline{\neg R}$$

then $\langle c_1, c_2\rangle : k_1 \in \overline{P}$, where

$$\texttt{supp}(p(c_1, c_2)) = k_1 \geq \lfloor \alpha \times F(k_2, k_3) \rfloor \ .$$

Note that the support k_1 should be computed by taking into account all clauses of a rough program, as shown in example 6.1. This is the reason for writing $k_1 \geq \lfloor \alpha \times F(k_2, k_3) \rfloor$ instead of writing $k_1 = \lfloor \alpha \times F(k_2, k_3) \rfloor$.

In contrast with the language presented in chapter 4, the head of a rough clause cannot refer to the boundary region of a rough relation, i.e. an expression as $\overline{p}(X_1, X_2)$ could not be the head of a rough clause. However, this is not a real restriction as shown in example 6.1. If rough literals referring to the boundary region were allowed in the head of a rough clause then we could not know how many individuals computed from the body would be positive examples and how many would be negative examples. This is the motivation behind this restriction. Moreover, no testing literals (e.g. $l?(t)$) can occur in the body of a rough clause, although this feature could be easily added.

We argue now on the usefulness of having user parameterized rough clauses H :-$[\alpha, F]$ $B.$, where α and F are the parameters. An example illustrating the importance of parameter α is when the user wants to decrease his trust on certain data. For example, assume that the user strongly doubts of the reliability of the information carried by 20% of the examples belonging to any indiscernibility class only with positive examples of Q and for which the second attribute has value c. A rough clause like

$$\underline{q_1}(X, c) \texttt{:-}[0.8, -]\ \underline{q}(X, c).$$

could be used to express such doubt. The new predicate q_1 denotes the same rough relation as q except that any indiscernibility class described by a tuple $\langle t_1, c\rangle \in \underline{Q_1}$ has only 80% of the individuals in the corresponding indiscernibility class $\langle t_1, c\rangle \in \underline{Q}$.

We also think that the way the support information, obtained from the expressions in the body, should be combined strongly depends on the application. For instance, if the user wants to represent the join of two decision tables then parameter F should correspond to the product function. But, if he wants to define a new rough relation R that captures those indiscernibility classes belonging to the same region of two rough relations P and Q (i.e. having the same description) then, it might make more sense to use the minimum function. The rough clauses below could together express this

idea.

$$\begin{array}{l}
\overline{r}(X_1, X_2) \ \texttt{:-}[1, \texttt{min}] \ \overline{p}(X_1, X_2), \overline{q}(X_1, X_2). \\
\underline{r}(X_1, X_2) \ \texttt{:-}[1, \texttt{min}] \ \underline{p}(X_1, X_2), \underline{q}(X_1, X_2). \\
\overline{\neg r}(X_1, X_2) \ \texttt{:-}[1, \texttt{min}] \ \overline{\neg p}(X_1, X_2), \overline{\neg q}(X_1, X_2). \\
\underline{\neg r}(X_1, X_2) \ \texttt{:-}[1, \texttt{min}] \ \underline{\neg p}(X_1, X_2), \underline{\neg q}(X_1, X_2).
\end{array}$$

Thus, if $t : k_1 \in \underline{P}$ and $t : k_2 \in \underline{Q}$ then one may conclude that, for both relations, there is an indiscernibility class described by tuple t in the lower approximation and $\texttt{supp}(r(t)) \geq \texttt{min}(k_1, k_2)$.

Example 6.1 *We give an example of a rough program $\mathcal{P}$ and discuss informally its meaning.*

$$\begin{array}{rcl}
\mathcal{P} & = & \{\overline{p}(X_1, X_2) \ \texttt{:-}[1, \textit{min}] \ \underline{q}(X_1, X_2), \overline{\neg r}(X_1, X_2). , \\
& & \overline{p}(X, c) \ \texttt{:-}[1, -] \ \overline{q_1}(X, c). , \\
& & \overline{\neg p}(X, c) \ \texttt{:-}[1, -] \ \overline{\neg q_1}(X, c). , \\
& & \overline{q}(a, c) : 2. , \\
& & \overline{r}(a, c) : 3. , \overline{\neg r}(a, c) : 4. , \\
& & \overline{q_1}(a, c) : 3. , \overline{\neg q_1}(a, c) : 7. \}
\end{array}$$

The body of the first rough clause represents the intersection of the lower approximation of the rough relation Q, denoted by q, with the boundary of rough relation $\neg R$, denoted by $\neg r$. From the facts of $\mathcal{P}$, we get that $\langle a, c\rangle : 2 \in \overline{Q}$ and $\langle a, c\rangle : 4 : 3 \in \overline{\neg R}$. Hence, from the first rough clause can be concluded that $\texttt{supp}(p(a, c)) \geq 1 \times \textit{min}(2, 4)$ ($\texttt{supp}(q(a, c)) = 2$ and $\texttt{supp}(\neg r(a, c)) = 4$).

The second and third rough clauses together state that if an indiscernibility class $[t]$ belongs to the boundary of rough relation $\neg Q_1$ and its second attribute has value c then, $[t]$ also belongs to the boundary of P. Moreover, the same number of positive and negative examples in $[t]$ should be inherited by rough relation P (i.e. $\texttt{supp}(q_1(a, c)) = 3$ individuals should be considered as representing positive examples of P, while $\texttt{supp}(\neg q_1(a, c)) = 7$ individuals should be considered as representing negative examples of P). Since the body of each of the rough clauses has only one expression, the choice of function F is irrelevant. This can be represented by the use of '$-$' instead of some concrete function.

Putting all together, it can be concluded from $\mathcal{P}$ that $\texttt{supp}(p(a, c)) = \textit{min}(2, 4) + 3 = 5$ and $\texttt{supp}(\neg p(a, c)) = 7$.

As this example shows, information concerning an indiscernibility class may be obtained independently from different rough clauses. For instance, information related with indiscernibility class $\langle a, c\rangle \in \overline{P}$ is obtained from the

three rough clauses of $\mathcal{P}$. Note that $\texttt{supp}(p(a,c))$ *is computed by summing the support obtained from different rough clauses.*

□

Another important aspect of the language is the possibility of using *quantitative measure expressions* in the body of a rough clause. For example,

$$\texttt{acc}(p(c_1,c_2)) > \texttt{acc}(\neg q(c_1,c_2)) \quad \text{and}$$
$$\texttt{supp}(\neg p(c_1,c_2)) > 7$$

are quantitative measure expressions. The first quantitative measure expression states that the accuracy of the indiscernibility class described by $\langle c_1, c_2\rangle$ of rough relation P is larger than the accuracy of the indiscernibility class $[\langle c_1, c_2\rangle]$ of rough relation $\neg Q$. The second states that indiscernibility class $[\langle c_1, c_2\rangle]$ has more than 7 individuals with negative outcome for the decision attribute `p`.

The next step is to define formally the language supported by *RKBS*, to encode rough programs. To this end, we first define the notion of quantitative measure expression.

Definition 6.2 *Assume that m stands either for* `supp`*, or* `strength`*, or* `acc`*, or* `cov` *and that relOp is one of the relation symbols $=, <, \leq, >, \geq, \neq$. A* quantitative measure expression *is a formula of the form*

$$\begin{array}{llll} m(l(t_1,\ldots,t_n)) & relOp & k & or \\ m(l_1(t_1,\ldots,t_n)) & relOp & m(l_2(t_1,\ldots,t_n)) \ , & \end{array}$$

where

- *l, l_1, and l_2 are either p or $\neg p$, for some predicate symbol p; and*
- *$n \geq 0$; and*
- *k is a positive rational number.*

Note that not all quantitative measure expressions are meaningful. For example, $\texttt{acc}(q(a,b)) > \texttt{supp}(r(a,b))$ is meaningless because it does not make sense to compare accuracy with support.

Definition 6.3 *A* rough clause in *RKBS is any expression of the form*

$$H \ \texttt{:-}[\alpha, F] \ \ B_1,\ldots,B_i, M_1,\ldots,M_k. \ ,$$

where

- *H is either $\bar{l}(t_1, \ldots, t_n)$ or $\underline{l}(t_1, \ldots, t_n)$, with $n \geq 0$ and l being either p or $\neg p$, for some predicate symbol p; and*
- *α is a rational number such that $0 < \alpha \leq 1$; and*
- *F is a commutative and associative binary function; and*
- *each B_j is a rough literal, with $1 \leq j \leq i$; and*
- *each M_j is a quantitative measure expression such that all variables occurring in M_j also occur in some rough literal in the body of the rough clause, with $1 \leq j \leq k$; and*
- *$i, k \geq 0$.*

As expected, a rough fact in *RKBS* is a rough clause with empty body (i.e. $i = k = 0$) and a rough program supported by *RKBS* is a finite set of non-recursive rough clauses (as defined in 6.3).

Each predicate occurring in a rough program, supported by *RKBS*, denotes a rough relation as defined in section 6.1 (see definition 6.1).

6.3 Compilation of *RKBS* Programs

Our system transforms a rough program into a *Prolog* program, where the special predicates `bagof/3` and `findall/3` occur. These predicates are used for the following purposes.

- To count the number of individuals in each indiscernibility class of a rough relation that are positive (negative) examples of the underlying concept.
- To count the number of individuals in the universe of a rough relation D that have positive (negative) outcome for the decision attribute `d` (i.e. to compute $|d|$ and $|\neg d|$).

We first informally introduce those special predicates. We then discuss the details about the compilation of (non-recursive) rough programs with quantitative measures.

6.3.1 All Solutions Predicates in *Prolog*

Any standard *Prolog* system [DEBC96] has two built-in predicates, `bagof/3` and `findall/3`, to collect together all solutions to a problem. We explain their meaning through a couple of examples.

Example 6.2 *Consider the following (definite) logic program containing a number of facts about employees and their salaries.*

```
P = {salary(peter, 100).,
     salary(terry, 150).,
     salary(john, 100).,
     salary(susan, 150).,
     who(X, L) :- bagof(Y, salary(Y,X), L).
```

For instance, the fact `salary(peter, 100).` *states that "Peter's salary is 100.".*

The query

$$(\texttt{who(150, L)}, \mathcal{P})$$

requests a list of all employees who earn 150. *The answer is the set of substitutions*

$$\{\{[\ \texttt{terry}, \texttt{susan}\ ]/L\}\}\ .$$

Note that a list of items is represented between square brackets, '[' and ']'. Thus, [`terry`, `susan`] *represent the list with constants* `terry` *and* `susan`.

The query

$$(\texttt{who(X, L)}, \mathcal{P})$$

requests all salaries together with a list of people who earn each salary. The answer is the set of substitutions

$$\{\{100/X, [\ \texttt{peter}, \texttt{jonh}\ ]/L\}, \{150/X, [\ \texttt{terry}, \texttt{susan}\ ]/L\}\}\ .$$

□

The atom `bagof(Template, Goal, List)` can be interpreted as the "`List` of all instances of `Template` such that `Goal` is satisfied". Note that `List` may contain duplicates, if the same instance of `Template` can be proved in several ways.

Next example illustrates the use of predicate `findall`.

Example 6.3 *Consider the following (definite) logic program.*

```
P = {salary(peter, 100).,
     salary(terry, 150).,
     salary(john, 100).,
     salary(susan, 150).,
     allSalaries(L) :- findall(X, salary(Y,X), L).,
     total(T) :- findall(X, salary(Y,X), L), sum(L, T).}
```

Assume that predicate `sum(L, T)` *has the following meaning: "*`T` *is the sum of all numbers in list* `L`*". For example, the atom* `sum([ 2, 5, 4 ], 11)` *is true. This predicate can be easily defined in Prolog.*

The query

$$(\texttt{allSalaries(L)},\ \mathcal{P})$$

requests a list of all salaries paid to the employees. The answer is the set of substitutions

$$\{\{[\,100, 150, 100, 150\,]/L\}\}\ .$$

The query

$$(\texttt{total(T)},\ \mathcal{P})$$

requests the total amount spent in salaries. The answer is the set of substitutions $\{\{500/T\}\}$.

□

An empty list (i.e. a list with no elements) is represented as "[]". Moreover, the atom `sum`$([], 0)$ is always true.

The main difference between

```
bagof(Template, Goal, List)     and
findall(Template, Goal, List)
```

is that the former may produce as answer a set with several substitutions, if there are variables that occur in `Goal` but do not appear in `Template`, while the latter produces only singleton answers. A query

$$(\texttt{bagof(Template, Goal, List)},\ \mathcal{P})$$

produces a substitution for each possible instantiation of the variables of `Goal` that do not appear in `Template`.

6.3.2 The Compilation

For each indiscernibility class t of a rough relation R, we need now to compute the support measures, i.e. $\texttt{supp}(r(t))$ and $\texttt{supp}(\neg r(t))$. This point is complicated by the following. By applying one rough clause of the program, we may conclude that a certain number $k_1 > 0$ of individuals belong to $[t]$ and that they are positive examples. It may also be the case that by applying another rough clause of the same program, we conclude that other $k_2 > 0$ individuals belong to the same indiscernibility class and that they are also positive examples. Hence to compute the number of individuals belonging to $[t]$ that are positive (negative) examples of a rough relation, we may need to consider different rough clauses. To this end, the special atoms introduced in the previous section are used in the transformed program.

We start by giving an overview of the compilation procedure. Figure 6.1 shows the different functions that are called during compilation of a rough program in *RKBS*.

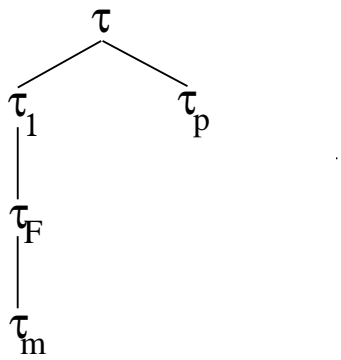


Figure 6.1: Compilation procedure of *RKBS*.

Function τ compiles a rough program $\mathcal{P}$ to a program $\tau(\mathcal{P})$ corresponding to a standard *Prolog* program. It calls two other functions, τ_1 and τ_p. Function τ_1 compiles each rough clause to a set of clauses and integrity constraints. τ_p generates a set of clauses for each predicate symbol q occurring in $\mathcal{P}$ that gather support information for each indiscernibility class of rough relation Q. The compilation $\tau_1(C)$ of a rough clause $C \in \mathcal{P}$ originates a call to function τ_F to compile the body of C. If the body of C contains

quantitative measure expressions then τ_F calls function τ_m to compile each quantitative measure. In the reminder of this section, we discuss in detail each of the functions mentioned above.

Without loss of generality, we assume that each rough relation defined in a rough program $\mathcal{P}$ has a different name. Thus, no two equal predicate symbols with different arity may occur in $\mathcal{P}$.

To simplify the presentation of the compilation procedure, we use some notation shortcuts.

- The compiled programs may contain clauses with explicit disjunction, represented as ';', in the body of a clause. These clauses can easily be re-written as clauses without disjunction in the body. For example, the clause

  $$p(X) \texttt{:-}\ q(a, X), r(X)\ ;\ q(b, X).$$

  can be replace by the following two clauses

  $$p(X) \texttt{:-}\ q(a, X), r(X).$$
  $$p(X) \texttt{:-}\ q(b, X).$$

- If we are not interested in the constant value with which a variable X gets instantiated then, we use '$-$' instead of X. For instance, if the value with which the first argument of q is instantiated is irrelevant then we write $q(-, Y)$.

- We may write $q(\overline{X})$ instead of $q(X_1, \ldots, X_n)$, for a predicate q/n.

- Given a pair $u = \langle c_1, c_2 \rangle$, we write $u.1st$ to denote c_1 and $u.2nd$ to denote c_2.

It should be stressed that the compiled program $\tau(\mathcal{P})$ may contain integrity constraints, given a rough program $\mathcal{P}$. Integrity constraints are not allowed in standard *Prolog* programs. However, each logic program $\tau(\mathcal{P})$ can be easily transformed into a standard *Prolog* program. Assume that no predicate symbol named `false` occurs in $\tau(\mathcal{P})$. Each integrity constraint `:- B.` $\in \tau(\mathcal{P})$ can be replaced by a clause `false :- B.` . If the atom `false` belongs to the least model of the *Prolog* program obtained this way then we can conclude that some integrity constraint cannot be satisfied.

Compiling a rough program $\mathcal{P}$ implies compilation of each rough clause and rough fact. For each predicate symbol p occurring in $\mathcal{P}$, p^*, p^π, $\neg p$, $\neg p^*$, and $\neg p^\pi$ should be seen as new predicate symbols not occurring in $\mathcal{P}$. Note that $\neg p$, $\neg p^*$, and $\neg p^\pi$ represent explicit negation.

Each predicate p in the compiled program has an extra argument that carries information about the support. For example, both atoms $p(c_1, c_2, k)$ and $p^*(c_1, c_2, k)$ ($\neg p(c_1, c_2, k)$ and $\neg p^*(c_1, c_2, k,)$) indicate that the indiscernibility class $[\langle c_1, c_2\rangle]$ belongs to $\overline{P}$ ($\overline{\neg P}$). However, the former states that the $\mathtt{supp}(p(c_1, c_2))$ ($\mathtt{supp}(\neg p(c_1, c_2))$) is exactly k, while the latter says that $\mathtt{supp}(p(c_1, c_2))$ ($\mathtt{supp}(\neg p(c_1, c_2))$) is at least k. An atom $p^\pi(c_1, c_2)$ ($\neg p^\pi(c_1, c_2)$) indicates that the indiscernibility class $[\langle c_1, c_2\rangle]$ belongs to the upper approximation of P ($\neg P$) but it does not keep any information about the support.

In the reminder of this section we assume that l (l_1, l_2, $\cdots$) is either p or $\neg p$ and that $\neg\neg p$ is equivalent to p, for some predicate symbol p.

We introduce first a function τ_m to compile quantitative measures in the body of a rough clause. For example, the compilation of a quantitative measure such as $\tau_m(\mathtt{supp}(q(t)))$ returns a pair u, where

- $u.1st$ is the body of a clause, and
- $u.2nd$ is a variable that will be instantiated with the support of atom $q(t)$.

If t describes an indiscernibility class of rough relation Q then literal $q(t, K)$ expresses that the $\mathtt{supp}(q(t)) = K$. The default negated literal $\mathtt{not}\ \underline{q}^\pi(t)$ is true, if t describes an indiscernibility class that does not belong to $\underline{Q}$. The conjunction of atoms

$$\mathtt{findall}(K_1, q(\overline{X}, K_1), L_1), \mathtt{sum}(L_1, K')$$

imposes that $K' = |q|$, i.e. K' represents the number of individuals in the universe of rough relation Q that have positive outcome for the decision attribute. Hence, the following conjunction of atoms expresses that K is the total number of individuals in universe of Q.

$$\begin{array}{l}\mathtt{findall}(K_1, q(\overline{X}, K_1), L_1),\\ \mathtt{findall}(K_2, \neg q(\overline{X}, K_2), L_2),\\ \mathtt{sum}(L_1, K_1'), \mathtt{sum}(L_2, K_2'), K = K_1' + K_2'\end{array}$$

Function τ_m that compiles quantitative measures is formally presented below.

$$
\begin{array}{rcl}
\tau_m(\texttt{supp}(l(t))) & = & \langle \texttt{code}\ ,\ K \rangle \\
\texttt{code} & = & l(t,K)\ ; \\
 & & \texttt{not}\ l^{\pi}(\bar{t}), K = 0
\end{array}
$$

$$
\begin{array}{rcl}
\tau_m(\texttt{strength}(l(\bar{t}))) & = & \langle \texttt{code}\ ,\ K \rangle \\
\texttt{code} & = & l(t,K_0), \\
 & & \texttt{findall}(K_1, l(\overline{X}, K_1), L_1), \\
 & & \texttt{findall}(K_2, \neg l(\overline{X}, K_2), L_2), \\
 & & \texttt{sum}(L_1, K_1'), \texttt{sum}(L_2, K_2'), K = \frac{K_0}{K_1'+K_2'}\ ; \\
 & & \texttt{not}\ l^{\pi}(t), K = 0
\end{array}
$$

$$
\begin{array}{rcl}
\tau_m(\texttt{acc}(l(t))) & = & \langle \texttt{code}\ ,\ K \rangle \\
\texttt{code} & = & l(t,K_1), \neg l(t,K_2), K = \frac{K_1}{K_1+K_2}\ ; \\
 & & l(t,K_1),\ \texttt{not}\ \neg l^{\pi}(t), K = 1\ ; \\
 & & \texttt{not}\ l^{\pi}(t), K = 0
\end{array}
$$

$$
\begin{array}{rcl}
\tau_m(\texttt{cov}(l(\bar{t}))) & = & \langle \texttt{code}\ ,\ K \rangle \\
\texttt{code} & = & l(t,K_1) \\
 & & \texttt{findall}(K_2, l(\overline{X}, K_2), L), \texttt{sum}(L, K_3), K = \frac{K_1}{K_3}\ ; \\
 & & \texttt{not}\ l^{\pi}(t), K = 0\ .
\end{array}
$$

Let us informally describe the first two cases above.

- The compilation of $\texttt{supp}(l(t))$ considers two possible cases. If atom $l(t,K)$ is true then $\texttt{supp}(l(t)) = K$. If t does not describe an indiscernibility class in the upper approximation, i.e. $\texttt{not}\ l^{\pi}(t)$ is true, then $\texttt{supp}(l(t)) = K = 0$.

- The compilation of $\texttt{strength}(l(t))$ considers also two cases. If t describes an indiscernibility class in the upper approximation then, atom $l(t,K_0)$ is true, $\texttt{supp}(l(t)) = K_0$, and the number of individuals in the universe is given by $K_1' + K_2'$. Otherwise, the default negated literal $\texttt{not}\ l^{\pi}(t)$ is true and $\texttt{supp}(l(t)) = \texttt{strength}(l(t)) = 0$.

The next step is to show how to compile the body of a rough clause (with quantitative measures). Let $\mathbf{I}^+$ denote the set of positive integers including zero and F be a commutative and associative binary function such that $F : \mathbf{I}^+ \rightarrow \mathbf{I}^+$. This function indicates how the support of the atoms $p(t)$

or $\neg p(t)$, occurring in rough literals of the body, should be combined to compute the support of the atom corresponding to the head. Given the body B of a rough clause, $\tau_F(B)$ returns a pair such that $(\tau_F(B)).1st$ is the body of a clause of a logic program and $(\tau_F(B)).2nd$ represents a variable. Assume that m is either `supp`, `strength`, `acc`, or `cov`. E_1 and E_2 represent quantitative measure expressions. B_1 and B_2 stand for rough literals.

$$
\begin{array}{rcl}
\tau_F(m(l(\bar{t}))\ relOp\ k) & = & \langle \texttt{code}\ ,\ - \rangle \\
\texttt{code} & = & \tau_m(m(l(\bar{t}))).1st, \\
 & & \tau_m(m(l(\bar{t}))).2nd\ \ relOp\ \ k
\end{array}
$$

$$
\begin{array}{rcl}
\tau_F(m(l_1(\overline{t_1}))\ relOp\ m(l_2(\overline{t_2}))) & = & \langle \texttt{code}\ ,\ - \rangle \\
\texttt{code} & = & \tau_m(m(l_1(\overline{t_1}))).1st, \tau_m(m(l_2(\overline{t_2}))).1st, \\
 & & \tau_m(m(l_1(\overline{t_1}))).2nd \\
 & & \qquad relOp \\
 & & \tau_m(m(l_2(\overline{t_2}))).2nd
\end{array}
$$

$$
\begin{array}{rcl}
\tau_F((E_1, E_2)) & = & \langle \texttt{code}\ ,\ - \rangle \\
\texttt{code} & = & (\tau_F(E_1)).1st, (\tau_F(E_2)).1st
\end{array}
$$

$$
\begin{array}{rcl}
\tau_F(\underline{p}(\bar{t})) & = & \langle \texttt{code}\ ,\ K \rangle \\
\texttt{code} & = & p(\bar{t}, K), \texttt{not}\ \neg p^{\pi}(\bar{t})
\end{array}
$$

$$
\begin{array}{rcl}
\tau_F(\underline{\neg p}(\bar{t})) & = & \langle \texttt{code}\ ,\ K \rangle \\
\texttt{code} & = & \neg p(\bar{t}, K), \texttt{not}\ p^{\pi}(\bar{t})
\end{array}
$$

$$
\begin{array}{rcl}
\tau_F(\overline{p}(\bar{t})) & = & \langle \texttt{code}\ ,\ K \rangle \\
\texttt{code} & = & p(\bar{t}, K)
\end{array}
$$

$$
\begin{array}{rcl}
\tau_F(\overline{\neg p}(\bar{t})) & = & \langle \texttt{code}\ ,\ K \rangle \\
\texttt{code} & = & \neg p(\bar{t}, K)
\end{array}
$$

$$
\begin{array}{rcl}
\tau_F(\overline{\underline{p}}(\bar{t})) & = & \langle \texttt{code}\ ,\ K \rangle \\
\texttt{code} & = & p(\bar{t}, K), \neg p^{\pi}(\bar{t})
\end{array}
$$

$$
\begin{array}{rcl}
\tau_F(\overline{\underline{\neg p}}(\bar{t})) & = & \langle \texttt{code}\ ,\ K \rangle \\
\texttt{code} & = & \neg p(\bar{t}, K), p^{\pi}(\bar{t})
\end{array}
$$

$$\begin{array}{rcl}\tau_F((B_1,B_2)) & = & \langle \texttt{code}\ ,\ K\rangle \\ \texttt{code} & = & (\tau_F(B_1)).1st, (\tau_F(B_2)).1st, \\ & & K = F((\tau_F(B_1)).2nd, (\tau_F(B_2)).2nd)\end{array}$$

$$\begin{array}{rcl}\tau_F((B_1,E_1)) & = & \langle \texttt{code}\ ,\ K\rangle \\ \texttt{code} & = & (\tau_F(B_1)).1st, \tau_m(E_1).1st, \\ & & K = (\tau_F(B_1)).2nd\ .\end{array}$$

The reader can confirm that the underlying idea to compile the body of a rough clause is similar to what was discussed in chapter 4.

We need to define a function that compiles a rough clause into a set of clauses and integrity constraints. This function is presented below and it is based on ideas similar to the ones discussed in chapter 4.

$$\begin{array}{rcl}\tau_1(\underline{p}(\bar{t})\texttt{:-}[\alpha,F]\ \ B.) & = & \{p^*(\bar{t},K)\texttt{:-}\ (\tau_F(B)).1st, \\ & & \qquad K = \lfloor \alpha \times (\tau_F(B)).2nd \rfloor .\ , \\ & & \ \texttt{:-}\ \neg p^\pi(\bar{t}), (\tau_F(B)).1st.\}\ , \\ \tau_1(\overline{p}(\bar{t})\texttt{:-}[\alpha,F]\ \ B.) & = & \{p^*(\bar{t},K)\texttt{:-}\ (\tau_F(B)).1st, \\ & & \qquad K = \lfloor \alpha \times (\tau_F(B)).2nd \rfloor .\}\ , \\ \tau_1(\underline{\neg p}(\bar{t})\texttt{:-}[\alpha,F]\ \ B.) & = & \{\neg p^*(\bar{t},K)\texttt{:-}\ (\tau_F(B)).1st, \\ & & \qquad K = \lfloor \alpha \times (\tau_F(B)).2nd \rfloor .\ , \\ & & \ \texttt{:-}\ p^\pi(\bar{t}), (\tau_F(B)).1st.\}\ , \\ \tau_1(\overline{\neg p}(\bar{t})\texttt{:-}[\alpha,F]\ \ B.) & = & \{\neg p^*(\bar{t},K)\texttt{:-}\ (\tau_F(B)).1st, \\ & & \qquad K = \lfloor \alpha \times (\tau_F(B)).2nd \rfloor .\}\ .\end{array}$$

The compilation of a rough clause might generate clauses with repeated literals. Thus, it is convenient to eliminate repeated literals from the bodies of compiled clauses.

Let $[t]$ be an indiscernibility class of a rough relation R. Application of different rough clauses may lead to the conclusion that a certain number of individuals belonging to $[t]$ are positive examples of a rough relation R ($\neg R$). Hence, it is needed to gather support information for each indiscernibility class of a rough relation. Function τ_p formalizes this idea. For each predicate symbol r occurring in the rough program, the following set of clauses is generated.

$$\begin{array}{lcl}\tau_p(r) & = & \{r(\overline{X},K)\texttt{:- bagof}(K',r^*(\overline{X},K'),L),\texttt{sum}(L,K).,\\ & & \neg r(\overline{X},K)\texttt{:- bagof}(K',\neg r^*(\overline{X},K'),L),\texttt{sum}(L,K).,\\ & & r^\pi(\overline{X})\texttt{:- }r(\overline{X},-).,\\ & & \neg r^\pi(\overline{X})\texttt{:- }\neg r(\overline{X},-).\}\ .\end{array}$$

Functions τ_1 and τ_p can also be applied to a rough program $\mathcal{P}$. Let $\beta_\mathcal{P}$ be the set of all predicate symbols occurring in $\mathcal{P}$.

$$\begin{array}{l}\tau_1(\mathcal{P}) = \bigcup_{C\,\in\,\mathcal{P}}\,\tau_1(C)\ ,\\ \tau_p(\mathcal{P}) = \bigcup_{p\,\in\,\beta_\mathcal{P}}\,\tau_p(p).\end{array}$$

Compilation of a rough program $\mathcal{P}$ is obtained by applying function τ to $\mathcal{P}$,

$$\tau(\mathcal{P}) = \tau_1(\mathcal{P}) \cup \tau_p(\mathcal{P})\ .$$

The example below illustrates the compilation of a simple rough program in *RKBS*.

Example 6.4 *Consider again the program*

$$\begin{array}{lcl}\mathcal{P} & = & \{\overline{p}(X_1,X_2)\ \texttt{:-[1,-]}\ \overline{q}(X_1,X_2),\texttt{acc}(q(X_1,X_2)) > 0.85.,\\ & & \overline{p}(X,c)\ \texttt{:-[1,-]}\ \overline{q_1}(X,c).,\\ & & \overline{\neg p}(X,c)\ \texttt{:-[1,-]}\ \overline{\neg q_1}(X,c).,\\ & & \overline{q}(a,c) : 2.,\\ & & \overline{q_1}(a,c) : 3.,\overline{\neg q_1}(a,c) : 7.\}\end{array}$$

- **Compilation of the first rough clause**

  $$\tau_1(\overline{p}(X_1,X_2)\ \texttt{:-[1,-]}\ \overline{q}(X_1,X_2),\texttt{acc}(q(X_1,X_2)) > 0.85.)$$

  adds the following clauses to $\tau(\mathcal{P})$.

  $$\begin{array}{l}p^*(X_1,X_2,K_1)\texttt{:- }q(X_1,X_2,K_1),\neg q(X_1,X_2,K_2),\\ \qquad\qquad K_3 = \frac{K_1}{K_1+K_2}, K_3 > 0.85.\end{array}$$

  $$\begin{array}{l}p^*(X_1,X_2,K_1)\texttt{:- }q(X_1,X_2,K_1),\mathit{not}\ \neg q^\pi(X_1,X_2),\\ \qquad\qquad K_2 = 1, K_2 > 0.85.\end{array}$$

  $$\begin{array}{l}p^*(X_1,X_2,K_1)\texttt{:- }q(X_1,X_2,K_1),\mathit{not}\ q^\pi(X_1,X_2),\\ \qquad\qquad K_2 = 0, K_2 > 0.85.\end{array}$$

- **Compilation of the second rough clause**

$$\tau_1(\overline{p}(X,c) \ \ \texttt{:-}[1,-] \ \ \underline{\overline{q_1}}(X,c).)$$

adds the following clause to $\tau(\mathcal{P})$.

$$p^*(X,c,K) \texttt{:-}\ q_1(X,c,K), \neg q_1^\pi(X,c).$$

- **Compilation of the third rough clause**

$$\tau_1(\overline{\neg p}(X,c) \ \ \texttt{:-}[1,-] \ \ \underline{\overline{\neg q_1}}(X,c).)$$

adds the following clause to $\tau(\mathcal{P})$.

$$\neg p^*(X,c,K) \texttt{:-}\ \neg q_1(X,c,K), q_1^\pi(X,c).$$

- **Compilation of the rough facts** *adds the following facts to* $\tau(\mathcal{P})$.

$$q^*(a,c,2).$$

$$q_1^*(a,c,3).$$

$$\neg q_1^*(a,c,7).$$

Finally, function τ_p *is called for each predicate symbol.*

$$\begin{array}{lcl}
\tau_p(p) & = & \{p(X_1,X_2,K) \texttt{:-}\ \texttt{bagof}(K', p^*(X_1,X_2,K'),L), \texttt{sum}(L,K).\,, \\
& & \ \neg p(X_1,X_2,K) \texttt{:-}\ \texttt{bagof}(K', \neg p^*(X_1,X_2,K'),L), \texttt{sum}(L,K).\,, \\
& & \ p^\pi(X_1,X_2) \texttt{:-}\ p(X_1,X_2,-).\,, \\
& & \ \neg p^\pi(X_1,X_2) \texttt{:-}\ \neg p(X_1,X_2,-).\}\ . \\
\tau_p(q) & = & \{q(X_1,X_2,K) \texttt{:-}\ \texttt{bagof}(K', q^*(X_1,X_2,K'),L), \texttt{sum}(L,K).\,, \\
& & \ \neg q(X_1,X_2,K) \texttt{:-}\ \texttt{bagof}(K', \neg q^*(X_1,X_2,K'),L), \texttt{sum}(L,K).\,, \\
& & \ q^\pi(X_1,X_2) \texttt{:-}\ q(X_1,X_2,-).\,, \\
& & \ \neg q^\pi(X_1,X_2) \texttt{:-}\ \neg q(X_1,X_2,-).\}\ . \\
\tau_p(q_1) & = & \{q_1(X_1,X_2,K) \texttt{:-}\ \texttt{bagof}(K', q_1^*(X_1,X_2,K'),L), \texttt{sum}(L,K).\,, \\
& & \ \neg q_1 X_1,X_2,K) \texttt{:-}\ \texttt{bagof}(K', \neg q_1^*(X_1,X_2,K'),L), \texttt{sum}(L,K).\,, \\
& & \ q_1^\pi(X_1,X_2) \texttt{:-}\ q_1(X_1,X_2,-).\,, \\
& & \ \neg q_1^\pi(X_1,X_2) \texttt{:-}\ \neg q_1(X_1,X_2,-).\}\ .
\end{array}$$

□

6.4 The Query Language of *RKBS*

Our system allows rough programs to be queried. Quantitative measure expressions may also appear in queries.

A rough query is a pair consisting of the query itself and a rough program from which an answer should be retrieved. We start by formally defining the query language of the system.

Definition 6.4 *A* rough query *in RKBS is a pair* $(\mathcal{Q}, \mathcal{P})$*, where* $\mathcal{P}$ *is a rough program supported by RKBS and* $\mathcal{Q}$ *is defined by the following abstract syntax rules*

$$
\begin{array}{lcl}
\mathit{RelOp} & \longrightarrow & = = | \neq | > | \geq | < | \leq \ . \\
\mathcal{Q}_1 & \longrightarrow & K = M \mid K = M, \mathcal{Q}_1 \mid K = M, K \ \ \mathit{RelOp} \ \ k, \mathcal{Q}_1 \mid \\
 & & K_1 = M, K_2 = M, K_1 \ \ \mathit{RelOp} \ \ K_2, \mathcal{Q}_1 \ . \\
\mathcal{Q}_2 & \longrightarrow & \mathtt{classify}(A) \ . \\
\mathcal{Q}_3 & \longrightarrow & L \mid L, \mathcal{Q}_3 \mid \mathcal{Q}_1 \ . \\
\mathcal{Q} & \longrightarrow & \mathcal{Q}_3 \mid \mathcal{Q}_2 \ .
\end{array}
$$

where M *is a quantitative measure,* K, K_1, K_2 *are variables,* k *is a rational number,* L *is a rough literal, and* A *is an objective literal. A rough query is* well-formed*, if all variables occurring in a quantitative measure expression also occur in some other rough literal of the query.*

In contrast to the query language presented in chapter 4, it is not possible to test whether a region of a rough relation (e.g. lower approximation) is a subset of another region of some other rough relation. Thus, rough inclusion and rough equality cannot be tested, either. On the other side, the system supports queries of the form $(\mathtt{classify}(d(t)) \ , \ \mathcal{P})$.

We describe informally the meaning of some (well-formed) rough queries. Consider the rough query

$$(\mathtt{classify}(d(c_1, X, c_3)) \ \mathtt{,} \ \mathcal{P}) \ .$$

Each tuple $t = \langle c_1, c_2, c_3 \rangle$ describing an indiscernibility class of a rough relation D can be seen as a decision rule. Assume that rough relation D corresponds (implicitly) to the decision table $\mathcal{D} = (U, \{a_1, a_2, a_3\}, \mathtt{d})$. If $t \in \overline{D}$ then it induces the decision rule $(a_1 = c_1) \wedge (a_2 = c_2) \wedge (a_3 = c_3) \rightarrow (\mathtt{d} = \mathtt{yes})$. If $t \in \overline{\neg D}$ then it induces the decision rule $(a_1 = c_1) \wedge (a_2 = c_2) \wedge (a_3 = c_3) \rightarrow (\mathtt{d} = \mathtt{no})$. The query above requests a prediction for the decision class to which a new individual i described by $(a_1 = c_1 \wedge a_3 = c_3)$

may belong. To answer this query the strategy described in section 2.5 is followed. The answer to the rough query is either the pair $(\mathtt{d} = \mathtt{yes}, C_F)$, or $(\mathtt{d} = \mathtt{no}, C_F)$, or $(\mathtt{d} = \mathtt{unknown}, 0)$, where C_F is the certainty factor of the prediction. The last case corresponds to the situation where no decision rule is fired.

Consider another rough query

$$(\underline{p}(X_1, X_2), K_1 = \mathtt{supp}(p(X_1, X_2)), K_2 = \mathtt{supp}(\neg p(X_1, X_2)) \ , \ \mathcal{P}) \ .$$

This rough query requests the description of all indiscernibility classes in the boundary region of P with indication, for each indiscernibility class, of how many individuals of that class are positive examples and how many individuals are negative examples. Hence, the substitution

$$\{a/X_1, b/X_2, K_1/5, K_2/7\}$$

could be an answer stating that $\langle a, b\rangle \in \overline{P}$, $\mathtt{supp}(p(a, b)) = 5$, and $\mathtt{supp}(\neg p(a, b)) = 7$.

The rough query

$$(\overline{p}(X_1, b), K = \mathtt{acc}(p(X_1, b)), K > 0.6 \ , \ \mathcal{P})$$

requests a description of all indiscernibility classes in the upper approximation of P such that the second attribute has value b and their corresponding accuracy is larger than 0.6.

The system answers a rough query for a rough program $\mathcal{P}$ by compiling it to one or more *Prolog* queries to the *Prolog* program corresponding to $\tau(\mathcal{P})$. The compilation of rough queries is based on ideas similar to the compilation functions τ_m and τ_F, presented in section 6.3.2. Answers to (rough) queries are sets of substitutions, like for the query language introduced in chapter 4. We give an example showing how a rough query can be answered by querying the compiled program.

Example 6.5 *Consider the rough query*

$$(\underline{p}(X_1, b), K = \mathit{supp}(r(X_1, b)), K < 10 \ , \ \mathcal{P})$$

The following two queries are generated for the compiled program.

$(p^\pi(X_1, b), \neg p^\pi(X_1, b), r(X_1, b, K), K < 10 \ , \ \tau(\mathcal{P}))$ *and*

$(p^\pi(X_1, b), \neg p^\pi(X_1, b), \mathit{not}\ r^\pi(X_1, b), K = 0, K < 10 \ , \ \tau(\mathcal{P}))$.

The union of the sets representing the answers to the queries above is the answer to the initial rough query.

□

As the example above shows, a rough query may be answered by issuing more than one query to the compiled program. The compilation of quantitative measures (see definition of function τ_m) involves more than one possible case. This explains why more than one query might be needed. Assume that variable X_1 is instantiated with a constant c. In the example above, the first query, obtained by compiling the rough query, corresponds to the case where the indiscernibility class described by $\langle c, b\rangle \in \overline{R}$, i.e. $\mathtt{supp}(r(c,b)) > 0$. The second query corresponds to the case where $\langle c, b\rangle \notin \overline{R}$, i.e. $\mathtt{supp}(r(c,b)) = 0$.

6.5 Application Examples

6.5.1 Variable Precision Rough Relations

We show below how quantitative measure expressions in the body of rough clauses can be used to build generalized rough approximations of a relation, in the spirit of the variable precision rough set model [Zia93], described in section 2.6. This aspect illustrates an important application of our language, since the VPRSM is the rough set model often used in practical applications.

Example 6.6 *Let q denote a rough relation Q (possibly obtained directly from a decision table and encoded as a set of rough facts in our language) , l and u be two precision control parameters such that $l < u$.*

We define then a new rough relation Q_1 such that

- *if $\mathtt{acc}(q(t)) \geq u$ then $t \in \underline{Q_1}$;*
- *if $\mathtt{acc}(q(t)) \leq l$ then $t \in \underline{\neg Q_1}$;*
- *If $l < \mathtt{acc}(q(t)) < u$ then $t \in \overline{\underline{Q_1}}$.*

$\overline{q_1}(X_1, X_2)$:-[1, −] $\underline{q}(X_1, X_2)$.

$\overline{\neg q_1}(X_1, X_2)$:-[1, −] $\underline{\neg q}(X_1, X_2)$.

%*Any indiscernibility class t in the boundary s.t.*
% $acc(q(t)) \geq u$ *is considered to be in* $\underline{Q}$
$\underline{q_1}(X_1, X_2)$:-[1, *sum*] $\overline{q}(X_1, X_2), \overline{\neg q}(X_1, X_2), \mathtt{acc}(q(X_1, X_2)) \geq u$.

% *Any indiscernibility class t in the boundary s.t.*
% $acc(q(t)) \leq l$ *is considered to be in* $\neg\underline{Q}$
$\underline{\neg q_1}(X_1, X_2)$:-[1, *sum*] $\overline{q}(X_1, X_2), \overline{\neg q}(X_1, \overline{X_2}), \mathtt{acc}(q(X_1, X_2)) \leq l$.

% *Any indiscernibility class t in the boundary s.t.*
% $l < acc(q(t)) < u$ *remains in the boundary*
$\overline{q_1}(X_1, X_2)$:-[1, −] $\overline{q}(X_1, X_2), \mathtt{acc}(q(X_1, X_2)) > l, \mathtt{acc}(q(X_1, X_2)) < u$.

$\overline{\neg q_1}(X_1, X_2)$:-[1, −] $\overline{\neg q}(X_1, X_2), \mathtt{acc}(q(X_1, X_2)) > l, \mathtt{acc}(q(X_1, X_2)) < u$.

Note that the use of $\overline{q}(X_1, X_2), \overline{\neg q}(X_1, X_2)$ *in the body of the third and fourth rough clauses captures those indiscernibility classes* $[t]$ *in the boundary of* Q*. Moreover, it is worth to note the use of function* `sum` *to combine* $\mathtt{supp}(q(t))$ *with* $\mathtt{supp}(\neg q(t))$*, since* $\mathtt{supp}(q(t)) + \mathtt{supp}(\neg q(t))$ *gives the total number of individuals in the indiscernibility class* $[t]$*.*

□

The rough program above shows that our framework caters for extending the VPRSM to implicitly defined rough relations.

6.5.2 Avoiding Expensive Tests Revisited

In section 5.2, we describe a possible technique to identify those individuals for who expensive tests, corresponding to some of the condition attributes, can (cannot) be avoided. We show now the same problem formulated in the language supported by our system. In addition, we illustrate how quantitative measure expressions can be used to retrieve relevant information.

Example 6.7 *Consider the decision table* $\mathcal{D}eathmi = (U, \{\mathtt{Age}, \mathtt{A}_1, \mathtt{A}_2\}, \mathtt{Deathmi})$*, where* U *is a set of patients with heart problems. This decision table is encoded as a set of rough facts shown below. Assume that the conditional attributes* $\mathtt{A}_1$ *and* $\mathtt{A}_2$ *represent two medical tests. Moreover, test* $\mathtt{A}_2$ *is usually considered as being expensive, and therefore, desirable to avoid.*

$\overline{\neg deathmi}$(>70, b1, c1): 2.
$\overline{\neg deathmi}$(>70, b1, c2): 2.
$\overline{\neg deathmi}$(>40 <70, b2, c4): 2.
$\overline{\neg deathmi}$(<40, b3, c5): 2.

$\overline{deathmi}$(>70, b1, c2): 3.
$\overline{deathmi}$(>40 <70, b2, c3): 4.
$\overline{deathmi}$(<40, b3, c5): 3.
$\overline{deathmi}$(<40, b4, c3): 8.

The following rough clauses monitor the impact in the boundary region of not considering test A_2.

(1) $\overline{d}$(Age, A1) :-[1,_] $\overline{deathmi}$(Age, A1, A2).

(2) $\overline{\neg d}$(Age, A1) :-[1,_] $\overline{\neg deathmi}$(Age, A1, A2).

(3) $\overline{migrate}$(Age, A1) :-[1,min] $\underline{\overline{d}}$(Age, A1),
$\underline{deathmi}$(Age, A1, A2).

(4) $\overline{migrate}$(Age, A1) :- [1,min] $\underline{\overline{d}}$(Age, A1),
$\underline{\neg deathmi}$(Age, A1, A2).

(5) $\overline{\neg migrate}$(Age, A1) :-[1,_] $\underline{\neg d}$(Age, A1).

(6) $\overline{\neg migrate}$(Age, A1) :-[1,_] $\underline{d}$(Age, A1).

(7) $\overline{\neg migrate}$(Age, A1) :-[1,sum] $\overline{deathmi}$(Age, A1, A2),
$\overline{\neg deathmi}$(Age, A1, A2).

Predicate migrate *denotes the rough relation*

$$Migrate = (\{\langle >70, b1\rangle : 2, \langle >40\ <70,\ b2\rangle : 6\}, \{\langle >70, b1\rangle : 5, \langle <40,\ b3\rangle : 5, \langle <40,\ b4\rangle : 8\}) .$$

We show some useful queries and their answers. Assume that rough program $\mathcal{P}$ *contains all rough facts above and the rough clauses* $(1) - (7)$.

- *"For which patients it may be useful to request the expensive test* A_2*? And what is the expected gain if only those patients undergo test* A_2*?"*

This request can be formulated by the rough query $\mathcal{Q}_1$

$$\mathcal{Q}_1 = (\overline{migrate}(Age, A1), K_1 = strength(migrate(Age, A1)), K_2 = strength(\neg migrate(Age, A1))\ ,\ \mathcal{P}) .$$

The answer to this rough query is the set

$$\{\{\texttt{>70}/\texttt{Age}, \texttt{b1}/\texttt{A1}, 0.0769/K_1, 0.1923/K_2\}\ ,\\ \{\texttt{>40 <70}/\texttt{Age}, \texttt{b2}/\texttt{A1}, 0.2308/K_1, 0/K_2\}\}\ .$$

This answer indicates that

- ⋆ *for patients who are more than* 70 *years old and have got result* `b1` *in the test* `A1`*, or*
- ⋆ *for patients who are between* 40 *and* 70 *years and have got result* `b2` *in the test* `A1`*,*

it may be advisable to perform additionally test `A2`*. Moreover, if only the patients suggested by this answer undergo the expensive test then, we may expect to avoid the test for about* 50% *of the patients. Notice that if*

$$\sum_{t\in\overline{Migrate}} (\texttt{strength}(\texttt{migrate}(t)) + \texttt{strength}(\neg\texttt{migrate}(t)))$$

would get too close to one then, this would indicate that not that much would be gained by not requesting test `A2` *for all patients.*

- *"Make a prediction of whether individuals with result* `b1` *for test* $\texttt{A}_1$ *need to be submitted to test* $\texttt{A}_2$*."*

This query could be formulated as follows

$$\mathcal{Q}_2 = (\texttt{classify}(\texttt{migrate}(Age, b1))\ \texttt{,}\ \mathcal{P})\ .$$

The answer is

$$(\texttt{migrate} = \texttt{no}\ ,\ 0.7193)$$

Thus, the prediction of the system is that the expensive test is not needed and the confidence factor on this prediction is 0.7 *.*

□

The example discussed in this section has also been tested in *RKBS*. The system has also been tested with the same problem but more realistic data was used: the `deathmi` table contained 418 patients and 12 condition attributes. Figure 6.2 shows the interface of the *RKBS*. On the top part there is a text area where the user can enter the rough clauses and rough facts of a rough program. In this case, the text area displays some of the

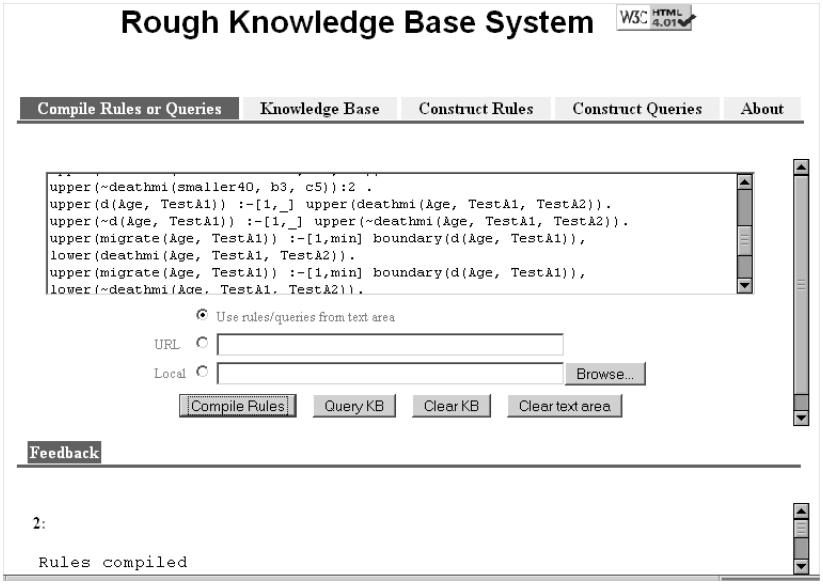
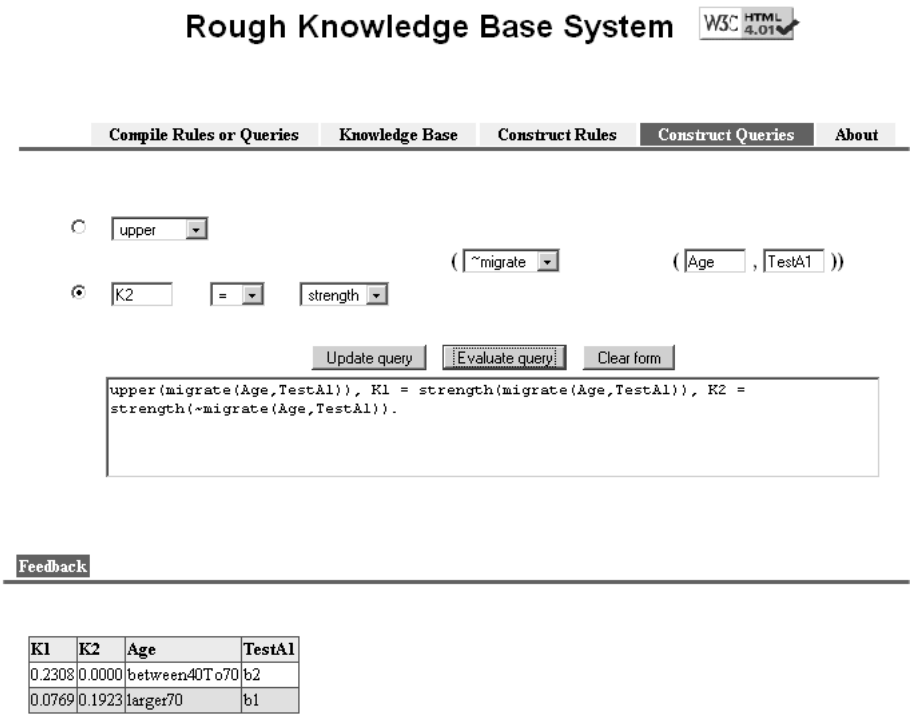


Figure 6.2: The Rough Knowledge Base System.

rough clauses belonging to the rough program presented in the example above. It is also possible to load a rough program from a local file or from a Web page.

Figure 6.3 displays the first rough query of the example above. On the bottom part, it is shown the answer to the rough query in table format. Rough queries can be directly entered in the text area or can be constructed with the help of several menus. The second rough query and its answer is shown in figure 6.4.



K1	K2	Age	TestA1
0.2308	0.0000	between40To70	b2
0.0769	0.1923	larger70	b1

Figure 6.3: The *RKBS* showing query $\mathcal{Q}_1$ and its answer.

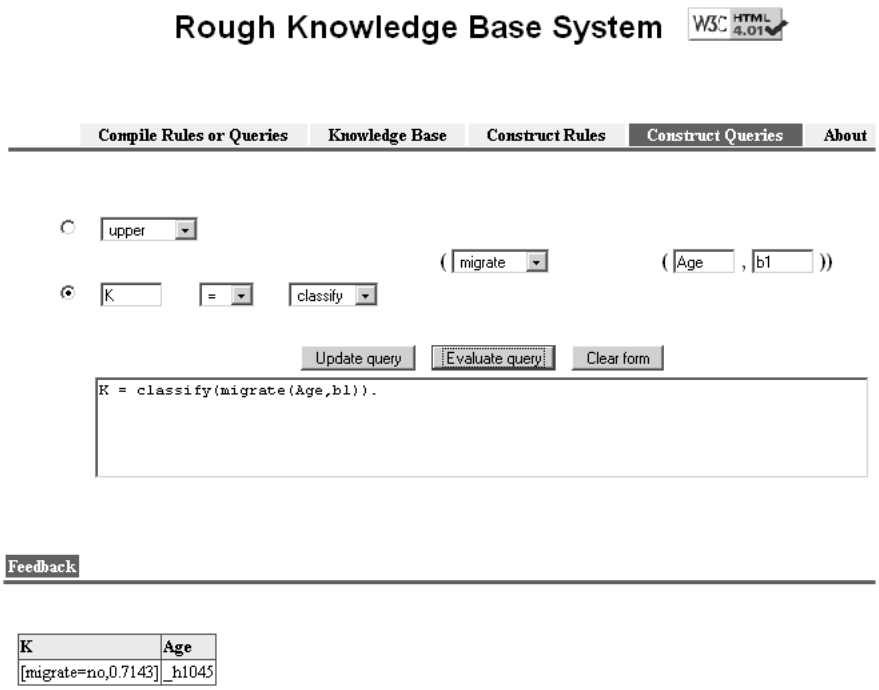


Figure 6.4: The *RKBS* showing query $\mathcal{Q}_2$ and its answer.

Chapter 7

Conclusions and Future Work

This thesis has introduced a language to define intensionally rough relations. An extension of the language supporting basic quantitative measures has also been considered.

This chapter is intended to summarize the work presented in this thesis (section 7.1) and point out possible directions for future research (section 7.2).

7.1 Concluding Remarks

Rough sets framework has two appealing aspects. First, it is a mathematical approach to deal with vague concepts. Second, rough set techniques can be used in data analysis to find patterns hidden in the data. The number of applications of rough sets to practical problems in different fields demonstrates the increasing interest in this framework and its applicability.

The first point above suggests that rough sets techniques can be generalized to knowledge bases. It is not uncommon that our knowledge about a concept bears contradictory information. For instance, one expert may state that a vehicle driven at medium speed in a wet road corresponds to a dangerous situation while another expert may consider that medium speed does not imply a dangerous traffic situation in general. Thus, the concept "dangerous traffic situation" cannot be defined precisely using the available knowledge. There will always be situations that are considered non-dangerous according

the knowledge provided by one expert while the same situation is classified as dangerous when considering the criteria provided by the other expert. Rough set based techniques can be used to represent and deal with contradictory knowledge, as in the situation depicted previously. To this end we have proposed a language that caters for implicit definitions of rough sets.

Different regions (e.g. lower approximation, upper approximation or boundary region) of several rough sets can be combined to define a region of another rough set. In this way, existing rough sets can be used in the definition of other rough sets. This is achieved in the proposed language through the use of rough clauses and rough facts, forming together a rough program. The main strengths of our language can be summarized as follows.

- The language captures and integrates in a uniform way vague knowledge with two possible sources: knowledge obtained directly from experimental data and encoded as rough facts; domain or expert knowledge expressed as rough rules. This contrasts with most of current rough set techniques that only allow definition of (vague) concepts to be obtained from experimental data.

- Several useful techniques and extensions to rough sets, reported in the literature [KØ99, Zia02a], and implemented in an "ad hoc" way can be naturally expressed in our language.

Another important aspect of the work presented in this thesis is the definition of a query language to retrieve information about the defined rough sets and patterns implicit in the data.

The computational basis for reasoning with the rough sets defined in a rough program is a program transformation. Rough programs are compiled to extended logic programs whose semantics is captured by paraconsistent stable models. Systems like *Smodels* [Sim] or *dlv* [Pro] can then be readily used to run extended logic programs. The correctness of the proposed compilation technique has been proved.

An extension of our language to quantitative measures has also been explored. Quantitative measures are particularly relevant in data mining applications. However, we have restricted this extension to non-recursive rough programs. We have implemented a system, called Rough Knowledge Base System, that can reason and answer queries about rough sets defined in this language. We also show that this extension allows to capture the variable precision rough set model [Zia93] and to extend its application to implicitly defined rough sets.

To our knowledge, besides our work, only system *CAKE* [DŁS02] addresses the problem of defining implicitly rough sets. We present below a brief comparison of *CAKE* with the framework presented in this thesis.

- Our language distinguishes tuples for which there is no information available from tuples for which there is contradictory evidence. The latter case corresponds to tuples in the boundary region. System *CAKE* does not support this distinction: the boundary region includes tuples about which there is no information at all and tuples about which there is contradictory information. Hence, our language is based on a $4-$valued logic while *CAKE* is based on a $3-$valued logic.

- In our framework quantitative measures can be easily supported. This extension seems less obvious to achieve in *CAKE*.

- Another important difference is that *CAKE* only supports a restricted type of recursively defined rough sets, corresponding to stratified programs, while our language supports any recursive (rough) program. However, this restriction of *CAKE* has the benefit of ensuring its tractability. Computing all models capturing the meaning of a rough program is an intractable problem.

- Most knowledge representation systems incorporate either the *open-world assumption* or the *closed-world assumption* in their reasoning procedures. Systems using the former assume that they may not have complete information about the world. Thus, information not known is assumed to be unknown. Under closed-world assumption, if a system cannot prove that a tuple belongs to a relation then it is assumed that the tuple does not belong to the relation. Both systems, the language we propose and *CAKE*, support reasoning under the *open-world assumption*. However, the latter also allows to apply the closed-world assumption locally in a particular context. This feature is achieved in *CAKE* through the use of *contextually closed queries* (CCQ) [DKS04] consisting of a query itself and a particular context for evaluating the query. This context specifies minimization (maximization) policies to be applied to selected relations and a number of integrity constraints. However, answering queries using CCQs is co-NPtime complete [DKS04].

7.2 Future Work

This section presents possible future directions of our research. We list below several aspects that can be improved and extensions of this work.

- We have discussed in chapter 6 an extension of our language supporting some basic quantitative measures. However, the declarative semantics of the language was not formally defined. Moreover, the compilation technique discussed only applies to non-recursive rough programs. Thus, we plan to formalize the declarative semantics of our extended language and investigate a computational technique that supports recursive rough programs with quantitative measures.
- We plan to investigate how the query language can be enriched. For instance, we would like to provide system support for formulating and testing hypothesis.
- More efficient implementation of our language is also one of our goals in the future. We plan to develop an implementation for recursive rough programs re-using the existing expertise in stable model systems, such as Smodels [Sim].
- We also plan to search for other concrete problems that can be formulated in our framework. To this end, we may take a closer look to rough mereology [PS94, PS97, PS01] applications. Mereology is a formal theory of parts and wholes. In rough mereology, "parthood" is a rough relation.
- A particular type of domain knowledge that often has to be considered when classifying objects of a universe is an ontology of decision classes. An interesting research direction is to investigate how ontologies can be represented in our framework. The problem of integrating learning algorithms based on rough set techniques with a gene ontology has been addressed in [MLK01, MK02, Mid03] to predict gene function. Several operators to build approximations of decision classes are proposed. An open question is whether these operators could be easily encoded in our language or which extensions need to be considered to achieve that goal.
- This thesis presents foundation and implementation principles for a rule language able to support reasoning on incomplete and imprecise information. The necessity of rule languages for handling imprecise and incomplete Web data seems to be obvious. The research

on this topic fits well with the objectives of the *EU FP*6 *Network of Excellence REWERSE* (`http://rewerse.net`) aiming at designing rule-based web reasoning languages. A topic of future research is deployment of the proposed language for web reasoning purposes.

Appendix A

Notation Summary

Notation	Meaning
$\|S\|$	the number of elements in a set S
$A \setminus B$	set difference between A and B
$\mathcal{D} = (U, A, d)$	a decision table such that U is a set of objects, A is a set of condition attributes, and d is a decision attribute
R_A	the indiscernibility relation induced by a set of attributes A
R_A^*	the set of equivalence classes induced by R_A
$\overrightarrow{E_A}$	the tuple describing indiscernibility class E
(U, R_A)	an approximation space
$\overline{X}$	the upper approximation of rough set X
$\underline{X}$	the lower approximation of rough set X
$\overline{\underline{X}}$	the boundary region of rough set X
$cond(r) \rightarrow dec(r)$	a decision rule r
$cover(c)$	the set of objects satisfying condition c
$cover(r)$	$cover(cond(r))$

Notation	Meaning
D_v	the set of objects having outcome v for the decision attribute
$red(r)$	a valued reduct for decision rule r
$\kappa(A, B)$	the degree of functional dependency between the sets of attributes A and B
$red(A, B)$	a relative reduct of the set of attributes A w.r.t. $\kappa(A, B)$
Var	an alphabet of variable symbols
$Const$	an alphabet of constant symbols
$Pred$	an alphabet of predicate symbols
p/n	a predicate p with arity n
$\mathcal{I}$	(a rough) interpretation
$\mathcal{M}$	a (rough) model
X/c	a binding of a variable X to a constant c
θ	a substitution
$\neg A$	an explicit negated atom A
$not\ A$	a default negated atom A
$ground(\mathcal{P})$	a ground (rough) program
H :- B.	a (rough) clause
:- B.	an integrity constraint
$sem(\mathcal{P})$	the semantics of a (rough) program
$(\mathcal{Q}, \mathcal{P})$	a (rough) query to a (rough) program $\mathcal{P}$
$\psi_{\mathcal{I}}(\mathcal{P})$	the reduct of the ELP $\mathcal{P}$
$\Psi_{\mathcal{I}}(\mathcal{P})$	the ground rough program, obtained from $\mathcal{P}$, without lower approximations or testing literals in the body of its rough clauses
$\mathcal{I} \models l$	(rough) literal l is true in $\mathcal{I}$
$\mathcal{P} \models l$	(rough) program $\mathcal{P}$ implies (rough) literal l
$Q^{\mathcal{I}}$	the rough set denoted by predicate q in interpretation $\mathcal{I}$
$\langle t_1, \ldots, t_n \rangle$	a tuple whose attribute values are $t_1, \ldots, t_n$
t	a tuple
$[t]$	the indiscernibility class described by tuple t
$\langle t_1, \ldots, t_n \rangle : k$	k objects belonging to the indiscernibility class $[\langle t_1, \ldots, t_n \rangle]$ are positive examples
$\langle t_1, \ldots, t_n \rangle : k_1 : k_2$	k_1 (k_2) objects belonging to the indiscernibility class $[\langle t_1, \ldots, t_n \rangle]$ are positive (negative) examples

Notation	Meaning
$q(\overline{X})$	$q(X_1, \ldots, X_1)$
$q(-, \overline{X})$	the first argument of predicate q is irrelevant
$u.1st$	if $u = \langle c_1, c_2 \rangle$ then $u.1st = c_1$
$u.2nd$	if $u = \langle c_1, c_2 \rangle$ then $u.2nd = c_2$
τ_1	the compilation function for rough clauses
τ_2	the compilation function for bodies of rough clauses belonging to rough programs without quantitative measures
τ_F	the compilation function for bodies of rough clauses in *RKBS*
τ_m	the compilation function for quantitative measures
τ_p	the compilation function for predicate symbols occurring in a rough program in *RKBS*
F	a commutative and associative binary function

Bibliography

[ALP+00] J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinska, and T. C. Przymusiski. Dynamic updates of non-monotonic knowledge bases. In *Journal of Logic Programming*, number 45, pages 43–70. Elsevier, 2000.

[And04a] R. Andersson. Implementation of a rough knowledge base system supporting quantitative measures. Master thesis, Linköping University, IDA, 2004.

[And04b] R. Andersson. Rough Knowledge Base System. Available at http://www.ida.liu.se/rkbs, 2004.

[Bar03] C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[Baz96] J. G. Bazan. Dynamic reducts and statistical inference. In *Proc. of the Sixth International Conference, Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMIU'96)*, volume 3, pages 1147–1152, 1996.

[Baz98] J. G. Bazan. Discovery of decision rules by matching new objects against data tables. In L. Polkowski and A. Skowron, editors, *Proc. of the First International Conference on Rough Sets and Current Trends in Computing (RSCTC'98)*, volume 1424 of *LNCS*, pages 521–528. Springer-Verlag, 1998.

[Bel77a] N. D. Belnap. How computer should think. In G. Rydle, editor, *Contemporary Aspets of Philosophy*, pages 30–56. Oriel Press, 1977.

[Bel77b] N. D. Belnap. A useful four-valued logic. In G. Epstein and J. M. Dunn, editors, *Modern Uses of Multiple-Valued Logic*, pages 7–37. Reidel Publishing Company, 1977.

[BL04] R. J. Brachman and H. J. Levesque. *Knowledge Representation and Reasoning*. Elsevier, 2004.

[CSW95] W. Chen, T. Swift, and D. S. Warren. Efficient top-down computation of queries under the well-founded semantics. *Journal of Logic Programming*, 24(3):161–199, September 1995.

[CW96] W. Chen and D. S. Warren. Tabled evaluation with delaying for general logic programs. *Journal of the ACM*, 43(1):20–74, January 1996.

[DEBC96] P. Deransart, A. Ed-Bali, and L. Cervoni. *Prolog: The Standard Reference Manual*. Springer-Verlag, 1996.

[DEGV01] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys*, 33(3):374–425, 2001.

[DKS04] P. Doherty, J. Kachniarz, and A. Szałas. Using contextually closed queries for local closed-world reasoning in rough knowledge databases. In S. K. Pal, L. Polkowski, and A. Skowron, editors, *Rough-Neural Computing*, pages 219–250. Springer-Verlag, 2004.

[DŁS02] P. Doherty, W. Łukaszewicz, and A. Szałas. CAKE: A Computer Aided Knowledge Engineering Technique. In F. van Harmelen, editor, *Proc. of the 15th European Conference on Artificial Intelligence, (ECAI'02)*, pages 220–224, Amsterdam, 2002. IOS Press.

[DP92] D. Dubois and H. Prade. Putting rough sets and fuzzy sets together. In R. Slowinski, editor, *Handbook of Applications and Advances of the Rough Sets Theory*, pages 204–232. Kluwer Academic Publishers, 1992.

[DP98] C. V. Damásio and L. M. Pereira. A survey of paraconsistent semantics for logic programs. In D. M. Gabbay and Ph. Smets, editors, *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, volume 2, pages 241–320. Kluwer Academic Publishers, 1998.

[ELM+98] T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The KR system `dlv`: Progress report, comparisons and benchmarks. In A. G. Cohn, L. Schubert, and S. C. Shapiro, editors, *KR'98: Principles of Knowledge Representation and Reasoning*, pages 406–417, San Francisco, California, 1998. Morgan Kaufmann.

[FLP04] W. Faber, N. Leone, and G. Pfeifer. Recursive aggregates in disjunctive logic programs: Semantics and complexity. In J. J. Alferes and J. A. Leite, editors, *Proc. of the Nineth European Conference on Logics in Artificial Intelligence (JELIA'04)*, volume 3229 of *LNCS*, pages 200–212. Springer-Verlag, 2004.

[GL88] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. Bowen, editors, *Proc. of the Fifth International Logic Programming Conference and Symposium*, pages 1070–1080, Seattle, USA, 1988. MIT Press.

[GL90] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In Warren and Szeredi, editors, *Proc. of the Seventh International Conference on Logic Programming*, pages 579–597. MIT Press, 1990.

[IS98] K. Inoue and C. Sakama. Negation as failure in the head. *Journal of Logic Programming*, 35(1):39–78, 1998.

[KØ99] J. Komorowski and A. Øhrn. Modelling prognostic power of cardiac tests using rough sets. *Journal of Artificial Intelligence in Medicine*, 15(2):167–191, 1999.

[KPPS99] J. Komorowski, Z. Pawlak, L. Polkowski, and A. Skowron. Rough sets: A tutorial. In S. K. Pal and A. Skowron, editors, *Rough Fuzzy Hybridization. A New Trend in Decision-Making*, pages 3–98. Springer-Verlag, 1999.

[Llo87] J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1987.

[LR04] L. Lazareck and S. Ramanna. Classification of swallowing sound signals: A rough set approach. In S. Tsumoto, R. Słowiński, and J. W. Grzymala-Busse, editors, *Proc. of the Fourth International Conference on Rough Sets and Current Trends in Computing (RSCTC'04)*, number 3066 in LNAI, pages 679–684. Springer-Verlag, 2004.

[Mid03] H. Midelfart. *Knowledge Discovery from cDNA Microarrays and a priori Knowledge*. PhD thesis, Department of Computer and Information Science, Norwegian University of Science and Technology, NO-7491 Trondheim, Norway, September 2003.

[MK00] H. Midelfart and J. Komorowski. A rough set approach to inductive logic programming. In W. Ziarko and Y. Y. Yao, editors, *Proc. of the Second International Conference on Rough Sets and Current Trends in Computing (RSCTC'00)*, volume 2005 of *LNCS*. Springer-Verlag, 2000.

[MK02] H. Midelfart and J. Komorowski. A rough set framework for learning in a directed acyclic graph. In J. J. Alpigini, J. F. Peters, J. Skowronek, and N. Zhong, editors, *Proc. of the Third International Conference on Rough Sets and Current Trends in Computing*, volume 2475 of *LNCS*, pages 144–155. Springer-Verlag, 2002.

[MLK01] H. Midelfart, A. Lægreid, and J. Komorowski. Classification of gene expression data in an ontology. In J. Crespo, V. Maojo, and F. Martin, editors, *Proc. of the Second International Symposium on Medical Data Analysis (ISMDA'01)*, volume 2199 of *LNCS*, pages 186–194. Springer-Verlag, 2001.

[MV02] J. Małuszyński and A. Vitória. Defining rough sets by extended logic programs. In *On-Line Proc. of Paraconsistent Computational Logic Workshop (PCL'02)*, 2002. http://floc02.diku.dk/PCL/ and http://arxiv.org/list/cs.lo/0207#cs.lo/0207089.

[NM95] U. Nilsson and J. Małuszynski. *Logic, Programming and Prolog, 2nd edition.* John Wiley & Sons, http://www.ida.liu.se/~ulfni/lpp/copyright.html, 1995.

[NNSS96] H. S. Nguyen, T. T. Nguyen, A. Skowron, and P. Synak. Knowledge discovery by rough set methods. In N. C. Callaos, editor, *Proc. of the International Conference on Information Systems Analysis and Synthesis (ISAS'96)*, pages 526–33, 1996.

[NS96] I. Niemelä and P. Simons. Efficient implementation of the well-founded and stable model semantics. In M. Maher, editor, *Proc. of the Joint International Conference and Symposium on Logic Programming*, pages 289–303, Bonn, Germany, 1996. MIT Press.

[NS97] I. Niemelä and P. Simons. Smodels - an implementation of stable model and the well-founded semantics for normal logic programs. In J. Dix, U. Furbach, and A. Nerode, editors, *Proc. of the Fourth International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'97)*, volume 1265 of *LNAI*, pages 420–429. Springer-Verlag, 1997.

[ØK97] A. Øhrn and J. Komorowski. ROSETTA: A rough set toolkit for analysis of data. In *Proc. of Third International Joint Conference on Information Sciences, Fifth International Workshop on Rough Sets and Soft Computing (RSSC'97)*, volume 3, pages 403–407, Durham, NC, USA, 1-5 March 1997.

[Pag97] P. Pagliani. Rough sets theory and logic-algebraic structures. In E. Orlowska, editor, *Incomplete Information: Rough Sets Analysis*, pages 109–190. Verlag-Physics, 1997.

[Paw82] Z. Pawlak. Rough sets. *International Journal of Information and Computer Science*, 11(5):341–356, 1982.

[Paw91] Z. Pawlak. *Rough sets. Theoretical Aspects of Reasoning about Data*. Kluwer Academic Publishers, Dordrecht, 1991.

[Pea93] D. Pearce. Answer sets and constructive logic, II: Extended logic programs and related non-monotonic formalisms. In L. M. Pereira and A. Nerode, editors, *Logic Programming and Nonmonotonic Reasoning - Proc. of the 2nd International Workshop*, pages 457–475. MIT Press, 1993.

[Pro] The DLV Project. Available at http://www.dbai.tuwien.ac.at/proj/dlv/.

[PS94] L. Polkowski and A. Skowron. Rough mereology. In *Proc. of the Eighth International Symposium on Methodologies for Intelligent Systems*, pages 85–94. Springer-Verlag, 1994.

[PS97] L. Polkowski and A. Skowron. Rough mereology: a new paradigm for approximate reasoning. *International Journal of Approximate Reasoning*, 15(4):333–4365, 1997.

[PS01] L. Polkowski and A. Skowron. Rough mereological calculi of granules: A rough set approach to computation. *Journal of Computational Intelligence*, 17(3):472–492, 2001.

[Rei80] R. Reiter. A logic for deafult reasoning. *Journal of Artificial Intelligence*, 13:81–132, 1980.

[SI95] C. Sakama and K. Inoue. Paraconsistent Stable Semantics for Extended Disjunctive Programs. *Journal of Logic and Computation*, 5(3):265–285, 1995.

[Sim] P. Simons. Smodels system. Available at http://www.tcs.hut.fi/Software/smodels/.

[SM] Java. Sun Mycrosystems. Available at http://java.sun.com/.

[SP97] A. Skowron and L. Polkowski. Synthesis of decison systems from data tables. In T. Y. Lin and N. Cercone, editors, *Rough Sets and Data Mining Analysis of Imprecise Data*, pages 259–300. Kluwer Academic Publishers, 1997.

[SS96] A. Skowron and J. Stepaniuk. Tolerance approximation spaces. *Fundamenta Informaticae*, 27:245–253, 1996.

[SS98] K. Sagonas and T. Swift. An abstract machine for tabled execution of fixed-order stratified logic programs. *Journal of the ACM TOPLAS*, 20(3):586–635, May 1998.

[Ste98] J. Stefanowski. On rough set approaches to induction of decision rules. In L. Polkowski and A. Skowron, editors, *Rough Sets in Knowledge Discovery 1: Methodology and Applications*, pages 501–529. Springer-Verlag, 1998.

[SV97] R. Slowinski and D. Vanderpooten. Similarity relations as a basis for rough set approximations. In P. P. Wang, editor, *Advances in Machine Intelligence and Soft Computing*, volume 4, pages 17–33, 1997.

[Swi99] T. Swift. Tabling for non-monotonic reasoning. *Annals of Mathematics and Artifcial Intelligence*, 25(3–4):201–240, 1999.

[Sys] XSB System. Available at http://xsb.sourceforge.net/.

[TS02] F. E. H. Tay and L. Shen. Economic and finantial prediction using rough sets model. *European Journal of Operational Research*, 141:641–659, 2002.

[VDM03a] A. Vitória, C. V. Damásio, and J. Małuszyński. From rough sets to rough knowledge bases. *Fundamenta Informaticae*, 57(2-4):215–246, October 2003.

[VDM03b] A. Vitória, C. V. Damásio, and J. Małuszyński. Query answering for rough knowledge bases. In G. Wang, Q. Liu, Y. Yao, and A. Skowron, editors, *Proc. of the Nineth International Conference on Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing (RSFDGrC'03)*, volume 2639 of *LNAI*, pages 197–204. Springer-Verlag, 2003.

[VDM04] A. Vitória, C. V. Damásio, and J. Małuszyński. Toward rough knowledge bases with quantitative measures. In S. Tsumoto, R. Slowiński, J. Komorowski, and J. W. Grzymala-Busse, editors, *Proc. of the Fourth International Conference on Rough Sets and Current Trends in Computing (RSCTC'04)*, volume 3066 of *LNAI*, pages 153–158. Springer-Verlag, 2004.

[VM02] A. Vitória and J. Małuszyński. A logic programming framework for rough sets. In J. Alpigini, J. Peters, A. Skowron, and N. Zhong, editors, *Proc. of the Third International Conference on Rough Sets and Current Trends in Computing (RSCTC'02)*, number 2475 in LNAI, pages 205–212. Springer-Verlag, 2002.

[Wyg89] M. Wygralak. Rough sets and fuzzy sets - some remarks on interrelations. *Journal of Fuzzy Sets and Systems*, 29:241–243, 1989.

[YL96] Y. Y. Yao and T. Y. Lin. Generalizations of rough sets using modal logic. *Journal of Intelligent Automation and Soft Computing*, 2:103–120, 1996.

[Zan02] C. Zaniolo. Key constraints and monotonic aggregates in deductive databases. In A. C. Kakas and F. Sadri, editors, *Computational Logic: Logic Programming and Beyond, Essays in Honour of Robert Kowalski*, volume 2408 of *LNCS*, pages 109–134. Springer-Verlag, 2002.

[ZF02] W. Ziarko and X. Fei. VPRSM approach to WEB searching. In J. Alpigini, J. Peters, A. Skowron, and N. Zhong, editors, *Proc. of the Third International Conference on Rough Sets and Current Trends in Computing, RSCTC'02*, number 2475 in LNAI, pages 514–521. Springer-Verlag, 2002.

[Zia93] W. Ziarko. Variable precision rough set model. *Journal of Computer and Systems Science*, 46(1):39–59, 1993.

[Zia02a] W. Ziarko. Acquisition of Hierarchy-Structured Probabilistic Decision Tables and Rules from Data. In *Proc. of the World Congress on Computational Intelligence*, Honolulu, 2002.

[Zia02b] W. Ziarko. Rough set approaches for discovery of rules and attribute dependencies. In W. Kloesgen and J. Zytkow, editors, *Handbook of Data Mining and Knowledge Discovery*, pages 328–338. Oxford University Press, 2002.