A Four-Valued Logic for Rough Set-Like Approximate Reasoning

Jan Małuszyński[1], Andrzej Szałas[1,2], and Aida Vitória[3]

[1] Linköping University, Department of Computer and Information Science
581 83 Linköping, Sweden
janma@ida.liu.se
[2] The University of Economics and Computer Science
Olsztyn, Poland
andsz@ida.liu.se
[3] Dept. of Science and Technology, Linköping University
S 601 74 Norrköping, Sweden
aidvi@itn.liu.se

Abstract. This paper extends the basic rough set formalism introduced by Pawlak [1] to a rule-based knowledge representation language, called Rough Datalog, where rough sets are represented by predicates and described by finite sets of rules. The rules allow us to express background knowledge involving rough concepts and to reason in such a knowledge base. The semantics of the new language is based on a four-valued logic, where in addition to the usual values TRUE and FALSE, we also have the values BOUNDARY, representing uncertainty, and UNKNOWN corresponding to the lack of information. The semantics of our language is based on a truth ordering different from the one used in the well-known Belnap logic [2, 3] and we show why Belnap logic does not properly reflect natural intuitions related to our approach. The declarative semantics and operational semantics of the language are described. Finally, the paper outlines a query language for reasoning about rough concepts.

1 Introduction

The seminal ideas of Pawlak [1, 4, 5, 6] on the treatment of imprecise and incomplete data opened a new area of research, where the notion of rough sets is used in theoretical studies as well as practical applications.

Rough sets are constructed by means of approximations obtained by using elementary sets which partition a universe of considered objects. The assumption as to partitioning of the universe has been relaxed in many papers (see, e.g., [7, 8, 9, 10, 11, 12, 13, 14, 15]), however the Pawlak's idea of approximations has remained the same.

This paper extends the basic rough set formalism to a rule-based language, where rough sets are represented by predicates and are described by finite sets of rules. The rules allow one to express background knowledge concerning rough concepts and to reason in such a knowledge base. The new language is different from that proposed in [14, 15], where the rules described rough sets by combining their regions (lower approximation, upper approximation and boundary region). In contrast to the language

J.F. Peters et al. (Eds.): Transactions on Rough Sets VI, LNCS 4374, pp. 176–190, 2007.

described in this paper, the rules expressed in the language presented in [14, 15] refer explicitly to different regions of a rough set.

Lifting the level of description makes the definitions easier to understand, also for the people not familiar with the technicalities of rough sets. The semantics of the new language is based on a four-valued logic, where in addition to the usual values TRUE and FALSE we have the values BOUNDARY representing uncertain/inconsistent information and UNKNOWN corresponding to the lack of information. As discussed in Section 3.2, the well-known four-valued Belnap logic [3, 2] does not properly reflect the natural intuitions related to our approach. We propose instead a slightly different truth ordering and use it, together with the standard knowledge ordering, for defining a declarative semantics of our language.

By using the four-valued logic we propose, we are then able to deal with some important issues.

First of all, we are able to provide a natural semantics for Datalog-like rules where negation can be used freely, both in the bodies and in the heads of rules. This, in previous approaches to various variants of negation, has always been problematic either due to the high computational complexity of queries or to a nonstandard semantics of negation, often leading to counterintuitive results (for an overview of different approaches to negation see, e.g., [16]). Our semantics reflects intuitions of fusing information from various independent sources. If all sources claim that a given fact is true (respectively, false) then we have an agreement and attach TRUE (respectively FALSE) to that fact. If information sources disagree in judgement of a fact, we attach to it the value BOUNDARY. If no source provides an information about a given fact, we then make it UNKNOWN.

Second, we are able to import knowledge systems based on the classical logic without any changes and make them work directly within the rough framework. In such cases these systems would act as single information sources providing answers TRUE, FALSE, when queried about facts. Possible conflicting claims of different systems would then be solved by the same, uniform four-valued approach we propose. This might be useful in combining low level data sources, like classifiers as well as higher level expert systems.

Third, one can import rough set-based systems, or systems supporting approximate reasoning, like for example, those described in [14, 15], or [17, 18]. In the latter three-valued logics are used (identifying BOUNDARY and UNKNOWN).

The paper is structured as follows. First, in Section 2, we recall basic definitions related to rough sets and approximations. Next, in Section 3, we discuss our choice of four-valued logic. In Section 4 we introduce Rough Datalog and provide its semantics. Section 5 outlines a query language and discusses its implementation in logic programming. Finally, Section 6 concludes the paper.

2 Rough Sets

According to Pawlak's definition (see, e.g., [19]), a rough set S over a universe U is characterized by two subsets of U:

Table 1. Test results considered in Example 1

car	station	safe
a	s1	yes
a	s2	no
b	s2	no
c	s1	yes
d	s1	yes

- the set $\underline{S}$, of all objects which can be *certainly* classified as belonging to S, called *the lower approximation* of S, and
- the set $\overline{S}$, of all objects which can be *possibly* classified as belonging to S, called *the upper approximation* of S.

The set difference between the upper approximation and the lower approximation, denoted by $\overline{S}$, is called the *boundary* region.

In practice, in order to describe a given reality, one chooses a set of attributes and the elements of the underlying universe are described by tuples of attribute values. Rough sets are then defined by *decision tables* associating membership decisions with attribute values. The decisions are not exclusive: a given tuple of attribute values may be associated with the decision "yes", with the decision "no", with both or with none, if the tuple does not appear.

Example 1. Consider a universe consisting of cars. If a car passed a test then it may be classified as safe (and as not safe, if it failed the test). Tests may be done independently at two test stations. The upper approximation of the rough set of safe cars would then include cars which passed at least one test. The lower approximation of the set would include the cars which passed all tests (and therefore, they did not fail at any test). The boundary region consists of the cars which passed one test and failed at one test. Notice that there are two other categories of cars, namely those which were not tested and those which failed all tests.

As an example consider the situation described in Table 1, where the first column consists of cars, the second column consists of test stations and the third one contains test results. Denote by "Safe" the set of safe cars. Then:

- the upper approximation of Safe consists of cars for which there is a decision "yes", i.e., $\overline{\text{Safe}} = \{a, c, d\}$
- the lower approximation of Safe consists of cars for which all decisions are "yes", i.e., $\underline{\text{Safe}} = \{c, d\}$
- the boundary region of Safe consists of cars for which there are both decisions "yes" and "no", i.e., $\overline{\underline{\text{Safe}}} = \{a\}$. ◁

A decision table, representing a concept t, may be represented as a finite set of literals of the form $t(y)$ or $\neg t(x)$, where y ranges over the tuples of attribute values associated with the decision "yes" and x ranges over the tuples of attribute values associated with the decision "no".

Example 2. For the Example 1 with the universe of cars $\{a, b, c, d, e\}$ and with two test stations, we may have the decision table, shown in Table 1, encoded as

$$\{\texttt{safe}(a), \neg\texttt{safe}(a), \neg\texttt{safe}(b), \texttt{safe}(c), \texttt{safe}(d)\} .$$

Notice that the literal $\texttt{safe}(a)$ indicates that car a has passed a safety test in one of the stations while literal $\neg\texttt{safe}(a)$ states that the same car as failed a safety test in another test station.

In this case the rough set Safe has the approximations

$$\overline{\text{Safe}} = \{a, c, d\} \text{ and } \underline{\text{Safe}} = \{c, d\}.$$

The rough set ¬Safe, describing those cars that have failed some test, has the approximations $\overline{\neg\text{Safe}} = \{a, b\}$ and $\underline{\neg\text{Safe}} = \{b\}$.

Note that it is totally unknown what is the status of car e. ◁

We notice that a decision table $\mathcal{T}$ of this kind defines two rough sets, T and $\neg T$, with a common boundary region which is the intersection of the upper approximations of both sets, i.e. $\overline{T} \cap \overline{\neg T}$. As rough sets are usually defined by decision tables, we then adopt the following definition (used also in [20, 14, 15]).

Definition 1. *A rough set S over a universe U is a pair $\langle \overline{S}, \overline{\neg S} \rangle$ of subsets of U.* ◁

Intuitively, the rough set S describes those elements of U having certain property. The set $\overline{S}$ is the upper approximation of S, and consists of the elements of U for which there is an indication of having the given property. On the other hand, the set $\overline{\neg S}$ consists of the elements for which there is an indication of not having the property. In Example 2, $\overline{\text{Safe}} = \{a, c, d\}$ and $\overline{\neg\text{Safe}} = \{a, b\}$.

Remark 1

1. Observe that Definition 1 differs from the understanding of rough sets as defined by Pawlak. In fact, the definition of Pawlak requires the underlying elementary sets used in approximations to be based on equivalence relations, while Definition 1 relaxes this requirement. Such differences are examined and discussed in depth in [12].
2. Since relations are sets of tuples, we further on also use the term *rough relation* to mean a rough set of tuples. ◁

3 A Four-Valued Logic for Rough Sets

3.1 The Truth Values for Rough Membership

Our objective is to define a logical language for rough set reasoning. The vocabulary of the language includes predicates to be interpreted as rough relations and constants to be used for representing attribute values. Consider an atomic formula of the form $p(t_1, \cdots, t_n)$, where p is a predicate, denoting a rough set P, and $t_1, \ldots, t_n$ (with $n > 0$) are constants. We now want to define the truth value represented by an atom $p(t_1, \cdots, t_n)$. Let $v = \langle t_1, \ldots, t_n \rangle$ and "$-$" denote the set difference operation. Then, the following cases are possible:

- $v \in \overline{P} - \overline{\neg P}$: intuitively, we only have evidence that the element of the universe described by the attributes v has property P. Thus, the truth value of $p(v)$ is defined to be TRUE.
- $v \in \overline{\neg P} - \overline{P}$: intuitively, we only have evidence that the element of the universe described by the attributes v does not have property P. Thus, the truth value of $p(v)$ is defined to be FALSE.
- $v \in \overline{P} \cap \overline{\neg P}$: in this case, we have contradictory evidences, i.e. an evidence that an element of the universe described by the attributes v has property P and an evidence that it does not have the property P. This is an uncertain information and we use the additional truth value BOUNDARY to denote it.
- $v \notin \overline{P} \cup \overline{\neg P}$: in this case, we have no evidence whether the element of the universe described by the attributes v has property P. We then use another truth value called UNKNOWN.

3.2 Is Belnap Logic Suitable for Rough Reasoning?

The truth values emerging from our discussion have been studied in the literature outside of the rough set context for defining four-valued logic. A standard reference is the well-known Belnap's logic [2]. We now recall its basic principles and we discuss whether it is suitable for rough set reasoning.

The Belnap logic is defined by considering a distributive *bilattice* of truth values and introducing logical connectives corresponding to the operations in the bilattice.

Bilattices have been introduced in [21, 22]. They generalize the notion of Kripke structures (see, e.g., [23]). A *bilattice* is a structure $B = \langle U, \leq_t, \leq_k \rangle$ such that U is a non-empty set, $\leq_t$ and $\leq_k$ are partial orderings each making set U a lattice. Moreover, there is usually a useful connection between both orderings.

We follow the usual convention that $\wedge_t$ and $\vee_t$ stand respectively for the meet and join, with respect to $\leq_t$. The symbols $\wedge_k$ and $\vee_k$ stand respectively for the meet and join, with respect $\leq_k$. Operations $\wedge_t$ and $\vee_t$ are also called the *conjunction* and *disjunction*, and $\wedge_k$ and $\vee_k$ are often designated as the *consensus* and *accept all* operators, respectively.

The bilattice used in Belnap's logic is shown in Fig 1. In the *knowledge ordering,* $\leq_k$, UNKNOWN is the least value, reflecting total lack of knowledge. Each of the values TRUE and FALSE provide more information than UNKNOWN. Finally, the INCONSISTENT value corresponds to the situation when there is evidence for both TRUE and FALSE.[1] The *truth ordering* $\leq_t$ (see Fig 1) has TRUE as its largest element, and FALSE as its smallest element.

Example 3. Assume that a family owns two cars: a and e. We want to check if the family has a safe car. This corresponds to the logical value of the expression

$$\texttt{safe}(a) \vee_t \texttt{safe}(e)\ . \tag{1}$$

[1] Observe that INCONSISTENT is replaced in our approach by BOUNDARY, which is closer to intuitions from rough set theory.

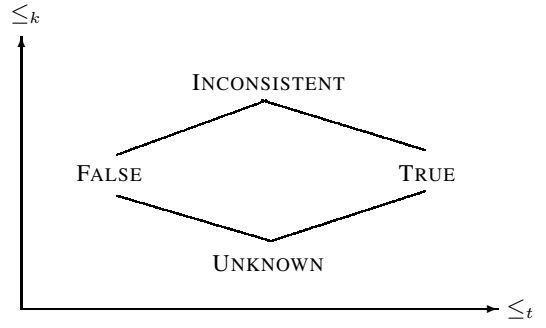

Fig. 1. The bilattice corresponding to Belnap's logic

The truth values of $\texttt{safe}(a)$ and $\texttt{safe}(e)$ are determined by the results of the tests, as specified in Example 2. Thus $\texttt{safe}(a)$ has the value BOUNDARY and $\texttt{safe}(e)$ has the value UNKNOWN. If the join operation $\vee_t$ is defined by Belnap's logic, as shown in Fig 1, then

$$\text{INCONSISTENT} \vee_t \text{UNKNOWN} = \text{TRUE} .$$

This contradicts our intuitions. We know that the safety of car a is unclear, since the results of both safety tests are contradictory, and we know nothing about safety of car e.

Asking instead if all cars of the family are safe,

$$\texttt{safe}(a) \wedge_t \texttt{safe}(e) , \tag{2}$$

would in Belnap's logic result in the answer FALSE. However, we really do not know whether both cars are safe because we do not have any information about the safety of car e. In contrast to the answer obtained in the Belnap's logic, UNKNOWN seems to be a more intuitive answer in this case. ◁

The example above shows that the truth ordering of Fig 1, and consequently Belnap's logic are not suitable for rough set-based reasoning. On the other hand, the knowledge ordering of Fig. 1 is adequate for our purposes. Indeed, the values TRUE and FALSE show that only one kind of evidence, either positive or negative, is known while the value BOUNDARY indicates existence of contradictory evidence, both positive and negative.

3.3 A Four-Valued Logic for Rough Set Reasoning

We now define a four-valued logic suitable for rough set-based reasoning by modifying the bilattice of Fig.1. As discussed in Section 3.2, only the truth ordering is to be changed. We will use the new truth ordering to define conjunction ($\wedge_t$) as the greatest lower bound in this ordering. The ordering should preserve the usual meaning of conjunction for the truth values TRUE and FALSE. Intuitively, the value UNKNOWN represents the lack of information. Thus, the result of its conjunction with any other

truth value is accepted here to be UNKNOWN. A new information may arrive, replacing UNKNOWN by either TRUE, or FALSE, or BOUNDARY, providing in each case a different result. On the other hand, BOUNDARY represents existence of contradictory information. Its conjunction with TRUE would not remove this contradiction. Thus, we define the result of such a conjunction to be BOUNDARY. It also seems natural, that the conjunction of FALSE with TRUE or BOUNDARY gives FALSE. Consequently the truth ordering, $\leq_t$, is redefined in our framework as

$$\text{UNKNOWN} \leq_t \text{FALSE} \leq_t \text{BOUNDARY} \leq_t \text{TRUE} . \tag{3}$$

The new structure $\mathcal{R} = \langle U, \leq_t, \leq_k \rangle$, where U is the universe of objects of interest, $\leq_t$ is the truth ordering defined in (3), and $\leq_k$ is the knowledge ordering as in the Belnap's logic, gives the meaning of the logical connectives and is used in our approach.

Example 4. Referring to Example 3, we then compute the logical values associated with the queries (1) and (2) by considering the new truth ordering above.

The first query, (1) of Example 3,

$$\text{BOUNDARY} \vee_t \text{UNKNOWN} ,$$

returns the logical BOUNDARY which better corresponds to the intuitions.

For the second query, (2) of Example 3, we have that

$$\text{BOUNDARY} \wedge_t \text{UNKNOWN} = \text{UNKNOWN} .$$

In contrast to Belnap's logic, it is not excluded that some cars of the family of Example 3 are safe, but to be sure we need to obtain some information about the safety of car e. So, the answer UNKNOWN adequately reflects our intuitions. ◁

The proposition below shows that there is a connection between the knowledge ordering and the truth ordering. In this sense, the structure $\mathcal{R}$ can then be seen as a bilattice.

Proposition 1. *Consider the bilattice $\mathcal{R} = \langle U, \leq_t, \leq_k \rangle$ and that $x, y \in U$. The operation $\wedge_t$ is monotonic with respect to $\leq_k$ on both arguments, i.e. if $x \leq_k y$ then, for every $z \in U$, we have $(z \wedge_t x) \leq_k (z \wedge_t y)$ and $(x \wedge_t z) \leq_k (y \wedge_t z)$.*

Proof. Table 2 shows the result. Operation $\wedge_t$ is obviously commutative. ◁

We now define formally the logic underlying our work, called *Rough Logic*.

Definition 2. *Consider the following negation operation $\neg$.*

$$\neg\text{TRUE} \stackrel{\text{def}}{=} \text{FALSE}, \quad \neg\text{FALSE} \stackrel{\text{def}}{=} \text{TRUE},$$
$$\neg\text{BOUNDARY} \stackrel{\text{def}}{=} \text{BOUNDARY}, \quad \neg\text{UNKNOWN} \stackrel{\text{def}}{=} \text{UNKNOWN}.$$

The propositional four-valued logic defined by the bilattice $\mathcal{R}$ together with negation $\neg$ is called the Rough Logic. ◁

Table 2. The table considered in the proof of Proposition 1

z	x	y	$z \wedge_t x$	$z \wedge_t y$
BOUNDARY	UNKNOWN	TRUE	UNKNOWN	BOUNDARY
BOUNDARY	UNKNOWN	FALSE	UNKNOWN	FALSE
BOUNDARY	FALSE	BOUNDARY	FALSE	BOUNDARY
BOUNDARY	TRUE	BOUNDARY	BOUNDARY	BOUNDARY
BOUNDARY	UNKNOWN	BOUNDARY	UNKNOWN	BOUNDARY

z	x	y	$z \wedge_t x$	$z \wedge_t y$
FALSE	UNKNOWN	TRUE	UNKNOWN	FALSE
FALSE	UNKNOWN	FALSE	UNKNOWN	FALSE
FALSE	FALSE	BOUNDARY	FALSE	FALSE
FALSE	TRUE	BOUNDARY	FALSE	FALSE
FALSE	UNKNOWN	BOUNDARY	UNKNOWN	FALSE

z	x	y	$z \wedge_t x$	$z \wedge_t y$
TRUE	UNKNOWN	TRUE	UNKNOWN	TRUE
TRUE	UNKNOWN	FALSE	UNKNOWN	FALSE
TRUE	FALSE	BOUNDARY	FALSE	BOUNDARY
TRUE	TRUE	BOUNDARY	TRUE	BOUNDARY
TRUE	UNKNOWN	BOUNDARY	UNKNOWN	BOUNDARY

z	x	y	$z \wedge_t x$	$z \wedge_t y$
UNKNOWN	UNKNOWN	TRUE	UNKNOWN	UNKNOWN
UNKNOWN	UNKNOWN	FALSE	UNKNOWN	UNKNOWN
UNKNOWN	FALSE	BOUNDARY	UNKNOWN	UNKNOWN
UNKNOWN	TRUE	BOUNDARY	UNKNOWN	UNKNOWN
UNKNOWN	UNKNOWN	BOUNDARY	UNKNOWN	UNKNOWN

4 Rough Datalog Language

We now define a rule language, called *Rough Datalog*, such that its semantics is based on the Rough Logic. Intuitively, Rough Datalog corresponds to the usual logic programming language Datalog. While predicates in the latter denote crisp relations, in Rough Datalog a predicate p denotes a rough relation P. Thus, Rough Datalog caters for uncertainty in the knowledge.

A *rough literal* is any expression of the form $p(t_1, \ldots, t_n)$ or $\neg p(t_1, \ldots, t_n)$. In Rough Datalog, knowledge is represented in the form of rough clauses,

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

where H and every B_i ($0 \leq i \leq n$) is a rough literal. A rough clause with the empty body (i.e. $n = 0$) is called a *rough fact*. A *rough program* $\mathcal{P}$ is a finite set of rough clauses.

Rough clauses are used to specify rough relations as explained next. Intuitively, a rough clause is to be understood as the knowledge inequality $\leq_k$ stating that the truth value of the body is less than or equal to the truth value of the head. The comma

symbol "," is interpreted as the meet in the truth ordering $\leq_t$. Notice that the arguments of $\leq_k$ are the truth values UNKNOWN, BOUNDARY, TRUE, or FALSE but the logical value associated with a rough clause is either TRUE or FALSE. Information obtained from different rough clauses with heads referring to the same rough relation P (i.e. p or $\neg p$ occurs in the head) is combined using the knowledge join operation $\vee_k$.

Example 5. The following rough clauses belong to an exemplary rough program $\mathcal{P}$.

(1) `¬useful(a) :- red(a), squared(a).`
"Object `a` *is not useful if it is red and squared."*
(2) `squared(a) :- useful(a).` —*"Object* `a` *is squared if it is useful."*
(3) `¬squared(a).` —*"Object* `a` *is not squared."* ◁

4.1 Semantics of Rough Datalog Programs

We now define notions of four-valued interpretation and model, extend the knowledge ordering to interpretations and show that each rough program has the least model in this ordering.

Let $\mathcal{P}$ be a rough program and L be the set of all constant symbols occurring in $\mathcal{P}$. Then, the *Herbrand base* $\mathcal{H}_\mathcal{P}$ is the set of all literals whose predicate symbols occur in $\mathcal{P}$ and whose arguments belong to L.

A four-valued *interpretation* $\mathcal{I}$ of a rough program $\mathcal{P}$ associates with each atom a occurring in $\mathcal{P}$ a logical value

$$\mathcal{I}(a) \in \{\text{UNKNOWN}, \text{TRUE}, \text{FALSE}, \text{BOUNDARY}\}$$

and $\neg\mathcal{I}(a) = \mathcal{I}(\neg a)$.

The notion of interpretation extends naturally to conjunction (disjunction) of literals. Let $l_1, \ldots, l_n$, with $n > 0$, be rough literals.

$$\mathcal{I}(l_1 \wedge_t \cdots \wedge_t l_n) = \mathcal{I}(l_1) \wedge_t \cdots \wedge_t \mathcal{I}(l_n)\ .$$

Definition 3. *An interpretation $\mathcal{I}$ of a rough program $\mathcal{P}$ is any subset of the* Herbrand base $\mathcal{H}_\mathcal{P}$. *Moreover, the rough relation $\mathcal{I}(p)$ is defined as*

$$\mathcal{I}(p) = \langle \overline{\mathcal{I}(p)}, \overline{\neg\mathcal{I}(p)} \rangle = \langle \{t \mid p(t) \in \mathcal{I}\},\ \ \{t \mid \neg p(t) \in \mathcal{I}\} \rangle\ \ . \qquad \triangleleft$$

Intuitively, an interpretation associates each predicate p occurring in a program $\mathcal{P}$ with a rough set. Notice that $\neg\mathcal{I}(p) = \mathcal{I}(\neg p)$. Moreover, we have that

- $\mathcal{I}(p(t)) = \text{UNKNOWN}$, if $t \notin \overline{\mathcal{I}(p)} \cup \overline{\neg\mathcal{I}(p)}$.
- $\mathcal{I}(p(t)) = \text{FALSE}$, if $t \in \underline{\neg\mathcal{I}(p)}$.
- $\mathcal{I}(p(t)) = \text{TRUE}$, if $t \in \underline{\mathcal{I}(p)}$.
- $\mathcal{I}(p(t)) = \text{BOUNDARY}$, if $t \in \overline{\mathcal{I}(p)}$.

Notice that we only consider variable-free rough programs. However, the results presented below can be also extended to rough programs with variables.

An interpretation $\mathcal{I}$ of a rough program $\mathcal{P}$ *satisfies* a rough clause H :- $B. \in \mathcal{P}$ if $\mathcal{I}(B) \leq_k \mathcal{I}(H)$. A *model* $\mathcal{M}$ of $\mathcal{P}$ is any interpretation that satisfies every rough clause belonging to $\mathcal{P}$.

Notice also that the Herbrand base $\mathcal{H}_\mathcal{P}$ is a model of any rough program $\mathcal{P}$. In this model the truth value of every literal is BOUNDARY. However, usually a program has more models. For comparing them we introduce a partial order on interpretations based on the knowledge ordering relation, $\leq_k$.

Definition 4. *Let $\mathcal{I}_1 \subseteq \mathcal{H}_\mathcal{P}$ and $\mathcal{I}_2 \subseteq \mathcal{H}_\mathcal{P}$ be two interpretations. Then, $\mathcal{I}_1 \leq_k \mathcal{I}_2$, if and only if $\mathcal{I}_1(l) \leq_k \mathcal{I}_2(l)$, for every literal $l \in \mathcal{H}_\mathcal{P}$.* ◁

It can be checked that the knowledge ordering on interpretations corresponds to set inclusion.

Proposition 2. *$\mathcal{I}_1 \leq_k \mathcal{I}_2$ if and only if $\mathcal{I}_1 \subseteq \mathcal{I}_2$.* ◁

We show now that there is the least model for every rough program.

Proposition 3. *Let $\mathcal{P}$ be a rough program. Then, $\mathcal{P}$ has the least model with respect to $\leq_k$.*

Proof. To prove that $\mathcal{P}$ has a least model with respect to $\leq_k$, we show that the intersection of all models of $\mathcal{P}$ is also a model of $\mathcal{P}$.

Let $\mathcal{M} = \bigcap_i^n M_i$, where $\{M_1, \ldots, M_n\}$ $(n \geq 1)$ is the set of all models of $\mathcal{P}$. Notice that, by Proposition 2, $\mathcal{M} \leq_k M_i$, with $M_i \in \{M_1, \ldots, M_n\}$. We prove that $\mathcal{M}$ is a model of $\mathcal{P}$. For this we have to show that, for any clause H :- B. $\in \mathcal{P}$, we have $\mathcal{M}(H) \geq_k \mathcal{M}(B)$. We prove this by cases, considering possible truth values of the body of a clause.

(a) If $\mathcal{M}(B)$ = UNKNOWN then $\mathcal{M}$ satisfies the rough clause, since UNKNOWN is the least element in the knowledge ordering.
(b) If $\mathcal{M}(B)$ = TRUE then $W(B) \geq_t$ BOUNDARY, for every model W of $\mathcal{P}$. Hence, $W(H) \geq_k$ TRUE, for every model W of $\mathcal{P}$. Consequently, $\mathcal{M}(H) \geq_k$ TRUE because the literal occurring in the head belongs to every model W. We conclude then that $\mathcal{M}$ satisfies the rough clause.
(c) If $\mathcal{M}(B)$ = FALSE then B includes a literal l that is FALSE in some model of $\mathcal{P}$ and l is either FALSE or BOUNDARY in the other models. Obviously, no literal occurring in B can be UNKNOWN in any model. Consequently, $\mathcal{M}(H) \geq_k$ FALSE because $\neg H$ belongs to every model W. We conclude then that $\mathcal{M}$ satisfies the rough clause.
(d) If $\mathcal{M}(B)$ = BOUNDARY then $W(B) =$ BOUNDARY, for every model W of $\mathcal{P}$. Notice that if $\mathcal{I}(B) =$ BOUNDARY, for some interpretation $\mathcal{I}$ of $\mathcal{P}$, then we have that either $\mathcal{I}(l) =$ TRUE or $\mathcal{I}(l) =$ BOUNDARY, for every literal l in the body B. Hence, $W(H) =$ BOUNDARY, for every model W of $\mathcal{P}$. Consequently, $\mathcal{M}(H) =$ BOUNDARY because $\{H, \neg H\} \subseteq W$, for every model W. We conclude then that $\mathcal{M}$ satisfies the rough clause. ◁

The semantics of a rough program $\mathcal{P}$ is captured by its least model, with respect to $\leq_k$.

Example 6. Consider again the rough program of Example 5. Its least model is $M = \{\neg\texttt{squared(a)}\}$. Hence, `useful`(a) and `red`(a) are UNKNOWN, while `squared`(a) is FALSE. ◁

4.2 A Fixpoint Characterization of the Least Model

We now give a fixpoint characterization of the least model which makes it possible to compute the semantics of a program. We define an operator on interpretations, considered as sets of literals. We show that the operator is monotonic with respect to set inclusion. Thus, it has the least fixpoint (with respect to set inclusion) which can be obtained by iterations of the operator starting with the empty interpretation. We also show that the least fixpoint is a model. Taking into account Proposition 2, we can then conclude that the least fixpoint is also the least model of the program with respect to knowledge ordering. In the following definition if l is a negative literal of the form $\neg a$, then $\neg l$ denotes a.

Definition 5. *Let $\mathcal{P}$ be a rough program. A total function $T_{\mathcal{P}}$ mapping interpretations into interpretations is defined as follows:*

$$\begin{aligned} T_{\mathcal{P}}(\mathcal{I}) = {} & \{l \mid l \text{ :- } B.\ \in \mathcal{P} \text{ and } \mathcal{I}(B) = \text{TRUE}\} \quad \cup \\ & \{\neg l \mid l \text{ :- } B.\ \in \mathcal{P} \text{ and } \mathcal{I}(B) = \text{FALSE}\} \quad \cup \\ & \{l, \neg l \mid l \text{ :- } B.\ \in \mathcal{P} \text{ and } \mathcal{I}(B) = \text{BOUNDARY}\}\ . \end{aligned}$$ ◁

Thus, the set $T_{\mathcal{P}}(\mathcal{I})$ consists of the heads of the rough clauses whose bodies are TRUE or BOUNDARY in $\mathcal{I}$ and, the negated heads of the rules whose bodies are FALSE or BOUNDARY in $\mathcal{I}$. Such a way to gather heads of rules corresponds to defining the result by the disjunction of heads w.r.t. knowledge ordering $\leq_k$.

Proposition 4. *Given a rough program $\mathcal{P}$, the operator $T_{\mathcal{P}}$ is monotonic with respect to set inclusion.*

Proof. The bodies of the program clauses are conjunctions of atoms. By Proposition 1 the conjunction is monotonic with respect to knowledge ordering. Hence by Proposition 2, it is also monotonic with respect to set inclusion of the interpretations. Thus, $\mathcal{I} \subseteq T_{\mathcal{P}}(\mathcal{I})$, for every interpretation $\mathcal{I}$. ◁

The proposition above guarantees that $T_{\mathcal{P}}$ has a least fixpoint (with respect to set inclusion), denoted as LFP($T_{\mathcal{P}}$).

Proposition 5. *Given a rough program $\mathcal{P}$, the LFP($T_{\mathcal{P}}$) coincides with the least model of $\mathcal{P}$.*

Proof. It is easy to see that the interpretation $\mathcal{I} = \text{LFP}(T_{\mathcal{P}})$ is a model of $\mathcal{P}$. Assume the contrary. Then, there exists a clause H:- B. such that $\mathcal{I}(H) <_k \mathcal{I}(B)$. The possible cases are as follows.

- $\mathcal{I}(B)$ = TRUE and $\mathcal{I}(H) \leq_k$ FALSE.
- $\mathcal{I}(B)$ = FALSE and $\mathcal{I}(H) \leq_k$ TRUE.
- $\mathcal{I}(B)$ = BOUNDARY and $\mathcal{I}(H) <_k$ BOUNDARY.

In the first two cases, we immediately obtain the contradiction with the assumption $\mathcal{I} = \text{LFP}(T_{\mathcal{P}})$, since $T_{\mathcal{P}}(\mathcal{I})$ would then include, respectively, the literal H ($\neg H$). A similar contradiction is obtained for the third case, since $\mathcal{I}(B)$ = BOUNDARY means that $T_{\mathcal{P}}(\mathcal{I})$ would then include both literals H and $\neg H$. In any case, we conclude that $\mathcal{I}$ is not a fixpoint of $T_{\mathcal{P}}$.

It remains to prove that the model LFP($T_{\mathcal{P}}$) is the least model in the knowledge ordering. This follows directly from Proposition 2. ◁

Proposition 5 shows that the least model of a program $\mathcal{P}$ can be computed by applying iteratively operator $T_{\mathcal{P}}$, starting from the empty interpretation until the fixpoint is reached. Notice that in the empty interpretation, all literals of the Herbrand base have the truth value UNKNOWN.

We show below a simple example of a rough program, based on a classical example from logic programming, and it illustrates the use of $T_{\mathcal{P}}$ for computation of its semantics.

Example 7. Consider the rough program consisting of the following rough clauses.

(1) `fly(tweety) :- bird(tweety).`
(2) `bird(tweety) :- penguin(tweety).`
(3) ¬`fly(tweety) :- penguin(tweety).`
(4) ¬`dangerous(tweety) :- red(tweety), fly(tweety).`
(5) `penguin(tweety).`
(6) `red(tweety).`

Application of $T_{\mathcal{P}}$ to the empty interpretation gives

$$\mathcal{I}_1 = \texttt{penguin(tweety), red(tweety).}$$

Further iterations of $T_{\mathcal{P}}$ give

$$\begin{aligned}
\mathcal{I}_2 &= \mathcal{I}_1 \cup \{\texttt{bird(tweety)}, \neg\texttt{fly(tweety)}\}, \\
\mathcal{I}_3 &= \mathcal{I}_2 \cup \{\texttt{fly(tweety)}\}, \\
\mathcal{I}_4 &= \mathcal{I}_1 \cup \{\texttt{dangerous(tweety)}, \neg\texttt{dangerous(tweety)}\}, \\
\mathcal{I}_5 &= \mathcal{I}_4 .
\end{aligned}$$

Thus, we conclude that `tweety` belongs to the lower approximations of the rough relations Bird, Penguin and Red and it belongs to the boundary region of rough relations Fly and Dangerous. ◁

5 A Query Language and Its Implementation

In this section we describe a query language for rough programs. We start by defining the notions of rough query and answer. Then, we briefly describe how the query language can be implemented in a logic programming as queries to a definite logic program. Existing systems like Prolog [24], XSB [25], or SModels [26, 27] can then be used to compute the answers. We assume that the reader is familiar with the basics of logic programming [28].

Definition 6. *A* rough query *is a pair* $\langle$:- $l_1, \ldots, l_n$, $\mathcal{P}\rangle$*, with* $n \geq 1$*, where* $\mathcal{P}$ *is a rough program and each* l_i *is a (variable-free) rough literal.* ◁

We need now to define the notion of answer to a rough query.

Definition 7. *Let* $\langle$:- $l_1, \ldots, l_n$, $\mathcal{P}\rangle$ *be a rough query. The* answer *to the rough query is defined as the logical value of* $LFP(T_{\mathcal{P}})(l_1 \wedge_t \cdots \wedge_t l_n)$. ◁

Example 8. Consider the rough program of Example 7. The answer to the rough query

$$\langle \text{:- } \texttt{bird(tweety)}, \mathcal{P}\rangle$$

is TRUE, while the answer to the rough query

$$\langle \text{:- } \texttt{fly(tweety)}, \texttt{penguin(tweety)} \ , \ \mathcal{P}\rangle$$

is BOUNDARY. ◁

Rough programs can be compiled to definite logic programs as described below. A definite logic program is a non-empty set of clauses H:- $A_1, \cdots, A_n$., where each A_i is an atom, $(0 \leq i \leq n)$. Clauses can informally be understood as implications: if every atom A_i is true then H must also be true. Therefore, the comma symbol "," is interpreted as conjunction. Notice that predicates in a logic program denote crisp relations and each atom is either TRUE or FALSE.

Any fact (a clause of the form H:- .) remains unchanged.

Let $C \equiv H$:- $l_1, \cdots, l_n$., where $n \geq 1$, be a rough clause and φ be a function transforming C into a non-empty set of clauses such that $\varphi(C) = \{C\} \cup \phi(C)$, where

$$\phi(C) = \{\neg H\text{:- } l'_1, \cdots, l'_n. \mid (\forall 1 \leq i \leq n : \ l'_i \in \{l_i, \neg l_i\}) \text{ and } H\text{:- } l'_1, \cdots, l'_n. \not\equiv C\} \ . \tag{4}$$

Hence, a rough program is compiled to a definite logic program by applying function φ to each rough clause, i.e. $\varphi(\mathcal{P}) = \bigcup_{C \in \mathcal{P}} \varphi(C)$. We assume that, in the compiled programs, $\neg p$ is treated as a new predicate symbol and $\neg\neg p$ is replaced with p, for any symbol p.

Informally, the main idea underlying the compilation of rough programs is that the body of a rough clause is associated with TRUE, if all literals occurring in it are TRUE. The body of a rough clause is associated with FALSE, if $\neg l$ is TRUE, for at least one literal l occurring in the body, and all other literals in the body are provable. The body of a rough clause is associated with BOUNDARY, if we can prove that it is TRUE and FALSE. If for some literal l in the body, it is neither possible to prove l nor $\neg l$ then the body of a rough clause is associated with UNKNOWN. It can be easily seen that the least model of $\mathcal{P}$ coincides with the least model of the definite logic program $\varphi(\mathcal{P})$.

Remark 2. The transformation expressed by formula (4) results in the exponential blow up of the number of clauses. Namely, if a body of a rule consists of n literals then we have 2^n resulting clauses. However, in practice n is rather small. Moreover, the transformation we have provided is the simplest one and we only intend to show that the required compilation can be done. ◁

Definite logic programs can also be queried. A query for a definite logic program has the same syntax as a rough query. However in logic programming, queries are answered YES or NO depending whether the query is provable or not. A rough query $\langle \mathcal{Q}, \mathcal{P} \rangle$ can be compiled to several queries to $\varphi(\mathcal{P})$. Thus,

- If the query $\langle \mathcal{Q}, \varphi(\mathcal{P}) \rangle$ is answered YES and all queries $\langle \mathcal{Q}', \varphi(\mathcal{P}) \rangle$ are answered NO, with $\mathcal{Q}' \in \phi(Q)$, then the answer to the rough query $\langle \mathcal{Q}, \mathcal{P} \rangle$ is TRUE.
- If the query $\langle \mathcal{Q}, \varphi(\mathcal{P}) \rangle$ is answered NO and some query $\langle \mathcal{Q}', \varphi(\mathcal{P}) \rangle$ is answered YES, with $\mathcal{Q}' \in \phi(Q)$, then the answer to the rough query $\langle \mathcal{Q}, \mathcal{P} \rangle$ is FALSE.
- If the query $\langle \mathcal{Q}, \varphi(\mathcal{P}) \rangle$ is answered YES and some query $\langle \mathcal{Q}', \varphi(\mathcal{P}) \rangle$ is also answered YES, with $\mathcal{Q}' \in \phi(Q)$, then the answer to the rough query $\langle \mathcal{Q}, \mathcal{P} \rangle$ is BOUNDARY.
- Otherwise, the answer to the rough query $\langle \mathcal{Q}, \mathcal{P} \rangle$ is UNKNOWN.

6 Conclusions

In the paper we have presented a four-valued logic which we found adequate for approximate reasoning based on Pawlak's ideas of approximations. The four-valued approach reflects intuitions of fusing information from various, possibly independent data sources.

We have proposed a database language involving approximate concepts and provided its formal semantics. Lifting the level of description from approximations to sets/relations themselves facilitates the use of the language as well as the import of rules from other databases, including those based on two-valued and three-valued logics. A corresponding query language and its implementation have also been discussed.

As noticed in Remark 2, the transformation defined by formula (4) is rather inefficient. We plan to address its improvement in our future work.

References

1. Pawlak, Z.: Information systems – theoretical foundations. Information Systems **6** (1981) 205–218
2. Belnap, N.: A useful four-valued logic. In Eptein, G., Dunn, J., eds.: Modern Uses of Many Valued Logic, Reidel (1977) 8–37
3. Belnap, N.: How a computer should think. In Ryle, G., ed.: Contemporary Aspects of Philosophy, Stocksfield, Oriel Press (1977) 30–55
4. Pawlak, Z.: Rough sets. International Journal of Computer and Information Sciences **11** (1982) 341–356
5. Pawlak, Z.: Rough logic. Bull. Polish Acad. Sci. Tech **35** (1987) 253–258
6. Pawlak, Z.: Rough Sets. Theoretical Aspects of Reasoning about Data. Kluwer Academic Publishers, Dordrecht (1991)
7. Skowron, A., Stepaniuk, J.: Tolerance approximation spaces. Fundamenta Informaticae **27** (1996) 245–253
8. Słowiński, R., Vanderpooten, D.: Similarity relation as a basis for rough approximations. In Wang, P., ed.: Advances in Machine Intelligence & Soft Computing, Raleigh NC, Bookwrights (1997) 17–33

9. Słowiński, R., Vanderpooten, D.: A generalized definition of rough approximations based on similarity. IEEE Trans. on Data and Knowledge Engineering **12(2)** (2000) 331–336
10. Doherty, P., Łukaszewicz, W., Szałas, A.: Tolerance spaces and approximative representational structures. In Günter, A., Kruse, R., Neumann, B., eds.: Proc. 26th German Conf. on AI, KI'2003. Volume 2821 of LNAI., Springer-Verlag (2003) 475–489
11. Doherty, P., Łukaszewicz, W., Szałas, A.: Approximate databases and query techniques for agents with heterogenous perceptual capabilities. In: Proceedings of the 7th Int. Conf. on Information Fusion, FUSION'2004. (2004) 175–182
12. Doherty, P., Szałas, A.: On the correspondence between approximations and similarity. In Tsumoto, S., Slowinski, R., Komorowski, J., Grzymala-Busse, J., eds.: Proceedings of 4th International Conference on Rough Sets and Current Trends in Computing, RSCTC'2004. Volume 3066 of LNAI., Springer-Verlag (2004) 143–152
13. Doherty, P., Łukaszewicz, W., Skowron, A., Szałas, A.: Knowledge Representation Techniques. A Rough Set Approach. Volume 202 of Studies in Fuziness and Soft Computing. Springer-Verlag (2006)
14. Andersson, R., Vitória, A., Małuszyński, J., Komorowski, J.: Rosy: A rough knowledge base system. In: RSFDGrC (2). Volume 3642 of Lecture Notes in Computer Science., Springer (2005) 48–58
15. Vitória, A.: A framework for reasoning with rough sets. Transactions on Rough Sets IV **3700** (2005) 178–276
16. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley Pub. Co. (1996)
17. Doherty, P., Łukaszewicz, W., Szałas, A.: CAKE: A computer-aided knowledge engineering technique. In van Harmelen, F., ed.: Proc. 15th European Conference on Artificial Intelligence, ECAI'2002, Amsterdam, IOS Press (2002) 220–224
18. Doherty, P., Magnusson, M., Szałas, A.: Approximate databases: A support tool for approximate reasoning. Journal of Applied Non-Classical Logics **16** (2006) 87–118 Special issue on Implementation of logics.
19. Pawlak, Z.: A treatise on rough sets. Transactions on Rough Sets IV **3700** (2005) 1–17
20. Vitória, A., Damásio, C., Małuszyński, J.: Query answering for rough knowledge bases. In Wang, G., Liu, Q., Yao, Y., Skowron, A., eds.: Proceedings of 9th Internatinal Conference on Rough Sets, Fuzzy Sets, Data Mining and Granular Computing. Volume 2639 of LNCS., Springer-Verlag (2003) 197–204
21. Ginsberg, M.: Multi-valued logics. In: Proceedings of AAAI-86, Fifth National Conference on Artificial Intelligence. (1986) 243–247
22. Ginsberg, M.: Multivalued logics: a uniform approach to reasoning in ai. Computational Intelligence **4** (1988) 256–316
23. Fitting, M.: Bilattices are nice things. In: Proc. PhiLog Conference on Self-Reference, Copenhagen, The Danish Network for Philosophical Logic and Its Applications (2002)
24. Deransart, P., Ed-Bali, A., Cervoni, L.: Prolog: The Standard Reference Manual. Springer-Verlag (1996)
25. : XSB system. (Available at http://xsb.sourceforge.net/)
26. Niemelä, I., Simons, P.: Smodels - an implementation of stable model and the well-founded semantics for normal logic programs. In Dix, J., Furbach, U., Nerode, A., eds.: Proc. of the 4th Int. Conf. on Logic Programming and Nonmonotonic Reasoning (LPNMR'97). Volume 1265 of LNAI., Springer-Verlag (1997) 420–429
27. Simons, P.: Smodels system. (Available at http://www.tcs.hut.fi/Software/smodels/)
28. Nilsson, U., Małuszynski, J.: Logic, Programming and Prolog, 2nd edition. John Wiley & Sons, http://www.ida.liu.se/~ulfni/lpp/copyright.html (1995)