Non-Commutativity and Expressive Deductive Logic Databases

S. Krajči,[1], R. Lencses,[1], J. Medina,[2]
M. Ojeda-Aciego,[2⋆], A. Valverde,[2], and P. Vojtáš[3]

[1] Institute of Informatics. P.J. Šafárik University. Slovakia
[2] Dept. Matemática Aplicada. Universidad de Málaga. Spain
[3] Inst. Computer Science, Acad. Sci. of the Czech Republic

Abstract. The procedural semantics of multi-adjoint logic programming is used for providing a model-theoretic semantics for a data model. A translation method for deductive logic databases is presented for obtaining a relational algebra with classical projection and enriched parametric join operator with aggregations. The use of non-commutative conjunctors allows for a model of different degrees of granulation and precision, whereas expressiveness is achieved by using multiple-valued connectives.

Keywords. deductive databases, datalog, multiple-valued logic, expressiveness, non-commutative connectives.

1 Introduction

The handling of imprecision in databases is a topic which is getting growing attention, for knowledge-base systems must typically deal with the intrinsic imprecision of the data, vagueness and imperfection in knowledge, in particular, in the form of incompleteness, inconsistency, and uncertainty.

Several frameworks for manipulating data and knowledge have been proposed. Our approach here is a lattice-valued logic programming and/or Datalog paradigm, which permits the articulation of vague concepts and, moreover, has the property that the truth of an argument can diminish as the number of inferences in it increases.

Multi-adjoint logic programming was introduced in [8] as a refinement of both initial work in [13] and residuated logic programming [2]. It allows for very general connectives in the body of the rules, and sufficient conditions for the continuity of its semantics are known.

Such an approach is interesting for applications: for instance, in [10] a system is presented where connectives are learnt from different users' examples and, thus, one can imagine a scenario in which knowledge is described by a many-valued logic program where connectives have many-valued truth functions representing conjunctions, disjunctors or, more generally, aggregation operators (arithmetic mean, weighted sum, ...) where different implications could be

⋆ Corresponding author. aciego@uma.es

needed for different purposes, and different aggregators are defined for different users, depending on their preferences.

It is important to recall that many different "and" and "or" operations have been proposed for use in fuzzy logic. It is therefore important to select the operations which are the best for each particular application.

Several papers discuss the optimal choice of "and" and "or" operations for fuzzy control, when the main criterion is to get the stablest control. In reasoning application, however, it is more appropriate to select operations which are the best in reflecting human reasoning, i.e., operations which are "the most logical".

2 Motivating examples

In fuzzy logic there is a well developed theory of t-norms, t-co-norms and residual implications. The objective of this section is to show some interesting non-standard connectives to motivate the consideration of a more general class of connectives in fuzzy logic. The motivation is the following:

When evaluating the relevance of answers to a given query it is common to use some subjective interpretation of human preferences in a granulated way. This is, fuzzy truth-values usually describe steps in the degree of perception (numerous advocations of this phenomenon have been pointed out by Zadeh). This is connected to the well-known fact that people can only distinguish finitely many degrees of quality (closeness, cheapness, ...) or quantity in control. Thus, in practice, although we use the product t-norm $\&_p(x, y) = x \cdot y$, we are actually working with a piece-wise constant approximation of it. In this generality, it is possible to work with approximations of t-norms and/or conjunctions learnt from data by a neural net like, for instance, those in [10].

We are looking for a hotel which is close to downtown, with reasonable price and being a new building. Classical fuzzy approaches would assign a user "his" interpretation of "close", "reasonable" and "new". In practice, we recognize finitely many degrees of being close, reasonable, new, so the fuzzy sets have a stepwise shape. Actually, we are working on intervals of granulation and/or indistinguishability. It is just a matter of representation that the outcome is done by means of intervals.

Namely, the set of truth-values will be considered to be a lattice. This motivates our lattice-valued approach. It is easy to obtain examples in which the lattice can be:

- Generated by a partition of the real unit interval $[0, 1]$.
- All subintervals of $[0, 1]$.
- All the probability distributions on $[0, 1]$.

Regarding the use of non-standard connectives, just consider that a variable represented by x can be observed with m different values, then surely we should be working with a regular partition of $[0, 1]$ of m pieces. This means that a given value x should be fitted to this 'observation' scale as the least upper bound with the form k/m (analytically, this corresponds to $\big(\lceil m \cdot x \rceil\big)/m$ where $\lceil \_ \rceil$ is the

ceiling function). A similar consideration can be applied to both, variable y and the resulting conjunction; furthermore, it might be possible that each variable has different granularity.

Formally, assume in x-axis we have a partition into n pieces, in y-axis into m pieces and in z-axis into k pieces. Then the approximation of the product conjunction looks like

Definition 1. *Denote* $(z)_p = \frac{\lceil p \cdot z \rceil}{p}$ *and define, for naturals* $n, m, k > 0$

$$C^k_{n,m}(x, y) = \big((x)_n \cdot (y)_m\big)_k$$

Example 1. Connectives $C^k_{n,m}(x, y)$ need be neither associative nor commutative:

1. For instance $C^{10}_{10,10}$, denoted simply as C, is not associative

$$C(0.7, C(0.7, 0.3)) = C(0.7, (0.21)_{10}) = C(0.7, 0.3) = (0.21)_{10} = 0.3$$
$$C(C(0.7, 0.7), 0.3) = C((0.49)_{10}, 0.3) = C(0.5, 0.3) = (0.15)_{10} = 0.2$$

2. $C^4_{10,5}(x, y)$ is not commutative.

$$C^4_{10,5}(0.82, 0.79) = ((0.82)_{10} \cdot (0.79)_5)_4 = (0.9 \cdot 0.8)_4 = (0.72)_4 = 0.75$$
$$C^4_{10,5}(0.79, 0.82) = ((0.79)_{10} \cdot (0.82)_5)_4 = (0.8 \cdot 1)_4 = 1$$

As previously stated, to model precision and granularity, it is reasonable to work with partitions of $[0, 1]$. In fact, in this case, the set of truth values is a finite linearly ordered set. In practical applications it happens that we change the pespective and work with finer and/or coarser partition. This is a special case studied in domain theory [1], in which one of the most fundamental questions is about the representation of a real number: a common approach to this problem is to identify each real number r with a collection of intervals whose intersection is $\{r\}$. In such a representation a smaller interval gives more information about a number than a bigger interval. So an interval I carries more information than an interval J, which we represent by writing $J \leq I$, provided that $I \subseteq J$.

Consider now the following interval extension of the connectives $C^k_{n,m}$.

Definition 2. *For naturals* $n, m, k > 0$ *and* $a \leq n, b \leq m$ *we define*

$$K^k_{n,m}\left(\left\langle \frac{a-1}{n}, \frac{a}{n} \right\rangle, \left\langle \frac{b-1}{m}, \frac{b}{m} \right\rangle\right) = \left\langle (a \cdot b)_k - \frac{1}{k}, (a \cdot b)_k \right\rangle$$

Several authors propose to model precision, uncertainty of our knowledge with the set of truth values being all closed subintervals of the real interval $[0, 1]$. In some papers the set of truth values is the set of pairs of closed intervals: the first one modeling our belief, confidence, or probability estimation, and the second modeling our doubts or disbelief on the content of information.

Our intuitive model will consider the lattice L of all closed subintervals of the interval $[0,1]$, the ordering being the "truth" ordering, see [2–4, 7]

$$\langle a,b\rangle \leq \langle c,d\rangle \qquad \text{iff} \qquad a \leq c \quad \text{and} \quad b \leq d$$

This is a complete lattice with $\bot = \langle 0,0\rangle$ and $\top = \langle 1,1\rangle$ and

$$\langle a,b\rangle \wedge \langle c,d\rangle = \langle \min(a,c), \min(b,d)\rangle$$
$$\langle a,b\rangle \vee \langle c,d\rangle = \langle \max(a,c), \max(b,d)\rangle$$

the references cited above generally use t-norms as operations acting on endpoints of intervals. We would like to extend the set of connectives to arbitrary approximations of t-norms. The computations in the probabilistic and lattice valued fuzzy logic are the same, the only difference is handling and/or ignorance of probabilistic constraints.

Example 2. Another example justifying the use of (lattices of) intervals as truth-values, together with possibly some source of inconsistency problems comes, for instance, when considering the results of polls.

Some days before the polling day for the recent elections in Hungary, the following vote expectancy data was published:

- Between 35–45% of voters will favor party 1.
- Between 45–55% of voters will favor party 2.
- Between 5–10% of voters will favor party 3.
- Between 5–10% of voters will favor party 4.

Problems with probabilistic restrictions should be seen as a sign of possible inconsistencies, just note that if the predictions were correct for all parties with values are taken at the right end of the interval of probability, then we would have a total 120% of votes!

In our approach, the computation will be processed by lattice valued truth-functions. Instead of working with probabilistic constraints as cuts we could add a new degree with a measure of violation of probabilistic constraints. In this example, the degree of inconsistency could be for instance $\frac{2}{\pi}\arctan(0.2)$ because of the sum of upper bounds of estimations was 120%.

Furthermore, these measurements usually generates inconsistencies, because the questioned group could not be representing the whole population and/or people do not answer their real interests. The outcome of the election was the following:

$$P1\text{: } 42\% \qquad P2\text{: } 41\% \qquad P3\text{: } 6\% \qquad P4\text{: } 4\%$$

in which the results for two parties were outside the predicted interval.

3 Biresiduated multi-adjoint logic programming

We would like to build our semantics of deductive logic databases on a lattice valued logic with non-commutative conjunctions and based on residuation. There

exists already the algebraic notion of biresiduated lattices [12], which seems to be a natural candidate for our model.

The preliminary concepts required to formally define the syntax of biresiduated multi-adjoint logic programs are built on those of the 'monoresiduated' multi-adjoint case [9]; to make this paper as self-contained as possible, the necessary definitions are included below.

Definition 3. *Let $\langle L, \preceq\rangle$ be a complete lattice. A* biresiduated multi-adjoint lattice $\mathcal{L}$ *is a tuple $(L, \preceq, \swarrow^1, \nwarrow_1, \&_1, \ldots, \swarrow^n, \nwarrow_n, \&_n)$ satisfying the following items:*

1. *$\langle L, \preceq\rangle$ is bounded, i.e. it has bottom and top elements;*
2. *$\top \,\&_i\, \vartheta = \vartheta \,\&_i\, \top = \vartheta$ for all $\vartheta \in L$ for $i = 1, \ldots, n$;*
3. *$(\swarrow^i, \nwarrow_i, \&_i)$ satisfies the following properties, for all $i = 1, \ldots, n$; i.e.*
   (a) *Operation $\&_i$ is increasing in both arguments,*
   (b) *Operations $\swarrow^i, \nwarrow_i$ are increasing in the first argument and decreasing in the second argument,*
   (c) *For any $x, y, z \in P$, we have that*

$$\begin{array}{lll} x \preceq y \swarrow^i z & \text{if and only if} & x \,\&_i\, z \preceq y \\ x \preceq y \nwarrow_i z & \text{if and only if} & z \,\&_i\, x \preceq y \end{array}$$

The existence of multiple pairs satisfying property 3 in the previous definition justifies the term *multi-adjoint*. The biresiduated structure comes from the fact that, for each $\&_i$ (called *adjoint conjunctor*) there exist two 'sided' *adjoint implications* denoted as $\swarrow^i$ and $\nwarrow_i$, satisfying the following properties:[1]

$$\begin{array}{lll} x \preceq y \swarrow^i z & \text{if and only if} & x \,\&_i\, z \preceq y \\ x \preceq y \nwarrow_i z & \text{if and only if} & z \,\&_i\, x \preceq y \end{array}$$

We will be working with two languages: the first one, $\mathfrak{F}$, to define the syntax of our programs, and the second one, $\mathfrak{L}$, to host the manipulation of the truth-values of the formulas in the programs. To avoid possible name-clashes, we will denote the interpretation of an operator symbol ω under $\mathfrak{L}$ as $\dot{\omega}$ (a dot on the operator), whereas ω itself will denote its interpretation under $\mathfrak{F}$.

In the sequel, we will omit the adjectives biresiduated and multi-adjoint if no confusion could arise. Furthermore, we will use the symbol $\leftarrow_i$ to denote either $\swarrow^i$ or $\nwarrow_i$, whenever the 'side' of the implication is not relevant to the case.

In the next definition we will consider a language $\mathfrak{F}$ which may contain some additional connectives, especially several aggregations, disjunctors and some additional conjunctors.

Definition 4. *A* biresiduated multi-adjoint logic program *(in short* a program*) on a language $\mathfrak{F}$ with values in a lattice $\mathfrak{L}$ is a set $\mathbb{P}$ of rules of the form $\langle A \swarrow^i \mathcal{B}, \vartheta\rangle$ or $\langle A \nwarrow_i \mathcal{B}, \vartheta\rangle$ such that:*

[1] Note that if the connective $\&_i$ turns out to be commutative, then $\swarrow^i$ and $\nwarrow_i$ must coincide.

1. *The* head *of the rule, A, is a propositional symbol;*
2. *The* body *formula, $\mathcal{B}$, is a formula of $\mathfrak{F}$ built from propositional symbols $B_1, \ldots, B_n$ ($n \geq 0$) and monotone operators (and no implications);*
3. *The* confidence factor *ϑ is an element (a truth-value) of L.*

As usual, facts *are rules with body $\top$, and a* query *(or* goal*) is a propositional symbol intended as a question $?A$ prompting the system.*

As usual, an *interpretation* is a mapping $I\colon \Pi \to L$. Note that each of these interpretations can be uniquely extended to the whole set of formulas, $\hat{I}\colon \mathfrak{F} \to L$. The set of all interpretations of the formulas defined by $\mathfrak{F}$ in is denoted $\mathcal{I}_{\mathfrak{L}}$.

The ordering $\preceq$ of the truth-values L can be easily extended to $\mathcal{I}_{\mathfrak{L}}$, which also inherits the structure of complete lattice. The minimum element of the lattice $\mathcal{I}_{\mathfrak{L}}$, which assigns $\bot$ to any propositional symbol, will be denoted $\triangle$.

A weighted rule of a biresiduated multi-adjoint logic program is satisfied whenever its truth-value is greater or equal than the confidence factor:

Definition 5.

1. *An interpretation $I \in \mathcal{I}_{\mathfrak{L}}$* satisfies *$\langle A \swarrow^i \mathcal{B}, \vartheta\rangle$ if and only if $\vartheta \&_i \hat{I}(\mathcal{B}) \preceq \hat{I}(A)$.*
2. *An interpretation $I \in \mathcal{I}_{\mathfrak{L}}$* satisfies *$\langle A \nwarrow_i \mathcal{B}, \vartheta\rangle$ if and only if $\hat{I}(\mathcal{B}) \&_i \vartheta \preceq \hat{I}(A)$.*
3. *An interpretation $I \in \mathcal{I}_{\mathfrak{L}}$ is a* model *of a program $\mathbb{P}$ iff all weighted rules in $\mathbb{P}$ are satisfied by I.*
4. *An element $\lambda \in L$ is a* correct answer *for a query $?A$ and a program $\mathbb{P}$ if for any interpretation $I \in \mathcal{I}_{\mathfrak{L}}$ which is a model of $\mathbb{P}$ we have $\lambda \preceq I(A)$.*

The immediate consequences operator, given by van Emden and Kowalski, can be easily generalised to the framework of biresiduated multi-adjoint logic programs. This operator will serve as our database query evaluation operator.

Definition 6. *Let $\mathbb{P}$ be a program, the* immediate consequences operator *$T_{\mathbb{P}}$ maps interpretations to interpretations, and for an interpretation I and propositional variable A is defined by*

$$T_{\mathbb{P}}(I)(A) = \sup\left\{\{\vartheta \mathbin{\dot{\&}_i} \hat{I}(\mathcal{B}) \mid \langle A \swarrow^i \mathcal{B}, \vartheta\rangle \in \mathbb{P}\} \cup \{\hat{I}(\mathcal{B}) \mathbin{\dot{\&}_i} \vartheta \mid \langle A \nwarrow_i \mathcal{B}, \vartheta\rangle \in \mathbb{P}\}\right\}$$

The semantics of a biresiduated multi-adjoint logic program is characterised by the post-fixpoints of $T_{\mathbb{P}}$; that is, an interpretation I of $\mathcal{I}_{\mathfrak{L}}$ is a model of a program $\mathbb{P}$ iff $T_{\mathbb{P}}(I) \sqsubseteq I$.

Our operator is a generalisation of that in [7] (although they work with pairs of intervals). Their operator is continuous because they consider only connectives generated by Łukasiewicz, product and Gödel connectives acting on the endpoint of intervals. In general, when working with aggregations and constant approximations this is not always the case. We will justify this claim by using the example below, modified from [5]. A detailed description of this is out of the scope of this paper. More details on relations between annotated and fuzzy programs are in [6].

Example 3. In the language of lattice valued generalized annotated programs with restricted semantics consider the following program:

$$p\colon \langle 0,0\rangle \leftarrow \qquad p\colon \left\langle 0, \frac{1+2y}{4}\right\rangle \leftarrow p\colon \langle 0,y\rangle \qquad q\colon \left\langle 0,\frac{1}{2}\right\rangle \leftarrow p\colon \left\langle 0,\frac{1}{2}\right\rangle$$

The $T_{\mathbb{P}}$ operator iterates as follows:

$$T_{\mathbb{P}}{}^1(p) = \left\langle 0,\frac{1}{4}\right\rangle \ , \ T_{\mathbb{P}}{}^2(p) = \left\langle 0,\frac{3}{8}\right\rangle, \ldots, T_{\mathbb{P}}{}^n(p) = \left\langle 0,\frac{2^{n-1}-1}{2^n}\right\rangle, \ldots$$

$$\ldots, T_{\mathbb{P}}{}^\omega(p) = \left\langle 0,\frac{1}{2}\right\rangle, T_{\mathbb{P}}{}^{\omega+1}(p) = \left\langle 0,\frac{1}{2}\right\rangle$$

$$T_{\mathbb{P}}{}^1(q) = \langle 0,0\rangle = T_{\mathbb{P}}{}^2(q) = T_{\mathbb{P}}{}^3(q) = \cdots = T_{\mathbb{P}}{}^\omega(q), T_{\mathbb{P}}{}^{\omega+1}(q) = \left\langle 0,\frac{1}{2}\right\rangle$$

so the operator is not continuous. Indeed, there is an upward directed set of interpretations of L for which $T_{\mathbb{P}}(\bigvee X) \not\leqslant \bigvee\{T_{\mathbb{P}}(I) \mid I \in X\}$. Namely, take $X = \{I_n \mid n \in \omega\}$ where $I_n\colon \{p,q\} \to L$ is defined as

$$I_n(p) = \left\langle 0,\frac{2^{n-1}-1}{2^n}\right\rangle \qquad I_n(q) = \langle 0,0\rangle$$

then we have

$$\bigvee X(p) = \left\langle 0,\frac{1}{2}\right\rangle \quad T_{\mathbb{P}}(\bigvee X)(p) = \left\langle 0,\frac{1}{2}\right\rangle = \bigvee\{T_{\mathbb{P}}(I_n)(p) \mid n \in \omega\} = \left\langle 0,\frac{1}{2}\right\rangle$$

$$\bigvee X(q) = \langle 0,0\rangle \quad T_{\mathbb{P}}(\bigvee X)(q) = \left\langle 0,\frac{1}{2}\right\rangle \not\leqslant \bigvee\{T_{\mathbb{P}}(I_n)(q) \mid n \in \omega\} = \langle 0,0\rangle$$

The program of this example can be translated to

$$(p \leftarrow @_1(p), \top) \qquad (q \leftarrow @_2(p), \top)$$

where the truth-function $\dot{@}_i\colon L \to L$ are defined as

$$\dot{@}_1(\langle x,y\rangle) = \left\langle 0,\frac{1+2y}{4}\right\rangle \qquad \dot{@}_2(\langle x,y\rangle) = \begin{cases} \langle 0,\frac{1}{2}\rangle & \text{if } \langle x,y\rangle \geq \langle 0,\frac{1}{2}\rangle \\ \langle 0,0\rangle & \text{if } \langle x,y\rangle \not\geqslant \langle 0,\frac{1}{2}\rangle \end{cases}$$

The jump operator $\dot{@}_2$ is not lattice continuous: simply consider the following upward directed set

$$A \subset L \qquad A = \left\{ \left\langle 0,\frac{2^{n-1}-1}{2^n}\right\rangle \mid n \in \omega \right\}$$

Note that $\bigvee A = \langle 0,\frac{1}{2}\rangle$, therefore $\dot{@}_2(\bigvee A) = \langle 0,\frac{1}{2}\rangle$. Now, for every element of A, $\dot{@}_2(\left\langle 0,\frac{2^{n-1}-1}{2^n}\right\rangle) = \langle 0,0\rangle$, therefore

$$\dot{@}_2\left(\bigvee A\right) = \left\langle 0,\frac{1}{2}\right\rangle \not\leqslant \bigvee\left\{\dot{@}_2\left(\left\langle 0,\frac{2^{n-1}-1}{2^n}\right\rangle\right) \mid n \in \omega\right\} = \bigvee\{\bot\} = \bot$$

A possible solution to this, as proposed in [6], can be obtained by replacing constant annotations by annotation terms as functions, in order to work with lattice continuous connectives in the body.

3.1 Procedural semantics

It can be shown that the $T_{\mathbb{P}}$ operator is continuous under very general hypotheses (the proof in [8] can be easily generalised to this framework), therefore the least model can be reached in at most countably many iterations. Now, it is worth to define a procedural semantics which allows us to actually construct the answer to a query against a given program.

Our computational model will work in an extended language in which we allow to use jointly propositional symbols and elements of the lattice as basic formulas (these 'mixed' formulas will be called *extended formulas*). Given a query, and by using the rules below, will provide a lower bound of the value of A under any model of the program. Intuitively, the computation proceeds by, somehow, substituting propositional symbols by lower bounds of their truth-value until, eventually, an extended formula with no propositional symbol is obtained, which will be interpreted in the lattice to get the computed answer.

Given a program $\mathbb{P}$, we define the following admissible rules for transforming any extended formula.

Definition 7. Admissible rules *are defined as follows:*

R1a *Substitute an atom A in an extended formula by $(\vartheta \,\&_i\, \mathcal{B})$ whenever there exists a rule $\langle A \swarrow^i \mathcal{B}, \vartheta\rangle$ in $\mathbb{P}$.*

R1b *Substitute an atom A in an extended formula by $(\mathcal{B} \,\&_i\, \vartheta)$ whenever there exists a rule $\langle A \nwarrow_i \mathcal{B}, \vartheta\rangle$ in $\mathbb{P}$.*

R2 *Substitute an atom A in an extended formula by $\bot$.*

R3 *Substitute an atom A in an extended formula by ϑ whenever there exists a fact $\langle A \swarrow^i \top, \vartheta\rangle$ or $\langle A \nwarrow_i \top, \vartheta\rangle$ in $\mathbb{P}$.*

Note that if an extended formula turns out to have no propositional symbols, then it can be directly interpreted in the computation as an element in $\mathfrak{L}$, rather than like a formula. This justifies the following definition of *computed answer*.

Definition 8. *Let $\mathbb{P}$ be a program in a language interpreted on a lattice $\mathcal{L}$ and let $?A$ be a goal. An element $\lambda \in L$ is said to be a* computed answer *if there is a sequence $G_0, \ldots, G_{n+1}$ such that*

1. *$G_0 = A$ and $G_{n+1} = @(r_1, \ldots, r_m)$ where[2] $r_i \in L$ for all $i = 1, \ldots m$, and $\lambda = \dot{@}(r_1, \ldots, r_m)$.*
2. *Every G_i, for $i = 1, \ldots, n$, is an extended formula.*
3. *Every G_{i+1} is inferred from G_i by exactly one of the admissible rules.*

[2] Here the r_i represent all the variables occurring in the G_{n+1}. Therefore we are abusing the notation, for @ represents the composition, as functions in the lattice, of all the operators inserted by rules R1a and R1b.

Note that our procedural semantics, instead of being refutation-based (this is not possible, since negation is not allowed in our approach), is oriented to obtaining a bound of the optimal correct answer of the query.

3.2 Greatest Answers and Reductants

The definition of correct answer is not entirely satisfactory in that $\bot$ is always a correct answer. Actually, we should be interested in the greatest confidence factor we can assume on the query, consistently with the information in the program, instead of in the set of its lower bounds.

By proving that models are post-fixpoint of $T_{\mathbb{P}}$ together with the Knaster-Tarski theorem, it is possible to obtain, the following result:

Theorem 1. *Given a complete lattice L, a program $\mathbb{P}$ and a propositional symbol A, we have that $T^{\omega}_{\mathbb{P}}(\triangle)(A)$ is the greatest correct answer.*

Regarding the computation of the greatest correct answer, it might well be the case that for some lattices, our procedural semantics cannot compute the greatest correct answer. We can cope with this problem by generalising the concept of reductant [5]; any rule $\langle A \leftarrow_{j_i} \mathcal{D}_i, \vartheta_i \rangle$ contributes with a value such as either $\vartheta_i \,\dot{\&}_i\, b_i$ or $b_i \,\dot{\&}_i\, \vartheta_i$ in the calculation of the lower bound for the truth-value of A, thus we would like to have the possibility of reaching the supremum of all the contributions, in the computational model, in a single step.

Definition 9. *Let $\mathbb{P}$ be a program; assume that the set of rules in $\mathbb{P}$ with head A can be written as $\langle A \swarrow^{i_j} \mathcal{B}_j, \vartheta_j \rangle$ for $j = 1, \ldots, n$, and $\langle A \nwarrow_{k_l} \mathcal{C}_l, \theta_l \rangle$ for $l = 1, \ldots, m$, and contains at least a proper rule; a* reductant for A *is any rule*

$$\langle A \swarrow \dot{@}(\mathcal{B}_1, \ldots, \mathcal{B}_n, \mathcal{C}_1, \ldots, \mathcal{C}_m), \top \rangle$$

where $\swarrow$ is any implication symbol and the operator $@$ is defined as

$$\dot{@}(b_1, \ldots, b_n, c_1, \ldots, c_m) = \sup\{\vartheta_1 \&_{i_1} b_1, \ldots, \vartheta_n \&_{i_n} b_n,\ c_1 \&_{k_1} \theta_1, \ldots, c_m \&_{k_m} \theta_m\}$$

If there were just facts with head A, but no proper rule, then the expression above does not give a well-formed formula. In this case, the reductant is defined to be a fact which aggregates all the knowledge about A, that is,

$$\langle A \swarrow \top, \sup\{\vartheta_1, \ldots, \vartheta_n\} \rangle$$

As a consequence of the definition, and the boundary conditions in the definition of biresiduated multi-adjoint lattice, the choice of the implication to represent the corresponding reductant is irrelevant for the computational model. Therefore, in the following, we will assume that our language has a distinguished implication to be selected in the construction of reductants, leading to the so-called *canonical reductants*.

It will be interesting to consider only programs which contain all its reductants; since its set of models is not modified, we can assume that a program contains all its reductants.

3.3 Completeness results

The two quasi-completeness theorems given in [9], can be extended to this more general framework as follows:

Theorem 2. *For every correct answer $\lambda \in L$ for a program $\mathbb{P}$ and a query $?A$, there exists a chain of elements λ_n such that $\lambda \preceq \sup \lambda_n$, such that for arbitrary n_0 there exists a computed answer δ such that $\lambda_{n_0} \preceq \delta$.*

A more standard statement of the quasi-completeness result can be obtained under the assumption of the following property:

Definition 10. *A lattice L is said to satisfy the* supremum property *if for all directed set $X \subset L$ and for all ε we have that if $\varepsilon < \sup X$ then there exists $\delta \in X$ such that $\varepsilon < \delta \leq \sup X$.*

Theorem 3 below states that any correct answer can be approximated up to any lower bound.

Theorem 3. *Assume L has the supremum property, then for every correct answer $\lambda \in L$ for a program $\mathbb{P}$ and a query $?A$, and arbitrary $\varepsilon \prec \lambda$ there exists a computed answer δ such that $\varepsilon \prec \delta$.*

4 A Non-Commutative Expressive Deductive Data Model

In this section a relational algebra is defined for the semantics above. More expressiveness is obtained because it is built on a richer set of connectives. Furthermore, as its fixpoint semantics is built on a continuous operator, the computed answer to queries is reached in at most countably many steps and, thus, no transfinite computation phenomena appears here. Moreover, our semantics is replacing the constraint resolution semantics of [5] by a more effective one calculating the best possible answer. The underlying idea is to generalize the fuzzy relational algebra from [11] to the lattice-case and give a simplified presentation. The definitions of operations do not change substantially; what changes is the continuity problem for lattices.

The semantics of rules (without negation) is given by expressing them in a positive fuzzy relational algebra (as an extension of the classical positive relational algebra) which consists of selection, natural join, classical projection and union. The new lattice-valued relations are described as classical relations with one additional attribute, TV, for the truth value. The following notation will be used: $\mathcal{R}$ will denote the set of records, R will denote relations, and r will denote predicates.

Selection Expressions in selections are allowed to use TV. The result of such a selection, e.g. $\sigma_{TV \geq t}(r)$, consists of those tuples from $\mathcal{R}$ which have the value of the truth value attribute at least t. This is the only extension of classical selection.

Join The operation of a natural join is in our relational algebra defined wrt to crisp equality and an aggregation operator which tells us how to calculate the truth value degree of a tuple in the join. Assume we have relations $R_1, ..., R_n$ which evaluate predicates $r_1, ..., r_n$. Moreover, assume that the first k-attributes in each R_i are the same and $(b_1, \ldots, b_k, b^i_{k+1}, \ldots, b^i_{m_i}, \beta^i) \in R_i$ then $(b_1, \ldots, b_k, b^1_{k+1}, \ldots, b^1_{m_1}, \ldots, b^n_{k+1}, \ldots, b^n_{m_n}, \dot{@}(\beta^1, \ldots, \beta^n))$ is in the relation $\bowtie_{@} (R_1, ..., R_n)$, that is the truth value attributes in our join do not behave as in classical join, they disappear, forwarding the respective truth values to the new aggregated truth value. This way it is possible to obtain the following

Theorem 4. *The operation $\bowtie_{@} (R_1, ..., R_n)$ is a sound and complete (wrt the satisfaction of fuzzy logic) evaluation of $@(r_1, ..., r_n)$ provided $R_1, ..., R_n$ are evaluations of $r_1, ..., r_n$.*

Projection In [6] we have presented several transformations of generalised annotated programs and fuzzy logic programs (see also Example 3). Motivated by this, observe that a multi-adjoint rule

$$\langle H \leftarrow_i @(\mathcal{B}_1, \ldots, \mathcal{B}_n), \vartheta \rangle$$

is semanticaly equivalent to

$$\langle H \leftarrow \vartheta \,\&_i\, @(\mathcal{B}_1, \ldots, \mathcal{B}_n), \top \rangle$$

where $\&_i$ is the residuated conjunctor of the implication $\leftarrow_i$, and $\leftarrow$ is a fixed implication, having residuated conjunctor $\&$ satisfying $\top \,\&\, b = b$.

Note that

$$y \,\&_i\, \dot{@}(x_1, \ldots, x_n) = \dot{@}'(y, x_1, \ldots, x_n)$$

is also an aggregation in our lattice.

So all Datalog programs can be translated to programs with all rules having truth value $\top$ and using only one fixed implication with above property. All the richness of our multiadjoint biresidual programs with truth vlaues of rules is now hidden in a richer set of aggregations used to evaluate bodies of rules.

This enables us to work with the classical projection

Having the evaluation R of the whole body of the rectified rule we use the classical projection to get the evaluation of the (IDB) predicate $H(X_1, ..., X_h)$, So for $(b_1, ..., b_m, \beta) \in R$ the projection consists of tuples

$$(b_1, ..., b_h, \beta) \in \Pi_{X_1,...,X_h}(R)$$

assuming the variables selected where at the beginning of the enumeration of the tuple.

Here is a substantial difference with some models of probabilistic databases which calculate the truth value of projection as a measure of a union of events (events being all records in R which equal on attributes $X_1, ..., X_h$).

Union For the case there are more rules with the same predicate in the head we have to guarantee that different witnesses to the same conclusion are not lost. In the case of different rules it means that these witnesses are aggregating together with the lattice join operator. This is determined by our semantics saying the graded statement is true in an interpretation if its truth value is bigger than or equal than the grade given in the syntactical part. Hence they unify wrt the union which calculates the truth value as a lattice join (recall the need of reductants here). Assume $R_1, \ldots, R_n$ are relations with same attributes and $(b_1, \ldots, b_k, \beta_i) \in R_i$ then

$$(b_1, \ldots, b_k, \bigvee\{\beta_1, \ldots, \beta_n)\} \in \bigcup_{i=1}^{n} R_i.$$

Theorem 5. *The operation*

$$\bigcup_{j=1}^{k} \Pi_{X_1,\ldots,X_h} \left(\sigma_{X_h=a_h^j} \left(\ldots \sigma_{X_1=a_1^j} \left(\bowtie_{@} (R_1^j, \ldots, R_{n_j}^j)\right) \ldots\right)\right)$$

is a sound and complete evaluation of the $H(X_1, \ldots, X_h)$ (wrt the satisfaction of our logic) wrt all rules with head H ranging from $j = 1$ to k

$$\langle H(X_1, \ldots, X_h) \longleftarrow X_1 = a_1^j \wedge \ldots \wedge X_h = a_h^j \wedge @(B_1^j, \ldots, B_{n_j}^j), \top \rangle$$

provided $R_1^j, ..., R_{n_j}^j$ were evaluations of $B_1^j, ..., B_{n_j}^j$.

Now the expressive power of our lattice valued relational algebra is given by the $T_{\mathbb{P}}$ operator and its fixpoint. The iteration of $T_{\mathbb{P}}$ is used to prove that the expressive power of lattice Datalog is the same as that of the relational algebra.

So if our declarative and procedural semantics is sound and complete for ground queries, it will be also for queries with free variables. It suffices for every unbound variable we can extend our language by a new constant which will be evaluated by infimum of all truth values ranging through this variable. This is again a model of our fuzzy theory and the result holds. So our systems allows recursion and the relational algebra calculates it correctly.

Theorem 6. *Every query over the knowledge base represented by a lattice Datalog program (possibly with recursion, without negation) can be evaluated up to any arbitrary accuracy, by iterating described operations of lattice relational algebra.*

Every correct answer to a query is obtained by finite iteration of the $T_{\mathbb{P}}$ operator with any prescribed precision. This operator evaluates relations in interpretation in a same way as our relational algebra does.

5 Conclusions and future work

Our motivation comes from the usage to model some subjective interpretation of human preferences in a granulated way, e.g. by truth value set consisting

of intervals. When restricting multi-valued connectives to intervals we observe they need not be neither associative nor commutative. To cover all these phenomena a general framework of multi-adjoint lattice valued logic programming which allows rather general set of connectives in the bodies has been introduced (including some non-commutative ones).

A procedural semantics for this framework of multi-adjoint biresiduated logic programs has been presented and a quasi-completeness theorem proved. The generality of the framework allows for better expresiveness, as well as a means for handling imprecission in databases. The computational model, especially the corresponding generalisation to our lattice-valued case of the $T_{\mathbb{P}}$ operator, is used to construct a positive relational algebra.

In the final section we prove that the expressive power of our relational algebra is the same as that of our Datalog programs (up to some approximation of the best answer).

References

1. K. Ciesielski, R. Flagg, and R. Kopperman. Polish spaces, computable approximations, and bitopological spaces. *Topology and applications*, 119(3):241–256, 2002.
2. C.V. Damásio and L. Moniz Pereira. Monotonic and residuated logic programs. In *Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU'01*, pages 748–759. Lect. Notes in Artificial Intelligence, 2143, 2001.
3. A. Dekhtyar and V. S. Subrahmanian. Hybrid probabilistic programs. *J. of Logic Programming*, 43:187–250, 2000.
4. T. Eiter, T. Lukasiewicz, and M. Walter. A data model and algebra for probabilistic complex values. *Annals of Mathematics and Artificial Intelligence*, 33(2–4):205–252, 2001.
5. M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *J. of Logic Programming*, 12:335–367, 1992.
6. S. Krajči, R. Lencses, and P. Vojtáš. A comparison of fuzzy and annotated logic programming. *Fuzzy Sets and Systems*, 2002. Submitted.
7. L.V.S. Lakshmanan and F. Sadri. On a theory of probabilistic deductive databases. *Theory and Practice of Logic Programming*, 1(1):5–42, 2001.
8. J. Medina, M. Ojeda-Aciego, and P. Vojtáš. Multi-adjoint logic programming with continuous semantics. In *Logic Programming and Non-Monotonic Reasoning, LPNMR'01*, pages 351–364. Lect. Notes in Artificial Intelligence 2173, 2001.
9. J. Medina, M. Ojeda-Aciego, and P. Vojtáš. A procedural semantics for multi-adjoint logic programming. In *Progress in Artificial Intelligence, EPIA'01*, pages 290–297. Lect. Notes in Artificial Intelligence 2258, 2001.
10. E. Naito, J. Ozawa, I. Hayashi, and N. Wakami. A proposal of a fuzzy connective with learning function. In P. Bosc and J. Kaczprzyk, editors, *Fuzziness Database Management Systems*, pages 345–364. Physica Verlag, 1995.
11. J. Pokorný and P. Vojtáš. A data model for flexible querying. In *Advances in Databases and Information Systems, ADBIS'01*, pages 280–293. Lect. Notes in Computer Science 2151, 2001.
12. C.J. van Alten. Representable biresiduated lattices. *J. Algebra*, 247:672–691, 2002.
13. P. Vojtáš and L. Paulík. Soundness and completeness of non-classical extended SLD-resolution. In *Extensions of Logic Programming, ELP'96*, pages 289–301. Lect. Notes in Comp. Sci. 1050, 1996.