Fundamenta Informaticae ??? (2003) 1–32 1
IOS Press

From Rough Sets to Rough Knowledge Bases

Aida Vitória
Dept. of Science and Technology, Linköping University,
S 601 74 Norrköping, Sweden
aidvi@itn.liu.se

Carlos Viegas Damásio
Centro de Inteligência Artificial (CENTRIA),
Dept. Informática, Faculdade de Ciências e Tecnologia,
Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
cd@di.fct.unl.pt

Jan Małuszyński
Dept. of Computer and Information Science,
Linköping University, S 581 83 Linköping, Sweden
janma@ida.liu.se

Abstract. This paper presents an expressive language for representing knowledge about vague concepts. It is based on the rough set formalism and it caters for implicit definition of rough relations by combining different regions (e.g. upper approximation, lower approximation, boundary) of other rough relations. The semantics of the proposed language is obtained by translating it to the language of extended logic programs whose meaning is captured by paraconsistent stable models. A query language is also discussed to retrieve information about the defined rough relations. Motivating examples illustrating the use of the language are described.

Keywords: Rough sets, logic programming, stable models, uncertain reasoning, incomplete reasoning.

1. Introduction

This paper investigates an approach to representation of - and reasoning about imprecise knowledge. The approach is based on the notion of *rough set*, introduced by Pawlak (see e.g. [13]) and on logic

programming. We present a language that makes it possible to define and to reason about rough sets, adopting techniques developed by logic programming community. The paper generalizes our previous work [18] and it is an extended version of the conference paper [17]. The extension includes a better presentation of the logical foundations of the defined language, treatment of undefined objects and a more general query language. An important aspect of the extension is to demonstrate that some useful techniques introduced in rough set literature in an "ad hoc" way, are naturally expressible in the proposed language. This shows that the language provides a natural framework for defining extensions of the traditional rough set techniques.

The notion of rough set is motivated by applications where the objects of a universe of discourse have to be classified on the basis of some associated information, such as values of the *attributes* of a given object. This assumption makes the universe granular in the sense that the objects with identical attribute values are indiscernible. For example, it is often the case that two patients have the same symptoms but only one of them has a specific disease. Thus the symptoms are not sufficient for a reliable diagnosis. The rough set theory [13] provides a formal framework for modelling and analysis of such situations.

Intuitively, a rough set is an approximation of a subset of the universe. It is usually defined by a finite decision table, whose rows correspond to some objects of the universe. Such a table can be encoded by a finite set of atomic formulae, which can be seen as a logic program consisting of a finite number of unit clauses (or facts). A natural extension discussed in our previous work [17, 18] is to admit also non-unit clauses for that purpose. This gives direct support for integration of background (e.g. expert) knowledge, reasoning, and possibility to combine different rough sets for defining new ones. The language of [18] is very restricted. In this paper we give an extended presentation of a language for definition and querying of rough sets, which is based on the ideas of the conference paper [17], with an extension concerning treatment of undefined objects. The main idea is similar to our previous work [18] but the language is much more expressive, and much more suitable for defining concepts in terms of the notions of rough set theory, as illustrated by the examples in this paper. The precise meaning of the definitions is provided by translation to extended logic programs, under the *paraconsistent stable model semantics* [14, 16]. In this way we link rough sets with paraconsistent logic programming and with Nelson's constructive logic N^- [2]. A program may have several (paraconsistent) stable models (or no model at all). Each of the models describes a family of rough sets and can be seen as a possible scenario of rational beliefs supported by the knowledge represented by the program. The price we pay for the extra expressiveness is the increase in time complexity: to know whether a literal belongs to a stable model is a NP-complete problem. However, the complexity is polynomial when there is no recursion through negation. Furthermore, efficient implementations exist of the stable model semantics (see e.g. [5, 11]) and these systems can be readily used for querying rough sets defined in our language.

Recently, a system called **CAKE** [4] has been proposed. This system is a generalization of deductive databases to knowledge bases also defining rough relations. Although **CAKE** and the framework we propose in this paper seem to address related problems and both extend the basic rough set formalism, there are some important differences. First, **CAKE** focuses on visualization of the represented knowledge, while we concentrate more on the language and its semantics. Second, our language is 4-valued but **CAKE** is 3-valued, since there is no distinction between undefined knowledge and contradictory information. Third, tractability is guaranteed in **CAKE** because the underlying language is stratified. Fourth, in **CAKE** it is possible to use the closed-world assumption locally in a particular context through local closure context policies, in addition to the open-world assumption. Answers to queries in our language are computed using the open-world assumption. At last, our framework can be extended to incorporate

quantitative measures (ongoing work), what is an important aspect from the point of view of practical applications. This extension seems less obvious to achieve in **CAKE**.

The rest of this paper is organized as follows. Section 2 gives a brief introduction to rough sets and to extended logic programs. Section 3 introduces the rough set definition language and the query language used for querying the defined rough sets. It gives the semantics of these languages by defining a formal transformation of rough set definitions and queries to extended logic programs with the paraconsistent stable model semantics. Section 4 illustrates the usefulness of the proposed definition language as a general framework for rough set applications. It discusses three example applications of rough sets selected from the literature. Each of them requires a different "ad hoc" extension to classical rough set techniques. We show that each of these extensions can be easily and naturally encoded in our language. Finally, Section 5 presents conclusions and further planned work.

2. Preliminaries

This section surveys well-known basic notions related to rough sets [10, 12] and (extended) logic programs [3, 14, 16] needed in the context of our work . Thus, in this section, it is not our aim to present any novel work, but to introduce preliminaries in a way facilitating understanding of the paper.

2.1. Rough sets

Rough sets emerge naturally in the context of supervised learning, where we deal with a universe U of objects described by attributes. Each object belongs to a predefined class determined by one of the attributes, often called decision attribute. For instance, an object can be a patient, an attribute might be the result of a certain medical test, and a decision attribute could tell whether the patient suffers from a certain disease or not. Usually, the description of objects in terms of attributes is provided in the form of a decision table. Next, we formalize these notions.

An attribute a can be seen as a total function $a : U \to V_a$, where V_a is called the *value domain* of a. In this paper we assume that we do not have missing (unknown) attribute values. Thus, every object is associated with a tuple of attributes. We assume that this tuple is the only way of referring to the object: different objects with same attribute values are indiscernible (cf. [10]).

The special constant `null` may belong to the value domain V_a off an attribute a. If for an object $o \in U$, $a(o) = \texttt{null}$, 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 this is different from a missing value for $a(o)$, since a missing value means that the value exists but we do not know it. Thus, in this last situation it is sometimes reasonable to apply some technique to recover the lost value (or a suitable approximation of it), while for the former case this does not make so much sense.

Formally, a decision table $\mathcal{D}$ is a triple (U, A, d), where U is a set of objects, A is a set of conditional 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 aim is to be able to characterize the set of objects with positive (negative) outcome for the decision attribute in terms of the values of the conditional attributes. As it is shown below, a precise description may not always be possible.

Example 2.1. Consider the decision table $\mathcal{D}\texttt{eathmi} = (U, A, \texttt{Deathmi})$, represented by table 1, where $U = \{o_1, \cdots, o_8\}$ is a set of patients and $A = \{\texttt{Age}, \texttt{Hypert}, \texttt{Scanabn}\}$ is the set of conditional

attributes specifying the age of a patient, whether he/she has hypertension, and if he/she has a positive or negative result on a particular medical test to the heart (this is the meaning of attribute `Scanabn`). The decision attribute `Deathmi` indicates whether the patient had a major heart problem (e.g. cardiac death or infarction) in a certain follow up period.

Table 1.

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

Note that using the attributes from A, patients o_6 and o_7 are indiscernible from each other because they are represented by the same tuple $\langle$`>70`, `yes`, `no`$\rangle$. However, they have different outcomes for the decision attribute: patient o_6 had a major heart problem, while patient o_7 did not. This points out that the information available in the decision table may be contradictory.

□

Definition 2.1. Given a decision table $\mathcal{D} = (U, A, d)$, an object $o_i \in U$ is *indiscernible* from object $o_j \in U$ iff, for all conditional 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 (also called indiscernibility classes). If $a(o) =$ `null`, for some object o of an indiscernibility class, then $a(o') =$ `null`, for every other object o' in the same equivalence class. Note that each equivalence class can be described by a unique tuple of $\prod_{a_i \in A} V_{a_i}$[1]. These partitions are like atomic information granules that are used to define new subsets of the universe U. The next example illustrates this point.

Example 2.2. Consider again the decision table of example 2.1. The equivalence classes induced by this table are

[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 $A = \{a_1, \cdots, a_k\}$.

$$\begin{array}{lcllcl} [o_1] & = & \{o_1, o_8\}, & [o_4] & = & \{o_4, o_5\}, \\ [o_2] & = & \{o_2\}, & [o_6] & = & \{o_6, o_7\}, \\ [o_3] & = & \{o_3\}. & & & \end{array}$$

Each equivalence class can be described by a boolean formula. For instance, take $[o_6]$. It can be described by

$$(\texttt{Age} = \texttt{>70}) \wedge (\texttt{Hypert} = \texttt{yes}) \wedge (\texttt{Scanabn} = \texttt{no}) .$$

Alternatively, we can simply use the tuple $\langle \texttt{>70}, \texttt{yes}, \texttt{no} \rangle$ to describe the same equivalence class (this is the approach followed in this paper).

Assume we want to define the set (concept) of patients who had a serious heart problem during the follow up period ($\texttt{Deathmi} = \texttt{yes}$), i.e. $S = \{o_3, o_4, o_5, o_6\}$, in terms of the description of the equivalence classes above. We remind the reader that each equivalence class can be described by a tuple of (conditional) attribute values. The problem here is that $[o_6]$ is neither contained in *S* nor it is disjoint from *S*. This indicates that concept *S* cannot be defined precisely using the available information. Thus, set *S* is considered to be rough. However $[o_4] \subseteq S$, leading us to conclude that, in light of he available information, patients with more than 70 years old, with hypertension and testing positive for the heart test 'scanabn' run definitely a risk of having a serious heart problem.

□

As the example above shows, it may not be possible to define precisely a concept (set of objects) using the induced indiscernibility classes. It is in this context that the notion of rough set is useful, since it introduces the idea of set approximations.

A subset *S* of U, representing a concept, can be characterized by two sets of tuples: $\overline{S}$ and $\overline{\neg S}$.

- $\overline{S}$ includes the tuples describing the objects in *S*.
- $\overline{\neg S}$ includes the tuples representing the objects in the complement of *S*.

This motivates the following definition.

Definition 2.2. 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. Moreover, the *rough complement* of a rough set $S = (\overline{S}, \overline{\neg S})$ is the rough set $\neg S = (\overline{\neg S}, \overline{S})$.

Usually, we do not distinguish between a concept and the rough set representing that concept.

We stress that $\overline{S}$ and $\overline{\neg S}$ need not be disjoint: the indiscernibility class described by a tuple may include objects in the concept S and objects in its complement. Moreover, $\overline{S}$ and $\overline{\neg S}$ need not to cover the universe: for the remaining tuples there is no information whether the corresponding indiscernibility class includes elements of S.

The set $\overline{S}$ ($\overline{\neg S}$) can also be interpreted as a description of the objects that *possibly* belong (do not belong) to a certain concept S and is often called the *upper approximation* of S ($\neg S$). Notice that some doubt exists in this description because there may exist a tuple $t \in \overline{S} \cap \overline{\neg S}$. The *lower approximation* of S is the set $\underline{S} = \overline{S} - \overline{\neg S}$ and it should essentially be viewed as a descriptions of those objects that *definitely* belong to the concept. The set $\underline{\overline{S}} = \overline{S} \cap \overline{\neg S}$ is called the *boundary* and it corresponds to the conflicting cases (i.e. the doubtful ones).

It is easy to see that a decision table $\mathcal{D} = (U, A, d)$ is an alternative representation of a rough set that we designate as $D = (\overline{D}, \overline{\neg D})$, 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.

Given a rough set $S = (\overline{S}, \overline{\neg S})$, a tuple t *belongs to* S, represented as $t \in S$, iff $t \in \overline{S}$. Similarly, $t \notin S$ iff $t \notin \overline{S}$. Notice that the fact $t \notin S$ does not imply that $t \in \neg S$ because there may exist no information if the objects described by this tuple belong to the concept S or not.

Example 2.3. Consider once more the decision table of example 2.1 and the concept $S = \{o_3, o_4, o_5, o_6\}$ representing the patients who had a major heart problem. Since it is not possible to define this concept precisely using the available information, instead we represent it by the rough set $S = (\overline{S}, \overline{\neg S})$, where

$$\begin{aligned} \overline{S} &= \{\langle \texttt{>70}, \texttt{no}, \texttt{yes}\rangle, \langle \texttt{>70}, \texttt{yes}, \texttt{yes}\rangle, \langle \texttt{>70}, \texttt{yes}, \texttt{no}\rangle\} \text{ and} \\ \overline{\neg S} &= \{\langle \texttt{<70}, \texttt{no}, \texttt{no}\rangle, \langle \texttt{>70}, \texttt{no}, \texttt{no}\rangle, \langle \texttt{>70}, \texttt{yes}, \texttt{no}\rangle\} . \end{aligned}$$

The upper approximation of S, given by $\overline{S}$, describes those patients who possibly may have heart problems.

The lower approximation of S, given by

$$\underline{S} = \{\langle \texttt{>70}, \texttt{no}, \texttt{yes}\rangle, \langle \texttt{>70}, \texttt{yes}, \texttt{yes}\rangle\} ,$$

describes those patients who certainly run a major risk of having a heart problem.

The lower approximation of $\neg S$, given by

$$\underline{\neg S} = \{\langle \texttt{<70}, \texttt{no}, \texttt{no}\rangle, \langle \texttt{>70}, \texttt{no}, \texttt{no}\rangle\} ,$$

describes those patients who certainly do no run a significant risk of having a heart problem.

□

We want to stress that in our work a rough set is not defined in terms of objects of the universe, but instead in terms of the tuples that describe each equivalence class (i.e. class of indiscernible objects) to which the objects belong. Moreover, the terms "rough set" and "rough relation" are used interchangeably.

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 "$\mathcal{D}$eathmi", a name usually starting with calligraphic letter, then its decision attribute is called "`Deathmi`" and it defines the rough relation "$Deathmi$".

2.2. Extended Logic Programs

The ability to describe rough relations in terms of other ones is fundamental for the construction of rough knowledge bases. Extended Logic Programming supplies the appropriate mechanisms to represent and reasoning with rough sets, and to express the definition of derived rough relations in the form of rules. In particular, the *paraconsistent stable model semantics* of extended logic programs [14, 16] provides two forms of negation, explicit and default, allowing both open-world and closed-world reasoning. Explicit negation describes negative evidence, for instance negative examples in decision tables, while default negation allows reasoning with lack of information, e.g. for defining lower approximations of rough sets.

The semantics is paraconsistent and non-monotonic: a piece of information and its explicit negation can simultaneously hold without trivializing the models; and by adding a new statement to a program the conclusions entailed might decrease, e.g. by adding a negative example conflicting an already positive one, the lower approximation of the rough relation is reduced. This section surveys the logic programming concepts needed in the sequel, and is self-contained. For additional details and rationale the reader is referred to [3, 7, 14, 16].

We start by presenting the syntax of *extended logic programs* and defining the *paraconsistent stable model semantics* of extended logic programs [14, 16], generalizing Answer Set Semantics [7] to the paraconsistent case. Extended Logic Programming is the target language of the transformations discussed in the following sections, and we resort only to the disjunctive free fragment of the languages described in [14, 16].

The syntax of logic programs is built-up from ordinary first-order atomic formulae (or atoms). An atom has the form $a(t_1, \ldots, t_n)$ where each t_i ($0 \leq i \leq n$) is a first-order term (an atom with zero arguments is simply written as a). A first-order term is either a variable, a constant, or a function symbol applied to terms. We assume throughout the paper that the language has no function symbols. An usual convention in logic programming is that variables start with capital letters. The set of atoms is denoted by At, and 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 $\neg At = \{\neg A : A \in At\}$. Similarly, 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

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

Intuitively, an objective literal represents a (positive or negative) evidence, while the default negated literal represents a 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. We allow coexistence of positive and negative evidence and for formalizing this we will use a paraconsistent logic.

Definition 2.3. A *program 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 $0 \leq m \leq n$. The left side of the clause (w.r.t. :-) is called the *head* and the right side is designated as *body* of the clause. A fact is a program clause with empty body ($n = m = 0$), succinctly represented by L_0. .

If no variables occur in the literals of a program clause, then the program clause is *ground* (this notion can be trivially extended to literals, sets of literals, logic programs, etc.). Without loss of generality, a non-ground program clause stands for all its ground instances, obtained by substituting variables by ground terms, in our case constant symbols. Therefore, variables are implicitly universally quantified. Informally, a program clause represents an implication: if every literal in the body is true then the head must also be true. Therefore, the comma symbol is interpreted as conjunction. 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 can be seen as a clause with the head being the atom *false* (or $\bot$) representing falsehood.

Definition 2.4. An *extended logic program* (ELP) is a set of program clauses and integrity constraints. Moreover, a *definite extended logic program* is simply an ELP without integrity constraints and occurrences of default negated literals.

Example 2.4. Assume that the following clauses belong to the extended logic program $\mathcal{P}$.

`guilty :- guilty.`	*"Someone is guilty if he is guilty"*
`innocent :- not guilty.`	*"Someone is innocent if we cannot prove he is guilty."*
`¬guilty :- innocent.`	*"Someone is not guilty if he is innocent."*
`male :- not female.`	*"Someone is a male if we cannot prove he is a female."*
`female :- not male.`	*"Someone is a female if we cannot prove she is a male."*

The last two clauses express the idea that a person is either a male or female.

□

Example 2.5. Consider the following non-ground clause belonging to an ELP $\mathcal{P}$ and that the set of all constants occurring in $\mathcal{P}$ is $\{$`tweety`, `piu`$\}$.

(1) `fly(X) :- bird(X), not penguin(X).` *"Every bird flies, if it cannot be proved it is a penguin."*

The non-ground clause above stands for the two ground clauses below, also called ground instances of clause (1).

```
fly(tweety) :- bird(tweety), not penguin(tweety).
fly(piu) :- bird(piu), not penguin(piu).
```

The atom `fly(tweety)` is a ground literal, whereas `fly(X)` is not. We can also say that the atom `fly(tweety)` is a ground instance of `fly(X)` (i.e. `fly(tweety)` $\in ground($`fly(X)`$)$).

□

Informally, the set $ground(At)$ contains all ground instances of any atom $A \in At$. Moreover, given an ELP $\mathcal{P}$, $ground(\mathcal{P})$ is the set of all ground instances of any clause $c \in \mathcal{P}$. It is easy to see that the definition of $ground$ can be also extended to a sequence of literals $L_1, \ldots, L_n$ ($n \geq 1$), i.e. $ground(L_1, \ldots, L_n)$.

The declarative semantics of a program captures its meaning and it 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 2.5. An *interpretation* $\mathcal{I}$ of an extended logic program $\mathcal{P}$ is any subset of $ground(OLit) =$

$ground(At) \cup \neg ground(At)$. An interpretation induces the following consequence relation:

$$\begin{array}{lll} \mathcal{I} \models L & \text{iff} & L \in \mathcal{I} \\ \mathcal{I} \models \text{not } L & \text{iff} & L \notin \mathcal{I} \\ \mathcal{I} \models C_1, \ldots, C_n & \text{iff} & \mathcal{I} \models C_1 \text{ and } \ldots \text{ and } \mathcal{I} \models C_n \end{array}$$

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

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*.

Example 2.6. Consider the extended logic program $\mathcal{P}$ of example 2.4 and a possible interpretation is

$$\mathcal{I} = \{\texttt{innocent}, \neg\texttt{guilty}, \texttt{male}\} .$$

For instance, the literals `guilty`, `female`, *not* `male` do not hold in $\mathcal{I}$ and thus are considered to be *false*, i.e. $\mathcal{I} \not\models$ `guilty`, $\mathcal{I} \not\models$ `female`, $\mathcal{I} \not\models not$ `male`. Obviously, $\mathcal{I} \models \neg$`guilty`, not `female`.

□

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*. Formally:

Definition 2.6. A *model* $\mathcal{M}_\mathcal{P}$ of an extended logic program $\mathcal{P}$ is any interpretation that satisfies every program clause and integrity constraint of $ground(\mathcal{P})$, i.e.

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 2.6 is also a model of the program. Intuitively, an integrity constraint discards all model candidates that make every literal in its body *true*.

A program may have several models and they are ordered by set inclusion. 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. Any definite ELP has a least model, other ELPs may have several minimal models. This intuition is captured by the following definition.

Definition 2.7. 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 extended logic program $\mathcal{P}^\mathcal{I}$ such that $L_0 \texttt{:-} L_1, \ldots, L_m.$ is in $\mathcal{P}^\mathcal{I}$ iff there is a program clause of the form $L_0 \texttt{:-} L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n.$ from $ground(\mathcal{P})$ such that $\mathcal{I} \models not\ L_{m+1}, \ldots, not\ L_n$. The interpretation $\mathcal{I}$ is a *paraconsistent stable model* of $\mathcal{P}$ iff $\mathcal{I}$ is the least model of $\mathcal{P}^\mathcal{I}$ and $\mathcal{I}$ satisfies all integrity constraints of $\mathcal{P}$.

Example 2.7. Consider once more the extended logic program $\mathcal{P}$ of example 2.4 and assume that $\mathcal{P}_1 = \mathcal{P} \cup \{\texttt{:- female.}\}$. Although, $\mathcal{I}_1 = \{\texttt{innocent}, \neg\texttt{guilty}, \texttt{female}\}$ is a model of $\mathcal{P}$, it cannot be a model of $\mathcal{P}_1$ because the integrity constraint rejects any interpretation containing the literal `female`.

The following interpretations

$$\begin{aligned} \mathcal{M}_1 &= \{\texttt{guilty}, \texttt{male}\} , \\ \mathcal{M}_2 &= \{\texttt{innocent}, \neg\texttt{guilty}, \texttt{male}\} \end{aligned}$$

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` is a justification to itself (see first clause). But, we can justify having `innocent` in $\mathcal{M}_2$ because there is no evidence that `guilty` is *true* (i.e. by default `guilty` is *false*) and then, by the second clause, we must have that `innocent` is *true*.

Formally, taking definition 2.7, the reduct of $\mathcal{P}_1$ w.r.t. $\mathcal{M}_1$ is the definite extended logic program $\mathcal{P}_1^{\mathcal{M}_1}$ consisting of the following clauses.

```
guilty :- guilty.
¬guilty :- innocent.
male.
```

The least model of $\mathcal{P}_1^{\mathcal{M}_1}$ is $\{\texttt{male}\}$. Since $\mathcal{M}_1 \neq \{\texttt{male}\}$, we conclude that $\mathcal{M}_1$ is not a paraconsistent stable model. It can easily be checked that $\mathcal{M}_2$ is a paraconsistent stable model.

□

Whenever confusion does not arise, we sometimes use the term model instead of paraconsistent stable model.

Extended logic programs can also be queried. Next, we introduce the notion of query.

Definition 2.8. A *query* is a pair $(L_1, \ldots, L_n \ , \ \mathcal{P})$, with $n \geq 1$, where $\mathcal{P}$ is an ELP and each L_i is a literal.

We consider two cases for defining answers.

- The *answer to a ground query* $(L_1, \ldots, L_n \ , \ \mathcal{P})$ is `yes` iff there is a paraconsistent stable model $\mathcal{M}$ of $\mathcal{P}$ such that $\mathcal{M} \models L_1, \ldots, L_n$.
- An *answer to a non-ground query* $(L_1, \ldots, L_n \ , \ \mathcal{P})$ (i.e. L_i is a non-ground literal, for some $1 \leq i \leq n$) is the set

  $$\{L'_1, \ldots, L'_n \ \mid \ L'_1, \ldots, L'_n \in ground(L_1, \ldots, L_n) \text{ and } \mathcal{M} \models L'_1, \ldots, L'_n\} ,$$

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

Example 2.8. Consider the ELP $\mathcal{P}$ of example 2.4 and the queries $(\texttt{innocent}, \mathcal{P})$ and $(\texttt{guilty}, \mathcal{P})$. The answer to the first query is `yes`, since `innocent` belongs to a model of $\mathcal{P}$ (see $\mathcal{M}_2$ in example 2.7). However, we should answer negatively to the second question because the literal `guilty` does not belong to any paraconsistent stable model of $\mathcal{P}$.

□

The paraconsistent stable model semantics coincides with the stable model semantics [1, 6, 8] whenever explicit negated literals do not occur in the program. It has close connections to Nelson's constructive logic system N^- [2], as shown in [14].

Finally, we refer that *Smodels* [11] and *dlv* [5] 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.

3. A Language for Defining Rough Relations

In this section, we present a language for defining and querying rough relations, based on logic programming. It substantially extends our previous work [18] by allowing lower approximations and boundaries of rough relations in the head and in the body of clauses.

The main intuitive idea underlying our work is as follows. A rough relation S divides the universe in four regions: $\underline{S}$, $\underline{\neg S}$, $\overline{S}$, and the remaining part of the universe not contained in any of those. Then, using clauses we can combine regions of different rough relations to define implicitly a new rough relation.

We give the semantics of the new language, as the least model semantics used in [18] is no longer applicable.

3.1. The syntax

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 either a variable or a constant. We follow the usual convention that variables start with upper case letter (e.g. `Distance`) and constants begin with lower case (e.g. `small`). Moreover, an n-ary predicate p is usually designated as p/n.

Given a predicate p/n, expressions 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 3.1. A *rough clause* is any formula of the form

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

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

A *rough program* $\mathcal{P}$ is a finite set of rough clauses. Moreover, clauses with an empty body (i.e. $n = 0$ and $m = 0$) are usually called *facts*.

Intuitively, each predicate p denotes a rough relation P and we use rough literals to represent evidence about tuples. For instance, the facts $\overline{p}(t_1, \ldots, t_n).$ and $\overline{\neg p}(t_1, \ldots, t_n).$ express the information that the tuple $\langle t_1, \ldots, t_n \rangle$ belongs both to the rough relation P and to its complement $\neg P$ (thus, to the boundary of P, denoted by $\underline{\overline{p}}(\ldots)$). Obviously, $\underline{\overline{p}}(\ldots)$ and $\underline{\overline{\neg p}}(\ldots)$ have the same meaning.

The lower (upper) approximation of P is represented by $\underline{p}(\ldots)$ ($\overline{p}(\ldots)$). For example, the fact $\underline{p}(t_1, \ldots, t_n).$ states that the tuple $\langle t_1, \ldots, t_n \rangle$ is a positive example of rough relation P but cannot be a negative example of it (i.e. $\langle t_1, \ldots, t_n \rangle \in P$ and $\langle t_1, \ldots, t_n \rangle \notin \neg P$).

A decision table $\mathcal{D}$ can be easily represented in our language. A row $\langle c_1, \ldots, c_n \rangle$ of $\mathcal{D}$ corresponding to positive (negative) example, where each c_i is the value of a conditional attribute, is represented as the fact $\overline{d}(c_1, \ldots, c_n)$. ($\overline{\neg d}(c_1, \ldots, c_n)$.)

Since rough clauses allow lower and upper approximations of a relation as well as boundaries to occur both in the body and 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 represent that the boundary of a rough relation Q is contained in the lower approximation of another rough relation P. If predicates q and p denote the rough relations Q and P, respectively, then the rough clause

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

captures such information.

Given a rough relation P with n attributes, an n-ary tuple t is *undefined w.r.t.* P iff t is neither a positive nor a negative example of the relation, i.e.

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

We can test in the body of a rough clause whether a tuple $\langle t_1, \ldots, t_n \rangle$ is undefined w.r.t. P, by using the literal $p?(t_1, \ldots, t_n)$.

The following example motivates the potential usefulness of our language. More examples are presented in the next section.

Example 3.1. A relation $Train$ has two arguments (attributes) representing time and location, respectively. Two (or more) sensors automatically detect presence/absence of an approaching train at a crossing, producing facts like $\overline{\texttt{train}}\texttt{(12:50,montijo).}$ automatically added to the knowledge base. A malfunction of a sensor may result in the contradictory fact $\overline{\neg\texttt{train}}\texttt{(12:50,montijo).}$ 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.

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

3.2. The Semantics

In this section, we present a transformation of rough programs into extended logic programs, introduced in section 2.2. In this way, we obtain a semantics for our language.

The intuition is as follows. Assume that P and Q are the rough relations denoted by predicates p and q, respectively. Then, the literal $p(t_1, \ldots, t_n)$ is a statement that the tuple $\langle t_1, \ldots, t_n \rangle$ is in 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 ELPs, as formalized by the following transformation τ_2. This transformation can be used to compile rough literals in the bodies of source (i.e. rough) program clauses.

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

The translation above is not directly applicable to the heads, since the heads in the target programs can contain neither conjunctions of literals nor default negated literals. Therefore, it may be necessary to compile a clause in the source program into a clause and an integrity constraint of the target program, as described below. For example, consider a rough clause like $\underline{p}(X1,\ldots,Xn)\texttt{:-}\ \underline{\overline{q}}(X1,\ldots,Xn).$ stating that the boundary of Q is contained in the lower approximation of P. 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, $p(X1,\ldots,Xn)\texttt{:-}\ q(X1,\ldots,Xn), \neg q(X1,\ldots,Xn).$ and $\texttt{:-}\ \neg p(X1,\ldots,Xn), q(X1,\ldots,Xn), \neg q(X1,\ldots,Xn).$ capture the same information as the rough clause above. The program clause states that tuples belonging to both Q and $\neg Q$ also belong to P, while the integrity constraint 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 $\neg p$ should essentially be viewed as a new predicate symbol representing explicit negation.

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

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 3.2. Assume that we have two decision tables

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

similar to the one of example 2.1, but each referring to a different period of time (e.g. year 1 and year 2, respectively). 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 belongs to the boundary of table $\mathcal{D}\text{eathmi}_1$ and to the lower approximation of the rough relation represented by table $\mathcal{D}\text{eathmi}_2$, then we may interpret this fact as an increase of risk for those patients.

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 $\underline{\overline{Risk}}$ correspond to an increase, decrease, and stability of the risk of a cardiac event, respectively.

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

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

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

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).
```

□

Let $\mathcal{P}$ be a rough program and $\mathcal{P}' = \tau_1(\mathcal{P})$. Each predicate symbol q with arity n (i.e. q/n), occurring in $\mathcal{P}$, denotes the rough relation

$$Q_{\mathcal{M}_{\mathcal{P}'}} = (\{\langle c_1, \ldots, c_n\rangle \mid q(c_1, \ldots, c_n) \in \mathcal{M}_{\mathcal{P}'}\}, \{\langle c_1, \ldots, c_n\rangle \mid \neg q(c_1, \ldots, c_n) \in \mathcal{M}_{\mathcal{P}'}\}) \ ,$$

in a model $\mathcal{M}_{\mathcal{P}'}$ of $\mathcal{P}'$. Recall that $\tau_1(\mathcal{P})$ is an extended logic program and, therefore, may have several paraconsistent stable models (or none). In each model, the predicate q may denote a different rough relation. Consequently, the denotation of a predicate is always with respect to a model.

Finally, we give an example illustrating a situation where a rough program has more than one paraconsistent stable model.

Example 3.3. Consider that we want to represent some expert knowledge saying that

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

Moreover, three 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`, `diseaseB`, or an `infection`. Symptoms and clinical test results form the conditional attributes. The decision tables associated with `diseaseA` and `diseaseB` have the same conditional attributes and some of these also appear in the decision table related with `infection`. For instance, a conditional attribute is `temperature` that can have the values `low`, `normal`, or `high`. Another conditional attribute is the identification (`Id`) of the patient.

The expert knowledge can be represented as follows. Assume that predicates `diseaseA` and `diseaseB` have arity 4 (i.e. decision tables have four conditional attributes) and `infection` has arity 3.

```
diseaseA(Id, X, normal, Z)   :-   infection(Id, normal, Y),
                                  ¬diseaseB(Id, X, normal, Z).

diseaseB(Id, X, normal, Z)   :-   infection(Id, normal, Y),
                                  ¬diseaseA(Id, X, 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 plus the following facts encoding the information contained in the decision tables.

$\overline{\neg\texttt{diseaseA}}(36, \texttt{c}_1, \texttt{normal}, \texttt{c}_3)$. $\quad$ $\overline{\neg\texttt{diseaseB}}(36, \texttt{c}_1, \texttt{normal}, \texttt{c}_3)$.

$\overline{\texttt{infection}}(36, \texttt{normal}, \texttt{c}_2)$.

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

(i) In one model $\langle 36, c_1, \text{normal}, c_3\rangle \in \underline{\overline{DiseaseA}}$ and $\langle 36, c_1, \text{normal}, c_3\rangle \in \underline{\neg DiseaseB}$.

(ii) In another model $\langle 36, c_1, \text{normal}, c_3\rangle \in \underline{\overline{DiseaseB}}$ and $\langle 36, c_1, \text{normal}, c_3\rangle \in \underline{\neg DiseaseA}$.

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 plan to address the issue by extending to clauses the quantitative measures associated with decision tables. In the example above, the model referred in (i) is obtained by applying the first clause above, while model (ii) is obtained by applying the second clause. If the tuple $\langle \texttt{36}, \texttt{c}_1, \texttt{normal}, \texttt{c}_3\rangle$ 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.

3.3. Queries

This section proposes a language to query rough programs. This can be achieved by adapting existing systems based on the stable model semantics [5, 11], which is a topic we are currently working on. Here, we only present queries and their expected answers. Since there might exist more than one model, answers are computed w.r.t. one paraconsistent stable model of the compiled rough program. If a program has a unique paraconsistent stable model, which may often be the case[2], the answers will refer to this model.

Definition 3.2. 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 an atom and each L_i ($i = 1, 2$) is a rough literal. Moreover, a rough query is *well-formed* if the following conditions are satisfied.

[2]For instance, any definite extended logic program or rough program whose clauses do not contain lower approximations in their bodies, has a unique model.

(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, both rough literals L_1 and L_2 must have the same arguments[3] (i.e. the first argument term of L_1 should be equal to the first argument term of L_2, the second argument term of L_1 should be equal to the second argument term of L_2, and so on).

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

The three cases $(\overline{L}, \mathcal{P})$, $(\underline{L}, \mathcal{P})$, and $(\overline{\underline{L}}, \mathcal{P})$ of a rough query have already been introduced in our previous work [18]. Let us take one case and, informally, explain what is being queried and the corresponding answer. For instance, with the query $(\overline{\underline{q}}(c_1, c_2), \mathcal{P})$ 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 $\tau_1(\mathcal{P})$. 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 the upper approximation of the rough relation denoted by predicate q in some model of $\tau_1(\mathcal{P})$.

Formally, a rough query $(\mathcal{Q}, \mathcal{P})$, where $\mathcal{Q}$ is of the form $\mathcal{Q}_1$ or $\mathcal{Q}_2$, is translated into a query to the ELP $\tau_1(\mathcal{P})$

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

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 query is answered positively (i.e. the test succeeds) if $\mathcal{P}'$ has at least one paraconsistent stable model. Otherwise, the query is answered negatively (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 *ELP* 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 ELP with integrity constraints, for each possible case of L_2 (i.e. $\overline{L}, \underline{L}, \overline{\underline{L}}$).

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

It is trivial to extend function τ_3 for compiling queries of the form $(\mathcal{Q}_3, \mathcal{P})$. Thus, given a rough program $\mathcal{P}$, we have that the answer to the query $(\mathcal{Q}_3, \mathcal{P})$ is `yes` iff the ELP $\tau_3((\mathcal{Q}_3, \mathcal{P}))$ has a model.

The query language proposed here is slightly more general than the one presented in [17], since now we allow for testing arbitrary inclusions between lower and upper approximations. For instance in [17], we could not test whether the lower approximation of one rough relation R_1, denoted by predicate $\mathtt{r}_1/n$, was included in the upper approximation of another rough relation R_2, denoted by another predicate

[3] In reality, this constraint can be lifted as we discuss in the conclusion.

$\mathtt{r}_2/n$. With the rough query language discussed in this paper, 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 (see e.g. [21]) it is necessary to check rough inclusion or rough equality of given rough relations. We recall the notions of rough inclusion and rough equality [13]. 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 $\overline{Q_1} \subseteq \overline{Q_2}$. 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 $\mathtt{q}_1/n$ and $\mathtt{q}_2/n$ 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 question

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

4. Application Examples

4.1. 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 [20], in the context of the variable precision rough set model [19] . 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 conditional attributes to the table. Alternatively, if some attributes have been subject to discretization, 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.

The main idea described in [20] 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 with the aim of reducing this region of the rough relation. 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.

Each equivalence 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 equivalence 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 the equivalence 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.

Example 4.1. Consider the decision table shown in table2.

Table 2.

	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

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

$\neg$deathmi(<70, no, no).

$\neg$deathmi(>70, no, no).

deathmi(>70, yes, yes).

$\vdots$

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

It is easy to see that the equivalence classes

$$\begin{aligned} [o_4] &= \{o_4, o_5, o_6\} \ , \\ [o_7] &= \{o_7, o_8\} \end{aligned}$$

are in the boundary region.

In order to reduce the boundary region, a different set of conditional attributes can be considered for some of these equivalence classes (in the boundary area), i.e. the new set of attributes considered for one equivalence class may be different from the set of attributes considered for another equivalence 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 equivalence classes the same attributes as in the original table have been considered, but increased discretization precision of the conditional attributes has been applied to each of these equivalence 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 $[o_4]$: instead of the age, it was considered whether the patient was a smoker. For patients in $[o_7]$ only different discretization for the Age attribute was applied.

As the reader can see from tables 3 and 4, the boundary region has been reduced to one equivalence class with two patients only, $[o_4] = \{o_4, o_6\}$.

Table 3.

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

Table 4.

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

The decision tables 3 and 4 can be represented by the following facts. Note that each decision table is recorded under a different predicate name (deathmi, $\texttt{deathmi}_1$, and $\texttt{deathmi}_2$).

$\texttt{deathmi}_1$(yes, no, yes).
$\neg\texttt{deathmi}_1$(yes, no, no).
$\neg\texttt{deathmi}_1$(yes, no, yes).
$\neg\texttt{deathmi}_2$(<40, yes, yes).
$\texttt{deathmi}_2$(>40, yes, yes).

Putting together all the above decision tables, we create a new rough relation shown below. It corresponds to the initial decision table $\mathcal{D}$eathmi with a reduced boundary, by integrating tables 3 and 4.

Table 5.

	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

Using rough clauses, the rough relation corresponding to table 5 can be easily encoded. Predicate $\texttt{deathmi}_3$ denotes this rough relation.

(1) $\overline{\texttt{deathmi}_3}$(Age,Hypert,Scanabn,null) :- deathmi(Age,Hypert,Scanabn).

(2) $\overline{\neg\texttt{deathmi}_3}$(Age,Hypert,Scanabn,null) :- $\neg$deathmi(Age,Hypert,Scanabn).

(3) $\overline{\texttt{deathmi}_3}$(null,Hypert,Scanabn,Smoke) :- $\overline{\texttt{deathmi}_1}$(Hypert,Scanabn,Smoke).

(4) $\overline{\neg\texttt{deathmi}_3}$(null,Hypert,Scanabn,Smoke) :-
$\overline{\neg\texttt{deathmi}_1}$(Hypert,Scanabn,Smoke).

(5) $\overline{\texttt{deathmi}_3}$(Age,Hypert,Scanabn,null) :- $\overline{\texttt{deathmi}_2}$(Age,Hypert,Scanabn).

(6) $\overline{\neg\texttt{deathmi}_3}$(Age,Hypert,Scanabn,null) :- $\overline{\neg\texttt{deathmi}_2}$(Age,Hypert,Scanabn).

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

□

The example above raises the question whether it would be possible to apply the same refinement technique to part of the boundary (i.e. to some indiscernibility classes in the boundary), instead of applying it to whole boundary. In our current framework, we do not envisage a simple way to achieve this. In our opinion, this problem could be easily addressed, if we would generate unique identifiers for each indiscernibility class. This issue is by now a matter of future research.

A slightly different method (also proposed in [20]) 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 easily encoded in our language.

Example 4.2. Consider the decision table 2 of example 4.1. 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 conditional 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).

Table 6.

	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 7.

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

Looking at the table 6, we see that the boundary has been reduced to two objects $[o_7] = \{o_7, o_8\}$. Experts decided to try a different discretization for the age attribute, with the aim to eventually eliminate the boundary region. Thus, another decision table (see table 7) was associated with the boundary of table 6 .

Decision tables 6 and 7 are represented in our language as a set of facts under predicates $\texttt{deathmi}_1$ and $\texttt{deathmi}_2$, respectively.

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

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

The relation between the decision tables 2, 6, and 7, a hierarchical linear structure, can be easily encoded using rough clauses. All these decision tables are related to the same rough set (relation) that we designate by predicate $\texttt{deathmi}_3$.

$\overline{\texttt{deathmi}_3}$(Age,Hypert,Scanabn,null,null) :- $\underline{\texttt{deathmi}}$(Age,Hypert,Scanabn).
$\overline{\neg\texttt{deathmi}_3}$(Age,Hypert,Scanabn,null,null) :- $\underline{\neg\texttt{deathmi}}$(Age,Hypert,Scanabn).
$\overline{\texttt{deathmi}_3}$(Age,null,Scanabn,Sex,Smoke):- $\underline{\neg\texttt{deathmi}_1}$(Age,Scanabn,Sex,Smoke).
$\overline{\neg\texttt{deathmi}_3}$(Age,null,Scanabn,Sex,Smoke):- $\underline{\neg\texttt{deathmi}_1}$(Age,Scanabn,Sex,Smoke).
$\overline{\texttt{deathmi}_3}$(Age,null,Scanabn,Sex,Smoke) :- $\overline{\texttt{deathmi}_2}$(Age,Scanabn,Sex,Smoke).
$\overline{\neg\texttt{deathmi}_3}$(Age,null,Scanabn,Sex,Smoke) :- $\overline{\neg\texttt{deathmi}_2}$(Age,Scanabn,Sex,Smoke).

When considering a graphical interface, eventually showing the hierarchical structure of the tables, the clauses above could be generated automatically.

4.2. Avoiding Potentially 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 intuitively, we may 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 for which objects o the knowledge about attributes B is absolutely necessary for making a decision, i.e. for determining $d(o)$.

This problem has been studied in [9] 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).

Assume 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 underlying decision table $\mathcal{D}_A$ and $\mathcal{D}_{A\setminus B}$, respectively. Intuitively, this means that when considering the set of attributes $A \setminus B$, several equivalence classes may be merged into one single equivalence class. Thus, when considering less attributes, the approximation space may be formed by a smaller number of larger equivalence classes. Consequently, the lower approximation of D_A tends to shrink and the upper approximation tends to enlarge.

Moreover, it can also be easily concluded that

$$\begin{aligned} \underline{D}_{A\setminus B} &\subseteq \underline{D}_A \ , \\ \underline{\neg D}_{A\setminus B} &\subseteq \underline{\neg D}_A \ , \end{aligned}$$

where D_A is the rough set obtained from decision table $\mathcal{D}_A$ (i.e. considering all conditional attributes), and $D_{A\setminus B}$ is the rough set obtained from table $\mathcal{D}_{A\setminus B}$ (i.e. only taking into account the attributes ($A \setminus B$)).

When considering the reduced set of attributes $A \setminus B$, the boundary region of D_A also increases, since some equivalence 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 equivalence 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 $\overline{\underline{D_A}}$, where they already belong. Interpreting each of the rough set regions as a subset of the universe [4], the set of migrating objects can be defined as

$$Migrate(A, B, D) = (\overline{\underline{D}}_{A\setminus B} \cap \underline{D}_A) \cup (\overline{\underline{D}}_{A\setminus B} \cap \underline{\neg D}_A) \ . \quad (1)$$

Although the definition above looks different from the one used in [9], 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

$$\neg Migrate(A, B, D) = \underline{\neg D}_{A\setminus B} \cup \underline{D}_{A\setminus B} \cup \overline{\underline{D}}_A \ . \quad (2)$$

[4] In this particular context, it makes more sense to interpret each rough set region as a set of objects instead of a set of tuples characterizing each equivalence class belonging to that region.

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 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 and it is ongoing work.

Both expressions above, the set of migrating and non-migrating objects, can be translated to a set of rough clauses. The next example illustrates this application.

Example 4.3. 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 eventually with heart problems. Assume that the conditional 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.

Table 8.

Age	Test $\texttt{A}_1$	Test $\texttt{A}_2$	**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

From table 8, 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) $\{\langle \texttt{>70, b1, c2} \rangle, \langle \texttt{<40, b3, c5} \rangle\} \subseteq \overline{\underline{Deathmi}}$;

The table is encoded as facts in our language.

$\overline{\neg\texttt{deathmi}}$(>70, b1, c1). $\overline{\texttt{deathmi}}$(>70, b1, c2).
$\overline{\neg\texttt{deathmi}}$(>70, b1, c2). $\overline{\texttt{deathmi}}$(>40 <70, b2, c3).
$\overline{\neg\texttt{deathmi}}$(>40 <70, b2, c4). $\overline{\texttt{deathmi}}$(<40, b3, c5).
$\overline{\neg\texttt{deathmi}}$(<40, b3, c5). $\overline{\texttt{deathmi}}$(<40, b3, c3).

Moreover, the following clauses monitor the impact in the boundary region of not considering test $\texttt{A}_2$. 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.

(1) $\overline{\texttt{d}}$(Age,Test_A1) :- $\overline{\texttt{deathmi}}$(Age,Test_A1,Test_A2).
(2) $\overline{\neg\texttt{d}}$(Age,Test_A1) :- $\overline{\neg\texttt{deathmi}}$(Age,Test_A1,Test_A2).
(3) $\overline{\texttt{migrate}}$(Age,Test_A1) :- $\underline{\overline{\texttt{d}}}$(Age,Test_A1),
$\underline{\texttt{deathmi}}$(Age,Test_A1,Test_A2).
(4) $\overline{\texttt{migrate}}$(Age,Test_A1) :- $\underline{\overline{\texttt{d}}}$(Age,Test_A1),
$\underline{\neg\texttt{deathmi}}$(Age,Test_A1,Test_A2).
(5) $\overline{\neg\texttt{migrate}}$(Age,Test_A1) :- $\underline{\neg\texttt{d}}$(Age,Test_A1).
(6) $\overline{\neg\texttt{migrate}}$(Age,Test_A1) :- $\underline{\texttt{d}}$(Age,Test_A1).
(7) $\overline{\neg\texttt{migrate}}$(Age,Test_A1) :- $\underline{\overline{\texttt{deathmi}}}$(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 equivalence classes, one originating from the lower approximation and the other from the outside ($\underline{\neg Deathmi}$).

By clause (6), $\langle$<40, b4$\rangle \in \overline{\neg Migrate}$. This indiscernibility class remains in the lower approximation, even after dropping attribute $\texttt{A}_2$. 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 3rd argument of deathmi corresponding to the expensive medical test.

Let $\mathcal{P}$ be the rough program obtained from the set of facts encoding decision table 8 together with the rough clauses $(1) - (7)$. The query ($\overline{\texttt{migrate}}$(Age,A1), $\mathcal{P}$) returns all tuples that belong to the upper approximation of rough relation denoted by migrate (i.e. $\overline{Migrate}$). Thus, the tuples $\langle$>40 <70, b2$\rangle$ and $\langle$>70, b1$\rangle$ are the answer to the given query. Then, for instance, we may conclude that for a new patient whose age is between 40 and 70 and who obtained the result b2 for the test $\texttt{A}_1$, the medical test $\texttt{A}_2$ should be also preformed.

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 $(\overline{\texttt{migrate}}\texttt{(>70,A1)},\ \mathcal{P})$. As answer we get the tuple $\langle$`>70, b1`$\rangle$ indicating that only patients with outcome `b1` for test $\texttt{A}_1$ should be submitted to test $\texttt{A}_2$.

Another relevant question is

- *"Which patients should not be submitted to test A_2?"*

that can be represented by the rough query $(\overline{\neg\texttt{migrate}}\texttt{(Age,A1)},\ \mathcal{P})$. As answer we get both tuples $\langle$`<40, b3`$\rangle$ and $\langle$`<40, b4`$\rangle$. This answer can be interpreted as stating that if a patient conforms to the case

$$((\texttt{Age} < 40) \wedge (\texttt{Test A}_1 = \texttt{b3})) \vee ((\texttt{Age} < 40) \wedge (\texttt{Test A}_1 = \texttt{b4}))$$

then test $\texttt{A}_2$ is rather irrelevant.

4.3. 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** [4], 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 as proposed default logic [15]. We would like to stress that we are not aiming to show a complete correspondence between default logic and the language proposed in our framework, but only to illustrate that certain forms of default knowledge can be also encoded in our language. The default knowledge used in the following examples correspond to normal default rules in the context of default logic.

Example 4.4. Consider table 9

$$\mathcal{D}\texttt{istance} = (U, \{\texttt{Dif}, \texttt{Road Conditions}, \texttt{Physical Distance}\}, \texttt{Distance}).$$

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 (captured by attribute `Dif`);
- road conditions (dry, wet, snow, or ice); and
- the distance between the vehicle and the one after it.

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,

Table 9.

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

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 coded 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)  large̅(X1,X2,X3):- p̄(X1,X2,X3,large).
(2)  ¬large̅(X1,X2,X3):- medium̅(X1,X2,X3).
(3)  ¬large̅(X1,X2,X3):- small̅(X1,X2,X3).

(4)  medium̅(X1,X2,X3):- p̄(X1,X2,X3,medium).
(5)  ¬medium̅(X1,X2,X3):- small̅(X1,X2,X3).
(6)  ¬medium̅(X1,X2,X3):- large̅(X1,X2,X3).

(7)  small̅(X1,X2,X3):- p̄(X1,X2,X3,small).
(8)  ¬small̅(X1,X2,X3):- medium̅(X1,X2,X3).
(9)  ¬small̅(X1,X2,X3):- large̅(X1,X2,X3).
```

Another decision table, see table 10,

$$\mathcal{D}\text{anger} = (U, \{\texttt{Dif}, \texttt{Road Conditions}, \texttt{Distance}\}, \texttt{Danger}) \ ,$$

shows whether a number or traffic situations has been classified as dangerous by an expert.

As usual, this table is represented as a set of facts.

Table 10.

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

$\overline{\neg\texttt{danger}}$(10, dry, large). $\overline{\texttt{danger}}$(20, ice, small).
$\overline{\neg\texttt{danger}}$(0, wet, medium). $\overline{\texttt{danger}}$(10, wet, medium).
$\overline{\neg\texttt{danger}}$(10, wet, medium). $\overline{\texttt{danger}}$(-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.

$$\frac{\texttt{small}(X1, X2, X3) : \texttt{danger}(X1, X2, \texttt{small})}{\texttt{danger}(X1, X2, \texttt{small})} \ . \tag{1}$$

Technically, the default rule (1) can be read as follows: if in a certain situation $\texttt{small}(x1, x2, x3)$ holds (i.e. it can be proved) and $\texttt{danger}(x1, x2, \texttt{small})$ is consistent with the current knowledge, then (by default) we assume that $\texttt{danger}(x1, x2, \texttt{small})$ holds, too. Note that $\texttt{danger}(x1, x2, \texttt{small})$ is consistent with the current knowledge, if we have no evidence that $\neg\texttt{danger}(x1, x2, \texttt{small})$ is true. Thus, no contradiction with the available knowledge arises by the fact that $\texttt{danger}(x1, x2, \texttt{small})$ 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 (2) and (3).

$$\frac{\texttt{medium}(X1, X2, X3) : \neg\texttt{danger}(X1, X2, \texttt{medium})}{\neg\texttt{danger}(X1, X2, \texttt{medium})} , \tag{2}$$

$$\frac{\texttt{large}(X1, X2, X3) : \neg\texttt{danger}(X1, X2, \texttt{large})}{\neg\texttt{danger}(X1, X2, \texttt{large})} . \tag{3}$$

Next, we show that the above default rules can be expressed in our language.

(10) $\overline{\texttt{danger}_1}\texttt{(X1,X2,small):-}\ \overline{\texttt{small}}\texttt{(X1,X2,X3),danger?(X1,X2,small).}$

(11) $\overline{\neg\texttt{danger}_1}\texttt{(X1,X2,medium):-}\ \overline{\texttt{medium}}\texttt{(X1,X2,X3),danger?(X1,X2,medium).}$

(12) $\overline{\neg\texttt{danger}_1}\texttt{(X1,X2,large):-}\ \overline{\texttt{large}}\texttt{(X1,X2,X3),danger?(X1,X2,large).}$

Rough clauses (10)−(12) express the default rules (1)−(3), respectively. 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 $\texttt{medium}(\texttt{-10,snow,9})$ 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 (2)) cannot be applied because $\texttt{danger}(\texttt{-10,snow,medium})$ holds and consequently $\langle\texttt{-10,snow,medium}\rangle$ is not undefined w.r.t. rough relation $Danger$.

However, $\texttt{small}(\texttt{-10,snow,9})$ 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 (1)), we conclude $\texttt{danger}(\texttt{-10,snow,small})$.

□

Next example, illustrates the use of priorities between default rules.

Example 4.5. Consider once more the (decision) tables of the previous example and default rules (1) and (2). Moreover, we also assume that

- *"If distance between vehicles is medium and the road conditions are icy, then we may assume that the traffic situation is dangerous (unless it can be proved otherwise)."*

Using a default rule, we could express this statement as

$$\frac{\texttt{medium}(X1, \texttt{ice}, X3) : \texttt{danger}(X1, \texttt{ice}, \texttt{medium})}{\texttt{danger}(X1, \texttt{ice}, \texttt{medium})} . \tag{4}$$

Informally, looking at the second row of the first table, we conclude that $\texttt{medium}(\texttt{10,ice,15})$ holds. Moreover, from the second table we can conclude that the tuple $\langle\texttt{10,ice,medium}\rangle$ is not considered as a dangerous (or non-dangerous) situation (actually, there is no such tuple in the second table). Thus, default rule (2) can be applied and we conclude that $\neg\texttt{danger}(\texttt{10,ice,medium})$ also holds. Similarly default rule (4) can be applied, to conclude that $\texttt{danger}(\texttt{10,ice,medium})$ 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 (2) and (4) are applicable, then we may give priority to (4) and block application of default (2) for safety reasons.

First, we define a new rough relation expressing the default rules (1) and (4).

$$(10)\quad \overline{\texttt{danger}_1}\texttt{(X1,X2,small)}\texttt{:-}\ \overline{\texttt{small}}\texttt{(X1,X2,X3)},\texttt{danger?(X1,X2,small).}$$

$$(11)\quad \overline{\texttt{danger}_1}\texttt{(X1,ice,medium)}\texttt{:-}\ \overline{\texttt{medium}}\texttt{(X1,ice,X3)},\texttt{danger?(X1,ice,medium).}$$

Then, we put together the knowledge coming from the table $\mathcal{D}$anger with the default knowledge. The last rough clause, (15), encodes default rule (2) and gives it lower priority than rough clause (11) (default rule (4)).

$$(12)\quad \overline{\texttt{danger}_2}\texttt{(X1,X2,X3)}\texttt{:-}\ \overline{\texttt{danger}}\texttt{(X1,X2,X3).}$$

$$(13)\quad \overline{\neg\texttt{danger}_2}\texttt{(X1,X2,X3)}\texttt{:-}\ \overline{\neg\texttt{danger}}\texttt{(X1,X2,X3).}$$

$$(14)\quad \overline{\texttt{danger}_2}\texttt{(X1,X2,small)}\texttt{:-}\ \overline{\texttt{danger}_1}\texttt{(X1,X2,small).}$$

$$(15)\quad \overline{\neg\texttt{danger}_2}\texttt{(X1,X2,medium)}\texttt{:-}\ \overline{\texttt{medium}}\texttt{(X1,X2,X3)},\texttt{danger?(X1,X2,medium)},\ \texttt{danger}_1\texttt{?(X1,X2,medium).}$$

5. Conclusions and Future Work

We introduced a language for representing a vague knowledge in the framework of the rough set theory and an expressive query language.

We defined a natural translation of our language to extended logic programming under the paraconsistent stable model semantics. This opens for re-use of the existing logic programming systems based on the stable model semantics for answering rough set queries.

Our language uses common notions of rough set theory: lower and upper approximations, and boundaries for implicit definitions of rough sets. The usual technique of defining rough sets by decision tables is embedded as a special case. The language is flexible enough to allow separate definitions of each region of a rough set. Since a program $\mathcal{P}$ in our language may have different models, the rough sets defined by $\mathcal{P}$ may not be unique. This fact can be useful in some applications.

The proposed query language is very expressive. It makes it possible not only to search for elements in particular regions of rough relations but also to test for rough inclusion and rough equality, which is essential in some applications. For a matter of simplicity, we have not really defined a declarative semantics for rough queries. This could be achieved by defining the notion of logical consequence ($\models$) for rough programs. For this reason, we restricted the type of rough queries that are allowed to well--formed ones, since we believe that the intuitive meaning of these queries is obvious. Assume that R_1

and R_2 are two binary rough relations. Our definition of rough query does not allow to test whether for each pair $\langle c_1, c_2\rangle \in \underline{R_1}$, we have that $\langle c_2, c_1\rangle \in \overline{R_2}$, since the query $(\underline{r_1}(X_1, X_2) \subseteq \overline{r_2}(X_2, X_1), \mathcal{P})$ is not well-formed. However, the restriction (ii) of being well-formed could be easily lifted and these queries could be answered by applying the same strategy, as explained in section 3.3.

Development of a system based on the presented ideas, integrating quantitative techniques [10], and testing it on real life examples is ongoing work.

Acknowledgements

The authors would like to thank to Witold Łukaszewicz and Andrzej Szałas for the insightful discussion about system **CAKE** and also for the suggestions that contributed to the examples presented in section 4.3 .

References

[1] Alferes, J. J., Leite, J. A., Pereira, L. M., Przymusinska, H., Przymusiski, T. C.: Dynamic Updates of Non-monotonic Knowledge Bases, *Journal of Logic Programming*, number 45, Elsevier, 2000.

[2] Almukdad, A., Nelson, D.: Constructible Falsity and Inexact Predicates, *Journal of Symbolic Logic*, **49**, 1984, 231–233.

[3] Apt, K., Bol, R.: Logic Programming and Negation: A Survey, *Journal of Logic Programming*, 19/20, Elsevier, May/July 1994.

[4] Doherty, P., Lukaszewicz, W., Szalas, A.: CAKE: A Computer Aided Knowledge Engineering Technique, *Proc. of the 15th European Conference on Artificial Intelligence, (ECAI'02)*, IOS Press, Amsterdam, 2002.

[5] Eiter, T., Leone, N., Mateis, C., Pfeifer, G., Scarcello, F.: The KR System `dlv`: Progress Report, Comparisons and Benchmarks, *KR'98: Principles of Knowledge Representation and Reasoning* (A. G. Cohn, L. Schubert, S. C. Shapiro, Eds.), Morgan Kaufmann, San Francisco, California, 1998.

[6] Gelfond, M., Lifschitz, V.: The Stable Model Semantics for Logic Programming, *Proc. of the 5th International Conference on Logic Programming* (R. A. Kowalski, K. Bowen, Eds.), Seattle, USA, 1988.

[7] Gelfond, M., Lifschitz, V.: Logic programs with Classical Negation, *Proc. of the 7th International Conference on Logic Programming* (Warren, Szeredi, Eds.), MIT Press, 1990.

[8] Inoue, K., Sakama, C.: Negation as Failure in the Head, *Journal of Logic Programming*, **35**(1), 1998, 39–78.

[9] Komorowski, J., Øhrn, A.: Modelling Prognostic Power of Cardiac Tests using Rough Sets, *Journal of Artificial Intelligence in Medicine*, **15**(2), 1999, 167–191.

[10] Komorowski, J., Pawlak, Z., Polkowski, L., Skowron, A.: Rough Sets: A Tutorial, *Rough Fuzzy Hybridization. A New Trend in Decision-Making*, Springer-Verlag, 1999.

[11] Niemelä, I., Simons, P.: Efficient Implementation of the Well-founded and Stable Model Semantics, *Proc. of the Joint International Conference and Symposium on Logic Programming* (M. Maher, Ed.), MIT Press, Bonn, Germany, 1996.

[12] Pawlak, Z.: Rough Sets, *International Journal of Information and Computer Science*, **11**(5), 1982, 341–356.

[13] Pawlak, Z.: *Rough sets. Theoretical Aspects of Reasoning about Data*, Kluwer Academic Publishers, Dordrecht, 1991.

[14] Pearce, D.: Answer sets and constructive logic, II: Extended logic programs and related non-monotonic formalisms, *Logic Programming and Nonmonotonic Reasoning - Proc. of the 2nd International Workshop* (L. Pereira, A. Nerode, Eds.), MIT Press, 1993.

[15] Reiter, R.: A Logic for Deafult Reasoning, *Journal of Artificial Intelligence*, **13**, 1980, 81–132.

[16] Sakama, C., Inoue, K.: Paraconsistent Stable Semantics for Extended Disjunctive Programs, *Journal of Logic and Computation*, **5**(3), 1995, 265–285.

[17] Vitória, A., Damásio, C. V., Małuszyński, J.: Query Answering for Rough Knowledge Bases, *Proc. of the 9th International Conference on Rough Sets, Fuzzy Sets, Data Mining, and Granular Computing, , RSFDGrC'03, Chongqing, China* (G. Wang, Q. Liu, Y. Yao, A. Skowron, Eds.), 2639, Springer, 2003, ISBN 3-540-14040-9.

[18] Vitória, A., Małuszyński, J.: A Logic Programming Farmework for Rough Sets, *Proc. of the 3rd International Conference on Rough Sets and Current Trends in Computing, RSCTC'02* (J. Alpigini, J. Peters, A. Skowron, N. Zhong, Eds.), number 2475 in LNCS/LNAI, Springer-Verlag, 2002.

[19] Ziarko, W.: Variable Precision Rough Set Model, *Journal of Computer and Systems Science*, **46**(1), 1993, 39–59.

[20] Ziarko, W.: Acquisition of Hierarchy-Structured Probabilistic Decision Tables and Rules from Data, *Proc. of the World Congress on Computational Intelligence*, Honolulu, 2002.

[21] Ziarko, W., Fei, X.: VPRSM Approach to WEB Searching, *Proc. of the 3rd International Conference on Rough Sets and Current Trends in Computing, RSCTC'02* (J. Alpigini, J. Peters, A. Skowron, N. Zhong, Eds.), number 2475 in LNCS/LNAI, Springer-Verlag, 2002.