Paraconsistent Logic Programs with Four-Valued Rough Sets*

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

[1] The College of Economics and Computer Science
10-061 Olsztyn, Poland
janma@ida.liu.se
[2] Institute of Informatics, Warsaw University
02-097 Warsaw, Poland
andsz@mimuw.edu.pl
[3] Department of Science and Technology, Linköping University
S 601 74 Norrköping, Sweden
aidvi@itn.liu.se

Abstract. This paper presents a language for defining four-valued rough sets and to reason about them. Our framework brings together two major fields: rough sets and paraconsistent logic programming. On the one hand it provides a paraconsistent approach, based on four-valued rough sets, for integrating knowledge from different sources and reasoning in the presence of inconsistencies. On the other hand, it also caters for a specific type of uncertainty that originates from the fact that an agent may perceive different objects of the universe as being indiscernible. This paper extends the ideas presented in [9]. Our language allows the user to define similarity relations and use the approximations induced by them in the definition of other four-valued sets. A positive aspect is that it allows users to tune the level of uncertainty or the source of uncertainty that best suits applications.

1 Introduction

We present a language for defining four-valued rough sets and to reason about them. Our framework relates and brings together two major fields: rough sets [8] and paraconsistent logic programming [3]. On the one hand the work discussed here provides a paraconsistent approach, based on four-valued rough sets, for integrating knowledge from different sources and reasoning with possible inconsistent knowledge resulting from this integration. On the other hand, it also caters for a specific type of uncertainty that originates from the fact that an agent may perceive different objects of the universe as being indiscernible. This type of uncertainty has been widely studied in the rough set field. To this end, the proposed language allows the user to define similarity relations modeling indiscernibility and use the similarity-based approximations in definitions of new four-valued sets.

The language discussed in this paper is based on ideas of our previous work [9] that presents a four-valued framework for rough sets. In this approach membership

* Supported in part by the MNiSW grant N N206 399334.

function, set containment and set operations are four-valued, where logical values are **t** (true), **f** (false), **i** (inconsistent) and **u** (unknown). Moreover, the similarity relations used to define approximations of a set are also four-valued. Consequently, we also define four-value notions of upper and lower approximations that extend the usual notion of approximations in rough set theory [8].

In contrast to the standard rough set framework, our framework allows different types of boundary cases to be identified and, consequently, different degrees of uncertainty.

We now briefly compare our work with some of the work in the field of paraconsistent logic programming[1]. From a syntactic perspective, the logic programs introduced in Section 3.1 correspond to Fitting programs [5] which do not involve $\otimes$ and $\forall$ in the right-hand side (body) of the rules. Although both frameworks are intended to deal with inconsistencies and use a four-valued logic, there are some major differences at the semantic level. First, the Belnap's logic underlies the semantics of Fitting programs (see [5], Def. 18). This contrasts with our approach since we use a different truth ordering. Second, the semantics of Fittings programs allows to derive conclusions from false premises. For instance, rule *danger* :– *hot.* can be used to derive that there is no danger, i.e. *danger* is **f**, if *hot* is **f**. In contrast to our framework, a rule of a Fitting program is satisfied if and only if the truth values assigned to the head and to the body are equal.

Paper [1] describes a paraconsistent approach, called P-Datalog, for knowledge base integration based on a four-valued logic and the total order of the four logical values presented there coincides with our truth ordering. However, there are several important differences. First, in contrast to [1], we do not follow the closed-world assumption, i.e. a formula $\neg p(\overline{d})$ is **t** only if some agent states it explicitly and no agent claims that $p(\overline{d})$ is **t**. Second, our language allows explicit negation in the head and bodies of the rules while P-Datalog programs only allow negation by default $\sim$ in the rule's bodies. Consequently, knowledge ordering is used in our framework as a more natural way to combine knowledge from different sources while P-Datalog uses the truth ordering presented in Section 2.1. Third, the rules are interpreted differently. In our language a rule is interpreted as the implication $\rightarrow_k$ defined in Table 1, while the implication $\rightarrow$ underlying the rules of P-Datalog is another. For example, in P-Datalog, the truth-value of $\mathbf{t} \rightarrow \mathbf{i}$ is **f** while in our framework $\mathbf{t} \rightarrow_k \mathbf{i}$ is **t**. Finally, the language we propose allows disjunction $\vee_t$ (join under truth ordering) to be used in the body of a rule.

The paper is organized as follows. Section 2 summarizes the main results of [9]. Section 3 gives a formal definition of the language. Section 4 sketches an implementation proposal. Finally, Section 5 summarizes the paper.

2 The Four-Valued Framework

2.1 Logics Reflecting Truth Ordering and Knowledge Ordering

To construct the language we use two orderings on truth vales, namely the *truth ordering* and *knowledge ordering*. Truth ordering is used for calculations within a single information source while knowledge ordering is used for gathering knowledge from different sources. This approach has been considered in [2] and in the framework of

[1] A detailed comparison is outside of the scope of this paper.

bilattices, in [4,6]. The *truth ordering* $\leq_t$ and the *knowledge ordering* $\leq_k$ are defined as the smallest reflexive and transitive relations satisfying $\mathbf{f} \leq_t \mathbf{u} \leq_t \mathbf{i} \leq_t \mathbf{t}$, $\mathbf{u} \leq_k \mathbf{f} \leq_k \mathbf{i}$, and $\mathbf{u} \leq_k \mathbf{t} \leq_k \mathbf{i}$. The knowledge ordering above coincides with Belnap's knowledge ordering [2]. However, our truth ordering is different from the Belnap's truth ordering. This change is motivated by the fact that Belnap's truth ordering can give counterintuitive results when used for reasoning, as shown in [7].

Having two orderings on truth values, we also have two logics: L_t based on truth ordering and L_k based on knowledge ordering. We denote by $\wedge_t$, $\vee_t$ and $\rightarrow_t$ the connectives of L_t and by $\wedge_k$, $\vee_k$ and $\rightarrow_k$ the corresponding connectives in L_k. Negation, denoted as $\neg$, in both logics has the same semantics. Let GLB^t (GLB^k) and LUB^t (LUB^k) denote the greatest lower bound and the least upper bound of a set of logical values w.r.t truth (knowledge) ordering, respectively. Then, $(a \wedge_t b) = \text{GLB}^t\{a, b\}$ ($(a \wedge_k b) = \text{GLB}^k\{a, b\}$) and $(a \vee_t b) = \text{LUB}^t\{a, b\}$ ($(a \vee_k b) = \text{LUB}^k\{a, b\}$), where a and b are two logical values. Table 1 provides the semantics for implication in both logics, L_t and L_k. Observe that the implication $\rightarrow_t$, introduced in [9], is a four-valued extension of the usual logical implication, suitable for determining set containment and approximations in the case of four-valued sets.

Table 1. Truth tables for $\rightarrow_t$, $\rightarrow_k$, and $\neg$

$\rightarrow_t$	**f**	**u**	**i**	**t**
f	**t**	**t**	**t**	**t**
u	**u**	**u**	**i**	**t**
i	**i**	**i**	**i**	**t**
t	**f**	**u**	**i**	**t**

$\rightarrow_k$	**f**	**u**	**i**	**t**
f	**t**	**t**	**t**	**t**
u	**t**	**t**	**t**	**t**
i	**f**	**f**	**t**	**f**
t	**f**	**f**	**t**	**t**

$\neg$
t
u
i
f

The semantics of quantifier $\forall$ and $\exists$ is given below.

$$\forall x[P(x)] \overset{\text{def}}{=} \underset{x \in U}{\text{GLB}^t}\{P(x)\} \text{ and } \exists x[P(x)] \overset{\text{def}}{=} \underset{x \in U}{\text{LUB}^t}\{P(x)\} .$$

Intuitively, $P(x)$ denotes whether an element x has a property P (i.e. membership of x in a four-valued set P) and it is evaluated to one of the four logical values.

We have the following important propositions.

Proposition 1. *The disjunction $\vee_t$ is monotonic w.r.t. the knowledge ordering.* $\lhd$

Proposition 2. *The conjunction $\wedge_t$ is not monotonic w.r.t. the knowledge ordering.* $\lhd$

This is because ($\mathbf{f} \wedge_t \mathbf{u}$) = **f** but ($\mathbf{i} \wedge_t \mathbf{u}$) = **u**. This shows the lack of monotonicity, since $\mathbf{f} <_k \mathbf{i}$ and $\mathbf{f} >_k \mathbf{u}$. However, we have the following proposition.

Proposition 3. *Let p and q be truth values such that $(p \wedge_t q) \geq_t$ **i**. If $p' \geq_k p$ then $(p' \wedge_t q) \geq_k (p \wedge_t q)$. If $q' \geq_k q$ then $(p \wedge_t q') \geq_k (p \wedge_t q)$.* $\lhd$

Thus, the conjunction is monotonic w.r.t. knowledge ordering for arguments greater or equal than **i**.

2.2 Operations on Four-Valued Sets

Let us now formalize the notion of four-valued sets. Given a universe U, we introduce a new set, disjoint with U, denoted by $\neg U$ and defined by $\neg U \overset{\text{def}}{=} \{\neg x \mid x \in U\}$, where $\neg x$ denotes elements in $\neg U$. A *four-valued set* A on U is any subset of $U \cup \neg U$. Intuitively, $x \in A$ represents the fact that there is an evidence that x is in A and $(\neg x) \in A$ represents the fact that there is an evidence that x is not in A.

In our framework, set membership is four-valued and it extends the usual two-valued membership. We assume that $\neg(\neg x)$ is equal to x.

Set membership, denoted as $\epsilon : U \times 2^{U \cup \neg U} \rightarrow \{\mathbf{f}, \mathbf{u}, \mathbf{i}, \mathbf{t}\}$, is defined by

$$x \,\epsilon\, A = \begin{cases} \mathbf{t} \text{ if } x \in A \text{ and } (\neg x) \notin A \\ \mathbf{i} \text{ if } x \in A \text{ and } (\neg x) \in A \\ \mathbf{u} \text{ if } x \notin A \text{ and } (\neg x) \notin A \\ \mathbf{f} \text{ if } x \notin A \text{ and } (\neg x) \in A \,. \end{cases} \tag{1}$$

The *complement* $\neg A$ *of a four-valued set* A, is defined by $\neg A \overset{\text{def}}{=} \{\neg x \mid x \,\epsilon\, A\}$ and the four-valued set inclusion is defined by $X \Subset Y \overset{\text{def}}{=} \forall x \in U[x \,\epsilon\, X \rightarrow_t x \,\epsilon\, Y]$.

The four-valued operations of intersection and union, defined as

$x \,\epsilon\, (X \Cap Y) \overset{\text{def}}{=} (x \,\epsilon\, X) \wedge_t (x \,\epsilon\, Y)$ and $x \,\epsilon\, (X \Cup Y) \overset{\text{def}}{=} (x \,\epsilon\, X) \vee_t (x \,\epsilon\, Y)$,

generalize the respective standard set operations.

A four-valued extension of rough sets is then defined by four-valued set approximations as follows (cf. [9]). Note that (four-valued) relations are (four-valued) sets of tuples.

Definition 1. A *four-valued similarity relation* σ is any four-valued binary relation on a universe U, satisfying the reflexivity condition, i.e., for any element x of the universe $(x, x) \,\epsilon\, \sigma = \mathbf{t}$. The *neighborhood of element* $x \in U$ *w.r.t.* σ, is the four-valued set $\sigma(x)$ such that $y \,\epsilon\, \sigma(x) \overset{\text{def}}{=} (x, y) \,\epsilon\, \sigma$. $\lhd$

Definition 2. Let A be a four-valued set. Then, the *lower and upper approximations of* A *w.r.t.* σ, denoted by A_σ^+ and $A_\sigma^\oplus$, respectively, are defined by $(x \,\epsilon\, A_\sigma^+) \overset{\text{def}}{=} \sigma(x) \Subset A$ and $(x \,\epsilon\, A_\sigma^\oplus) \overset{\text{def}}{=} \exists y \in U[y \,\epsilon\, (\sigma(x) \Cap A)]$. $\lhd$

Note that approximations are also four-valued. For example, let $U = \{o_1, o_2\}$, the set $A = \{o_1, \neg o_2\}$, and $\sigma(o_1, o_2) = \mathbf{u}$. Then, we have that membership of o_1 in A_σ^+ is unknown ($\mathbf{u}$). It might later appear that $\sigma(o_1, o_2)$ is $\mathbf{f}$ and we then conclude that $(o_1 \,\epsilon\, A_\sigma^+) = \mathbf{t}$. Or, it might appear that $\sigma(o_1, o_2)$ is $\mathbf{t}$ and we then get that $(o_1 \,\epsilon\, A_\sigma^+) = \mathbf{f}$.

3 A Rule Language for Defining Four-Valued Sets

Our aim is to present a rule language for defining four-valued sets. A rule consists of an head and a body. The head and the body are formulae of the four-valued logic of Section 2. Thus, each of them gets one of the four truth values, under a given interpretation. A rule is satisfied in a given interpretation iff whenever the body is $\mathbf{t}$ or $\mathbf{i}$ then the truth value of the head is greater or equal than the truth value of the body, w.r.t. $\leq_k$. Thus, rules reflect the semantics of implication $\rightarrow_k$ as provided in Table 1. This choice corresponds to the intuition that a rule is to be used for increasing knowledge by drawing conclusions. No conclusions are drawn from false or unknown premises (bodies).

3.1 The Syntax

The rules are constructed from:

- *literals* of the form $P(\bar{d})$, $\neg P(\bar{d})$, where P is a relation symbol and $\bar{d}$ is a tuple of terms (variables or constants denoting objects of the universe).
- *truth symbols* : $false,\ unknown,\ incons,\ true.$

Rules are of the form

$$head \text{ :- } l_{11},\ldots,l_{1k_1}\,;l_{21},\ldots,l_{2k_2}\,;\,\ldots\,;l_{m1},\ldots,l_{mk_m}. \tag{2}$$

where $m, k_i \geq 1$, for $1 \leq i \leq m$, $head$ and each l_{ij} $(1 \leq j \leq k_i)$ are literals or truth symbols.

A rule of the form $head \text{ :- } true.$, called a *fact*, is abbreviated as $head.$. A *program* is a finite set of rules. A *ground instance* of a rule is obtained by replacing each variable of the rule by a selected constant occurring in the program.

Example 1. Consider two robots, r_1 and r_2, recognizing similarities between objects on the basis of their shape. Assume that the only shapes are *round, rectangular, square* and *oval*. Due to perceptual limitations r_1 does not recognize the difference between round and oval, and r_2 does not recognize the difference between rectangular and square. The following rules can be used to express (partially) the similarities between objects, as perceived by the robot r_1.

$$\begin{aligned} sim(x,y) \text{ :- } & shape(x,round), shape(y,oval)\ ; \\ & shape(x,oval), shape(y,round). \\ \neg sim(x,y) \text{ :- } & shape(x,square), shape(y,rectangular)\ ; \\ & shape(x,square), shape(y,round). \end{aligned}$$

For the robot r_2 one can consider, e.g., the following rule.

$$\begin{aligned} sim(x,y) \text{ :- } & shape(x,square), shape(y,rectangular)\ ; \\ & shape(x,rectangular), shape(y,square). \end{aligned}$$

As a similarity relation is required to be reflexive (cf. Definition 2), the program also includes the fact $sim(x,x)$. ◁

3.2 The Declarative Semantics

Let $\mathcal{P}$ be a 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 relation symbols occur in $\mathcal{P}$ and whose arguments belong to L.

A four-valued *interpretation* $\mathcal{I}$ of a program $\mathcal{P}$ is any subset of $\mathcal{H}_{\mathcal{P}}$. It associates each ground literal l with a truth value such that $\mathcal{I}(l) \stackrel{\mathrm{def}}{=} (l \epsilon \mathcal{I})$ (ϵ is defined in (1)).

An interpretation $\mathcal{I}_1$ is *smaller or equal* than an interpretation $\mathcal{I}_2$, denoted by $\mathcal{I}_1 \sqsubseteq \mathcal{I}_2$, iff $\mathcal{I}_1$ is a (classical) subset of $\mathcal{I}_2$. Observe that if $\mathcal{I}_1 \sqsubseteq \mathcal{I}_2$ then, for every literal l of the Herbrand base, $\mathcal{I}_1(l) \leq_k \mathcal{I}_2(l)$. The interpretation $\emptyset$, called the empty interpretation, is the least interpretation in this ordering. It assigns **u** to every literal in $\mathcal{H}_{\mathcal{P}}$.

The notion of interpretation is extended to rules of the form (2) by interpreting ‘,’ as conjunction $\wedge_t$, ‘**;**’ is interpreted as the disjunction $\vee_t$, and a rule H :– $B.$ is interpreted as $B \rightarrow_k H$, with the semantics provided in Table 1. The truth symbols are interpreted as **f**, **u**, **i** and **t**. Thus, a given four-valued interpretation determines the truth values of the head and of the body of each rule. If several rules have the same literal H in their head, then H takes the truth value being the disjunction $\vee_k$ of the values assigned to each body's rule. More precisely, if H :– $B_1.$, ..., H :– $B_m.$ are all rules with the head H then the value of H is obtained from $(B_1 \vee_k \ldots \vee_k B_m)$. Note that different rules with the same literal H in their head gather knowledge about H according to knowledge ordering, using $\vee_k$. On the other hand, one often needs to define cases using truth ordering and this motivates the need of **;** in the bodies of the rules.

Definition 3. An interpretation $\mathcal{I}$ *satisfies* a rule H :– $B.$ if the implication $(B \rightarrow_k H)$ is **t** in $\mathcal{I}$. An interpretation is said to be a four-valued *Herbrand model* of a program $\mathcal{P}$ iff it satisfies each rule of $\mathcal{P}$. $\lhd$

Theorem 1. *The (classical) intersection of four-valued Herbrand models of a program $\mathcal{P}$ is a four-valued Herbrand model.*

Proof. Assume the theorem does not hold and let $\mathcal{M}$ be the intersection of the Herbrand models $\mathcal{M}_1$ and $\mathcal{M}_2$ of $\mathcal{P}$. Then, there is a rule H :– $B. \in \mathcal{P}$ such that $\mathcal{M}(H) <_k \mathcal{M}(B)$ and $\mathcal{M}(B) \in \{\mathbf{i}, \mathbf{t}\}$. If $\mathcal{M}(B) = \mathbf{i}$ then the truth value of B must have been **i** both in $\mathcal{M}_1$ and in $\mathcal{M}_2$. Hence, the truth values of the head must also have been **i** in both $\mathcal{M}_1$ and in $\mathcal{M}_2$, and consequently, in $\mathcal{M}$. If $\mathcal{M}(B) = \mathbf{t}$ then the body is **t** in one of the models, assume $\mathcal{M}_1$, and **t** or **i** in the other, $\mathcal{M}_2$. Thus, $\mathcal{M}_1(H) \geq_k \mathbf{t}$ and $\mathcal{M}_2(H) \geq_k \mathbf{t}$. Consequently, H must be **t** or **i** in $\mathcal{M}$. We can then conclude that there is no case under which $\mathcal{M}(H) <_k \mathcal{M}(B)$. This implies that $\mathcal{M}$ must be a model of $\mathcal{P}$. $\lhd$

Corollary 1. *For every program $\mathcal{P}$ there exists the least (w.r.t. $\sqsubseteq$) four-valued model.* $\lhd$

We denote this model by $\mathcal{M}_{\mathcal{P}}$ and consider it the declarative semantics of the program.

3.3 The Fixpoint Semantics

We now define the semantics of a program $\mathcal{P}$ as a fixpoint of an operator on interpretations. We consider here variable-free programs $\mathcal{P}$. If the program has variables then we consider instead all ground instances of its rules. The operator will be denoted $T_{\mathcal{P}}$ and it is a four-valued extension of the classical $T_{\mathcal{P}}$ operator used in logic programming. The operator formalizes the intuition of drawing conclusions with rules.

$$T_{\mathcal{P}}(\mathcal{I}) = \{l \mid l \text{ :- } B. \in \mathcal{P} \text{ and } \mathcal{I}(B) = \mathbf{t}\} \cup \{l, \neg l \mid l \text{ :- } B. \in \mathcal{P} \text{ and } \mathcal{I}(B) = \mathbf{i}\} .$$

Thus, the operator collects all the heads of the ground rules whose bodies are **t** in a given interpretation $\mathcal{I}$. In addition, it takes the heads and the negations of the heads of the rules whose bodies are inconsistent in $\mathcal{I}$.

The following theorem shows that the operator $T_{\mathcal{P}}$ is monotonic w.r.t. $\leq_k$. It follows from Propositions 1 and 3, as $T_{\mathcal{P}}$ uses only the rules for which the body is **t** or **i**.

Theorem 2. *Given a program $\mathcal{P}$ and two four-valued interpretations $\mathcal{I}_1$ and $\mathcal{I}_2$, if $\mathcal{I}_1 \sqsubseteq \mathcal{I}_2$ then $T_{\mathcal{P}}(\mathcal{I}_1) \sqsubseteq T_{\mathcal{P}}(\mathcal{I}_2)$.* $\triangleleft$

Corollary 2. *$T_{\mathcal{P}}$ has the least fixpoint, denoted $\mathrm{LFP}(T_{\mathcal{P}})$, which can be computed by iterating $T_{\mathcal{P}}$ starting from the empty interpretation.* $\triangleleft$

It can be shown that the least fixpoint of $T_{\mathcal{P}}$ is the least model of the program, wr.t. $\leq_k$.

Example 2. Consider the rules of Example 1 and a database with five objects: o_1 and o_2 are oval, o_2 is also considered to be round, o_3 is square, o_4 is rectangular, and o_5 has unknown shape. Note that object o_2 is associated with different shapes, perhaps, because different robots perceive it differently. The successive iterations of $T_{\mathcal{P}}$ are given below. Note that $\mathcal{I}_2 = T_{\mathcal{P}}(\mathcal{I}_2)$.

$$\begin{aligned} &\mathcal{I}_1 = \emptyset \\ &\mathcal{I}_2 = T_{\mathcal{P}}(\mathcal{I}_1) = \{sim(o_1,o_1), sim(o_2,o_2), sim(o_3,o_3), sim(o_4,o_4), sim(o_5,o_5), \\ &\qquad\qquad sim(o_1,o_2), sim(o_2,o_1), \neg sim(o_3,o_2), \\ &\qquad\qquad \neg sim(o_3,o_4), sim(o_3,o_4), sim(o_4,o_3)\}. \end{aligned}$$

According to the definition of four-valued interpretation, we have that both $sim(o_1,o_2)$ and $sim(o_2,o_1)$, as well as for any $sim(x,x)$, receive the value **t**; $sim(o_3,o_2)$ receives the value **f** but $sim(o_2,o_3)$ is **u**; $sim(o_3,o_4)$ receives the value **i** but $sim(o_4,o_3)$ is **t**; for all the remaining pairs (x,y), the value of $sim(x,y)$ is **u**. $\triangleleft$

3.4 Using Approximations

A Hierarchy of Uncertainty. In our framework, lower and upper approximations are also four-valued sets. Figure 1 shows the truth ordering of pairs $(o \mathrel{\epsilon} A^{+}_{\sigma}, o \mathrel{\epsilon} A^{\oplus}_{\sigma})$. Note that $(t_1,t_2) \leq_t (t_3,t_4)$ iff $(t_1 \leq_t t_3)$ and $(t_2 \leq_t t_4)$. In the figure this is indicated by an edge from pair (t_1,t_2) to (t_3,t_4). Moreover, not all pairs of logical values are allowed because $(o \mathrel{\epsilon} A^{+}_{\sigma}) \leq_t (o \mathrel{\epsilon} A^{\oplus}_{\sigma})$ [9], for any object $o \in U$.

The pair $(\mathbf{t},\mathbf{t})$ corresponds to the case where an object o certainly belongs to a given set A, while $(\mathbf{f},\mathbf{f})$ indicates that o certainly does not belong to A. The remaining pairs of logical values in Figure 1 correspond to boundary cases where the object may belong to A. In the standard rough set framework [8], boundary cases correspond to the pair $(\mathbf{f},\mathbf{t})$ since approximations are two-valued sets. In contrast to the standard rough set framework, our framework allows different types of boundary cases to be identified and different degrees of uncertainty. For instance, the pair $(\mathbf{i},\mathbf{t})$ indicates that we can be more certain that object o has a property A than the pair $(\mathbf{f},\mathbf{i})$, although both pairs indicate that there is a possibility of object o having property A. Note that $(\mathbf{f},\mathbf{i}) <_t (\mathbf{i},\mathbf{t})$. However, as Figure 1 shows, not all pairs are comparable, e.g., pairs $(\mathbf{f},\mathbf{t})$ and $(\mathbf{i},\mathbf{i})$. But,

$$\begin{tikzcd}
 & & (\mathbf{t},\mathbf{t}) & \\
 & & (\mathbf{i},\mathbf{t}) \arrow[u] & \\
 & (\mathbf{u},\mathbf{t}) \arrow[ur] & & (\mathbf{i},\mathbf{i}) \arrow[ul] \\
(\mathbf{f},\mathbf{t}) \arrow[ur] & & (\mathbf{u},\mathbf{i}) \arrow[ul] \arrow[ur] & \\
 & (\mathbf{f},\mathbf{i}) \arrow[ul] \arrow[ur] & & (\mathbf{u},\mathbf{u}) \arrow[ul] \\
 & & (\mathbf{f},\mathbf{u}) \arrow[ul] \arrow[ur] & \\
 & & (\mathbf{f},\mathbf{f}) \arrow[u] &
\end{tikzcd}$$

Fig. 1. Truth ordering for $(o \in A^{+}_{\sigma}, o \in A^{\oplus}_{\sigma})$

these pairs point then to different sources (types) of uncertainty. For example, the pair $(\mathbf{f}, \mathbf{t})$ indicates that there is at least one object similar to o that does not have property A, i.e. $(o \in A^{+}_{\sigma}) = \mathbf{f}$ (see the case (1) of Lemma 1 in Section 4), but there is another object similar to o that has property A, i.e. $(o \in A^{\oplus}_{\sigma}) = \mathbf{t}$ (see the case (5) of Lemma 1 in Section 4). Therefore, in the neighborhood of o there are objects that have property A and others that do not have property A. The pair $(\mathbf{i}, \mathbf{i})$ points to a different source of uncertainty, e.g., for all objects in the neighborhood of o there is contradictory evidence about their membership in A (see cases (3) and (6) of Lemma 1 in Section 4).

The informal ideas presented above are reflected in our rule language. Thus, the language allows the user to choose the level of uncertainty or the type of uncertainty that best suits his application.

Extending the Language with Approximations. The rule language makes it possible to define four-valued relations. A defined relation can then be used to specify approximations of another four-valued relation, as discussed in [9] and in Section 2. Such an approximation is itself a four-valued relation. The rule language can thus be extended by allowing approximations of a rough relation (set) to appear in rule bodies. To this end, we need to extend the language with a notation for such symbols. In this paper, the lower approximation (upper approximation) of a relation A w.r.t. a similarity relation σ is denoted A^{+}_{σ} $(A^{\oplus}_{\sigma})$. Such approximation symbols can only be used in a program including rules defining A and σ. Programs must also not use recursion through approximations. Intuitively, the relations are not to be defined by referring to their own approximations. Such programs are considered *well-formed.*

Example 3. Consider the rules of Example 1 and the database of objects in Example 2. Based on the accessible knowledge, the robots may be given the task to remove from a given place all round and square objects. Let us introduce an additional unary relation rsq (standing for "round or square") defined as follows.

$$\begin{array}{ll}
\neg shape(x,y) & :\!-\ shape(x,z), y \neq z. \\
rsq(x) & :\!-\ shape(x, round)\ \textbf{;}\ shape(x, square). \\
\neg rsq(x) & :\!-\ \neg shape(x, round), \neg shape(x, square).
\end{array}$$

Observe that $rsq(o_2)$ is **i**, in the declarative semantics of the program, because o_2 is associated with both shapes round and oval.

The required rule expressing the task to be done may be expressed in various non-equivalent ways, according to the intended meaning.

1. $remove(x)$:– $rsq(x)$. – a traditional formulation where neighborhoods are not taken into account.
2. $remove(x)$:– $(rsq(x)^{+}_{sim} = true)$. – x is to be removed only when it surely is round or square.
3. $remove(x)$:– $(rsq(x)^{\oplus}_{sim} = true), rsq(x)^{+}_{sim}$. – x is to be removed if $(x \mathrel{\epsilon} rsq(x)^{+}_{sim}, x \mathrel{\epsilon} rsq(x)^{\oplus}_{sim}) \geq_t (\mathbf{i}, \mathbf{t})$.
4. $remove(x)$:– $rsq(x)^{\oplus}_{sim}$. – x is to be removed if there is a possibility that it might be round or square.

From the first rule above, we conclude that the truth value of $remove(o_2)$ is **i**, $remove$ (o_3) is **t**, and **u** for all other objects.

The reader can verify[2] that the membership in $rsq(x)^{+}_{sim}$ is **f** for o_1, o_2 and o_4 and it is **u** for o_3 and o_5. The membership in $rsq(x)^{\oplus}_{sim}$ is **i** for o_1 and o_2, **t** for o_3 and o_4, but **u** for o_5. Note that $(o_2 \mathrel{\epsilon} rsq(x)^{+}_{sim}) = \mathbf{f}$ because $(o_1 \mathrel{\epsilon} sim(o_2)) = \mathbf{t}$ but $rsq(o_1) = \mathbf{f}$, i.e. there is an object similar to o_2 that is neither round nor square, although there is also information indicating that o_2 is round. Consequently, it is not possible to conclude with certainty that o_2 is round (or square).

Using the second rule instead, the truth value of $remove(x)$, is **u** for all objects since for no object in the database it can be proved that it is surely round or square. Note that there is no object o in the database such that $(o \mathrel{\epsilon} rsq^{+}_{sim}) = \mathbf{t}$.

The third rule imposes that there must be a quite high believe that an object is round or square in order to remove it, although some uncertainty is acceptable. The rule forces that $(x \mathrel{\epsilon} rsq^{\oplus}_{sim}) = \mathbf{t}$. Thus, there must be an object similar to x that is round or square and $(x \mathrel{\epsilon} rsq^{+}_{sim}) \geq_t \mathbf{i}$. Remember that no conclusions are drawn from rules with bodies evaluated to false or unknown. If $(x \mathrel{\epsilon} rsq^{+}_{sim}) <_t \mathbf{i}$ then the rule does not fire, since the whole body becomes evaluated to **f** or **u**. Consequently with this rule, the truth value of $remove(x)$ is **u**, for all objects x in the database.

With the fourth rule, $remove(o_1)$ and $remove(o_2)$ are **i**, $remove(o_5)$ is **u**, and **t** for all remaining objects. In particular, $remove(o_1)$ is **i** because $(o_1 \mathrel{\epsilon} rsq(x)^{\oplus}_{sim}) = \mathbf{i}$. In contrast with the first rule, if this rule is used then o_4 is removed. $\lhd$

4 Implementation

For a program not using approximations the least model can be computed by iterating the $T_{\mathcal{P}}$ operator, as illustrated in Example 2.

The following lemma, which follows from the definition of approximations, shows how to compute the truth value of an approximation literal under a given interpretation, by consecutive check of simple conditions.

Lemma 1. Let A be a four-valued set on a universe U, σ be a four-valued similarity relation, and $x \in U$.

[2] Detailed calculation for the lower and upper approximations are not shown for space reasons.

1. $x \in A^{+}_{\sigma} = \mathbf{f}$ iff $(y \in \sigma(x) = \mathbf{t}$ and $y \in A = \mathbf{f})$, for some $y \in U$.
2. $x \in A^{+}_{\sigma} = \mathbf{u}$ iff (1) does not hold and
   (a) $(y \in \sigma(x) = \mathbf{u}$ and $y \in A \leq_t \mathbf{u})$, for some $y \in U$, or
   (b) $(y \in \sigma(x) = \mathbf{t}$ and $y \in A = \mathbf{u})$, for some $y \in U$.
3. $x \in A^{+}_{\sigma} = \mathbf{i}$ iff 1. and 2. does not hold and
   (a) $(y \in \sigma(x) = \mathbf{i}$ and $y \in A \leq_t \mathbf{i})$, for some $y \in U$, or
   (b) $(y \in \sigma(x) = \mathbf{t}$ and $y \in A = \mathbf{i})$, for some $y \in U$.
4. $x \in A^{+}_{\sigma} = \mathbf{t}$ iff $y \in A = \mathbf{t}$, for all $y \in U$ such that $\sigma(x, y) \geq_t \mathbf{u}$.

Moreover,

5. $x \in A^{\oplus}_{\sigma} = \mathbf{t}$ iff $(y \in \sigma(x) = \mathbf{t}$ and $y \in A = \mathbf{t})$, for some $y \in U$.
6. $x \in A^{\oplus}_{\sigma} = \mathbf{i}$ iff (1) does not hold and
   (a) $(y \in \sigma(x) = \mathbf{i}$ and $y \in A \geq_t \mathbf{i})$, for some $y \in U$, or
   (b) $(y \in \sigma(x) = \mathbf{t}$ and $y \in A = \mathbf{i})$, for some $y \in U$.
7. $x \in A^{\oplus}_{\sigma} = \mathbf{u}$ iff 1. and 2. does not hold and
   (a) $(y \in \sigma(x) = \mathbf{u}$ and $y \in A \geq_t \mathbf{u})$, for some $y \in U$, or
   (b) $(y \in \sigma(x) \geq_t \mathbf{i}$ and $y \in A = \mathbf{u})$, for some $y \in U$.
8. $x \in A^{\oplus}_{\sigma} = \mathbf{f}$ iff $y \in A = \mathbf{f}$ or $y \in \sigma(x) = \mathbf{f}$, for all $y \in U$. $\lhd$

Observe that any well-formed program has a least model which can be computed by a combined use of the $T_{\mathcal{P}}$ operator and Lemma 1.

5 Conclusions

In the current paper, we proposed a rule language for defining four-valued rough sets. The language allows us to work with four-valued approximations which appear frequently in practice. The techniques discussed in the paper open the space for implementation as well as for a pragmatic use of the language. The language is suitable for applications where some information might be unknown or inconsistent, in particular for databases or expert systems.

References

1. Amo, S., Pais, M.S.: A paraconsistent logic approach for querying inconsistent databases. International Journal of Approximate Reasoning 46, 366–386 (2007)
2. Belnap, N.: A useful four-valued logic. In: Epstein, G., Dunn, J.M. (eds.) Modern Uses of Many Valued Logic, pp. 8–37. Reidel (1977)
3. Damásio, C.V., Pereira, L.M.: A survay of paraconsistent semantics for logic programs. In: Gabbay, D.M., Smets, P. (eds.) Handbook of Defeasible Reasoning and Uncertainty Management Systems, vol. 2, pp. 241–320. Kluwer Academic Publishers, Dordrecht (1998)
4. Fitting, M.C.: Bilattices in logic programming. In: Epstein, G. (ed.) Proc. of the 20th International Symposium on Multiple-Valued Logic, pp. 238–246. IEEE CS Press, Los Alamitos (1990)
5. Fitting, M.C.: Fixpoint semantics for logic programming a survey. Theoretical Computer Science 278(1-2), 25–51 (2002)

6. Ginsberg, M.: Multivalued logics: a uniform approach to reasoning in AI. Computational Intelligence 4, 256–316 (1988)
7. Maluszyński, J., Szałas, A., Vitória, A.: A four-valued logic for rough set-like approximate reasoning. In: Duntsch, I., Grzymala-Busse, J., Orlowska, E., Polkowski, L. (eds.) Transactions on Rough Sets VI. LNCS, vol. 4374, pp. 176–190. Springer, Heidelberg (2007)
8. Pawlak, Z.: Rough Sets. Theoretical Aspects of Reasoning about Data. Kluwer Academic Publishers, Dordrecht (1991)
9. Vitória, A., Szałas, A., Maluszyński, J.: A four-valued extension of rough sets. In: Wang, G., Li, T., Grzymala-Busse, J., Miao, D., Skowron, A., Yao, Y. (eds.) RSKT 2008. LNCS (LNAI), vol. 5009, pp. 106–114. Springer, Heidelberg (2008)