A Multi-Adjoint Logic Approach to Abductive Reasoning

Jesús Medina,[1] Manuel Ojeda-Aciego,[1] and Peter Vojtáš[2]

[1] Dept. Matemática Aplicada. Universidad de Málaga.***
{jmedina,aciego}@ctima.uma.es
[2] Dept. Mathematical Informatics. P.J. Šafárik University.†
vojtas@kosice.upjs.sk

Abstract. Multi-adjoint logic programs has been recently introduced [9, 10] as a generalization of monotonic logic programs [2, 3], in that simultaneous use of several implications in the rules and rather general connectives in the bodies are allowed.
This paper discusses abductive reasoning—that is, reasoning in which explanatory hypotheses are formed and evaluated. To model uncertainty in human cognition and real world applications; we use multi-adjoint logic programming to introduce and study a model of abduction problem.

1 Introduction

Broadly speaking, abduction aims at finding explanations for, or causes of, observed phenomena or facts; it is inference to the best explanation, a pattern of reasoning that occurs in such diverse places as medical diagnosis, scientific theory formation, accident investigation, language understanding, and jury deliberation. More formally, abduction is an inference mechanism where given a knowledge base and some observations, the reasoner tries to find hypotheses which together with the knowledge base explain the observations. Reasoning based on such an inference mechanism is referred to as abductive reasoning.

Abductive reasoning has been recognized as an important form of reasoning with incomplete information that is appropriate for many problems in Artificial Intelligence. These problems include updates in databases, belief revision, planning, diagnosis, natural language understanding, default reasoning, user modelling and, in general, problems requiring reasoning with incomplete information.

The purpose of this work is to provide a theoretical framework for abduction in multi-adjoint logic programming [9]. The special feature of multi-adjoint logic programs is that it is possible to use a number of different implications in the rules of our programs. Specifically, the language and semantics of monotonic logic programs are generalized in order to encompass more complex rules. For

*** Partially supported by Spanish DGI project BFM2000-1054-C02-02 and Junta de Andalucía project TIC-115.
† and partially supported by Slovak project VEGA 1/7557/20

simplicity in the presentation, only the propositional (ground) case will be considered. A whole class of abduction problems with uncertainty expressed within the language of multiadjoint programs can be solved by our method.

A general theory of logic programming which allows the simultaneous use of different implications in the rules and rather general connectives in the bodies is presented in [9], where models of these programs are proved to be post-fixpoints of the immediate consequences operator, which turns out to be monotonic under very general hypotheses. In addition, the continuity of the immediate consequences operator is studied, and some sufficient conditions for its continuity are obtained. A procedural semantics, under these conditions, for multi-adjoint logic programs, together its completeness result was given in [10].

The structure of the paper is as follows: In Section 2, the preliminary definitions are introduced; later, in Section 3, the syntax and semantics of multi-adjoint logic programs are given, and the results about the continuity of the immediate consequences operator are presented. In Section 4, the procedural semantics of multi-adjoint logic programs is defined and the completeness results are stated. Section 5 is the main part of this paper. We give definitions of the abduction problem and of correct and computed explanations. We prove soundness and completeness of our abduction semantics. The computation of the cheapest explanation wrt a price function can be implemented, in determined lattices, by a logic programming computation followed by a linear programming optimization. The paper finishes with some conclusions and pointers to future work.

2 Preliminary definitions

In order to make this paper as self-contained as possible, the preliminary definitions required to formally define multi-adjoint logic programs are given in this section, which contains the approach given in [1, 3, 9].

We assume the reader is familiar to constructions and terminology of universal algebra such as graded set, Ω-algebra and subalgebra of an Ω-algebra, which are used to define formally the syntax and the semantics of the languages we will deal with.

The main concept we use in this section is that of *adjoint pair*, firstly introduced in a logical context by Pavelka [12], who interpreted the poset structure of the set of truth-values as a category, and the relation between the connectives of implication and conjunction as functors in this category. The result turned out to be another example of the well-known concept of adjunction, introduced by Kan in the general setting of category theory in 1950 (see also the notion of a relative pseudo-complement in lattice theory e.g. in Rasiowa and Sikorski's 'Mathematics of metamathematics' (1968)).

Definition 1 (Adjoint pair). *Let $\langle P, \preceq\rangle$ be a partially ordered set and $(\leftarrow, \&)$ a pair of binary operations in P such that:*

(a1) Operation & is increasing in both arguments, i.e. if $x_1, x_2, y \in P$ such that $x_1 \preceq x_2$ then $(x_1 \& y) \preceq (x_2 \& y)$ and $(y \& x_1) \preceq (y \& x_2)$;

(a2) Operation $\leftarrow$ is increasing in the first argument (the consequent) and decreasing in the second argument (the antecedent), i.e. if $x_1, x_2, y \in P$ such that $x_1 \preceq x_2$ then $(x_1 \leftarrow y) \preceq (x_2 \leftarrow y)$ and $(y \leftarrow x_2) \preceq (y \leftarrow x_1)$;

(a3) For any $x, y, z \in P$, we have that $x \preceq (y \leftarrow z)$ holds if and only if $(x \mathbin{\&} z) \preceq y$ holds.

Then we say that $(\leftarrow, \&)$ forms an adjoint pair *in $\langle P, \preceq \rangle$.*

The property (a3) corresponds to the categorical adjointness; and can be adequately interpreted in terms of multiple-valued inference as both the assertion that the truth-value of $y \leftarrow z$ is the maximal x satisfying $x \mathbin{\&} z \preceq_P y$, and the validity of a generalized modus ponens rule [5].

Extending the results in [1, 3, 13, 14] to a more general setting, in which different implications (Łukasiewicz, Gödel, product) and thus, several modus ponens-like inference rules are used, naturally leads to considering several *adjoint pairs* in the lattice. More formally,

Definition 2 (Multi-Adjoint Lattice). *Let $\langle L, \preceq \rangle$ be a complete lattice. A* multi-adjoint lattice *$\mathcal{L}$ is a tuple $(L, \preceq, \leftarrow_1, \&_1, \dots, \leftarrow_n, \&_n)$ satisfying the following items:*

(l1) $\langle L, \preceq \rangle$ is bounded, i.e. it has bottom ($\bot$) and top ($\top$) elements;
(l2) $(\leftarrow_i, \&_i)$ is an adjoint pair in $\langle L, \preceq \rangle$ for $i = 1, \dots, n$;
(l3) $\top \mathbin{\&_i} \vartheta = \vartheta \mathbin{\&_i} \top = \vartheta$ for all $\vartheta \in L$ for $i = 1, \dots, n$.

Remark 1. Note that residuated lattices are a special case of multi-adjoint lattice, in which the underlying poset has a complete lattice structure, has monoidal structure wrt $\otimes$ and $\top$, and only one adjoint pair is present.

From the point of view of expressiveness, it is interesting to allow extra operators to be involved with the operators in the multi-adjoint lattice. The structure which captures this possibility is that of a multi-adjoint algebra.

Definition 3 (Multi-Adjoint *Ω*-Algebra). *Let Ω be a graded set containing operators $\leftarrow_i$ and $\&_i$ for $i = 1, \dots, n$ and possibly some extra operators, and let $\mathfrak{L} = (L, I)$ be an Ω-algebra whose carrier set L is a complete lattice under $\preceq$.*

We say that $\mathfrak{L}$ is a multi-adjoint *Ω*-algebra *with respect to the pairs $(\leftarrow_i, \&_i)$ for $i = 1, \dots, n$ if $\mathcal{L} = (L, \preceq, I(\leftarrow_1), I(\&_1), \dots, I(\leftarrow_n), I(\&_n))$ is a multi-adjoint lattice.*

In practice,the extra operators will be assumed to be either conjunctors or disjunctors or aggregators.

Example 1. Consider $\Omega = \{\leftarrow_P, \&_P, \leftarrow_G, \&_G, \wedge_L, @\}$, the real unit interval $U = [0, 1]$ with its lattice structure, and the interpretation function I defined as:

$$I(\leftarrow_P)(x, y) = \min(1, x/y) \qquad I(\&_P)(x, y) = x \cdot y$$

$$I(\leftarrow_G)(x, y) = \begin{cases} 1 & \text{if } y \leq x \\ x & \text{otherwise} \end{cases} \qquad I(\&_G)(x, y) = \min(x, y)$$

$$I(@)(x, y, z) = \tfrac{1}{6}(x + 2y + 3z) \qquad I(\wedge_L)(x, y) = \max(0, x + y - 1)$$

that is, connectives are interpreted as product and Gödel connectives, a weighted sum and Łukasiewicz conjunction; then $\langle U, I\rangle$ is a multi-adjoint Ω-algebra with one aggregator and one additional conjunctor (denoted $\wedge_L$ to make explicit that its adjoint implicator is not in the language). $\square$

The syntax of the propositional languages we will work with will be defined by using the concept of Ω-algebra. To begin with, the concept of alphabet of the language is introduced below.

Definition 4 (Alphabet). *Let Ω be a graded set, and Π a countably infinite set. The* alphabet $A_{\Omega,\Pi}$ *associated to Ω and Π is defined to be the disjoint union $\Omega \cup \Pi \cup S$, where S is the set of auxiliary symbols "(", ")" and ",".*

In the following, we will use only A_Ω to designate an alphabet, for deleting the reference to Π cannot lead to confusion.

Definition 5 (Expressions). *Given a graded set Ω and alphabet A_Ω. The Ω-algebra $\mathfrak{E} = \langle {A_\Omega}^*, I\rangle$ of* expressions *is defined as follows:*

1. *The carrier ${A_\Omega}^*$ is the set of strings over A_Ω.*
2. *The interpretation function I satisfies the following conditions for strings $a_1, \dots, a_n$ in ${A_\Omega}^*$:*
   - *$c_{\mathfrak{E}} = c$, where c is a constant operation ($c \in \Omega_0$).*
   - *$\omega_{\mathfrak{E}}(a_1) = \omega\, a_1$, where ω is an unary operation ($\omega \in \Omega_1$).*
   - *$\omega_{\mathfrak{E}}(a_1, a_2) = (a_1 \omega\, a_2)$, where ω is a binary operation ($\omega \in \Omega_2$).*
   - *$\omega_{\mathfrak{E}}(a_1, \dots, a_n) = \omega(a_1, \dots, a_n)$, where ω is a n-ary operation ($\omega \in \Omega_n$) and $n > 2$.*

Note that a expression is only a string of letters of the alphabet, that is, it needn't be a well-formed formula. Actually, the well-formed formulas is the subset of the set of expressions defined as follows:

Definition 6 (Well-formed formulas). *Let Ω be a graded set, Π a countable set of propositional symbols and $\mathfrak{E}$ the algebra of expressions corresponding to the alphabet $A_{\Omega,\Pi}$. The* well-formed formulas *(in short, formulas) generated by Ω over Π is the least subalgebra $\mathfrak{F}$ of the algebra of expressions $\mathfrak{E}$ containing Π.*

The set of formulas, that is the carrier of $\mathfrak{F}$, will be denoted F_Ω. It is well-known that least subalgebras can be defined as an inductive closure, and it is not difficult to check that it is freely generated, therefore it satisfies the unique homomorphic extension theorem.

3 Syntax and Semantics of Multi-Adjoint Logic Programs

Multi-adjoint logic programs are constructed from the abstract syntax induced by a multi-adjoint algebra on a set of propositional symbols. Specifically, we will consider a multi-adjoint Ω-algebra $\mathfrak{L}$ whose extra operators are either conjunctors, denoted $\wedge_1, \dots, \wedge_k$, or disjunctors, denoted $\vee_1, \dots, \vee_l$, or aggregators, denoted $@_1, \dots, @_m$. (This algebra will host the manipulation the truth-values of the formulas in our programs.)

In addition, let Π be a set of propositional symbols and the corresponding algebra of formulas $\mathfrak{F}$ freely generated from Π by the operators in Ω. (This algebra will be used to define the syntax of a propositional language.)

Remark 2. As we are working with two Ω-algebras, and to discharge the notation, we introduce a special notation to clarify which algebra an operator belongs to, instead of continuously using either $\omega_{\mathfrak{L}}$ or $\omega_{\mathfrak{F}}$. Let ω be an operator symbol in Ω, its interpretation under $\mathfrak{L}$ is denoted $\dot{\omega}$ (a dot on the operator), whereas ω itself will denote $\omega_{\mathfrak{F}}$ when there is no risk of confusion.

The definition of multi-adjoint logic program is given, as usual, as a set of rules and facts. The particular syntax of these rules and facts is given below:

Definition 7 (Multi-Adjoint Logic Programs). *A* multi-adjoint logic program *is a set $\mathbb{P}$ of rules of the form $\langle (A \leftarrow_i \mathcal{B}), \vartheta \rangle$ such that:*

1. *The* rule $(A \leftarrow_i \mathcal{B})$ *is a formula of* $\mathfrak{F}$*;*
2. *The* confidence factor ϑ *is an element (a truth-value) of* L*;*
3. *The* head *of the rule A is a propositional symbol of Π.*
4. *The* body *formula $\mathcal{B}$ is a formula of $\mathfrak{F}$ built from propositional symbols $B_1, \dots, B_n$ $(n \geq 0)$ by the use of conjunctors $\&_1, \dots, \&_n$ and $\wedge_1, \dots, \wedge_k$, disjunctors $\vee_1, \dots, \vee_l$ and aggregators $@_1, \dots, @_m$.*
5. Facts *are rules with body* $\top$*.*
6. *A* query *(or* goal*) is a propositional symbol intended as a question ?A prompting the system.*

Note that an arbitrary composition of conjunctors, disjunctors and aggregators is also an aggregator.

Sometimes, we will represent the above pair as $A \overset{\vartheta}{\leftarrow}_i @[B_1, \dots, B_n]$, where[1] $B_1, \dots, B_n$ are the propositional variables occurring in the body and @ is the aggregator obtained as a composition.

Example 2. The following program $\mathbb{P}$, where the subscripts G, L, P on the connectives mean Gödel, Łukasiewicz and product connectives, is an example of a [0,1]-valued multi-adjoint logic program consisting of five rules and three facts.

$$\texttt{high\_fuel\_consumption} \overset{0.8}{\leftarrow}_G \texttt{rich\_mixture} \wedge_L \texttt{low\_oil} \tag{1}$$
$$\texttt{overheating} \overset{0.5}{\leftarrow}_P \texttt{low\_oil} \tag{2}$$
$$\texttt{noisy\_behaviour} \overset{0.8}{\leftarrow}_P \texttt{rich\_mixture} \tag{3}$$
$$\texttt{overheating} \overset{0.9}{\leftarrow}_L \texttt{low\_water} \tag{4}$$
$$\texttt{noisy\_behaviour} \overset{1}{\leftarrow}_P \texttt{low\_oil} \tag{5}$$
$$\texttt{low\_oil} \overset{0.2}{\leftarrow}_P \tag{6}$$

[1] Note the use of square brackets.

$$\texttt{low\_water} \overset{0.2}{\longleftarrow}_P \tag{7}$$

$$\texttt{rich\_mixture} \overset{0.5}{\longleftarrow}_P \tag{8}$$

This program is intended to represent some general knowledge about the behaviour of a car.

Definition 8 (Interpretation). *An* interpretation *is a mapping $I\colon \Pi \to L$. The set of all interpretations of the formulas defined by the Ω-algebra $\mathfrak{F}$ in the Ω-algebra $\mathfrak{L}$ is denoted $\mathcal{I}_{\mathfrak{L}}$.*

Note that by the unique homomorphic extension theorem, each of these interpretations can be uniquely extended to the whole set of formulas F_Ω.

The ordering $\preceq$ of the truth-values L can be easily extended to the set of interpretations as usual:

Definition 9 (Lattice of interpretations). *Consider two interpretations $I_1, I_2 \in \mathcal{I}_{\mathfrak{L}}$. Then, $\langle \mathcal{I}_{\mathfrak{L}}, \sqsubseteq \rangle$ is a complete lattice where $I_1 \sqsubseteq I_2$ iff $I_1(p) \preceq I_2(p)$ for all $p \in \Pi$. The least interpretation $\vartriangle$ maps every propositional symbol to the least element $\bot$ of L.*

A rule of a multi-adjoint logic program is satisfied whenever the truth-value of the rule is greater or equal than the confidence factor associated with the rule. Formally:

Definition 10 (Satisfaction, Model). *Given an interpretation $I \in \mathcal{I}_{\mathfrak{L}}$, a weighted rule $\langle A \leftarrow_i \mathcal{B}, \vartheta \rangle$ is* satisfied *by I iff $\vartheta \preceq \hat{I}\,(A \leftarrow_i \mathcal{B})$. An interpretation $I \in \mathcal{I}_{\mathfrak{L}}$ is a* model *of a multi-adjoint logic program $\mathbb{P}$ iff all weighted rules in $\mathbb{P}$ are satisfied by I.*

Note the following equalities

$$\hat{I}(A \leftarrow_i \mathcal{B}) = \hat{I}(A) \mathrel{\dot{\leftarrow}_i} \hat{I}(\mathcal{B}) = I(A) \mathrel{\dot{\leftarrow}_i} \hat{I}(\mathcal{B})$$

and the evaluation of $\hat{I}(\mathcal{B})$ proceeds inductively as usual, till all propositional symbols in $\mathcal{B}$ are reached and evaluated under I. For the particular case of a fact (a rule with $\top$ in the body) satisfaction of $\langle A \leftarrow_i \top, \vartheta \rangle$ means

$$\vartheta \preceq \hat{I}(A \leftarrow_i \top) = I(A) \mathrel{\dot{\leftarrow}_i} \top$$

by property *(a3)* of adjoint pairs this is equivalent to $\vartheta \mathbin{\dot{\&}_i} \top \preceq I(A)$ and this by assumption *(l3)* of multi-adjoint lattices gives $\vartheta \preceq I(A)$.

Definition 11. *An element $\lambda \in L$ is a* correct answer *for a program $\mathbb{P}$ and a query $?A$ if for an arbitrary interpretation $I\colon \Pi \to L$ which is a model of $\mathbb{P}$ we have $\lambda \preceq I(A)$.*

Example 3. The interpretation I defined by

$$\begin{aligned} I(\texttt{low\_oil}) &= 0.25 \\ I(\texttt{low\_water}) &= 0.35 \\ I(\texttt{overheating}) &= 0.45 \\ I(\texttt{rich\_mixture}) &= 0.90 \\ I(\texttt{noisy\_behaviour}) &= 0.75 \\ I(\texttt{high\_fuel\_consumption}) &= 0.55 \end{aligned}$$

is a model of the program given in Example 2.

If we add the query ?`overheating` to the program, then 0.1 is a correct answer. Actually, it is the greatest correct answer for the query.

The immediate consequences operator, given by van Emden and Kowalski in [15], can be generalised to the framework of multi-adjoint logic programs as follows:

Definition 12. *Let $\mathbb{P}$ be a multi-adjoint logic program. The* immediate consequences operator $T^{\mathfrak{L}}_{\mathbb{P}}: \mathcal{I}_{\mathfrak{L}} \to \mathcal{I}_{\mathfrak{L}}$*, mapping interpretations to interpretations, is defined by considering*

$$T^{\mathfrak{L}}_{\mathbb{P}}(I)(A) = \sup\left\{\vartheta \mathbin{\dot{\&}_i} \hat{I}(\mathcal{B}) \mid A \overset{\vartheta}{\leftarrow}_i \mathcal{B} \in \mathbb{P}\right\}$$

Note that all the suprema involved in the definition do exist because L is assumed to be a complete lattice.

As it is usual in the logic programming framework, the semantics of a multi-adjoint logic program is characterized by the post-fixpoints of $T^{\mathfrak{L}}_{\mathbb{P}}$.

Theorem 1 ([9]). *An interpretation I of $\mathcal{I}_{\mathfrak{L}}$ is a model of a multi-adjoint logic program $\mathbb{P}$ iff $T^{\mathfrak{L}}_{\mathbb{P}}(I) \sqsubseteq I$.*

Note that the fixpoint theorem works even without any further assumptions on conjunctors (definitely they need not be commutative and associative).

The monotonicity of the operator $T^{\mathfrak{L}}_{\mathbb{P}}$, for the case of only one adjoint pair, has been shown in [1]. The proof for the general case is similar.

Theorem 2 ([9]). *The operator $T^{\mathfrak{L}}_{\mathbb{P}}$ is monotonic.*

Due to the monotonicity of the immediate consequences operator, the semantics of $\mathbb{P}$ is given by its least model which, as shown by Knaster-Tarski's theorem, is exactly the least fixpoint of $T^{\mathfrak{L}}_{\mathbb{P}}$, which can be obtained by transfinitely iterating $T^{\mathfrak{L}}_{\mathbb{P}}$ from the least interpretation $\triangle$.

It is worth to investigate conditions which make the $T^{\mathfrak{L}}_{\mathbb{P}}$ operator to be continuous, in [9] it was proved that whenever every operator in Ω turns out to be continuous in the lattice, then $T_{\mathbb{P}}$ is also continuous and, consequently, its least fixpoint can be obtained by a countably infinite iteration from the least interpretation. Formally,

Theorem 3 ([9]). *If all the operators occurring in the bodies of the rules of a program $\mathbb{P}$ are continuous, and the adjoint conjunctions are continuous in their second argument, then $T^{\mathfrak{L}}_{\mathbb{P}}$ is continuous.*

4 Procedural semantics of multi-adjoint logic programs

Once shown that the $T_{\mathbb{P}}^{\mathfrak{L}}$ operator can be continuous under very general hypotheses, then the least model can be reached in at most countably many iterations. Therefore, it is worth to define a procedural semantics which allow us to actually construct the answer to a query against a given program.

In the following, we work in a hybrid Ω-algebra made up from the elements of the lattice, and the same alphabet of the language but the adjoint implicators.

For the formal description of the computational model, we will consider an extended the language $\mathfrak{F}'$ defined on the same graded set, but whose carrier is the disjoint union $\Pi \cup L$; this way we can work simultaneously with propositional symbols and with the truth-values they represent.

Definition 13. *Let $\mathbb{P}$ be a multi-adjoint logic program on a multi-adjoint Ω-algebra $\mathfrak{L}$ with carrier L and V the set of truth values of the rules in $\mathbb{P}$. The* extended language *$\mathfrak{F}'$ is the corresponding Ω-algebra of formulas freely generated from the disjoint union of Π and V.*

The formulas in the language $\mathfrak{F}'$ will be referred as *extended formulas.* An operator symbol ω interpreted under $\mathfrak{F}'$ will be denoted as $\bar{\omega}$.

Our computational model will take a query (an atom), and 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 multi-adjoint lattice to get the computed answer.

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

Definition 14. Admissible rules *are defined as follows:*

1. *Substitute an atom A in an extended formula by $(\vartheta \bar{\&}_i \mathcal{B})$ whenever there exists a rule $\langle A \leftarrow_i \mathcal{B}, \vartheta \rangle$ in $\mathbb{P}$.*
2. *Substitute an atom A in an extended formula by $\bot$.*
3. *Substitute an atom A in an extended formula by ϑ whenever there exists a fact $\langle A \leftarrow_i \top, \vartheta \rangle$ in $\mathbb{P}$.*

Note that if an extended formula turns out to have no propositional symbols, then it can be directly interpreted in the multi-adjoint Ω-algebra $\mathfrak{L}$. This justifies the following definition of *computed answer.*

Definition 15. *Let $\mathbb{P}$ be a program in a multi-adjoint language interpreted on a multi-adjoint lattice $\mathcal{L}$ and let $?A$ be a goal. An element $\dot{@}[r_1, \dots, r_m]$, with $r_i \in L$, for all $i \in \{1, \dots, m\}$ is said to be a* computed answer *if there is a sequence $G_0, \dots, G_{n+1}$ such that*

1. *$G_0 = A$ and $G_{n+1} = \bar{@}[r_1, \dots, r_m]$ where $r_i \in L$ for all $i = 1, \dots n$.*
2. *Every G_i, for $i = 1, \dots, n$, is a formula in $\mathfrak{F}'$.*

3. Every G_{i+1} is inferred from G_i by one of the admissible rules.

The idea of the computation is to consecutively apply admissible rules until an extended formula with no propositional symbols $\bar{@}[r_1,\dots,r_m]$ is obtained, which can be interpreted as the element $\dot{@}[r_1,\dots,r_m]$ in the lattice $\mathcal{L}$.

An alternative formalism of Generalized Annotated Logic Programs (GALP) was introduced in [7]. The procedural semantics of GALP uses a CLP-like procedure to solve a set of lattice inequalities to find a computed answer. Our procedural semantics replaces constraints in the form of inequalities by equalities building a final formula for $\dot{@}$ which is the best answer.

It might be the case that for some lattices it is not possible to get the correct answer, simply consider L to be the powerset of a two-element set $\{a,b\}$ ordered by inclusion. The requirement of the reductant property stated below will allow us to avoid these cases, see [7, 11] for details.

Definition 16 (Reductant, reductant property). *Let $\mathbb{P}$ be a program on a multi-adjoint Ω-algebra $\mathfrak{F}$ with values in a multi-adjoint lattice $\mathfrak{L}$; assume that all the rules in $\mathbb{P}$ with head A are $A \overset{\vartheta_i}{\longleftarrow}_i \mathcal{B}_i$ for $i = 1,\dots,k$. A* reductant *for A is a rule $A \overset{\vartheta}{\longleftarrow} @(\mathcal{B}_1,\dots,\mathcal{B}_n)$ such that for any $b_1,\dots,b_k$ we have*

$$\sup\{\vartheta_i \mathbin{\dot{\&}_i} b_i \mid i = 1,\dots,n\} = \vartheta \mathbin{\dot{\&}} \dot{@}(b_1,\dots,b_k)$$

A program $\mathbb{P}$ is said to have the reductant property *if there exist reductants for any atom A occurring in the head of some rule.*

Note that ϑ and $@$ should depend only on the (multi-)set of $\&_i$.

Certainly, it will be interesting to consider only programs which contain all its reductants, but this might be a too heavy condition on our programs; the following proposition shows that we can assume that our programs contain all the reductants, because the set of models is preserved.

Proposition 1 ([10]). *Any reductant $A \overset{\vartheta}{\longleftarrow} \mathcal{B}$ of $\mathbb{P}$ is satisfied by any model of $\mathbb{P}$. In short, $\mathbb{P} \models A \overset{\vartheta}{\longleftarrow} \mathcal{B}$.*

As a consequence of the proposition above, we can assume that a program contains all its reductants, since its set of models is not modified.

Definition 17. *Given a program $\mathbb{P}$ with the reductant property and a query $?A$, the* greatest computed answer *is a computed answer in which calculation admissible rules 1 and 3 are applied only with rules (and facts) reductants in $\mathbb{P}$, and the admissible rule 2 is applied if and only if no rule/fact exists for a given atom in the extended formula.*

Theorem 4. *Given a program $\mathbb{P}$, a query $?A$ and a computed answer λ'. If λ is the greatest computed answer, then $\lambda' \leq \lambda$.*

The theorem above shows that computed answers as in the previous definition are actually the greatest.

Theorem 5. *Given a program $\mathbb{P}$ with the reductant property, for all atom A let λ_A be the greatest computed answer for $\mathbb{P}$ and query $?A$, then $\lambda_A \preceq T^{\omega}_{\mathbb{P}}(\triangle)(A)$.*

It was proved in [10] that, given a program $\mathbb{P}$, then $T^{n}_{\mathbb{P}}(\triangle)(A)$ is a computed answer for all n and for all query $?A$. Now, in conjunction with the result above we straightforwardly obtain the to following corollaries which will be used later.

Corollary 1. *$\lambda \in L$ is the greatest computed answer for program $\mathbb{P}$ and query $?A$ if and only if $\lambda = T^{\omega}_{\mathbb{P}}(\triangle)(A)$.*

To finish the section, simply recall the following result from [10].

Corollary 2. *$\lambda \in L$ is a correct answer for program $\mathbb{P}$ and query $?A$ if and only if $\lambda \preceq T^{\omega}_{\mathbb{P}}(\triangle)(A)$.*

5 Abduction in multi-adjoint logic programs

To state the intuition behind an abduction problem, if will use the program in Example 2, but without the three facts; that is, we have no information about variables $\texttt{rich\_mixture}, \texttt{low\_oil}, \texttt{low\_water}$, which will turn out to be hypotheses to explain the behaviour of our car.

If we notice it is noisy, overheated and has a high fuel consumption, we would like to know why it is so. Let us call these assertions *observation variables* and let us denote it by

$$OV = \{\texttt{noisy\_behaviour}, \texttt{overheated}, \texttt{high\_fuel\_consumption}\}.$$

As noisiness can be subjective and the height of fuel consumption and temperature of the engine can take different (high) values, our observations are estimated (by an expert) by confidence factors. So the second parameter of our abduction problem are observations (sometimes called manifestations, symptoms, effects) represented by a theory, i.e. a partial mapping $OBS\colon OV \to L$ consisting of facts, which can be thought of as observation variables equipped with confidence factors

$$\texttt{high\_fuel\_consumption} \overset{0.25}{\longleftarrow_P}, \quad \texttt{overheating} \overset{0.25}{\longleftarrow_P}, \quad \texttt{noisy\_behaviour} \overset{0.5}{\longleftarrow_P}$$

Note that it is not necessary to assume a specific type of implication for observations - any will do. The full strength of multi-adjointness is needed for the logic programming part where specific implication describes a specific action of the truth value of the rule.

We would like to find explanations (causes) for given observations (symptoms) by means of semantical consequence. Namely, explanations are assertions which together with domain knowledge forces every model of them to be also a model of observations. That is, whenever in a real world situation represented by an interpretation I, both domain knowledge and explanations are true, then all observations are true in I.

Possible explanations will be L-fuzzy subsets of the set of hypotheses

$$H = \{\texttt{rich\_mixture}, \texttt{low\_oil}, \texttt{low\_water}\}$$

This makes sense, because in the realm of a theory and of observations suffering from uncertainty, we can expect that certain level of confidence of hypotheses can be (under the presence of the theory) an explanation of these (uncertain) observations. The formal definitions of abduction problem, explanation, etc. are given below.

Definition 18. *An* abduction problem *consists of a tuple $\mathcal{A} = \langle \mathbb{P}, OBS, H \rangle$, where*

1. *$\mathbb{P}$ is a multi-adjoint logic program.*
2. *$H \subseteq \Pi$ is the set of hypotheses.*
3. *$OBS\colon OV \to L$ is the L-fuzzy theory of observations (where OV is a set of observation variables such that $OV \cap H = \varnothing$)*

The intended meaning of $OV \cap H = \varnothing$ is that observation variables should not be explained by themselves.

A theory is a mapping assigning formulas a truth value. For two theories T and S, let $T \cup S$ denote the union of them as a theory defined by

$$(T \cup S)(A) = \max\{T(A), S(A)\}$$

Definition 19. *An L-fuzzy theory $E\colon H \to L$ is a* correct explanation *to an abduction problem $\mathcal{A} = \langle \mathbb{P}, OBS, H \rangle$ if*

1. *$\mathbb{P} \cup E$ is satisfiable.*
2. *$\mathbb{P} \cup E$ semantically implies OBS, that is every model of $\mathbb{P} \cup E$ is also a model of OBS.*

It will be useful to represent explanations as a subset of L^n, especially when $L = [0, 1]$. If $H = \{h_1, \dots, h_n\}$ is the set of hypotheses and $E\colon H \to L$ is an explanation, it is uniquely determined by its values

$$(E(h_1), \dots, E(h_n)) \in L^n$$

so, an element $\varepsilon = (\varepsilon_1, \dots, \varepsilon_n) \in L^n$ represents the mapping $E_\varepsilon(h_i) = \varepsilon_i$.

The set of correct explanations for $\mathcal{A}$ will be denoted by $SOL_d(\mathcal{A})$, where the subscript d resembles the declarative character of the explanations.

Using our motivating example we are illustrating a possible solution of a whole class of problems which are formulated within our formalism.

Example 4. Having our motor vehicle example and our multi-adjoint program $\mathbb{P}$ and the query `?high_fuel_consumption` by multi-adjoint logic program computation, using the first program rule we get

$$\min(0.8, \max(0, \texttt{rich\_mixture} + \texttt{low\_oil} - 1))$$

Now similarly as in the two valued logic abductive logic programming [6], our procedure instead of failing in a proof when a selected subgoal fails to unify with the head of any rule, the subgoal is viewed as a hypothesis, that is, if we know confidence factor for rich mixture and low oil (from an explanation) we have the computed answer for high fuel consumption. To fulfil

$$OBS(\texttt{high\_fuel\_consumption}) \geq 0.25$$

it should be

$$\min(0.8, \max(0, \texttt{rich\_mixture} + \texttt{low\_oil} - 1)) \geq 0.25$$

hence

$$\texttt{rich\_mixture} + \texttt{low\_oil} \geq 1.25$$

Note that if $OBS(\texttt{high\_fuel\_consumption})$ would be greater than 0.8, there is no explanation for this. To overcome this we could think of the possibility of calculating the truth value of the metamathematical assertion "E is an explanation for $\mathcal{A}$". Here we consider the case, when this truth value is 1, that is, E is an explanation with full confidence. This is why we are calculating the truth value of the following implication: Whenever I is a model of $\mathbb{P} \cup E$ then I is a model of OBS. In particular the truth value of the implication

$$\begin{aligned} \text{"If} \quad & I(\texttt{rich\_mixture}) + I(\texttt{low\_oil}) > 1.25 \\ \text{then} \quad & I(\texttt{high\_fuel\_consumption}) \geq 0.25\text{"} \end{aligned}$$

And again, the truth value of $x \leq y$ can be calculated as $\dot{\leftarrow}(y, x)$.

So our multi-adjoint logic programming abduction should run as a usual logic program with two exceptions

- it successfully ends without resolving variables which are in the set of hypotheses
- it is prompted by a query with threshold, which can serve as a cut (as we will see later).

Definition 20 (Procedural semantics for abduction). *Let us have an abduction problem $\mathcal{A} = \langle \mathbb{P}, OBS, H \rangle$ and consider $m \in OV$. A* successful abduction *for $\mathcal{A}$ and m is a sequence $\mathcal{G} = (G_0, G_1, \ldots, G_l)$ of extended formulas in the language of multi-adjoint logic program computations such that:*

1. *$G_0 = m$,*
2. *G_l contains only variables from H, and*
3. *(a) For all $i < l$, G_{i+1} is inferred from G_i by one of admissible rules, and*
   (b) For the constant interpretation $I_\top\colon \Pi \to \{\top\}$ the inequality $I_1(G_{i+1}) \geq OBS(m)$ holds.

The last condition is to be understood as a cut, because it allows to estimate the best possible computation of remaining propositional variables.

This definition allows the explanation of a single observation variable. Merging of several observation variables is in the Definition 21 bellow. Solutions are obtained as a combination (intersection) of the above set through all m's in observation variables. Moreover, the set of all solutions is the union through all possible combinations of all possible computational branches for all observation variables.

For $H = \{h_1, \dots h_n\}$, the expression G_l can be understood as a function of n variables from L^n to L and that is why can denote it by $G_l = \mathcal{G}_m(h_1, \dots, h_n)$.

Theorem 6 (Existence of solutions). *Let $\mathcal{A} = \langle \mathbb{P}, OBS, H \rangle$ be an abduction problem and assume that for each $m \in OV$ there is a successful abduction for $\mathcal{A}$ and m. Then $SOL_d(\mathcal{A}) \neq \varnothing$.*

Our definition of abduction gives us the possibility to define computed explanations for abduction problems $\mathcal{A}$.

Definition 21. *A tuple $\varepsilon = (\varepsilon_1, \dots, \varepsilon_n)$ is* a computed explanation *for an abduction problem $\mathcal{A} = \langle \mathbb{P}, OBS, H \rangle$ if for every $m \in OV$ there is an abduction $\mathcal{G}_m$ for $\mathcal{A}$ and m such that*

$$\mathcal{G}_m(\varepsilon_1, \dots, \varepsilon_n) \geq OBS(m)$$

The set of all computed explanations will be denoted by $SOL_p(\mathcal{A})$, where the subscript p resembles the procedural character of this definition.

Example 5. In our motor vehicle example we calculated that first rule of $\mathbb{P}$ gives $\texttt{rich\_mixture} + \texttt{low\_oil} \geq 1.25$, similarly second rule gives $\texttt{low\_oil} \geq 0.5$ and the third gives $\texttt{rich\_mixture} \geq 0.625$, hence the set (coordinates are ordered as $(\texttt{rich\_mixture}, \texttt{low\_oil}, \texttt{low\_water})$)

$$\{(\varepsilon_1, \varepsilon_2, \varepsilon_3) \in [0,1]^3 : \varepsilon_1 + \varepsilon_2 \geq 1.25 \text{ and } \varepsilon_1 \geq 0.625 \text{ and } \varepsilon_2 \geq 0.5\}$$

is a subset of $SOL_p(\mathcal{A})$.

This example shows also that the area got as an intersection of some computational branches for m's has the shape of a convex body. Moreover, the set of all solutions is the union of such areas. It seems reasonable that, to get the cheapest answer, we have just to run a linear programming optimization separately on each of these areas.

Theorem 7 (Soundness). *Assume $\mathcal{A}$ is a definite abduction problem, then $SOL_p(\mathcal{A}) \subseteq SOL_d(\mathcal{A})$ (that is, every computed explanation for $\mathcal{A}$ is also a correct explanation).*

In the completeness theorem below we need the assumption that our logical program has a finite computational tree, according to abductions. This is very often the case in practical applications of abduction, because e.g. observations should not be explained by themselves, and most of logic programs for abduction are layered. Moreover if the conjunctors are archimedean (also very often) then the abduction ends below the observation value threshold, and hence is cut.

Theorem 8 (Completeness). *Assume $\mathcal{A}$ is an abduction problem and the logical program has a finite computational tree according to abductions, then $SOL_d(\mathcal{A}) \subseteq SOL_p(\mathcal{A})$ (that is, every correct explanation for A is also a computed explanation).*

From now on we do not have to distinguish between two sets of solutions and we simply denote it $SOL(\mathcal{A})$.

We comment here very briefly the possibility of using linear programming in some cases to obtain the cheapest explanation to an abduction problem wrt a given cost function.

Example 6. Continuing the example of motor vehicles assume that checking $(\varepsilon_1, \varepsilon_2, \varepsilon_3) \in SOL(\mathcal{A})$ costs $2\varepsilon_1 + \varepsilon_2 + 0.1\varepsilon_3$. The space of solutions $SOL(\mathcal{A})$ is bounded by linear surfaces in $[0,1]^3$ and is the union of four convex bodies, obtained from the combinations of rules in the program for each observation variable.

Using the first three rules of the program $\mathbb{P}$ we get one of these four convex bodies, namely the set

$$S_1 = \{(\varepsilon_1, \varepsilon_2, \varepsilon_3) \in [0,1]^3 \mid \varepsilon_1 + \varepsilon_2 \geq 1.25 \text{ and } \varepsilon_1 \geq 0.625 \text{ and } \varepsilon_2 \geq 0.5\}$$

Applying a linear programming method for S_1 wrt our cost function we get in this convex body a minimal solution $(0.625, 0.625, 0)$ at cost of 1.875.

Actually, the cheapest solution of our running abduction problem $\mathcal{A}$ is $\varepsilon_{min} = (0.25, 1, 0.35)$ at cost of 1.535, obtained from the convex body of all solutions to the first, fourth and fifth program rule.

Eiter and Gottlob showed in [4] that deciding $SOL_d(\mathcal{A}) \neq \varnothing$ for two valued logic is complete for complexity classes at the second level of polynomial hierarchy, while the use of priorisation raises the complexity to the third level in certain cases (for arbitrary propositional theories).

One can ask how difficult is to decide, in our framework, whether $\varepsilon \in SOL(\mathcal{A})$ or not. Now we see that it substantially depends on the complexity of logic programming computation and the complexity of functions evaluating the truth values for connectives; thus, from a computational point of view, it makes sense to use simple connectives (e.g. linear in each coordinate, as product is, or even partly constant). So assuming connectives are easy this problem is in NP.

Regarding the linear programming approach to the cheapest explanations, as linear programming is polynomial and prolog is in NP, to find minimal solutions for an abduction problem (assuming connectives are coordinatewise linear) does not increase the complexity and remains in NP.

6 Conclusions and future work

In this work a theoretical framework for abduction in multi-adjoint logic programming is introduced; a sound and complete procedural semantics has been defined, and the possibility of obtaining the cheapest possible explanation to an abduction problem wrt a cost function by means of a logic programming computation followed by a linear programming optimization has been shown.

Future work on this area will be concerned with showing the embedding of different approaches to abduction in our framework, as well as the study of complexity issues in lattices more general than the unit interval.

Acknowledgements

We thank C. Damásio and L. Moniz Pereira for communicating the existence of the Technical Report [1], on which this research began.

References

1. C.V. Damásio and L. Moniz Pereira. A theory of logic programming. Technical report, Dept. Computer Science. Univ. Nova de Lisboa, 2000.
2. C.V. Damásio and L. Moniz Pereira. Monotonic and residuated logic programs, in *Proc. of ECSQARU'01*, Lect. Notes in Artif. Intelligence., 2143, pp. 748-759, 2001.
3. C.V. Damásio and L. Moniz Pereira. Hybrid probabilistic logic programs as residuated logic programs. In *Logics in Artificial Intelligence*, pages 57–73. Lect. Notes in AI, 1919, Springer-Verlag, 2000.
4. T. Eiter and G. Gottlob. The complexity of logic based abduction. *Journal of the ACM*, 42:3–42, 1995.
5. P. Hájek. *Metamathematics of Fuzzy Logic.* Trends in Logic. Studia Logica Library. Kluwer Academic Publishers, 1998.
6. A.C. Kakas, R.A. Kowalski, and F. Toni. The role of abduction in logic programming. In *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 5, pages 235–324. Oxford Univ. Press, 1998.
7. M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *J. of Logic Programming*, 12:335–367, 1992.
8. J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, second, extended edition, 1987.
9. J. Medina, M. Ojeda-Aciego, and P. Vojtaš. Multi-adjoint logic programming with continuous semantics, *Proc. of LPNMR'01*, Lect. Notes in Artif. Intelligence 2173, pp. 351–364, 2001.
10. J. Medina, M. Ojeda-Aciego, and P. Vojtáš. A procedural semantics for multi-adjoint logic programming. *Proc. of EPIA'01*, Lect. Notes in Artif. Intelligence 2258, 2001.
11. S. Morishita. A unified approach to semantics of multi-valued logic programs. Technical Report RT 5006, IBM Tokyo, 1990.
12. J. Pavelka. On fuzzy logic I, II, III. *Zeitschr. f. Math. Logik und Grundl. der Math.*, 25, 1979.
13. P. Vojtáš. Fuzzy logic programming. *Fuzzy sets and systems*, 124(3):361–370, 2001. Accepted.
14. P. Vojtáš and L. Paulík. Soundness and completeness of non-classical extended SLD-resolution. In *Proc. Extensions of Logic Programming*, pages 289–301. Lect. Notes in Comp. Sci. 1050, Springer-Verlag, 1996.
15. M. van Emden and R. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.