A new approach to completeness for multi-adjoint logic programming

Jesús Medina, Manuel Ojeda-Aciego*
Dept. Matemática Aplicada
E.T.S.I. Informática
Universidad de Málaga
{jmedina,aciego}@ctima.uma.es

Abstract

The hypotheses required for the quasi-completeness theorems [5, 7] are restated in terms of termination properties of a calculation algorithm.

1 Introduction

Several papers can be found in the literature on applications of definite fuzzy logic programming, which are based either on Łukasiewicz, or product, or Gödel implications on the unit real interval (an overview can be seen in [8]); if one is interested in more complex systems, it is reasonable to allow considering more general sets of truth-values to reflect uncertainty.

The following example, taken and simplified from [2], considers truth-values in a set other than the unit interval: Assume that if the CEO of a company sells the stock, and retires with a probability over 85%, then the probability that the stock of the company drops is 40-90%

$$\texttt{price\_drop} : [0.4, 0.9] \leftarrow$$
$$(\texttt{ch\_sells\_stock} \ \wedge \ \texttt{ch\_retires}) : [0.85, 1]$$

Several approaches to the generalisation of the set of truth-values can be found; for instance, that given by the structure of bilattice, which has been used to handle negation in logic programming; also in [1] the set of truth-values is generalised to a residuated lattice (in order to embed possibilistic, and hybrid probabilistic logic programs), but only one implication was considered and no study of continuity of its semantics was given.

Multi-adjoint logic programming was introduced in [6] as a generalisation of monotonic and residuated logic programming [1]. Its special features are that: it is possible to use a number of different implications in the rules of our programs; sufficient conditions for continuity of the immediate consequences operator are known; and the requirements on the lattice of truth-values are weaker than those on the residuated approach.

In practical systems we need a computational model, whose existence is guaranteed by the continuity of the semantics. The purpose of this work is to prove a completeness theorem for the computational model, which improves two quasi-completeness results introduced in [5, 7].

The structure of the paper is the following: in Section 2 the relevant definitions concerning multi-adjoint logic programming are presented. Then in Section 3, a study of the relationships between the greatest correct answer and the immediate consequence operator is given, and the concept of reductant for multi-adjoint logic programs is stated. In Section 4 the main theorem is proved, the greatest computed answer coincides with the greatest correct answer and, assuming termination, can be algorithmically obtained using the procedural semantics. The final section contains some conclusions and pointers to future work.

* Partially supported by Spanish DGI project BFM2000-1054-C02-02

2 Preliminary definitions

To make this paper as self-contained as possible, the necessary definitions about multi-adjoint structures are included in this section. For motivating comments, the interested reader is referred to [6].

Definition 1: Let $\langle L, \preceq \rangle$ be a complete lattice. A *multi-adjoint lattice* $\mathcal{L}$ is a tuple $(L, \preceq, \leftarrow_1, \&_1, \ldots, \leftarrow_n, \&_n)$ 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. $(\leftarrow_i, \&_i)$ is an adjoint pair in $\langle L, \preceq \rangle$ for $i = 1, \ldots, n$; i.e.
   (a) Operation $\&_i$ is increasing in both arguments,
   (b) Operation $\leftarrow_i$ is increasing in the first argument and decreasing in the second argument,
   (c) For any $x, y, z \in P$, we have that $x \preceq (y \leftarrow_i z)$ holds if and only if $(x \,\&_i\, z) \preceq y$ holds.

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 Ω-algebra* which can be understood as an extension of a multi-adjoint lattice containing a number of extra operators given by a signature Ω.

We will be working with two Ω-algebras: 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 in Ω under $\mathfrak{L}$ as $\dot{\omega}$ (a dot on the operator), whereas ω itself will denote its interpretation under $\mathfrak{F}$.

Syntax

Definition 2: A *multi-adjoint logic program on a multi-adjoint Ω-algebra $\mathfrak{F}$ with values in a multi-adjoint lattice $\mathfrak{L}$* (in short *multi-adjoint program*) is a set $\mathbb{P}$ of rules of the form $\langle A \leftarrow_i \mathcal{B}, \vartheta \rangle$.

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 $B_1, \ldots, B_n$ ($n \geq 0$) by the use of conjunctors $\wedge_1, \ldots, \wedge_k$, disjunctors $\vee_1, \ldots, \vee_l$ and aggregators $@_1, \ldots, @_m$.

A *fact* is rules with body $\top$. A rule which is not a fact will be called a *proper rule*. A *query* (or *goal*) is a propositional symbol intended as a question $?A$ prompting the system.

Model semantics

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, $\hat{I} \colon F_\Omega \to L$. The set of all interpretations of the formulas defined by the Ω-algebra $\mathfrak{F}$ in the Ω-algebra $\mathfrak{L}$ is denoted $\mathcal{I}_\mathfrak{L}$. 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.

Definition 3:

1. An interpretation I *satisfies* $\langle A \leftarrow_i \mathcal{B}, \vartheta \rangle$ if and only if $\vartheta \preceq \hat{I}\,(A \leftarrow_i \mathcal{B})$.
2. An interpretation I is a *model* of a multi-adjoint logic program $\mathbb{P}$ iff all weighted rules in $\mathbb{P}$ are satisfied by I.
3. An element $\lambda \in L$ is a *correct answer* for a program $\mathbb{P}$ and a query $?A$ if for any interpretation I which is a model of $\mathbb{P}$ we have $\lambda \preceq I(A)$.

Fix-point semantics

The immediate consequences operator, given by van Emden and Kowalski [3], can be easily

generalised to the framework of multi-adjoint logic programs.

Definition 4: Let $\mathbb{P}$ be a multi-adjoint program. The *immediate consequences operator* $T_{\mathbb{P}} \colon \mathcal{I}_{\mathfrak{L}} \to \mathcal{I}_{\mathfrak{L}}$, mapping interpretations to interpretations, is defined by

$$T_{\mathbb{P}}(I)(A) = \sup\left\{\vartheta \,\dot{\&}_i\, \hat{I}(\mathcal{B}) \mid A \overset{\vartheta}{\leftarrow}_i \mathcal{B} \in \mathbb{P}\right\}$$

As usual, the semantics of a multi-adjoint logic program is characterised by the post-fixpoints of $T_{\mathbb{P}}$, see [6]; that is, an interpretation I of $\mathcal{I}_{\mathfrak{L}}$ is a model of a multi-adjoint logic program $\mathbb{P}$ iff $T_{\mathbb{P}}(I) \sqsubseteq I$. It is remarkable the fact that this result is still true even without any further assumptions on conjunctors (definitely they need not be commutative and associative).

Regarding continuity, the following theorem was proved in [6].

Theorem 1

1. *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 $T_{\mathbb{P}}$ is continuous.*
2. *If the operator $T_{\mathbb{P}}$ is continuous for all program $\mathbb{P}$ on $\mathfrak{L}$, then any operator in the body of the rules is continuous, and the adjoint conjunctions are continuous in their second argument.*

Procedural semantics

Once we know that the $T_{\mathbb{P}}$ operator can be continuous under very general hypotheses then, by the Knaster-Tarski theorem, the least model can be reached in at most countably many iterations, that is $T^{\omega}_{\mathbb{P}}(\triangle)$. Therefore, it is worth to define a procedural semantics which allow us to actually construct the answer to a query against a given program.

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

Definition 5: Let $\mathbb{P}$ be a multi-adjoint program, and let $V \subset L$ be the set of truth values of the rules in $\mathbb{P}$. The *extended language* $\mathfrak{F}^e$ is the corresponding Ω-algebra of formulas freely generated from the disjoint union of Π and V.

We will refer to the formulas in the language $\mathfrak{F}^e$ simply as *extended formulas*, or *e-formulas*. An operator symbol ω interpreted under $\mathfrak{F}^e$ will be denoted as $\bar{\omega}$.

Our computational model takes a query (atom), and provides a lower bound of the value of A under any model of the program. Note that if an e-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 6: Let $\mathbb{P}$ be a multi-adjoint program, and let $?A$ be a goal. An element $\dot{@}(r_1, \ldots, r_m)$, with $r_i \in L$, for all $i \in \{1, \ldots, m\}$ 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 $G_{n+1} = \bar{@}(r_1, \ldots, r_m)$ where $r_i \in L$ for all $i = 1, \ldots n$.
2. Every G_{i+1} is inferred from G_i by one of the admissible rules below:
   (a) 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 $\mathbb{P}$.
   (b) Substitute an atom A in an extended formula by $\bot$.
   (c) 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 our procedural semantics is not refutation-based, for negation is not allowed in our approach. Instead, it is oriented to obtaining a bound of the optimal correct answer of the query.

Quasi-completeness results

Two quasi-completeness results for the procedural semantics of multi-adjoint logic programming were proved in [7]. Their proofs follow from some technical results. The first lemma below states that the least fix-point is also the least model of a program; the second states a characterisation of correct answers in terms of the $T_{\mathbb{P}}$ operator.

Lemma 1 *For all model I of $\mathbb{P}$ we have that $T^{\omega}_{\mathbb{P}}(\triangle) \sqsubseteq I$.*

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

Now, in order to match correct and computed answers, the proposition below, whose proof is based induction on n, shows that any iteration of the $T_{\mathbb{P}}$ operator is, indeed, a computed answer.

Proposition 1 *Let $\mathbb{P}$ be a program, then $T^{n}_{\mathbb{P}}(\triangle)(A)$ is a computed answer for all n and for all query $?A$.*

We have now all the required background to state the first quasi-completeness result.

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 $\lambda_{n_0} \preceq \delta$.*

The theorem above could be further refined under the assumption of the so-called supremum property [6]:

Definition 7: A cpo L is said to satisfy the *supremum property* if for all directed set $X \subset L$ and for all ε we have that if $\varepsilon \prec \sup X$ then there exists $\delta \in X$ such that $\varepsilon \prec \delta \preceq \sup X$.

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 $\varepsilon \prec \delta$.*

3 Greatest answers. Reductants

Note that 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, consistenly with the information in the program.

Definition 8: Given a complete lattice L, we define the *greatest correct answer*, λ_{CM}, for a program $\mathbb{P}$ and a query $?A$ as

$$\sup\{\lambda \mid \lambda \text{ is a correct answer for } \mathbb{P} \text{ and } ?A\}$$

The following theorem states that the greatest correct answer is reached by the least fix-point of the $T_{\mathbb{P}}$ operator.

Theorem 4 *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.*

Proof: It is indeed a correct answer, since for any model I of $\mathbb{P}$, we have by Lemma 1 that $T^{\omega}_{\mathbb{P}}(\triangle)(A) \preceq I(A)$, for $T^{\omega}_{\mathbb{P}}(\triangle)$ is the minimal model.

It is also the greatest, since by Lemma 2, for any correct answer λ, we have $\lambda \preceq T^{\omega}_{\mathbb{P}}(\triangle)(A)$, and taking supremum we obtain the following chain of inequalities:

$$T^{\omega}_{\mathbb{P}}(\triangle)(A) \preceq \lambda_{CM} \preceq T^{\omega}_{\mathbb{P}}(\triangle)(A)$$

□

Reductants

It might be the case that for some lattices, our procedural semantics cannot compute the greatest correct answer, simply consider L to be the powerset of a two-element set $\{a, b\}$ ordered by inclusion, and the following example from Morishita, used in [4]:

Example 1: Consider a multi-adjoint program $\mathbb{P}$ with rules $A \overset{a}{\leftarrow} B$ and $A \overset{b}{\leftarrow} B$ and fact $\langle B \leftarrow \top, \top \rangle$. Assuming that the adjoint conjunction to $\leftarrow$ has the usual boundary conditions, then the greatest correct answer to

the query $?A$ is $\top$, since it has to be an upper bound of all the models of the program, therefore it has to be greater than both a and b. But the only computed answers are either a or b. $\square$

The idea to cope with this problem is the generalisation of the concept of reductant [4]. Namely, that whenever we have a finite number of rules $A \overset{\vartheta_i}{\leftarrow_i} @_i(D^i_1, \ldots, D^i_{n_i})$ for $i = 1, \ldots, k$, then there should exist another rule which allows us to get the greatest possible value of A under the program.

As any rule $A \overset{\vartheta_i}{\leftarrow_i} @_i(D_1, \ldots, D_{n_i})$ contributes, by means of the adjoint property (3c) in Def. 1, with a value of the form $\vartheta_i \,\dot{\&}_i\, b_i$ for the calculation of the lower bound for the truth-value of A, we would like to have the possibility of reaching the supremum of all the contributions, in the computational model, in a single step. This leads to the following definition.

Definition 9: Let $\mathbb{P}$ be a multi-adjoint program; assume that the set of rules in $\mathbb{P}$ with head A can be written as $\langle A \leftarrow_i \mathcal{B}_i, \vartheta_i \rangle$ for $i = 1, \ldots, n$, and contains at least a proper rule; a *reductant for* A is any rule

$$\langle A \leftarrow @(\mathcal{B}_1, \ldots, \mathcal{B}_n), \top \rangle$$

where $\leftarrow$ is any implication with an adjoint conjunctor and the aggregator $@$ is defined as

$$\dot{@}(b_1, \ldots, b_n) = \sup\{\vartheta_1 \,\dot{\&}_1\, b_1, \ldots, \vartheta_n \,\dot{\&}_n\, b_n\}$$

If there were just facts with head A, but not a single 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,

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

As a consequence of the definition, and the boundary conditions in the definition of bi-residuated 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 is immediate to prove that the rule constructed in the definition above, in presence of proper rules, behaves as a reductant (in the standard sense) for A in $\mathbb{P}$, in that it provides in a single step the greatest amount of information about A which can be obtained from the rules (and facts) with head A.

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 it is not true, therefore we can assume that a program contains all its reductants, since its set of models is not modified.

Proposition 2 *Any reductant $A \overset{\vartheta}{\leftarrow} \mathcal{B}$ of $\mathbb{P}$ is satisfied by any model of $\mathbb{P}$.*

Note that we have followed just traditional techniques of logic programming, and discarded non-determinism by using reductants.

4 Completeness

It is time now to consider the possibility of obtaining a maximal (or greatest) computed answer for a query to a program. The definition of greatest computed answer follows directly its intended meaning.

Definition 10: Given a program $\mathbb{P}$ and a query $?A$, the *greatest computed answer* is a computed answer λ such that whenever λ' is any computed answer, then $\lambda' \preceq \lambda$.

Consider the following procedure, given a program $\mathbb{P}$ and a query $?A$:

1. *Apply rule R2 only on atoms B for which there are neither rules nor facts with head B.*
2. *Use the canonical reductant, otherwise.*

The proposition below states that the procedure can be used for obtaining the greatest computed answer.

Proposition 3 *Given a program $\mathbb{P}$ and a query $?A$, and assume that the procedure above terminates, then the output of the procedure is the greatest computed answer.*

Before starting the proof, we should recall that all the reductants can be assumed to belong to the program. On the other hand, note that, although rules R1 and R3 are not explicitly mentioned in the statement of the proposition, they are special cases of application of a reductant.

Proof: Firstly, note that only one computed answer can be obtained by following the procedure.

The maximality of the computed answer follows from the monotonicity of the operators in the bodies of our rules, together with the definition of reductant:

Let λ be the computed answer for program $\mathbb{P}$ and a query $?A$ obtained by the procedure, and let λ' be any computed answer for program $\mathbb{P}$ and a query $?A$. We will use induction on the length of the chain which computes λ' to prove that $\lambda' \preceq \lambda$.

If the length n of the chain is 1, it because either rule R2 or R3 has been applied. Therefore,

1. If R2 has been applied, then $\lambda' = \bot \preceq \lambda$;
2. If R3 has been applied on a fact, say $\langle A \leftarrow_{j_1} \top, \vartheta_1 \rangle$, then $\lambda' = \vartheta_1$, and is obtained by the chain

$$G'_0 = A \qquad G'_1 = \vartheta_1$$

Now, for λ there are two possibilities:

(a) If all the rules with head A are facts $\langle A \leftarrow_{j_i} \top, \vartheta_i \rangle$ with $i \in \{1, \ldots, l\}$, then the canonical reductant has the form $\langle A \leftarrow \top, \vartheta \rangle$ where $\vartheta = \sup\{\vartheta_1, \ldots, \vartheta_l\}$, and $\lambda = \vartheta$ has the chain of computation

$$G_0 = A \qquad G_1 = \vartheta$$

therefore, $\lambda' = \vartheta_1 \preceq \vartheta = \lambda$.

(b) Otherwise, if some of the rules with head A is not a fact, we will write $\langle A \leftarrow_{j_i} \mathcal{B}_i, \vartheta_i \rangle$ with $i = 1, \ldots, l$ to denote all the rules with head A. Then the canonical reductant is written as

$$\langle A \leftarrow @(\mathcal{B}_1, \ldots, \mathcal{B}_l), \top \rangle$$

where the operator @ is defined on each tuple $(b_1, \ldots, b_n) \in L^n$ by

$$\dot{@}(b_1, \ldots, b_l) = \sup\{\vartheta_1 \,\dot{\&}_{j_1}\, b_1, \ldots, \vartheta_l \,\dot{\&}_{j_l}\, b_l\}$$

where without loss of generality, we have considered the first rule $\langle A \leftarrow_{j_1} \mathcal{B}_1, \vartheta_1 \rangle$ as the fact $\langle A \leftarrow_{j_1} \top, \vartheta_1 \rangle$. Now, the following chain computes λ:

$$\begin{aligned} G_0 &= A \\ G_1 &= \top \bar{\&} \bar{@}(\top, \mathcal{B}_2, \ldots, \mathcal{B}_l) \\ &\vdots \\ G_r &= \top \bar{\&} \bar{@}(\top, \lambda_2, \ldots, \lambda_l) \end{aligned}$$

therefore:

$$\begin{aligned} \lambda' &= \vartheta_1 \\ &\preceq \sup\{\vartheta_1 \,\dot{\&}_{j_1}\, \top, \ldots, \vartheta_l \,\dot{\&}_{j_l}\, \lambda_l\} \\ &= \dot{@}(\top, \lambda_2, \ldots, \lambda_l) \\ &= \top \,\dot{\&}\, \dot{@}(\top, \lambda_2, \ldots, \lambda_l) \\ &= \lambda \end{aligned}$$

For the inductive step, let us assume that the result is true for all computed answer with chain of computation with length less than n, where $n > 1$.

The first rule applied in the computation of λ' should have been R1; let $\langle A \leftarrow_{j_1} \mathcal{B}_1, \vartheta_1 \rangle$ be the rule used in the application of R1 among all the rules with head A, which we will write $\langle A \leftarrow_{j_i} \mathcal{B}_i, \vartheta_i \rangle$ con $i = 1, \ldots, l$ (without loss of generality we have considered the first one to be selected in the computation of λ'). Then, we have the following chain of e-formulas:

$$\begin{aligned} G_0 &= A \\ G_1 &= \vartheta_1 \bar{\&} \mathcal{B}_1, \\ &\vdots \\ G_n &= \vartheta_1 \bar{\&}\, \mu' \end{aligned}$$

On the other hand, let

$$\langle A \leftarrow @(\mathcal{B}_1, \ldots, \mathcal{B}_l), \top \rangle$$

be the canonical reductant, then the chain of computation for λ is the following:

$$\begin{aligned} G_0 &= A \\ G_1 &= \top \bar{\&} \bar{@}(\mathcal{B}_1, \ldots, \mathcal{B}_l) \\ &\vdots \\ G_r &= \top \bar{\&} \bar{@}(\mu, \lambda_2, \ldots, \lambda_l) \end{aligned}$$

Clearly, the atoms occurring in $\mathcal{B}_1$ have a chain of computation with length less than n, then by the induction hypothesis the computed (sub)answers for them are less or equal than the corresponding greatest computed answers and, therefore $\mu' \leq \mu$, and consequently:

$$\begin{aligned} \lambda' &= \vartheta_1 \,\dot{\&}_{j_1}\, \mu' \\ &\preceq \vartheta_1 \,\dot{\&}_{j_1}\, \mu \\ &\preceq \sup\{\vartheta_1 \,\dot{\&}_{j_1}\, \mu, \vartheta_2 \,\dot{\&}_{j_2}\, \lambda_2 \ldots, \vartheta_l \,\dot{\&}_{j_l}\, \lambda_l\} \\ &= \dot{@}(\mu, \lambda_2, \ldots, \lambda_l) \\ &= \top \,\dot{\&}\, \dot{@}(\mu, \lambda_2, \ldots, \lambda_l) \\ &= \lambda \end{aligned}$$

□

The proposition above is interesting both from a practical and from a theoretical point of view: For the former, it gives an algorithm for, assuming termination, computing the greatest confidence factor for a query to a program; for the latter, it is the key to prove that, under the assumption of continuity of the fix-point semantics, the least fix-point is indeed the greatest computed answer.

Theorem 5 *Given a program $\mathbb{P}$ and an atom A, if λ_A is the greatest computed answer for $\mathbb{P}$ and query $?A$, then $\lambda_A = T^{\omega}_{\mathbb{P}}(\triangle)(A)$.*

Proof: By induction on the length n of the computation of λ_A.

If $n = 1$, then only rules R2 or R3 have been applied. In either case the result is obvious, as stated below:

- If R2 has been applied, then the characterisation of greatest computed answer says that no rule with head A exists in the program $\mathbb{P}$, therefore $T^{\omega}_{\mathbb{P}}(\triangle)(A) = \sup(\varnothing) = \bot = \lambda_A$.
- If R3 has been applied, the canonical reductant for A is a fact $\langle A \leftarrow \top, \vartheta \rangle$, therefore no proper rules with head A exist, and $\vartheta = \sup\{\vartheta_1, \ldots, \vartheta_n\}$, where the ϑ_i are all the confidence values for A. Now, as $T^{\omega}_{\mathbb{P}}(\triangle)$ is a model of the program $\mathbb{P}$ we have that

  $$\vartheta_i \preceq T^{\omega}_{\mathbb{P}}(\triangle)(A)$$

  for $i = 1, \ldots, n$. Furthermore, as it is the minimal model, then $T^{\omega}_{\mathbb{P}}(\triangle)(A)$ is the lower upper bound, that is

  $$T^{\omega}_{\mathbb{P}}(\triangle)(A) = \sup\{\vartheta_1, \ldots, \vartheta_n\} = \lambda_A$$

For the inductive step, let us assume the result is true for all computation with length less than n, with $n > 1$. In this case, the first applied rule is R1.

Let $A \overset{\vartheta}{\leftarrow} @(\mathcal{B}_1, \ldots, \mathcal{B}_l)$ be the canonical reductant for A. Its body will be more conveniently denoted as $@'[B_1, \ldots, B_k]$, where the B_i are the propositional symbols occurring in $@(\mathcal{B}_1, \ldots, \mathcal{B}_l)$.

After the first step in the computation, we get $\vartheta \,\bar{\&}\, \bar{@}'[B_1, \ldots, B_k]$; by the induction hypothesis we know that $\beta_i = T^{\omega}_{\mathbb{P}}(\triangle)(B_i)$, where β_i is the greatest computed answer for the query $?B_i$, for $i = 1, \ldots, k$ and, therefore, we get the following chain of e-formulas for each by using the rules in the chains of computation for each B_i

$$\begin{aligned} G_0 &= A \\ G_1 &= \vartheta \,\bar{\&}\, \bar{@}'[B_1, \ldots, B_k] \\ &\vdots \\ G_r &= \vartheta \,\bar{\&}\, \bar{@}'[\beta_1, \ldots, \beta_k] \end{aligned}$$

therefore, the greatest computed answer has the form

$$\lambda_A = \vartheta \,\dot{\&}\, \dot{@}'(\beta_1, \ldots, \beta_k)$$

Finally, interpreting all the function symbols

in the lattice $\mathfrak{L}$ we have

$$\begin{aligned}
&\vartheta \mathbin{\dot{\&}} \dot{@}'(\beta_1, \ldots, \beta_k) = \\
&\quad = \vartheta \mathbin{\dot{\&}} \dot{@}'(T^{\omega}_{\mathbb{P}}(\triangle)(B_1), \ldots, T^{\omega}_{\mathbb{P}}(\triangle)(B_k)) \\
&\quad = \vartheta \mathbin{\dot{\&}} \widehat{T^{\omega}_{\mathbb{P}}(\triangle)}(@'[B_1, \ldots, B_k]) \\
&\quad = \vartheta \mathbin{\dot{\&}} \widehat{T^{\omega}_{\mathbb{P}}(\triangle)}(@(\mathcal{B}_1, \ldots, \mathcal{B}_l)) \\
&\quad = \vartheta \mathbin{\dot{\&}} \dot{@}(\widehat{T^{\omega}_{\mathbb{P}}(\triangle)}(\mathcal{B}_1), \ldots, \widehat{T^{\omega}_{\mathbb{P}}(\triangle)}(\mathcal{B}_l)) \\
&\quad \overset{(\star)}{=} \sup\{\vartheta \mathbin{\dot{\&}_i} \widehat{T^{\omega}_{\mathbb{P}}(\triangle)}(\mathcal{B}) \mid A \overset{\vartheta}{\leftarrow}_i \mathcal{B} \in \mathbb{P}\} \\
&\quad = T^{\omega}_{\mathbb{P}}(\triangle)(A)
\end{aligned}$$

where the equality $(\star)$ follows from the application of a reductant. □

This is an interesting result, for it shows that the minimal model for a program $\mathbb{P}$ on each propositional symbol A is the greatest computed answer for program $\mathbb{P}$ and query $?A$.

5 Conclusions and future work

We have re-stated the hypothesis required for completeness on multi-adjoint logic programming in terms of termination of a particular procedure. As a consequence, we have new insight about the computational nature of the problem of completeness in the multi-adjoint case. As future work we will further investigate the algebraic properties of the lattice of truth-values which imply completeness.

The approach taken of considering reductants follows traditional techniques of logic programming, and non-determinism is discarded by using reductants. A possible disadvantage of this technique is that the full search space must be traversed (every rule of every atom must be evaluated), although this need not be necessary in many circumstances. It is clear that some evaluation strategies might start by executing non-deterministically the rules for a given atom, and finally the reductant. This joined with some memoizing or tabling technique would not have significant overhead, and could improve performance.

References

[1] 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.

[2] A. Dekhtyar and V. S. Subrahmanian. Hybrid probabilistic programs. *J. of Logic Programming*, 43:187–250, 2000.

[3] M. van Emden and R. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, 1976.

[4] M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *J. of Logic Programming*, 12:335–367, 1992.

[5] J. Medina, M. Ojeda-Aciego, and P. Vojtáš. A completeness theorem for multi-adjoint logic programming. In *Proc. FUZZ-IEEE'01*. The 10th IEEE International Conference on Fuzzy Systems, IEEE Press, 2001.

[6] 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.

[7] 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.

[8] P. Vojtáš. Fuzzy logic programming. *Fuzzy sets and systems*, 124(3):361–370, 2001.