International Journal of Approximate Reasoning 130 (2021) 126–149

Contents lists available at ScienceDirect

International Journal of Approximate Reasoning

www.elsevier.com/locate/ijar

Rough set reasoning using answer set programs

Patrick Doherty [a,b,*,1], Andrzej Szalas [c,b,2]

[a] School of Intelligent Systems and Engineering, Jinan University (Zhuhai Campus), Zhuhai, China
[b] Department of Computer and Information Science, Linköping University, SE-581 83 Linköping, Sweden
[c] Institute of Informatics, University of Warsaw, Banacha 2, 02-097 Warsaw, Poland

ARTICLE INFO

Article history:
Received 21 April 2020
Received in revised form 10 October 2020
Accepted 2 December 2020
Available online 10 December 2020

Keywords:
Rough sets
Approximate reasoning
Answer set programming
Knowledge representation

ABSTRACT

Reasoning about uncertainty is one of the main cornerstones of Knowledge Representation. Formal representations of uncertainty are numerous and highly varied due to different types of uncertainty intended to be modeled such as vagueness, imprecision and incompleteness. There is a rich body of theoretical results that has been generated for many of these approaches. It is often the case though, that pragmatic tools for reasoning with uncertainty lag behind this rich body of theoretical results. Rough set theory is one such approach for modeling incompleteness and imprecision based on indiscernibility and its generalizations. In this paper, we provide a pragmatic tool for constructively reasoning with generalized rough set approximations that is based on the use of Answer Set Programming (ASP). We provide an interpretation of answer sets as (generalized) approximations of crisp sets (when possible) and show how to use ASP solvers as a tool for reasoning about (generalized) rough set approximations situated in realistic knowledge bases. The paper includes generic ASP templates for doing this and also provides a case study showing how these techniques can be used to generate reducts for incomplete information systems. Complete, ready to run `clingo` ASP code is provided in the Appendix, for all programs considered. These can be executed for validation purposes in the `clingo` ASP solver.

1. Introduction and motivations

1.1. Rough sets and answer sets

Formal representations of uncertainty are numerous and highly varied due to different types of uncertainty intended to be modeled such as vagueness, imprecision and incompleteness. Rough set theory [8–10,24,30,31,33,39,41,42] is one such formal representation and it has been used to model incompleteness and imprecision using indiscernibility relations and approximations based on such relations. Rough sets and their generalizations often use *base* indiscernibility relations weaker than equivalence relations [10,13,14,38,39]. Consequently, there is large variation in characterization of rough sets themselves. Rough sets are additionally characterized by elements that are in such a set, elements that are not in such a

* Corresponding author at: Department of Computer and Information Science, Linköping University, SE-581 83 Linköping, Sweden.
E-mail addresses: patrick.doherty@liu.se (P. Doherty), andrzej.szalas@liu.se, andrzej.szalas@mimuw.edu.pl (A. Szalas).
[1] This work has been supported by the ELLIIT Network Organization for Information and Communication Technology, Sweden; the Swedish Foundation for Strategic Research SSF (Smart Systems Project RIT15-0097); and a distinguished guest professor grant from Jinan University (Zhuhai Campus).
[2] This work has been supported by grant 2017/27/B/ST6/02018 of the National Science Centre Poland.

https://doi.org/10.1016/j.ijar.2020.12.010
0888-613X/

set, and by a boundary region containing elements that may or may not be in such a set. This is determined by the type of indiscernibility relation associated with the particular set in question.

Answer Set Programming (ASP) [2,4,5,12,16,18,19,21,25,27,36] is a knowledge representation framework based on the logic programming and nonmonotonic reasoning paradigms that uses an answer set/stable model semantics for logic programs. Each answer set program contains a set or rules and a set of facts (rules without a body). A great deal of attention has been devoted to ASP implementations [17,21,25–27,36].[3]

Though Answer Set Programming mainly serves in the paper as a tool for computing approximations and related rough concepts, it may at the same time be used as a basis for different types of rough nonmonotonic reasoning. Here, one takes advantage of the ability to model rough sets in ASP together with powerful nonmonotonic features of ASP. In fact, answer sets, in some respects, are more general than rough sets and a suitable bridge between them is required to reason with both in an integrated manner. Such a bridge will be provided in the paper.

Heuristic rules, e.g., formalizing default reasoning, have demonstrated their strength in knowledge representation [3, 10,35]. In the context of ASP, such rules are represented by using default negation in the body of an ASP rule. Indeed, missing knowledge can many times be completed by default conclusions reflecting commonsense reasoning patterns such as the use of closed world assumption (CWA) or in the more general case local closed world assumption (LCWA), as will be demonstrated in the paper.

In the context of rough sets, the boundary regions of approximated rough concepts and relations characterize *missing* information associated with a rough relation. Recall that an element in the boundary region of a rough concept or relation may or may not be in that concept or relation, but could be. By combining rough relations as components in ASP rules, and using the heuristic techniques associated with ASP such as CWA or LCWA, the status of elements in the boundary region can be changed by default using commonsense or expert knowledge associated with an application at hand. This in turn can result in a substantial improvement of the informational quality of rough knowledge bases used in such applications. Such examples will be provided in the paper.

1.2. Contributions

In this paper we:

- provide an interpretation of answer sets as (generalized) approximations of crisp sets (when possible). This is done by leveraging the close relation between 3-valued logics used as a basis for ASP semantics and rough set semantics;
- provide an interpretation of rough (approximate) sets as answer sets;
- show how to use ASP solvers as a tool for reasoning about (generalized) rough set approximations situated in realistic knowledge bases. Here we show how rough set concepts and relations can be used to constructively extend classical answer set programs with such concepts and relations.

In particular, we address the following sub-problems where we assume that an underlying knowledge base, defined by means of answer set programs, is to be satisfied:

1. given a crisp set c and a base relation σ, compute the lower and upper approximation of c wrt σ;
2. given a base relation σ and a pair of sets $l \subseteq u$, compute a crisp set c whose lower approximation and upper approximation wrt σ are l and u, respectively;
3. given a crisp set c and a pair of sets $l \subseteq u$, compute the underlying base relation such that c's lower approximation and upper approximation wrt σ are l and u, respectively;
4. given a pair of sets $l \subseteq u$, compute the underlying base relation σ and a crisp set c, whose lower approximation and upper approximation wrt σ are l and u, respectively.

Since many sets/relations may satisfy the above requirements, we will provide constructive methods for enumerating all of them using ASP tools.

1.3. Paper structure

The paper is structured as follows. In Section 2 we present a landscape of rough set-inspired approximate reasoning techniques and associated definitions of approximations. In Section 3 we present a three-valued logic with default negation that is used as a bridge between rough sets and ASP. Section 4 reviews ASP constructs used in the paper to model rough concepts and relations. In Section 5 we interpret ASP programs in the approximate reasoning framework and show their constructive use as a tool for approximate reasoning. We also provide related complexity results. Section 6 is devoted to a case study illustrating the use of the developed techniques. Here we show how reducts for incomplete information systems

[3] To verify our examples we have used `clingo` (see https://potassco.org/). They can also be tested using an online interface https://potassco.org/clingo/run/.

Table 1
Properties of base relations in terms of approximations and first-order correspondences.

Notation	Property of c^+_σ, $c^\oplus_\sigma$	First-order correspondence	Property of σ
D	$c^+_\sigma \to c^\oplus_\sigma$	$\forall x \exists y\big(\sigma(x,y)\big)$	Seriality
T	$c^+_\sigma \to c$	$\forall x\big(\sigma(x,x)\big)$	Reflexivity
B	$c \to (c^\oplus_\sigma)^+_\sigma$	$\forall x \forall y\big(\sigma(x,y) \to \sigma(y,x)\big)$	Symmetry
4	$c^+_\sigma \to (c^+_\sigma)^+_\sigma$	$\forall x \forall y \forall z\big((\sigma(x,y) \wedge \sigma(y,z)) \to \sigma(x,z)\big)$	Transitivity
5	$c^\oplus_\sigma \to (c^\oplus_\sigma)^+_\sigma$	$\forall x \forall y \forall z\big((\sigma(x,y) \wedge \sigma(x,z)) \to \sigma(y,z)\big)$	Euclidicity

can be generated using Asp. Finally, Section 7 concludes by summarizing the results in the paper and considering future work. An appendix contains complete Asp code for running all examples described in the paper in an `clingo` Asp solver for purposes of validation.

2. Rough set reasoning landscape

2.1. Approximations and approximate/rough sets

In the paper we shall deal with rough approximations based on indiscernibility relations (being reflexive, symmetric and transitive), as well as their generalizations, where the requirements as to the underlying relation are relaxed. For example, rather than using indiscernibility as the base relation, one may require similarity (proximity, tolerance). In this case, transitivity appears too strong and is therefore rejected as a requirement.

In the rest of the paper we shall assume that:

- 'dom' is a fixed finite domain;
- $\sigma \subseteq \text{dom} \times \text{dom}$, possibly with an index, is a base relation intended to represent indiscernibility and its generalizations, proximity, similarity, tolerance, etc., among objects. It is assumed that $\sigma(x,y)$ is true if and only if $\langle x, y\rangle \in \sigma$ holds.

In the following definition we set no requirements on the base relation used to define approximations. The requirements used in subsequent parts of the paper are listed in Table 1.

Definition 2.1 *(Approximations, base relations, boundary regions, approximate sets).* Let $c \subseteq \text{dom}$ and σ be a binary relation on 'dom'. Then the *lower approximation* c^+_σ and the *upper approximation* $c^\oplus_\sigma$ *of* c *wrt* σ are:

$$c^+_\sigma \stackrel{\text{def}}{=} \{x \mid \forall y\big(\sigma(x,y) \to y \in c\big)\}; \tag{1}$$

$$c^\oplus_\sigma \stackrel{\text{def}}{=} \{x \mid \exists y\big(\sigma(x,y) \wedge y \in c\big)\}. \tag{2}$$

The difference $c^\oplus_\sigma \setminus c^+_\sigma$ is called the *boundary region* of c wrt σ. The relation σ is called the *base relation* for approximations c^+_σ, $c^\oplus_\sigma$. The pair $\langle c^+_\sigma, c^\oplus_\sigma\rangle$ is called an *approximate set*. □

Definition 2.2 *(Rough approximations, rough sets).* If the base relation σ of Definition 2.1 is an equivalence relation (reflexive, symmetric and transitive) then approximations defined by (1)–(2) are called *rough approximations*. A *rough set* is an approximate set with the base relation being an equivalence relation. □

Remark 2.3. Note that arbitrary (n-argument with $n > 1$) approximate relations can be modelled by assuming that the domain 'dom' consists of n-tuples of elements of "more elementary" domains. In the rest of the paper we will deal with a language with arbitrary relations. Of course, in view of this extended interpretation of relations, Definition 2.1 applies to this case as well. □

2.2. Correspondences between approximations and properties of base relations

Properties listed in Table 1 have been intensively investigated in the area of modal logics [6,20] and correspondence theory between modalities and Kripke accessibility relations [40]. They also relate properties of approximations to properties of the underlying relation σ (see, e.g., [13,41]). In particular:

- **D** ensures that the lower approximation of a set is included in its upper approximation;
- **T** ensures that the lower approximation is included in the approximated set;
- **B** ensures that the approximated set is included in the lower approximation of the upper approximation of the set;
- **4** ensures that the lower approximation of a set is included in the lower approximation of its lower approximation (so that iterating lower approximations do not change the result of its first application);

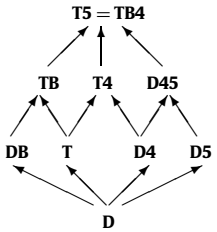

Fig. 1. Relationships among properties of base relations. An arrow $P \rightarrow Q$ indicates that the requirements P are weaker than the requirements Q.

- **5** ensures that the upper approximation of a set is included in the lower approximation of its upper approximation.

Remark 2.4. Note that the requirement **D** that for every set its lower approximation is included in its upper approximation is equivalent to the seriality of the base relation σ. Since the inclusion of lower approximation in the upper approximation is fundamental in approximate reasoning, we shall further assume (at least) the seriality of σ. □

Dependencies among properties of binary relations used as a basis for approximations, shown in Fig. 1, are well known (see, e.g., [6,40,41]). Note that the bottom of the figure, **D**, defines serial relations while the top, **T5**, defines equivalence relations. Note that **T5** = **TB4** (in modal correspondence theory they both correspond to **S5** [6]).

A user will have the ability to represent selections of these properties in ASP programs when formalizing different approximation relations. This is considered in Section 5.

3. Logical basis for rough set-based reasoning and ASP

In this section, we propose a three-valued logic intended to serve as the syntactic and semantic basis for an integrative approach to reasoning with rough sets and answer sets. The logic is an extension of Łukasiewicz logic $Ł_3$ [28] and Kleene logic K_3 [22], obtained by adding a new connective 'not' that will also be used in defining default negation in Answer Set Programs.[4] We will use $ŁK_3$ to refer to this logic throughout the paper. The language of $ŁK_3$ includes:

- truth constants T (true), U (unknown) and F (false) ordered by: F < U < T;
- individual constants $\mathcal{C}$ and individual variables $\mathcal{V}$;
- relation symbols $\mathcal{R}$;
- connectives: '-' (*strong negation*), 'not' (*default negation*), ',' (conjunction), ';' (disjunction).[5]

A special implicative operator for ASP rules will be defined in Section 4.

Remark 3.1. By convention, in ASP programs, constant identifiers are strings starting with lower case letters and variable identifiers start with upper case letters. For connectives we use standard ASP syntax. □

Definition 3.2 *(Literals, ground literals, default literals, formulas).* By a *positive literal* (or an *atom*) we mean any expression of the form $r(\bar{\tau})$, where $r \in \mathcal{R}$ and $\bar{\tau}$ consists of constants and/or variables. A *negative literal* is an expression of the form $\text{-}\ell$, where ℓ is a positive literal.[6] A *literal* is a positive or a negative literal. A *ground literal* is a literal without variables. By a *default literal* we understand a literal or an expression of the form not ℓ, where ℓ is a literal.

A *formula* is a literal, default literal or an expression of the form '$\text{-}A$' (strong negation), 'not A' (default negation), 'A, B' (conjunction) or '$A; B$' (disjunction), where A, B are formulas. □

Definition 3.3 *(Consistency, interpretations).* A set of literals is *consistent* if it does not contain a literal ℓ together with its negation $\text{-}\ell$. By an *interpretation* we mean any finite consistent set of ground literals. The set of constants occurring in I is denoted by $\mathcal{C}_I$. □

Let v be an assignment of constants to variables, $v : \mathcal{V} \longrightarrow \mathcal{C}$. In the rest of the paper we will use an extension of v to arbitrary (tuples of) expressions, $v(e) \stackrel{\text{def}}{=} e'$, where e' is an expression (a tuple of expressions) obtained from e by substituting each variable X occurring in e by the constant $v(X)$.

Interpretations assign truth values to formulas as shown in the following definition.

[4] Note that logics $Ł_3$ and K_3 are the same on the standard connectives we use.

[5] We use the ASP syntax for connectives. For the sake of readability, in formulas we sometimes use the notation $\wedge, \vee$.

[6] We always remove double strong negations using $\text{-}(\text{-}\ell) \stackrel{\text{def}}{=} \ell$.

Definition 3.4 *(Truth values of formulas, satisfiability).* Let I be an interpretation and $v : \mathcal{V} \longrightarrow \mathcal{C}_I$ be an assignment of constants to variables. A *truth value of a formula* A wrt I and v, denoted by $I^v(A)$, is inductively defined by:

- if $A = \ell$ is a literal then $I^v(A) \stackrel{\text{def}}{=} \begin{cases} \mathrm{T} & \text{when } v(\ell) \in I; \\ \mathrm{F} & \text{when } \text{-}v(\ell) \in I; \\ \mathrm{U} & \text{when } v(\ell) \notin I \text{ and } \text{-}v(\ell) \notin I; \end{cases}$
- $I^v(\text{-}A) \stackrel{\text{def}}{=} \begin{cases} \mathrm{T} & \text{when } I^v(A) = \mathrm{F}; \\ \mathrm{F} & \text{when } I^v(A) = \mathrm{T}; \\ \mathrm{U} & \text{when } I^v(A) = \mathrm{U}; \end{cases} \qquad I^v(\text{not } A) \stackrel{\text{def}}{=} \begin{cases} \mathrm{T} & \text{when } I^v(A) \in \{\mathrm{F}, \mathrm{U}\}; \\ \mathrm{F} & \text{when } I^v(A) = \mathrm{T}; \end{cases}$
- $I^v(A, B) \stackrel{\text{def}}{=} \min\{I^v(A), I^v(B)\}$, $I^v(A; B) \stackrel{\text{def}}{=} \max\{I^v(A), I^v(B)\}$ where min and max are the minimum and maximum wrt ordering $\mathrm{F} < \mathrm{U} < \mathrm{T}$.

We say that I, v *satisfy* formula A, denoted by $I, v \models A$, iff $I^v(A) = \mathrm{T}$. When v is irrelevant, we write $I(A)$ and $I \models A$ rather than $I^v(A)$ and $I, v \models A$. □

We will sometimes use quantifiers $\exists, \forall$. Since the considered domains are finite, quantifiers abbreviate disjunctions and conjunctions, where 'dom' is the domain of variable X:

$$\exists X\big((A(X)\big) \stackrel{\text{def}}{=} \bigvee_{a \in \text{dom}} A(a); \quad \forall X\big(A(X)\big) \stackrel{\text{def}}{=} \bigwedge_{a \in \text{dom}} A(a). \tag{3}$$

A variable occurrence is *bound* if it is in the scope of a quantifier. Otherwise it is *free*.

Let $l \subseteq u \subseteq \text{dom}$ and σ be a base relation. In the rest of the paper we will make use of the following interpretation of approximations:

the pair $\langle l, u\rangle$ represents all sets c such that $l = c^+_\sigma$ and $u = c^{\oplus}_\sigma$.

Of course, such a set c may not exist. Typically there may be more than one, even an exponential number of them (wrt the cardinality of the domain).

Remark 3.5. Given a base relation σ, an interpretation I and an assignment v of constants to variables, each $\text{Ł}K_3$ formula $A(\bar{x})$, with all free variables being in $\bar{x}$, defines the following pair of sets:

- $l \stackrel{\text{def}}{=} \big\{v(\bar{x}) \mid I^v(A(\bar{x})) = \mathrm{T}\big\}$ intended to be the lower approximation of a set;
- $u \stackrel{\text{def}}{=} \big\{v(\bar{x}) \mid I^v(A(\bar{x})) \in \{\mathrm{T}, \mathrm{U}\}\big\}$ intended to be the upper approximation of a set.

The pair $\langle l, u\rangle$ is an approximate/rough set provided that there is a (classical) set of n-tuples c such that $l = c^+_\sigma$ and $u = c^{\oplus}_\sigma$. □

4. Answer set programming

Answer Set Programming is a prominent rule-based logical tool used in the Knowledge Representation and Reasoning area. In this paper we will use normal Answer Set programs with the choice operator [29]. The choice operator chooses an arbitrary (possibly empty) subset of specified literals, satisfying a cardinality constraint, and adds it to the constructed interpretation. Below, for the sake of simplicity, we introduce a subset of Asp constructs needed for our purposes. Of course, one may use all other Asp constructs, too.

We first provide formal definitions for the syntax and semantics of Asp based on the use of definitions, syntax and semantics from $\text{Ł}K_3$ described in Section 3. We then describe some important concepts that are part of the Asp framework in addition to providing some illustrative examples of Asps.

4.1. Syntax of answer set programs

Let us start with the syntax of Answer Set Programs.

Definition 4.1 *(Rules, facts, constraints, heads, bodies).* By a *normal rule* (*rule*, for short) we understand an expression of the form:

$$H \text{ :- } \ell_1, \ldots, \ell_m, \text{not } \ell_{m+1}, \ldots, \text{not } \ell_n, \tag{4}$$

where H is a literal or the empty symbol, and for $1 \leq m \leq n$, $\ell_1, \ldots, \ell_n$ are (positive or negative) literals. The expression H is called the *head* and the expression at the righthand side of ' :- ' is called the *body* of the rule. A *fact* is a rule with the empty body. Facts are written without ' :- '. A *constraint* is a rule with the empty head. The empty body is equivalent to T and the empty head is equivalent to F. □

Definition 4.2 *(Choice rules).* By a *choice rule* we mean an expression of the form:

$$k\,\{\ell(\bar{r}) : \ell_1(\bar{s_1}), \ldots, \ell_l(\bar{s_l})\}\,m, \tag{5}$$

where:

- k, m are natural numbers such that $k \leq m$. When k is not provided, it is by default 0, when m is not provided, no upper limit on the number of chosen literals is placed;
- $\ell(\bar{r}), \ell_1(\bar{s_1}), \ldots, \ell_l(\bar{s_l})$ are literals, where $\bar{r}$ and $\bar{s_1}, \ldots, \bar{s_l}$ are tuples of constants and/or variables such that every variable in $\bar{r}$ occurs also among variables in $\bar{s_1}, \ldots, \bar{s_l}$. □

Each choice rule ϱ has two roles, where I is the computed interpretation:

(i) it acts as a constraint making sure that I contains at least k and at most m literals from the set specified in (5);
(ii) it allows an arbitrary number of literals from the set specified in (5) to be added to I provided that the constraint (i) remains satisfied.

Remark 4.3. In ASP a more general form of choice rules is considered, where rule bodies are allowed, too. For the sake of clarity we did not define these rules in full generality since we only need the simpler form provided by Definition 4.2. □

Definition 4.4 *(Programs, domains).* An *answer set program* (a *program*, for short) is a finite set of rules and/or choice rules. A program without choice rules is called *normal*. A *domain associated with a program* Π, denoted by $\mathcal{C}_\Pi$, is the set of constants occurring in Π. □

The following example illustrates the use of introduced ASP constructs. For the complete, executable `clingo` code, see Appendix A.8.

Example 4.5. Consider a real estate agency, REA, interested in making good matches between sellers and buyers. The REA's goal is to complete transactions trying to minimize time and effort spent both by clients and agents. They know that a potential buyer, Bob, is looking for a high quality house. He prefers houses close to the city center but may also consider cheaper but charming suburb residential areas within commuting distance of the city. A REA agent collects matching criteria which are then transformed to the following ASP rules (with self-explanatory relations):

$$may\_buy(bob, H) \text{ :- } house(H), high\_quality(H), located(H, center). \tag{6}$$

$$may\_buy(bob, H) \text{ :- } house(H), high\_quality(H), located(H, Loc), \tag{7}$$

$$residential(Loc), commuting\_dist(Loc), \tag{8}$$

$$\text{not } \text{-}charming(Loc). \tag{9}$$

$$\text{-}may\_buy(bob, H) \text{ :- } house(H), \text{ -}high\_quality(H). \tag{10}$$

$$\text{-}may\_buy(bob, H) \text{ :- } house(H), located(H, Loc), \text{ -}commuting\_dist(Loc). \tag{11}$$

Using these rules some houses may be classified as being houses that Bob may be willing to buy (using rules listed in (6)–(9)), some may be classified as being houses that Bob is not willing to buy (using the rules in (10)–(11)). The remaining houses would remain unclassified when neither premises in (6)–(9) nor in (10)–(11) evaluate to T.

Notice the role of the default negation 'not' in (9). At first glance it may seem that replacing (9) by '$charming(Loc)$' suffices. But that is too strong. Absence of truth for $\text{-}charming(Loc)$ is more appropriate in this case. In order to describe houses whose location is *charming* ($charming(Loc) = \mathrm{T}$) or whose location is unknown to be charming ($charming(Loc) = \mathrm{U}$), 'not' is used in the body of the second rule (7)–(9) instead. The expression '$not\ \text{-}charming(Loc)$' is T when '$\text{-}charming(Loc)$' is F or U, i.e., '$charming(Loc)$' is T or U.

To illustrate the use of choice rules, consider a situation when Bob wants to see at least 3 and no more than 5 houses. The following rule selects between 3 and 5 houses to visit:

$$3\,\{to\_visit(H) : may\_buy(bob, H)\}\,5. \tag{12}$$

If there are less than 3 such values of H then no answer set satisfies (12).

When some secondary conditions on Bob's criteria are known, e.g., "good school proximity", (12) can be refined to:

$$3\,\{to\_visit(H) : may\_buy(bob, H), located(H, Loc), good\_school\_in(Loc)\}\,5. \tag{13}$$

Indeed, rule (13) makes $to\_visit(H)$ true for at least 3 and at most 5 values of H such that the conjunction '$may\_buy(H)$, $located(H, Loc)$, $good\_school\_in(Loc)$' is T. □

4.2. Semantics of answer set programs

An Asp program can be viewed as a specification of answer sets, where answer sets are special interpretations satisfying an Asp program.

Definition 4.6 *(Rule Satisfiability).* An interpretation I *satisfies* a rule ϱ of the form (4), denoted by $I \models \varrho$, if for any assignment $v : \mathcal{V} \longrightarrow \mathcal{C}_I$, the conjunction $v(\ell_1), \ldots, v(\ell_m) \in I$ and $v(\ell_{m+1}), \ldots, v(\ell_n) \notin I$ implies $I^v(H) = \mathrm{T}$. In terms of the satisfaction relation $\models$ defined in Definition 3.4, $I, v \models \ell_1, \ldots, I, v \models \ell_m$ and $I, v \not\models \ell_{m+1}, \ldots, I, v \not\models \ell_n$ implies $I, v \models H$.

For ϱ being a choice rule of the form (5) and I being an interpretation, a *choice set* for ϱ, $L_I(\varrho)$, is defined by: $L_I(\varrho) \stackrel{\text{def}}{=} \{v(\ell(\bar{r})) \mid v : \mathcal{V} \longrightarrow \mathcal{C}_I \text{ and } I^v(\ell_1(\bar{s_1}), \ldots, \ell_l(s_l)) = \mathrm{T}\}$. We say that an interpretation I *satisfies* a choice rule ϱ, denoted by $I \models \varrho$, iff $k \leq |\, I \cap L_I(\varrho)\,| \leq m$. An interpretation I *satisfies* an Asp program Π, $I \models \Pi$, if for all $\varrho \in \Pi$,[7] $I \models \varrho$. □

The definition of answer sets consists of two parts [18,19]. The first part of the definition is for programs without default negation and choice rules. The second part explains how to remove default negation and choice rules so that the first part of the definition can be applied. To simplify definitions we assume that the considered Asp programs are grounded. That is, all variables are instantiated by constants representing domain elements.[8]

Definition 4.7 *(Answer Sets, Part I).* Let Π be a program not containing default negation nor choice rules (i.e., Π consists of rules of the form (4) without default literals 'not ℓ_i' for $i = m+1, \ldots, n$). An *answer set* of Π is an interpretation I such that $I \models \Pi$ and I is minimal (i.e., there is no $I' \subsetneq I$ such that $I' \models \Pi$). □

Answer sets can now be defined as follows.

Definition 4.8 *(Answer Sets, Part II).* Let Π be an Asp program, C all its choice rules, and I, J be interpretations such that $I \cup J \models C$ and $J \subseteq \bigcup_{\varrho \in C} L_I(\varrho)$, where $L_I(\varrho)$ is defined as in Definition 4.2.[9] By the *reduct of* Π *wrt* (I, J), denoted by $\Pi^{I,J}$, we mean the program obtained from Π by:

1. removing all choice rules and adding facts 'ℓ.' for $\ell \in J$;
2. removing all premises of the form 'not ℓ' such that $\ell \notin I \cup J$
3. removing all rules containing 'not ℓ' such that $\ell \in I \cup J$.

The interpretation $I \cup J$ is an *answer set* of Π if $I \cup J$ is an answer set of the reduct $\Pi^{I,J}$ in the sense of Definition 4.7. □

Remark 4.9. In Definition 4.8 we have defined *reducts* of Asp programs. In the context of rough set-based reasoning it is worth emphasizing that there is an unfortunate overlap of terminologies used: reducts of Asp programs *are not* the reducts of information systems as understood in the area of approximate reasoning. This latter type of reduct is considered in the case study in Section 6. □

Remark 4.10. Answer sets are interpretations consisting of positive and negative ground literals about relations, and can also be seen as sets of approximations of these relations. Therefore, if I is an answer set, r is a relation and $\bar{a}$ is a tuple of constants, then:

- $\{\bar{a} \mid I(r(\bar{a})) = \mathrm{T}\}$ represents the lower approximation of r;
- $\{\bar{a} \mid I(\bar{a}) \in \{\mathrm{T}, \mathrm{U}\}\}$ represents the upper approximation of c. □

The following example illustrates the use of Definitions 4.7 and 4.8. For `clingo` Asp code, see Appendix A.8.

[7] Including normal rules as well as choice rules.

[8] Some Asp implementations, including `clingo`, ground programs and use SAT solvers to compute answer sets.

[9] If such I, J do not exist then Π has no answer sets.

Example 4.11 *(Example 4.5 continued).* Assume the REA's database contains currently four high quality houses h_1, h_2, h_3, h_4. Houses h_1, h_2 are located in the city center, h_3, h_4 respectively in residential suburban areas a_3, a_4 with commuting distance to the city. It is unknown whether the area a_3 will appear charming for Bob. On the other hand, the area a_4 is uncharming. Both areas a_3, a_4 have good schools not far from the houses. That is, the REA's database contains (at least) the following facts:

$$\begin{array}{l} house(H), high\_quality(H) \text{ for } H \in \{h_1, h_2, h_3, h_4\}, \\ located(h_1, center), located(h_2, center), located(h_3, a_3), located(h_4, a_4), \\ residential(a_3), residential(a_4), commuting\_dist(a_3), commuting\_dist(a_4), \\ \text{-}charming(a_4), good\_school\_in(a3), good\_school\_in(a4), good\_school\_in(center). \end{array} \tag{14}$$

Consider an Answer Set program Π consisting of rules listed in (6)–(9), (13) and facts (14). According to our assumption, the rules are instantiated by constants. For example, rather than having the rule expressed by (6), one has four rules, where H is substituted by constants h_1, h_2, h_3, h_4:

$$may\_buy(bob, h_1) \text{ :- } house(h_1), high\_quality(h_1), located(h_1, center). \tag{15}$$

$$may\_buy(bob, h_2) \text{ :- } house(h_2), high\_quality(h_2), located(h_2, center). \tag{16}$$

$$may\_buy(bob, h_3) \text{ :- } house(h_3), high\_quality(h_3), located(h_3, center). \tag{17}$$

$$may\_buy(bob, h_4) \text{ :- } house(h_4), high\_quality(h_4), located(h_4, center). \tag{18}$$

Note that (13) is the only choice rule in Π and the default negation 'not' occurs only in (9). Let I be an interpretation containing (at least) facts listed in (14). Given that all rules and facts of Π are to be satisfied, the rules listed in (6)–(9) result in including in I, the conclusions '$may\_buy(bob, H)$' for $H \in \{h_1, h_2, h_3\}$. Notice that the premise (9) is false for h_4, so the rule does not support the conclusion $may\_buy(bob, h_4)$.

According to Definition 4.2, $L_I((13)) = \{to\_visit(h_1), to\_visit(h_2), to\_visit(h_3)\}$. One may choose J in Definition 4.8 to be any subset of $L_I((13))$. When one considers I and defines $J \stackrel{\text{def}}{=} L_I((13))$, the reduct $\Pi^{I,J}$ consists of facts (14), and:

- facts: '$to\_visit(h_1)$.', '$to\_visit(h_2)$.', '$to\_visit(h_3)$.', replacing the choice rule (13) using Point 1. of Definition 4.8;
- ground instances of rule (6), specified by (15)–(18);
- instances of rule (7)-(9) obtained by Point 2. of Definition 4.8[10]:

$$\begin{array}{l} may\_buy(bob, h_3) \text{ :- } house(h_3), high\_quality(h_3), located(h_3, a_3), \\ \qquad\qquad\qquad\qquad residential(a_3), commuting\_dist(a_3). \\ may\_buy(bob, h_4) \text{ :- } house(h_4), high\_quality(h_4), located(h_4, a_4), \\ \qquad\qquad\qquad\qquad residential(a_4), commuting\_dist(a_4). \end{array}$$

Observe that the instance:

$$\begin{array}{l} may\_buy(bob, h_4) \text{ :- } house(h_4), high\_quality(h_4), located(h_4, a_4), \\ \qquad\qquad\qquad\qquad residential(a_4), commuting\_dist(a_4), \\ \qquad\qquad\qquad\qquad \text{not } \text{-}charming(a_4). \end{array}$$

is removed from the reduct according to Point 3 of Definition 4.8, since 'not $\text{-}charming(a_4)$' occurs in the rule's premises and '$\text{-}charming(a_4)$' $\in I$.

According to Definition 4.8, $I \cup J$ is an answer set of Π, since it is an answer set for the reduct $\Pi^{I,J}$ in the sense of Definition 4.7. □

Due to the potentially large number of answer sets for a given program Π, generating all of them may be unfeasible or unwanted. In such cases one frequently uses the following standard methodology from ASP:

1. generate as many answer sets as possible given certain bounds on resources (time, memory, etc.); or,
2. select "the best" answer sets wrt some external criteria.

As examples, the external criteria may be related to:

[10] We only list relevant rules, contributing to the derived conclusions.

- minimizing costs involved or resource consumption resulting from using a specific answer set (e.g., as a basis for constructing a plan);
- minimizing the sizes of boundary regions of selected approximate relations.

This methodology can, e.g., be useful in the scenario considered in Example 4.5 where rules (12) and (13) can generate many answer sets. In such cases the real estate agent could rank answer sets using some heuristic criteria based on his/her experience.

4.3. Complexity issues and ASP

In this paper, we will deal with data complexity [1] of computing answer sets, where it is assumed that only the underlying database of facts can be changed and where queries are expressed by rules. Since in an answer set program, the database is given by a set of facts, we assume that the set of rules in a program is fixed while the set of facts may vary.

Definition 4.12 *(Data complexity).* Let the vocabulary (signature) of the language be fixed and Π be an answer set program. By the *data complexity of* ASP(Π), we mean the complexity of computing an answer set of Π' obtained from Π by changing at most the facts in Π, where the data size is $|C_{\Pi'}|$ (the cardinality of the set of constants occurring in Π'). □

The data complexity of ASP with choice rules is known to be NP-complete [29] and the number of answer sets may be exponential, as formalized in the following theorem.

Theorem 4.13 *([29]). Let Π be a program and m be the cardinality of the domain associated with Π.*

- *Computing an answer set for a program Π is NP-complete wrt the size of the domain $\mathcal{C}_\Pi$ associated with Π.*
- *The number of answer sets of Π may be exponential wrt the size of the domain $\mathcal{C}_\Pi$ associated with Π.* □

4.4. The CWA and OWA in ASP

In traditional databases, reasoning is often based on the assumption that information stored in a specific database contains a complete specification of the application environment at hand. If a tuple is not in a relational table, it is assumed not to have that specific property. In the case of deductive databases, if the tuple is not in a relational table or not among any conclusions generated implicitly by the application of intensional rules, it is again assumed not to have these properties. Under this assumption, an efficient means of representing negative information about the world depends on applying the *Closed-World Assumption* (CWA) [1,34]. In this case, atomic information about the world, absent in a world model (represented as a database), is assumed to be false.

On the other hand, for many applications such as autonomous systems applications and robotics, the assumption of complete information is not feasible nor realistic and the CWA cannot be used. In such cases an *Open-World Assumption* (OWA), where information not known by an agent is assumed to be unknown, is often accepted, as in the case of ASP. The CWA and the OWA represent two ontological extremes. Quite often, a reasoning agent does have or acquires additional information which permits the application of the CWA *locally* in a particular context. In addition, if it does have knowledge of what it does not know, this information is valuable because it can be used, for instance, in plan generation to acquire additional information through use of sensor actions during execution time. In such a context, various forms of *Local Closed World Assumptions* (LCWA) have been defined (see e.g. [11,15]). In ASP in general, and in some of the answer set programs that follow in the paper, we will frequently locally close the world using rules of the form:

$$\text{-}\ell(\bar{X}) \text{ :- } \mathrm{dom}(\bar{X}), \mathrm{not}\, \ell(\bar{X}), \tag{19}$$

where $\bar{X} = \langle X_1, \ldots, X_n \rangle$ and the expression $\mathrm{dom}(\bar{X})$ is an abbreviation for the conjunction of $\mathrm{dom}(X_1)$, …, $\mathrm{dom}(X_n)$. The $\mathrm{dom}(\bar{X})$ expression is used to fit the `clingo` syntax, where each variable occurring in a rule's head is required to occur in a positive literal in the rule's body.

The following example illustrates the use of LCWA. For the associated, executable `clingo` ASP code, see Appendix A.8.

Example 4.14 *(Example 4.5 continued).* To illustrate OWA and CWA, consider again the real estate agency scenario. Rules (6)–(9) allow one to infer positive facts about '*may_buy*' and rules (10)–(11) serve the purpose of inferring negative facts about '*may_buy*'. When OWA is accepted, as in ASP, the truth value of facts that are neither inferred positive nor negative remains U. On the other hand, rather than express specific rules for negative conclusions, one may want to use LCWA and, in addition to rules (6)–(9), use a single rule:

$$\text{-}may\_buy(bob, H) \text{ :- } house(H), \mathrm{not}\, may\_buy(bob, H). \tag{20}$$

Given a fixed value of H, the truth value of '$\mathrm{not}\, may\_buy(bob, H)$' is T when the truth value of '$may\_buy(bob, H)$' is F or U, i.e., when '$may\_buy(bob, H)$' is not inferred to be true. □

5. An ASP-based framework for approximate reasoning

5.1. General structure of ASP programs used for approximate reasoning

Notice that when reasoning about an application domain, one typically uses more than one concept/relation. If there are approximate relations involved, then each of them may call for the use of a specific base relation. For example, an indiscernibility relation among cars can hardly be the same as the one among houses. Therefore, for modelling convenience, we allow more than one base relation for a particular application domain.

The structure of answer set programs we will use for approximate reasoning is shown in Program 1 where, for better readability, we use program sections:

- **crisp set(s)** – to specify domain 'dom' in addition to explicitly specifying or implicitly generating crisp sets. For simplicity we assume that all constants occurring in a program have to belong to its domain 'dom';
- **base relation(s)** – to specify or generate base relations. The properties that may be considered (**T**, **B**, **4**, **5**) are listed in Lines 5–8 and should be selected or commented out in order to reflect the desired properties of relations, as depicted previously in Fig. 1. The property **D** is always required (see Remark 2.4).[11]
- **approximations** – to specify or generate lower and/or upper approximations for concepts and relations;
- **knowledge base** – to specify a background knowledge base using Asp rules.

Of course, these sections are not part of the Asp syntax and should be treated as comments. We will also use some `clingo` syntactic sugar, such as $r(k..m)$ to abbreviate the set of facts '$r(k)$.' …'$r(m)$.', where $k < m$ are natural numbers.

Program 1: The structure of Asp programs used in the paper. For a `clingo` code see Appendix A.1.

crisp set(s):		
1	dom(…).	% domain specification
2	…	
3	$c_1(\ldots)$. … $c_k(\ldots)$.	% facts about crisp sets/relations $c_1, \ldots, c_k$
4	$\text{-}c_i(X)$:- dom(X), not $c_i(X)$.	% LCWA applied to c_i $(1 \leq i \leq k)$
base relation(s):		
5	σ_{c_i}(X,X) :- dom (X).	% property **T** for the i-th base relation $(1 \leq i \leq k)$
6	σ_{c_i}(X,Y) :- σ_{c_i}(Y, X).	% property **B** for the i-th base relation $(1 \leq i \leq k)$
7	σ_{c_i}(X,Y) :- σ_{c_i}(X,Z), σ_{c_i}(Z,Y).	% property **4** for the i-th base relation $(1 \leq i \leq k)$
8	σ_{c_i}(X,Y) :- σ_{c_i}(Z,X), σ_{c_i}(Z,Y).	% property **5** for the i-th base relation $(1 \leq i \leq k)$
9	$\sigma_{c_1}(\ldots)$. … $\sigma_{c_k}(\ldots)$.	% further specifications of base relations for $c_1, \ldots, c_k$
approximations:		
10	$c_1^+{}_{\sigma_1}(\ldots)$. … $c_k{}_{\sigma_k}^+(\ldots)$.	% a specification of lower approximations of $c_1, \ldots, c_k$
11	$c_1^\oplus{}_{\sigma_1}(\ldots)$. … $c_k{}_{\sigma_k}^\oplus(\ldots)$.	% a specification of upper approximations of $c_1, \ldots, c_k$
12	:- $c_i{}_{\sigma_i}^+(\bar{X})$, $\text{-}c_i{}_{\sigma_i}^\oplus(\bar{X})$.	% ensuring property **D** for the i-th base relation $(1 \leq i \leq k)$
knowledge base:		
13	…	% other Asp rules

Properties **T**, **B**, **4**, **5**, formulated in Lines 5–8, directly reflect first-order conditions shown in Table 1. The property **D** is ensured by the constraint in Line 12: candidates for answer sets where the lower approximation of a set is not included in its upper approximation, are excluded. Indeed, the truth of $c_i{}_{\sigma_i}^+(\bar{X})$, $\text{-}c_i{}_{\sigma_i}^\oplus(\bar{X})$ indicates the existence of $\bar{X}$ belonging to $c_i{}_{\sigma_i}^+$ and not belonging to $c_i{}_{\sigma_i}^\oplus$. Therefore we have the following proposition.

Proposition 5.1 *(Correctness of Program 1). Rules in Lines 5–8 of Program 1 correctly reflect the first-order correspondents of **T**, **B**, **4**, **5** given in Table 1. The constraint in Line 12 ensures that* $c_\sigma^+ \subseteq c_\sigma^\oplus$.[12] □

Let $\mathcal{B}$ be an arbitrary knowledge base defined by means of answer set rules provided in the '**knowledge base**' section. We now address the problems outlined in Section 1:

1. computing approximations: given a crisp set c and a base relation σ_c, find c_σ^+ and $c_\sigma^\oplus$, satisfying $\mathcal{B}$. (Program 2 below);
2. computing crisp sets: given σ_c, c_σ^+ and $c_\sigma^\oplus$, find c satisfying $\mathcal{B}$. (Program 3 below);
3. computing base relations: given c, c_σ^+, $c_\sigma^\oplus$, find the underlying σ_c satisfying $\mathcal{B}$. (Program 4 below);
4. computing base relations and crisp sets: given c_σ^+ and $c_\sigma^\oplus$, find c and the underlying σ_c satisfying $\mathcal{B}$. (Program 5 below).

[11] Of course, if for some purpose needs to be avoided, the property **D** can easily be abandoned by removing the constraint in Line 12 from Program 1 (or programs using it as a template).

[12] Being trivially equivalent to property **D**.

Remark 5.2. The problems we deal with are in NP as shown by their encoding in the considered NP-complete fragment of ASP, guaranteeing nondeterministic polynomial data complexity. Since we always allow for arbitrary ASP knowledge bases, their NP-hardness is directly inherited from the NP-completeness of computing answer sets (see Theorem 4.13). To see NP-hardness, it simply suffices to use crisp ASP knowledge bases without approximate concepts and relations.

However, it is also interesting to consider the complexity of related reasoning problems with rough sets when the **knowledge base** part of the ASP programs considered is empty. As stated in [32], the majority of problems related to the "generation of reducts and their approximations, decision rules, association rules, discretization of real value attributes, symbolic value grouping, searching for new features defined by oblique hyperplanes or higher order surfaces, pattern extraction from data as well as conflict resolution or negotiation" are NP-complete or NP-hard. In particular, the problem of finding reducts, addressed in Section 6, is shown to be NP-hard in [37]. On the other hand, when the **knowledge base** section of the ASP programs considered is empty, particular problems we address in this paper may be tractable. For example, with an empty **knowledge base** section, computing approximations considered in Section 5.2 is in P, as follows from Corollary 5.4. □

5.2. Computing approximations

To keep the programs simple, we restrict the codes to a single crisp set and base relation. The extension to other crisp sets and related base relations is straightforward. Rather than using σ to denote the base relation, to make the code immediately runnable, we write 'sigma' wherever needed.

Program 2: A program for computing approximations. For the full code see Appendix A.2.

```
input  : a crisp set c and a crisp base relation σ_c
output : approximations low=c^+_{σ_c} and up=c^⊕_{σ_c}
crisp set(s):
1  | #const n=9.                                   % the size of the domain 'dom'
2  | dom (0..n).                                   % domain 'dom' consists of natural numbers 0...n
3  | c(2..6).                                      % c consists of natural numbers 2...6
4  | ...
5  | - c(X) :- dom (X), not c(X).                  % LCWA applied to c
base relation(s):
6  | % lines selected from Lines 5–8 of Program 1
7  | sigma(1,2).
8  | sigma(2,3).
9  | ...
10 | - sigma(X,Y) :- dom (X), dom (Y), not sigma(X,Y).   % LCWA applied to 'sigma'
approximations:
11 | aux(X):- sigma(X,Y), -c(Y).                   % used to compute 'low'
12 | low(X):- dom (X), not aux(X).                 % lower approximation of c wrt σ_c
13 | -low(X):- dom (X), not low(X).                % LCWA applied to 'low'
14 | up(X) :- sigma(X,Y), c(Y).                    % upper approximation of c wrt σ_c
15 | -up(X):- dom (X), not up(X).                  % LCWA applied to 'up'
16 | :- low(X), - up(X).                           % ensuring property D
knowledge base:
17 | % arbitrary rules using c, sigma, low, up and possibly other relations
```

Program 2 serves to compute approximations when a crisp set together with a crisp base relation are given:

- section '**crisp set(s)**' contains a specification of the domain (Line 2) and the input set (Lines 3–5). It is assumed that only positive literals are listed (Lines 3–4) and negative literals are defined by locally closing the world in Line 5. In the program below, we instantiate the crisp set as a set of integers for explanatory purposes;
- section '**base relation(s)**' contains a specification of properties selected from **T**–**5**. Additionally, Lines 7–9 contain positive facts about the base relation while Line 10 locally closes the world for the relation;
- section '**approximations**' contains rules for computing approximations;
- section '**knowledge base**' contains arbitrary ASP rules.

Note that the computed approximations are also crisp sets due to the application of Local Closed World Assumption in Lines 13 and 15 in Program 2.

To see the correctness of Program 2 note first that by Proposition 5.1, Lines 6 and 16 ensure that properties of base relations are handled properly.

Second, approximations are computed by rules listed in Lines 11–15, where 'aux' actually computes $\exists Y(\text{ sigma(X,Y)} \wedge \text{ - c(Y)})$ whose negation, used in Line 12, is equivalent to $\forall Y(\text{ sigma(X,Y)} \rightarrow \text{ c(Y) })$, i.e., to the lower approximation of 'c' (see Definition 2.1). Observe that 'c' and 'sigma' are crisp sets so laws of classical logic are in order here. The upper

approximation is computed by rules in Lines 14 and 15. Indeed, the rule in Line 14 defines 'up(X)' to be equivalent to $\exists Y\big(\text{sigma(X,Y)}\wedge\ \text{c(Y)}\big)$ (see Definition 2.1). Line 15 closes the world for 'up(X)'.

In consequence, we have the following proposition.

Proposition 5.3 *(Correctness of Program 2). Given a crisp set c and a crisp base relation σ_c, Program 2 correctly computes approximations low= $c^+_{\sigma_c}$ and up=$c^{\oplus}_{\sigma_c}$.* □

As stated before (see Definition 4.12), we deal with data complexity. That is, we assume that the vocabulary is fixed and we only allow facts to change which, in our case, appear in:

- the domain specification (in our programs always specified in Line 2);
- facts about crisp sets and base relations (e.g., Lines 3–4 and 7–9);
- facts in the '**knowledge base**' section.

Complexity is measured wrt the size of the domain 'dom'. Recall that, according to Definition 4.4, the domain consists of all constants occurring in the program.

Note that Program 2 is an answer set program (modulo syntactic sugar used for grouping rules in sections). As a consequence of Theorem 4.13 we have the following complexity results (see also Remark 5.2).

Corollary 5.4. *Data complexity of computing approximations is NP-complete. When c and σ are uniquely determined and can be computed in deterministic polynomial time, computing approximations are in P.* □

The assumption as to the uniqueness of c and σ is important since, in principle, there may be many resulting answer sets and they may differ in the evaluation of c and σ. Note also that when the **knowledge base** section is empty, Program 2 is stratified so, its answer sets, including approximations, can be computed in deterministic polynomial time.

5.3. Computing crisp sets

To compute crisp sets we use Program 3. In this program l and u, provided as input, refer to the lower and upper approximation of a crisp set c we would like to generate. First, a candidate for a crisp set is generated in Line 3. The sets low and up are generated from the candidate set c, using definitions of approximations provided in Program 2. The set c is rejected when premises of one of the constraints in Lines 17–20 are true. The premises express the inequality of sets 'l' and 'low' (sets 'u' and 'up'). For example, the premises of Line 17 are equivalent to:

$$\exists X\big(\text{low}(X)\wedge -\text{l}(X)\big),$$

meaning that there is a domain element X being a member of 'low' and not a member of 'l'. When this happens, the constraint in Line 17 rejects the generated candidate for an answer set.

As before, by Proposition 5.1, Lines 9 and 15 ensure that properties of base relations are handled properly. Therefore we have the following proposition.

Proposition 5.5 *(Correctness of Program 3). Given crisp base relation σ_c and crisp sets* $\text{l}\subseteq\text{u}\subseteq\text{dom}$*, Program 3 correctly computes all crisp sets c such that l= $c^+_{\sigma_c}$ and u=$c^{\oplus}_{\sigma_c}$.* □

As to the complexity, we have the following corollary of Theorem 4.13 (see also Remark 5.2).

Corollary 5.6. *Data complexity of computing crisp sets is NP-complete.* □

Remark 5.7. The complexity of computing crisp sets when the **knowledge base** section of Program 3 is empty does not appear to have been addressed in the literature. However, no matter whether the **knowledge base** section is empty or not, the number of crisp sets whose lower and upper approximations are given may be exponential in the size of the domain. For example, for total base relations, i.e., satisfying $\forall x\forall y(\sigma_c(x,y))$, and approximations $c^+_{\sigma_c}=\emptyset, c^{\oplus}_{\sigma_c}=\text{dom}$, every nonempty strict subset c of dom ($\emptyset\neq c\subsetneq\text{dom}$) is a crisp set whose lower and upper approximation wrt σ_c are respectively $\emptyset$ and dom. In these cases, the number of such crisp sets is $\mathcal{O}(2^{|\text{dom}|})$. □

5.4. Computing base relations

The method of computing base relations is analogous to that of computing crisp sets discussed in Section 5.3. However, rather than generating a crisp set, a candidate for a base relation is generated in Line 7 of Program 4. The candidate is accepted if the constraints do not reject it, i.e., it is indeed a base relation satisfying the requirements.

As before, we have the following proposition.

Program 3: A program for computing crisp sets. For the full code see Appendix A.3.

input : crisp base relation σ_c and crisp sets $l \subseteq u \subseteq dom$
output : crisp set c such that $l=c^{+}_{\sigma_c}$ and $u=c^{\oplus}_{\sigma_c}$

crisp set(s):
```
1  | #const n=9.
2  | dom (0..n).
3  | {c(X): dom (X)}.                                   % generate 'c'
4  | - c(X) :- dom (X), not c(X).                       % LCWA applied to 'c'
5  | l(0..1).
6  | -l(X):- dom (X), not l(X).                        % LCWA applied to 'l'
7  | u(0..2).
8  | -u(X):- dom (X), not u(X).                        % LCWA applied to 'u'
```
base relation(s):
```
9  | % lines selected from Lines 5–8 of Program 1
10 | sigma(1,2).
11 | sigma(2,3).
12 | ...
13 | - sigma(X,Y) :- dom (X), dom (Y), not sigma(X,Y).  % LCWA applied to 'sigma'
```
approximations:
```
14 | % Lines 11–15 of Program 2
15 | :- low(X), - up(X).                                % ensuring property D
16 | % Constraints rejecting candidates for answer sets with low≠l or up≠u:
17 | :- low(X), -l(X).
18 | :- -low(X), l(X).
19 | :- up(X), -u(X).
20 | :- -up(X), u(X).
```
knowledge base:
```
21 | % arbitrary rules using 'c', 'sigma', 'low', 'up' and possibly other relations
```

Proposition 5.8 *(Correctness of Program 4). Given a crisp set c and crisp sets* $l \subseteq u \subseteq dom$, *Program 4 correctly computes crisp base relation* σ_c *such that* $l= c^{+}_{\sigma_c}$, $u=c^{\oplus}_{\sigma_c}$. □

Program 4: A program for computing base relations. For the full code see Appendix A.4.

input : a crisp set c and crisp sets $l \subseteq u \subseteq dom$
output : crisp base relation σ_c such that $l=c^{+}_{\sigma_c}$, $u=c^{\oplus}_{\sigma_c}$

crisp set(s):
```
1  | #const n=9.
2  | dom (0..n).
3  | c(2..6).
4  | ...
5  | - c(X) :- dom (X), not c(X).
```
base relation(s):
```
6  | % lines selected from Lines 5–8 of Program 1
7  | {sigma(X,Y): dom (X), dom (Y)}.                    % generate 'sigma'
8  | - sigma(X,Y) :- dom (X), dom (Y), not sigma(X,Y).  % LCWA applied to 'sigma'
```
approximations:
```
9  | % section 'approximations' of Program 3
```
knowledge base:
```
10 | % arbitrary rules using 'c', 'sigma', 'low', 'up' and possibly other relations
```

As before, we have the following corollary regarding complexity.

Corollary 5.9. *Data complexity of computing base relations is NP-complete.* □

Remark 5.10. The complexity of computing base relations when the **knowledge base** section of Program 4 is empty, does not appear to have been addressed in the literature. However, no matter whether the **knowledge base** section is empty or not, the number of computed base relations may be exponential in the size of the domain. For example, for $c = c^{+}_{\sigma_c} = c^{\oplus}_{\sigma_c} = dom$, every (at least serial) binary relation satisfies the requirements of Program 4. In these cases, the number of such relations is $\mathcal{O}(2^{|dom|^2})$. □

5.5. Computing crisp sets and base relations

To compute base relations and crisp sets, in Program 5 we simply combine methods used in Program 3 and 4. Indeed, we have to generate a candidate for a crisp set and a candidate for a base relation and verify whether they satisfy the constraints.

As an immediate consequence of Propositions 5.5 and 5.8, we have the following proposition.

Proposition 5.11 *(Correctness of Program 5). Given crisp sets* $\mathrm{l} \subseteq \mathrm{u} \subseteq \mathrm{dom}$*, Program 5 correctly computes all crisp sets c and corresponding crisp base relations* σ_c *such that* $\mathrm{l}{=}\, c^{+}_{\sigma_c}$ *and* $\mathrm{u}{=}c^{\oplus}_{\sigma_c}$. □

Program 5: A program for computing crisp sets and base relations. For the full code see Appendix A.5.

input : crisp sets $\mathrm{l} \subseteq \mathrm{u} \subseteq \mathrm{dom}$
output : a crisp set c and a crisp base relation σ_c such that $\mathrm{l}{=}c^{+}_{\sigma_c}$ and $\mathrm{u}{=}c^{\oplus}_{\sigma_c}$
crisp set(s):
1 | % Section '**crisp set(s)**' of Program 3
base relation(s):
2 | % Section '**base relation(s)**' of Program 4
approximations:
3 | % Section '**approximations**' of Program 3
knowledge base:
4 | % arbitrary rules using 'c', 'sigma', 'low', 'up' and possibly other relations

As before, due to Theorem 4.13 and Remark 5.2, data complexity remains NP-complete. Note that the number of crisp sets and base relations may be exponential, as follows from Remarks 5.7 and 5.10.

Corollary 5.12. *Data complexity of computing crisp sets and base relations is NP-complete.* □

Programs 2–5 provide generic structures or ASP program templates for computationally generating and reasoning with rough set approximations and base relations using various input configurations. Since the lower and upper bounds of rough concepts and relations are syntactically represented, these approximate relations can be used together with standard ASP concepts and relations defined in the knowledge base section of the programs. This is a very powerful feature of the approach, where in modeling, some relations are classical and some approximate and each can constrain the other through use of ASP rules containing both types of relations.

6. A case study

In order to highlight concrete use of the computational tools for generating and using approximate relations considered in the generic ASP programs of the previous section, we will in this section consider a concrete reasoning case of some importance in data science where one would like to remove redundant attributes in incomplete datasets in order to increase the efficiency of the subsequent learning processes applied to such datasets. By incomplete datasets, we mean datasets, where for particular objects, attribute values are missing for one or more attributes associated with an object. If one thinks of the dataset as a large table where each row represents an object with attribute values and each column represents an attribute, identification of redundant (non-information providing) attributes would result in the removal of the associated columns leading to a smaller table without information loss.

Let us consider an incomplete information system, where some attributes for an object may be missing [23]. By an *incomplete information system* one means a pair $IS = \langle \mathrm{dom}, Attr\rangle$, where dom is a non-empty set of objects and $Attr$ is a set of attributes. Each attribute $a \in Attr$ is a function $a : \mathrm{dom} \longrightarrow Val \cup \{*\}$, where Val is a set of attribute values and '$*$' is a special value representing missing values. For simplicity we do not distinguish among domains of attributes.

In rough set terminology, minimal subsets of Attr, $A \subseteq \mathrm{Attr}$ such that, $\sigma^{IS}_{A} = \sigma^{IS}_{Attr}$ and for all $B \subset A, \sigma^{IS}_{B} \neq \sigma^{IS}_{Attr}$, are called *reducts* of IS.[13]

An example of an information system, considered in [23], is described in Table 2. This table gathers sample data about cars. The first column represents objects (cars) and columns 2–5 represent attribute values.

Given an information system $IS = \langle \mathrm{dom}, Attr\rangle$, each set of attributes $A \subseteq Attr$ determines a tolerance relation (a reflexive and symmetric relation, i.e., satisfying **TB**), σ^{IS}_{A}, as follows [23]:

$$\sigma^{IS}_{A}(x, y) \stackrel{\mathrm{def}}{=} \big\{\langle x, y\rangle \mid \forall a \in A\big(a(x) = a(y) \text{ or } a(x) = * \text{ or } a(y) = *\big)\big\}. \tag{21}$$

[13] As stated previously, this use of the term "reduct" has nothing to do with the use of the same term in ASP.

Table 2
Sample data about cars.

car	price	mileage	size	max_speed
1	high	high	full	low
2	low	∗	full	low
3	∗	∗	compact	high
4	high	∗	full	high
5	∗	∗	full	high
6	low	high	full	∗

σ_A^{IS} signifies a binary relation between objects that are possibly indiscernible relative to the particular subset of attributes A.

In Table 2, the set of attributes is Attr = {price, mileage, size, max_speed}. By IC let us denote the information system represented by Table 2. Our goal is to identify reducts of IC.

To represent information systems in Asp we assume:

- a domain of object identifiers 'dom';
- a one-argument relation 'attr()' gathering information about attributes;
- a three-argument relation val (X,Y,Z) where 'X' represents an object, 'Y' represents an attribute and 'Z' represents the attribute value for the object.

The null value '∗' signifying a missing attribute value is represented by a constant 'null'.

Program 6 is an auxiliary program needed for our main problem of removing redundant attributes from a dataset. This program computes $\sigma_{\text{attr}}(x, y)$, where:

- dom = $\{1, \ldots, n\}$ represents the domain of (car) object identifiers; 'n' (=6) is a constant specified in Line 1;
- attr_dom() = {price, mileage, size, max_speed} is the domain of attributes specified in Line 3;
- attr() specifies the attributes selected to reduce the information system; in Program 6 all attributes are selected in Line 4 but this can be simply adjusted for selecting their subsets;
- the domain of attribute values is vals() = {null, high, low, full, compact}; this domain is extracted from 'val' in Line 5;
- rules in Lines 6–11 reflect the contents of Table 2;
- rules in Lines 12–14 define the 'eq' relation to properly reflect (21). To this end we relax the requirement as to the equality of attributes by allowing them to be '∗', too. Note that closing the world for 'eq' is not needed;
- rules in Lines 15–16 ensure that 'sigma' is a tolerance relation. While Equation (21) enforces **T** and **B**, the lines are present since there may be user-defined clauses about 'sigma';
- Lines 15–19 define the relation 'sigma'. A `clingo` conditional is used here (see the explanation below).

Line 18 of Program 6 uses a `clingo` conditional 'aux(X, Y, A): attr(A)' which abbreviates the conjunction of literals 'aux(X, Y, A)' where 'A' are all elements satisfying 'attr(A)'. In the case of 'attr' defined in Lines 4 in Program 6, Line 18 encodes the rule:

```
sigma(X,Y):-  dom (X), dom (Y),
              aux(X, Y, price), % similar price
              aux(X, Y, mileage), % similar mileage
              aux(X, Y, size), % similar size.
              aux(X, Y, max_speed). % similar maximal speed
```

The base relation using all attributes for the information system given in Table 2, computed by Program 6 is:

```
sigma(1,1),  sigma(2,2),  sigma(3,3),  sigma(4,4),  sigma(5,5),  sigma(6,6),
sigma(2,6),  sigma(6,2),  sigma(4,5),  sigma(5,4),  sigma(5,6),  sigma(6,5).
```
(22)

Given that the base relation is computed, lower and upper approximations can be computed in the standard manner described in [23]. To constructively compute such approximations, one can use Program 2 (in Section 5), with Lines 7–9 substituted by facts obtained from (22). For example, the lower and upper approximation of the set of cars $\{1, 3, 4\}$ are:

$$low(1), low(3), up(1), up(3), up(4), up(5). \tag{23}$$

That is, the lower and upper approximation of $\{1, 3, 4\}$ wrt 'sigma' computed by Program 6 are respectively $\{1, 3\}$ and $\{1, 3, 4, 5\}$.

To compute σ_A with $A \subsetneq Attr$, it suffices to remove the rule, 'attr(X) :- attr_dom(X)' from Line 4 and replace it with a rule that generates the attributes A, or a set of facts representing A. For example, $\sigma_{\{price,size\}}$ can be computed by replacing Line 4 with:

Program 6: A program for computing σ_{attr} using attribute values. For the full code see Appendix A.6.

```
input  : relation 'val' encoding attribute values of an information system.
output : tolerance relation σ_attr.
crisp set(s):
1  | #const n=6.                                                          % the size of the domain (6 cars)
2  | dom (1..n).
3  | attr_dom(price).  attr_dom(mileage).  attr_dom(size).  attr_dom(max_speed).
4  | attr(X) :- attr_dom(X).                                              % all attributes are selected
5  | vals(X) :- val(_, _, X).                                             % extracting attribute values
6  | val(1, price, high).  val(1, mileage, high).  val(1, size, full).  val(1, max_speed, low).
7  | val(2, price, low).  val(2, mileage, null).  val(2, size, full).  val(2, max_speed, low).
8  | val(3, price, null).  val(3, mileage, null).  val(3, size, compact).  val(3, max_speed, high).
9  | val(4, price, high).  val(4, mileage, null).  val(4, size, full).  val(4, max_speed, high).
10 | val(5, price, null).  val(5, mileage, null).  val(5, size, full).  val(5, max_speed, high).
11 | val(6, price, low).  val(6, mileage, high).  val(6, size, full).  val(6, max_speed,null).
base relation(s):
12 | eq(X,Y) :- vals(X), vals(Y), X=Y.
13 | eq(X,Y) :- vals(X), vals(Y), Y=null.
14 | eq(X,Y) :- vals(X), vals(Y), X=null.
15 | sigma(X,X) :- dom (X).                                               % property T
16 | sigma(X,Y) :- sigma(Y,X).                                            % property B
17 | aux(X,Y,A) :- attr(A), val(X,A,Z1), val(Y,A,Z2), eq(Z1,Z2).
18 | sigma(X,Y) :- dom (X), dom (Y), aux(X,Y,A): attr(A).                 % all attributes are equal wrt 'eq'
19 | -sigma(X,Y) :- dom (X), dom (Y), not sigma(X,Y).                     % LCWA applied to 'sigma'
```

attr(price). attr(size).

The relation $\sigma_{\{price,size\}}$ computed by the suitably adjusted Program 6 is the relation shown in (22) extended by:

sigma(1,4), sigma(4,1), sigma(1,5), sigma(5,1),
sigma(2,5), sigma(5,2), sigma(5,6), sigma(6,5). (24)

It is clearly not the case that $\sigma_{\{price,size\}}$ is a reduct of σ_{Attr}.

Let us now focus on computing subsets of $A \subsetneq Attr$ preserving the original approximation based on $Attr$, but not necessarily minimal. That is, given an information system, we are interested in sets of attributes A of cardinality not greater than k, where $k > 0$ is a natural number, such that $\sigma_A = \sigma_{Attr}$.

This is achieved by Program 7, where we only have to generate suitable candidates for subsets of attributes as done in Line 3.

Program 7: A program for computing information systems reduced to at most 'k' attributes. For the full code see Appendix A.7.

```
input  : Relation 'val' encoding attribute values of an information system IS specified in terms of 'val' and a natural number k > 0.
output : Relation 'attrR' containing at most 'k' attributes such that the similarity relation for IS with attributes reduced to 'attrR' remains unchanged
         when compared to IS.
crisp set(s):
1  | % Section 'crisp set(s)' of Program 6
2  | #const k=3.
3  | 1 {attrR(X): attr_dom(X) } k.                                        % select at least 1 and at most k attributes
base relation(s):
4  | % Compute 'sigma' for IS using Section 'base relation(s)' of Program 6
5  | % Compute 'sigmaR' for the reduced IS using Lines 15–19 of Section 'base relation(s)'
6  | %       of Program 6 with 'sigma', 'aux' and 'attr' respectively replaced by
7  | %       'sigmaR', 'auxR' and attrR':
8  | sigmaR(X,X) :- dom (X).                                              % property T
9  | sigmaR(X,Y) :- sigmaR(Y,X).                                          % property B
10 | auxR(X,Y,A) :- attR(A), val(X,A,Z1), val(Y,A,Z2), eq(Z1,Z2).
11 | sigmaR(X,Y) :- dom (X), dom (Y), auxR(X,Y,A): attrR(A).              % all selected attributes are equal wrt 'eq'
12 | -sigmaR(X,Y) :- dom (X), dom (Y), not sigmaR(X,Y).                   % LCWA applied to 'sigmaR'
13 | % Constraints rejecting candidates for answer sets not preserving the similarity relation:
14 |  :- sigma(X,Y), - sigmaR(X,Y).
15 |  :- - sigma(X,Y), sigmaR(X,Y).
16 | % #minimize{1, X: attrR(X)}.                                          % to be uncommented when minimizing the set of attributes
```

For example, for k=3, Program 7 computes the attributes:

'attr(price)', 'attr(size)', 'attr(max_speed)'.

In order to compute minimal sets of attributes rather than those only restricted to at most 'k', one can use the `clingo` command #minimize:

```
#minimize{1, X: attr(X) }.
```
(25)

It suffices to add command (25) to Program 7 and to remove 'k' from Line 3. The resulting minimal set of attributes is again 'attr(price)', 'attr(size)', 'attr(max_speed)'. This is, in fact, the unique reduct for Table 2.

Remark 6.1. When modeling incomplete datasets, one can take advantage of some of the powerful features of ASP in the context of missing information. The assumption that the '$*$' value of an attribute can represent *any* legal value of the attribute can often be too broad. Indeed, in the area of knowledge representation many forms of reasoning address the problem of missing information. Generally, such information is assumed to represent "normal" situations and typical values become defaults. For example, one may add to the knowledge base:

knowledge base:

```
val(X, max_speed, high) :- val(X, max_speed, null), val(X, price, high).
val(X, price, low) :- val(X, price, null), val(X, mileage, high), val(X, max_speed, low).
...
```

The above rules represent a kind of default. ASP also offers default negation allowing one to formalize rules when unknown information is not present in the underlying knowledge base. That is, rather than listing all facts about 'val' with at least one attribute being '$*$', one could skip them. In such a case, the truth value of 'val' would become U and rules of the following form could be applicable:

```
val(X, max_speed, high) :- not -val(X, max_speed, high), val(X, price, high).
```

Of course, such rules represent a form of Reiter's default rules [35]. □

In [23] *reducts for objects* are defined, too. A set of attributes $A \subseteq Attr$ is a *reduct of IS for* $x \in \text{dom}$ iff A is a minimal subset of attributes A such that $S_A(x) = S_{Attr}(x)$, where S_A is the tolerance relation determined by A and $S_A(x) \stackrel{\text{def}}{=} \{y \in \text{dom} \mid \sigma_A(x, y)\}$. This notion can be generalized in a natural way to sets of objects as follows.

Definition 6.2 *(Reduct for a set).* Let $IS = \langle \text{dom}, Attr \rangle$ be an information system and $C \subseteq \text{dom}$ be a rough set. By a *reduct of IS for C* we mean a minimal subset of attributes A such that $S_A(C) = S_{Attr}(C)$, where S_A is the tolerance relation determined by A. □

Of course, a subset of attributes A is a reduct for an object o iff A is a reduct for the set $\{o\}$.

Note that reducts for sets are interesting from the point of view of optimizing queries to information systems. A query can be seen as a formula involving attributes, used to select objects of interest. For example, one may be interested in objects satisfying a formula:

$$mileage = high \vee size = compact.$$

The formula can be understood as a query for selecting cars with high mileage or compact size. Semantically such a formula represents a set of cars (elements from the domain of objects). Given that the query is frequently used, e.g., for creating reports for car sellers offering cars to clients, and the information system does not change that frequently, the reduct for the selected set of cars may be useful to reduce the information system to consists only of attributes relevant to the query (i.e., to the selected set of objects).

Another motivation of reducts for a set reflects a kind of rule mining sub-task where one tags a set of objects as belonging to a given class C, e.g., being of a good quality from some point of view, and is interested in selecting minimal sets of attributes of IS preserving the tolerance relation restricted to C. Note that a user or an expert may mark some objects as surely belonging to the set, some of them surely outside of the set, and some of them doubtful from that point of view.

For computing reducts of an information system IS for a set C it suffices to use Program 7 with facts about 'val' restricted to those being in the set C. This can be achieved by:

- adding facts/rules about the set C to Program 7's **crisp set(s)** section
- defining a new relation 'valR':

```
valR(X,Y,Z) :- c(X), val(X,Y,Z).
```
(26)

The relation 'valR' should then be used instead of 'val' in the Program's 7 **base relation(s)** section.

Remark 6.3. Note that when using (26) we implicitly apply the CWA, unifying negative with unknown information about C. To make an explicit use of negative information one would have to extend information systems with the part consisting of '- val' and develop the underlying machinery by slightly extending the provided ASP programs. □

7. Conclusions

In this paper, we have shown how ASPS, an established tool for reasoning about incomplete relations and nonmonotonic reasoning, can be leveraged to serve as a basis for reasoning about generalized rough set approximations. We provide an interpretation of answer sets as (generalized) approximations of crisp sets(when possible) and show how to use ASP solvers as a tool for reasoning about(generalized) rough set approximations situated in realistic knowledge bases. The paper includes generic ASP templates for doing this and it also provides a concrete case study describing how redundant attributes in an incomplete Information System can be identified and removed from the associated dataset without any information loss. In rough set theory, this is the process of generating reducts for information systems.

Another bridge between rough set approximations and ASP is the indirect characterization of rough sets as orthopairs [7] with constraints, and the relation of orthopairs to Answer Sets. In [7], orthopairs are used to characterize a diverse set of models for uncertainty such as Twofold sets, Shadowed sets, and Interval Sets. It would be interesting to apply the techniques in this paper to the development of new tools for reasoning about these diverse models of uncertainty.

Additionally, it has been shown how ASPS can be used for modeling rough set approximations and reasoning with them. It is clear from this that one can combine the representation of approximate relations as rough sets with the representation of incomplete classical relations used in ASP. This opens up new possibilities for an interesting form of hybrid reasoning. In fact, this is hinted at in Remark 6.1 in Section 6, where the expressive power of default rules is used to refine modeling of incomplete attribute values more concisely in Incomplete Information Systems.

Both of these avenues will be pursued in future work.

Declaration of competing interest

The authors certify that they have NO affiliations with or involvement in any organization or entity with any financial interest (such as honoraria; educational grants; participation in speakers' bureaus; membership, employment, consultancies, stock ownership, or other equity interest; and expert testimony or patent-licensing arrangements), or non-financial interest (such as personal or professional relationships, affiliations, knowledge or beliefs) in the subject matter or materials discussed in this manuscript.

Appendix A. ASP programs

In the appendix we enclose ASP program codes ready for copying and executing in a `clingo` solver.

A.1. The code of program 1

```
% The structure of programs assumed in the paper
% crisp set(s):
% dom(. . . ) - domain specification
% ... facts about crisp sets/relations
% ... Local Closed World Assumption

% base relation(s):

% properties to be selected:
sigma(X,X):- dom(X).                  % reflexivity (T)
sigma(X,Y):- sigma(Y,X).              % symmetry (B)
sigma(X,Y):- sigma(X,Z), sigma(Z,Y). % transitivity (4)
sigma(X,Y):- sigma(Z,X), sigma(Z,Y). % Euclidicity (5)

% ... further specifications of base relations ...

% approximations:
% a specification of lower and upper approximations
:- low(X), -up(X).                    % ensuring property D

% knowledge base:
% ... ASP rules ...
```

A.2. The code of program 2

```
% A program for computing approximations
% input:  a crisp set 'c' and a crisp base relation 'sigma'
% output: lower and upper approximation ('low', 'up') of 'c' wrt 'sigma'

% crisp set(s):
#const n=9.                    % the size of the domain 'dom'
dom (0..n).                    % domain 'dom' consists of natural numbers 0...n

c(2..6).                       % 'c' consists of natural numbers 2...6
-c(X) :- dom(X), not c(X).     % LCWA applied to 'c'

% base relation(s):

% properties selected from Program 1:
sigma(X,X):- dom(X).           % reflexivity (T)
sigma(X,Y):- sigma(Y,X).       % symmetry (B)

% further specifications of base relations:
sigma(1,2).
sigma(2,3).
-sigma(X,Y) :-  dom(X), dom(Y), not sigma(X,Y). % LCWA applied to 'sigma'

% approximations:
aux(X):- sigma(X,Y), -c(Y).    % used to compute 'low'
low(X):- dom(X), not aux(X).   % lower approximation of 'c'
-low(X):- dom(X), not low(X). % LCWA applied to 'low'
up(X) :-  sigma(X,Y), c(Y).    % upper approximation of 'c'
-up(X):- dom (X), not  up(X). % LCWA applied to 'up'
%:- low(X), -up(X).            % ensuring property D not needed (T is selected)

#show low/1.
#show up/1.
```

A.3. The code of program 3

```
% A program for computing crisp sets
% input:  a crisp base relation 'sigma' and crisp sets 'l', 'u'
% output: crisp set 'c' such that:
%         'l' is the lower approximation of 'c' wrt 'sigma' and
%         'u' is the upper approximation of 'c' wrt 'sigma'

% crisp set(s):

#const n=9.
dom(0..n).
{c(X): dom(X)}.                  % generate 'c'
-c(X):- dom(X), not c(X).        % LCWA applied to 'c'

l(0..1).
-l(X):- dom(X), not l(X).        % LCWA applied to 'l'
u(0..3).
-u(X):- dom(X), not u(X).        % LCWA applied to 'u'

% base relation(s):

% properties selected from Program 1:
sigma(X,X):- dom(X).             % reflexivity (T)
sigma(X,Y):- sigma(Y,X).         % symmetry (B)

% further specifications of base relations:
sigma(1,2).
sigma(2,3).
-sigma(X,Y):- dom(X), dom(Y), not sigma(X,Y). % LCWA applied to 'sigma'
```

```
% approximations:
aux(X):- sigma(X,Y), -c(Y).
low(X):- dom(X), not aux(X).
-low(X):- dom(X), not low(X).    % LCWA applied to 'low'
up(X):- sigma(X,Y), c(Y).
-up(X):- dom(X), not up(X).      % LCWA applied to 'up'
%:- low(X), -up(X).              % ensuring property D not needed (T is selected)

% constraints rejecting candidates for answer sets with low=/=l or up=/=u:

:- up(X), -u(X).
:- -up(X), u(X).
:- low(X), -l(X).
:- -low(X), l(X).

#show c/1.
```

A.4. The code of program 4

```
% A program for computing base relations
% input:  a crisp set 'c' and crisp sets 'l', 'u'
% output: crisp base relation 'sigma' such that:
%         'l' is the lower approximation of 'c' wrt 'sigma' and
%         'u' is the upper approximation of 'c' wrt 'sigma'

% crisp set(s):

#const n=9.
dom(0..n).
c(0..2).
-c(X):- dom(X), not c(X).                    % LCWA applied to 'c'
l(0..1).
-l(X):- dom(X), not l(X).                    % LCWA applied to 'l'
u(0..3).
-u(X):- dom(X), not u(X).                    % LCWA applied to 'u'

% base relation(s):

% lines selected from Program 1:
sigma(X,X):- dom(X).                         % reflexivity (T)
sigma(X,Y):- sigma(Y,X).                     % symmetry (B)

% further specifications of base relations:
{sigma(X,Y): dom(X), dom(Y)}.                % generate 'sigma'
-sigma(X,Y):- dom(X), dom(Y), not sigma(X,Y). % LCWA applied to 'sigma'

% approximations:
aux(X):- sigma(X,Y), -c(Y).
low(X):- dom(X), not aux(X).
-low(X):- dom(X), not low(X).                % LCWA applied to 'low'
up(X):- sigma(X,Y), c(Y).
-up(X):- dom(X), not up(X).                  % LCWA applied to 'up'
%:- low(X), -up(X).        % ensuring property D not needed (T is selected)

% constraints rejecting candidates for answer sets with low=/=l or up=/=u:
:- up(X), -u(X).
:- -up(X), u(X).
:- low(X), -l(X).
:- -low(X), l(X).

#show sigma/2.
```

A.5. The code of program 5

```
% A program for computing crisp sets and base relations
% input:  crisp sets 'l', 'u'
% output: a crisp set 'c' and a crisp base relation 'sigma' such that:
%         'l' is the lower approximation of 'c' wrt 'sigma' and
%         'u' is the upper approximation of 'c' wrt 'sigma'

% crisp set(s):

#const n=9.
dom(0..n).
{c(X): dom(X)}.                                % generate 'c'
-c(X):- dom(X), not c(X).                      % LCWA applied to 'c'

l(0..1).
-l(X):- dom(X), not l(X).                      % LCWA applied to 'l'
u(0..3).
-u(X):- dom(X), not u(X).                      % LCWA applied to 'u'

% base relation(s):

% lines selected from Program 1:
sigma(X,X):- dom(X).                           % reflexivity (T)
sigma(X,Y):- sigma(Y,X).                       % symmetry (B)

% further specifications of base relations:
{sigma(X,Y): dom(X), dom(Y)}.                  % generate 'sigma'
-sigma(X,Y) :- dom(X), dom(Y), not sigma(X,Y). % LCWA applied to 'sigma'

% approximations:
aux(X):- sigma(X,Y), -c(Y).
low(X):- dom(X), not aux(X).
-low(X):- dom(X), not low(X).                  % LCWA applied to 'low'
up(X):- sigma(X,Y), c(Y).
-up(X):- dom(X), not up(X).                    % LCWA applied to 'up'
%:- low(X), -up(X).       % ensuring property D not needed (T is selected)

% constraints rejecting candidates for answer sets with low=/=l or up=/=u:
:- up(X), -u(X).
:- -up(X), u(X).
:- low(X), -l(X).
:- -low(X), l(X).

#show c/1.
#show sigma/2.
```

A.6. The code of program 6

```
% A program for computing base relation using attribute values
% input:  relation 'val' encoding attribute values of an information system
% output: tolerance relation 'sigma'

% crisp set(s):

#const n=6.% the size of the domain (6 cars)
dom(1..n).

attr_dom(price).  attr_dom(mileage).  attr_dom(size).  attr_dom(max_speed).
attr(X)  :-  attr_dom(X).    % all attributes are selected

vals(X):-  val(_, _, X).     % extracting attribute values

val(1, price, high).   val(1, mileage, high).
val(1, size, full).    val(1, max_speed, low).
```

```
val(2, price, low).    val(2, mileage, null).
val(2, size, full).    val(2, max_speed, low).
val(3, price, null).   val(3, mileage, null).
val(3, size, compact). val(3, max_speed, high).
val(4, price, high).   val(4, mileage, null).
val(4, size, full).    val(4, max_speed, high).
val(5, price, null).   val(5, mileage, null).
val(5, size, full).    val(5, max_speed, high).
val(6, price, low).    val(6, mileage, high).
val(6, size, full).    val(6, max_speed, null).

% base relation(s):
eq(X,Y) :-  vals(X), vals(Y), X=Y.
eq(X,Y) :-  vals(X), vals(Y), Y=null.
eq(X,Y) :-  vals(X), vals(Y), X=null.

sigma(X,X) :-  dom (X).                          % reflexivity (T)
sigma(X,Y) :-  sigma(Y,X).                       % symmetry (B)

% further specifications of base relations:
aux(X,Y,A) :-  attr(A), val(X,A,Z1), val(Y,A,Z2), eq(Z1,Z2).
sigma(X,Y) :-  dom (X), dom (Y), aux(X,Y,A): attr(A).
% all attributes are equal wrt 'eq'
-sigma(X,Y) :-  dom (X), dom (Y), not  sigma(X,Y). % LCWA applied to 'sigma'

#show sigma/2.
```

A.7. The code of program 7

```
% A program for computing information systems reduced to at most 'k' attributes
% input:  relation 'val' encoding attribute values of an information system IS
%         specified in terms of 'val' and a natural \xch{number k}{number number k}>0
% output: relation 'attrR' containing at most 'k' attributes such that
%         the similarity relation for IS with attributes reduced to 'attrR'
%         remains unchanged when compared to IS.

% crisp set(s):

#const n=6.                    % the size of the domain (6 cars)
dom(1..n).

attr_dom(price).  attr_dom(mileage).  attr_dom(size).  attr_dom(max_speed).
attr(X)  :-  attr_dom(X).      % all attributes are selected
vals(X):-  val(_, _, X).       % extracting attribute values

val(1, price, high).   val(1, mileage, high).
val(1, size, full).    val(1, max_speed, low).
val(2, price, low).    val(2, mileage, null).
val(2, size, full).    val(2, max_speed, low).
val(3, price, null).   val(3, mileage, null).
val(3, size, compact). val(3, max_speed, high).
val(4, price, high).   val(4, mileage, null).
val(4, size, full).    val(4, max_speed, high).
val(5, price, null).   val(5, mileage, null).
val(5, size, full).    val(5, max_speed, high).
val(6, price, low).    val(6, mileage, high).
val(6, size, full).    val(6, max_speed, null).

#const k=3.
1{attrR(X): attr_dom(X)}k.     % select at least 1 and at most 'k' attributes

% base relation(s):

eq(X,Y) :-  vals(X), vals(Y), X=Y.
eq(X,Y) :-  vals(X), vals(Y), Y=null.
```

```
eq(X,Y) :-  vals(X), vals(Y), X=null.

sigma(X,X) :-  dom(X).          % property T
sigma(X,Y) :-  sigma(Y,X).      % property B
aux(X,Y,A) :-  attr(A), val(X,A,Z1), val(Y,A,Z2), eq(Z1,Z2).
sigma(X,Y) :-  dom(X), dom(Y), aux(X,Y,A): attr(A).
% all attributes are equal wrt 'eq'
-sigma(X,Y) :-  dom (X), dom (Y), not sigma(X,Y). %  LCWA applied to 'sigma'

sigmaR(X,X) :-  dom(X).         % property T
sigmaR(X,Y) :-  sigmaR(Y,X).    % property B
auxR(X,Y,A) :-  attrR(A), val(X,A,Z1), val(Y,A,Z2), eq(Z1,Z2).
sigmaR(X,Y) :-  dom(X), dom(Y), auxR(X,Y,A): attrR(A).
% all attributes are equal wrt 'eq'
-sigmaR(X,Y) :-  dom(X), dom(Y), not sigmaR(X,Y). %  LCWA applied to 'sigmaR'

% constraints rejecting candidates for answer sets not preserving
% the similarity relation:
:- sigma(X,Y), -sigmaR(X,Y).
:- -sigma(X,Y), sigmaR(X,Y).

% #minimize{1, X: attrR(X)}.    % to be uncommented when minimizing the set
% of attributes

#show attrR/1.
```

A.8. The code included in Examples 4.5, 4.11, 4.14

In the following code rules (12) and (13) can be used interchangeably by commenting out one of them. The LCWA rule (20) is stronger than rules (10)-(11) so can be used to replace them.

```
may_buy(bob, H):- house(H), high_quality(H), located(H, center).          % (6)
may_buy(bob, H):- house(H), high_quality(H), located(H, Loc),             % (7)
                  residential(Loc), commuting_dist(Loc),                  % (8)
                  not -charming(Loc).                                     % (9)
-may_buy(bob, H):- house(H), -high_quality(H).                            % (10)
-may_buy(bob, H):- house(H), located(H, Loc), -commuting_dist(Loc).       % (11)

3 {to_visit(H): may_buy(bob, H)} 5.                                        % (12)

% 3 {to_visit(H): may_buy(bob, H), located(H, Loc), good_school_in(Loc)} 5. % (13)

% Facts (14):

house(h1). high_quality(h1).
house(h2). high_quality(h2).
house(h3). high_quality(h3).
house(h4). high_quality(h4).

located(h1,center).   located(h2,center).
located(h3, a3).      located(h4, a4).
residential(a3).      residential(a4).
commuting_dist(a3).   commuting_dist(a4).
-charming(a4).
good_school_in(a3).   good_school_in(a4).   good_school_in(center).

% -may_buy(bob, H):- house(H), not may_buy(bob, H).                        % (20)
```

References

[1] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley Pub. Co., Boston, MA, USA, 1996.
[2] M. Alviano, W. Faber, N. Leone, S. Perri, G. Pfeifer, G. Terracina, The disjunctive datalog system DLV, in: O. de Moor, G. Gottlob, T. Furche, A. Sellers (Eds.), Datalog Reloaded, Springer, 2011, pp. 282–301.
[3] G. Antoniou, A tutorial on default logics, ACM Comput. Surv. 31 (1999) 337–359.
[4] C. Baral, Knowledge Representation, Reasoning, and Declarative Problem Solving, Cambridge University Press, 2003.

[5] G. Brewka, T. Eiter, M. Truszczyński, Answer set programming at a glance, Commun. ACM 54 (2011) 92–103.
[6] B.F. Chellas, Modal Logic - an Introduction, Cambridge University Press, 1980.
[7] D. Ciucci, Orthopairs: a simple and widely used way to model uncertainty, Fundam. Inform. 108 (2011) 287–304.
[8] D. Ciucci, D. Dubois, Three-valued logics, uncertainty management and rough sets, in: J. Peters, A. Skowron (Eds.), Transactions on Rough Sets XVII, Springer, Berlin Heidelberg, 2014, pp. 1–32.
[9] S. Demri, E. Orłowska, Incomplete Information: Structure, Inference, Complexity, EATCS Monographs, Springer, 2002.
[10] P. Doherty, W. Łukaszewicz, A. Skowron, A. Szałas, Knowledge Representation Techniques. A Rough Set Approach, Studies in Fuziness and Soft Computing, vol. 202, Springer Verlag, 2006.
[11] P. Doherty, W. Łukaszewicz, A. Szałas, Efficient reasoning using the local closed-world assumption, in: A. Cerri, D. Dochev (Eds.), Proc. 9th Int. Conference AIMSA 2000, Springer-Verlag, 2000, pp. 49–58.
[12] P. Doherty, A. Szałas, Stability, supportedness, minimality and Kleene Answer Set Programs, in: T. Eiter, H. Strass, M. Truszczyński, S. Woltran (Eds.), Advances in Knowledge Representation, Logic Programming, and Abstract Argumentation - Essays Dedicated to G. Brewka on the Occasion of His 60th Birthday, Springer, 2015, pp. 125–140.
[13] P. Doherty, A. Szałas, On the correspondence between approximations and similarity, in: S. Tsumoto, R. Słowiński, H. Komorowski, J. Grzymala-Busse (Eds.), Proc. 4th Conf. RSCTC Rough Sets and Current Trends in Computing, Springer, 2004, pp. 143–152.
[14] P. Doherty, A. Szałas, A correspondence framework between three-valued logics and similarity-based approximate reasoning, Fundam. Inform. 75 (2007) 179–193.
[15] O. Etzioni, K. Golden, D. Weld, Sound and efficient closed-world reasoning for planning, Artif. Intell. 89 (1997) 113–148.
[16] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Answer Set Solving in Practice. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan and Claypool Pub., 2012.
[17] M. Gebser, B. Kaufmann, A. Neumann, T. Schaub, clasp: A conflict-driven answer set solver, in: 9th Int. Conf. LPNMR.07, 2007, pp. 260–265.
[18] M. Gelfond, Y. Kahl, Knowledge Representation, Reasoning, and the Design of Intelligent Agents - The Answer-Set Programming Approach, Cambridge University Press, 2014.
[19] M. Gelfond, V. Lifschitz, The stable model semantics for logic programming, in: R. Kowalski, K. Bowen (Eds.), Proc. of Int'l Logic Programming, MIT Press, 1988, pp. 1070–1080.
[20] G.E. Hughes, M.J. Cresswell, An Introduction to Modal Logic, Methuen and Co. Ltd., London, New York, 1968.
[21] T. Janhunen, Cross-translating answer set programs using the ASPTOOLS collection, KI: Künstl. Intell. 32 (2018) 183–184.
[22] S. Kleene, On notation for ordinal numbers, J. Symb. Log. 3 (1938) 150–155.
[23] M. Kryszkiewicz, Rough set approach to incomplete information systems, Inf. Sci. 112 (1998) 39–49.
[24] A. Kumar, M. Banerjee, Kleene algebras and logic: Boolean and rough set representations, 3-valued, rough set and Perp semantics, Stud. Log. 105 (2017) 439–469.
[25] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, F. Scarcello, The DLV system for knowledge representation and reasoning, ACM Trans. Comput. Log. 7 (2006) 499–562.
[26] Y. Lierler, cmodels - SAT-based disjunctive answer set solver, in: Logic Programming and Nonmonotonic Reasoning, 8th International Conference, LPNMR 2005, 2005, pp. 447–451.
[27] F. Lin, Y. Zhao, ASSAT: computing answer sets of a logic program by SAT solvers, Artif. Intell. 157 (2004) 115–137.
[28] J. Łukasiewicz, O logice trójwartościowej (in Polish) Ruch Filoz. 5 (1920) 170–171;
English translation: On three-valued logic, in: L. Borkowski (Ed.), Selected works by Jan Łukasiewicz, North-Holland, Amsterdam, 1970, pp. 87–88.
[29] I. Niemelä, P. Simons, T. Soininen, Stable model semantics of weight constraint rules, in: M. Gelfond, N. Leone, G. Pfeifer (Eds.), Proc. LPNMR: 5th Conf. Logic Programming and Nonmonotonic Reasoning, Springer, 1999, pp. 317–331.
[30] Z. Pawlak, Rough sets, Int. J. Comput. Inf. Sci. 11 (1982) 341–356.
[31] Z. Pawlak, Rough Sets. Theoretical Aspects of Reasoning about Data, Kluwer Academic Publishers, Dordrecht, 1991.
[32] Z. Pawlak, A. Skowron, Rudiments of rough sets, Inf. Sci. 117 (2007) 3–27.
[33] L. Polkowski, Rough Sets, Mathematical Foundations. Advances in Intelligent and Soft Computing, vol. 15, Physica-Verlag, 2002.
[34] R. Reiter, On closed world data bases, in: H. Gallaire, J. Minker (Eds.), Logic and Data Bases, Plenum Press, 1978, pp. 55–76.
[35] R. Reiter, A logic for default reasoning, Artificial Intelligence 13 (1980) 81–132.
[36] P. Simons, I. Niemelä, T. Soininen, Extending and implementing the stable model semantics, Artificial Intelligence 138 (2002) 181–234.
[37] A. Skowron, C. Rauszer, The discernibility matrices and functions in information systems, in: R. Slowiński (Ed.), Intelligent Decision Support - Handbook of Applications and Advances of the Rough Sets Theory, in: Theory and Decision Library, vol. 11, Springer, 1992, pp. 331–362.
[38] R. Słowiński, D. Vanderpooten, Similarity relation as a basis for rough approximations, in: P. Wang (Ed.), Advances in Machine Intelligence & Soft Computing, Bookwrights, Raleigh NC, 1997, pp. 17–33.
[39] R. Słowiński, D. Vanderpooten, A generalized definition of rough approximations based on similarity, IEEE Trans. Knowl. Data Eng. 12 (2000) 331–336.
[40] J. Van Benthem, Correspondence theory, in: D. Gabbay, F. Guenthner (Eds.), Handbook of Philosophical Logic, D. Reidel Pub. Co., 1984, pp. 167–247.
[41] Y. Yao, S. Wong, T. Lin, A review of rough set models, in: T. Lin, N. Cercone (Eds.), Rough Sets and Data Mining, Springer, 1997, pp. 47–75.
[42] Q. Zhang, Q. Xie, G. Wang, A survey on rough set theory and its applications, CAAI Trans. Intell. Technol. 1 (2016) 323–333.