Homogenizing multi-adjoint logic programs*

Jesús Medina, Manuel Ojeda-Aciego
Dept. Matemática Aplicada. Univ. Málaga, Spain
{jmedina,aciego}@ctima.uma.es

Abstract

The concept of *homogeneous* multi-adjoint logic program is introduced, and a procedure to homogenize an arbitrary multi-adjoint logic program is presented. The procedure is proved to preserve models and, moreover, some complexity results are given.

Keywords: Fuzzy Logic Programming

1 Introduction

The fuzzy logic is a powerful mathematical tool for dealing with modelling and control aspects of complex processes, as well as with uncertain, incomplete and/or inconsistent information. The main advantages of fuzzy logic systems are the capability to express nonlinear input/output relationships by a set of qualitative if-then rules, and to handle both numerical data and linguistic knowledge, especially the latter, which is extremely difficult to quantify by means of traditional mathematics.

Multi-adjoint logic programming is a general theory of logic programming which allows the simultaneous use of different implications in the rules and rather general connectives in the bodies; in [4] a continuous fixpoint semantics was introduced. Regarding implementation issues, a neural-based approach to the implementation of the fixpoint semantics of multi-adjoint logic programming has been recently proposed [3] following some ideas from [1].

* This research was partially supported by Spanish DGI project BFM2000-1054-C02-02.

The implementation using neural networks needs some preprocessing of the initial program to transform it in a *homogeneous* program, form of homogeneous rules. These rules represent exactly the simplest type of (proper) rules we can have in our program. In this work, we introduce the concept of homogeneous multi-adjoint logic program, and present a procedure to homogenize an arbitrary multi-adjoint logic program. The procedure is proved to preserve models and, moreover, some complexity results are given.

2 Preliminary definitions

To make this paper as self-contained as possible, the necessary definitions about multi-adjoint structures are included in this section.

The first interesting feature of multi-adjoint logic programs is that a number of different implications are allowed in the bodies of the rules. Formally, the basic definition is given below:

Definition: A tuple $(L, \preceq, \leftarrow_1, \&_1, \ldots, \leftarrow_n, \&_n)$ is said to be a *multi-adjoint lattice* if $\langle L, \preceq\rangle$ is a lattice, and the following items are satisfied:

1. $\langle L, \preceq\rangle$ is bounded;
2. $\top \,\&_i\, \vartheta = \vartheta \,\&_i\, \top = \vartheta$ for all $\vartheta \in L$ and all i;
3. $(\leftarrow_i, \&_i)$ is an *adjoint pair* in $\langle L, \preceq\rangle$ for all i.

Definition: A *multi-adjoint program* is a set of weighted rules $\langle A \leftarrow_i \mathcal{B}, \vartheta\rangle$ satisfying:

1. The *head* A is a propositional symbol.
2. The *body* formula $\mathcal{B}$ is a formula of $\mathfrak{F}$ built from propositional symbols by the use of monotone operators.

3. The *weight* ϑ is an element of L.

Facts are rules with body $\top$ (which usually will not be written).[1]

Definition:

1. An *interpretation* is a mapping I from the set of propositional symbols Π to the lattice $\langle L, \preceq \rangle$.
2. An interpretation I *satisfies* $\langle A \leftarrow_i \mathcal{B}, \vartheta \rangle$ if and only if $\vartheta \preceq \hat{I}(A \leftarrow_i \mathcal{B})$.
3. An interpretation I is a *model* of a multiadjoint logic program $\mathbb{P}$ iff all weighted rules in $\mathbb{P}$ are satisfied by I.

3 Homogeneous programs

Regarding the implementation of the semantics, it is useful to introduce the concept of *homogeneous rules*. These rules represent exactly the simplest type of (proper) rules we can have in our program. In some sense, homogeneous rules allow a straightforward generalization of the standard logic programming framework, in that no operators other than $\leftarrow_i$ and $\&_i$ (and possibly some aggregators) are used.

Definition: A weighted formula is said to be *homogeneous* if it has one of the following forms:

- $\langle A \leftarrow_i B_1 \,\&_i \cdots \&_i\, B_n, \vartheta \rangle$
- $\langle A \leftarrow_i @(B_1, \ldots, B_n), \top \rangle$
- $\langle A \leftarrow_i B_1, \vartheta \rangle$

where $A, B_1, \ldots, B_n$ are propositional symbols.

In the following we present a procedure for transforming a given multiadjoint logic program into a homogeneous one.

3.1 Handling rules

We will state a procedure for transforming a given program into another (equivalent) one containing only facts and homogeneous rules. It is based on two types of transformations: The first one handles the main connective of the body of the rule, whereas the second one handles the subcomponents of the body.

$T1$. A formula $\langle A \leftarrow_i \&_j(\mathcal{B}_1, \ldots, \mathcal{B}_n), \vartheta \rangle$ is substituted by the following pair of formulas:

$$\langle A \leftarrow_i A_1, \vartheta \rangle$$
$$\langle A_1 \leftarrow_j \&_j(\mathcal{B}_1, \ldots, \mathcal{B}_n), \top \rangle$$

where A_1 is a fresh propositional symbol, and $\langle \leftarrow_j, \&_j \rangle$ is an adjoint pair.

For the case $\langle A \leftarrow_i @(\mathcal{B}_1, \ldots, \mathcal{B}_n), \vartheta \rangle$ in which the main connective of the body of the rule happens to be an aggregator, the transformation is similar:

$$\langle A \leftarrow_i A_1, \vartheta \rangle$$
$$\langle A_1 \leftarrow @(\mathcal{B}_1, \ldots, \mathcal{B}_n), \top \rangle$$

where A_1 is a fresh propositional symbol, and $\leftarrow$ is a designated implication.

$T2$. A weighted formula $\langle A \leftarrow_i \Theta(\mathcal{B}_1, \ldots, \mathcal{B}_n), \vartheta \rangle$, where Θ is either $\&_i$ or an aggregator, and a component $\mathcal{B}_k$ is assumed to be either of the form $\&_j(\mathcal{C}_1, \ldots, \mathcal{C}_l)$ or $@(\mathcal{C}_1, \ldots, \mathcal{C}_l)$, is substituted by the following pair of formulas in either case:

$$\langle A \leftarrow_i \Theta(\mathcal{B}_1, \ldots, \mathcal{B}_{k-1}, A_1, \mathcal{B}_{k+1}, \ldots, \mathcal{B}_n), \vartheta \rangle$$
$$\langle A_1 \leftarrow_j \&_j(\mathcal{C}_1, \ldots, \mathcal{C}_l), \top \rangle$$

or

$$\langle A \leftarrow_i \Theta(\mathcal{B}_1, \ldots, \mathcal{B}_{k-1}, A_1, \mathcal{B}_{k+1}, \ldots, \mathcal{B}_n), \vartheta \rangle$$
$$\langle A_1 \leftarrow @(\mathcal{C}_1, \ldots, \mathcal{C}_l), \top \rangle$$

The procedure to transform the rules of a program so that all the resulting rules are homogeneous, is presented in Fig. 1. It is based in the two previous transformations, and in its description by abuse the notation we use the terms T1-rule (resp. T2-rule) to mean an adequate input rule for transformation T1 (resp. T2):

Some applications of the algorithm below are presented in the examples below:

[1] We will consider one *designated implication* to be used for the representation of facts, which is denoted $\leftarrow$. This designated implication will be also used in the procedure of translation of a program into a homogeneous one.

```
Program Homogenization
  begin
    repeat
      for each T1-rule do
        Apply transformation T1
      end-for
      for each T2-rule do
        Apply transformation T2
      end-for
    until neither T1- nor T2-rules exist
  end
```

Figure 1: Pseudo-code for the procedure.

Example 1 *Consider the following T1-rule* $\langle A \leftarrow_P (B_1 \,\&_P\, B_2) \,\&_G\, B_3, \vartheta\rangle$ *(note that the main connective in the body is not the adjoint conjunctor to the implication). A first step of the previous algorithm gives:*

$$\langle A \leftarrow_P A_1, \vartheta\rangle \quad \textit{Homogeneous}$$
$$\langle A_1 \leftarrow_G (B_1 \,\&_P\, B_2) \,\&_G\, B_3, \top\rangle$$

Now, the second rule has to be modified, and the result is given below:

$$\langle A \leftarrow_P A_1, \vartheta\rangle \quad \textit{Homogeneous}$$
$$\langle A_1 \leftarrow_G A_2 \,\&_G\, B_3, \top\rangle \quad \textit{Homogeneous}$$
$$\langle A_2 \leftarrow_P B_1 \,\&_P\, B_2, \top\rangle \quad \textit{Homogeneous}$$

Example 2 *Consider the rule*

$$\langle A \leftarrow_P (B_1 \,\&_G\, B_2) \,\&_P\, @(B_3, B_4), \vartheta\rangle$$

The first step of the algorithm gives

$$\langle A \leftarrow_P A_1 \,\&_P\, @(B_3, B_4), \vartheta\rangle$$
$$\langle A_1 \leftarrow_G B_1 \,\&_G\, B_2, \top\rangle \quad \textit{Homogeneous}$$

The procedure continues with the first rule above

$$\langle A \leftarrow_P A_1 \,\&_P\, A_2, \vartheta\rangle \quad \textit{Homogeneous}$$
$$\langle A_2 \leftarrow @(B_3, B_4), \top\rangle \quad \textit{Homogeneous}$$
$$\langle A_1 \leftarrow_G B_1 \,\&_G\, B_2, \top\rangle \quad \textit{Homogeneous}$$

The idea of including new symbols and definitions for these symbols is a reformulation and adaptation of the technique introduced originally in the context of automated deduction in [5]. The original aim of this technique was to obtain a structure-preserving transformation of a formula into clause form.

3.2 Handling facts

After the exhaustive application of the previous procedure we can assume that all our rules are homogeneous. Regarding facts, it might be possible that the program contained facts about the same propositional symbol but with different weights.

Assume all the facts about A are

$$\langle A \leftarrow \top, \vartheta_j\rangle \qquad j \in \{1, \ldots, l\}$$

then, the following fact is substituted for the previous ones

$$\langle A \leftarrow \top, \sup\{\vartheta_j \mid j \in \{1, \ldots, l\}\rangle$$

the computed truth-value for A will be denoted ϑ_A.

The new program obtained from $\mathbb{P}$ after the homogenization of rules and facts is denoted $\mathbb{P}^*$. Note that in this new program there are new propositional symbols, if Π is the set of propositional symbols occurring in $\mathbb{P}$, then the set of propositional symbols occurring in $\mathbb{P}^*$ is denoted Π^*; obviously $\Pi \subseteq \Pi^*$.

3.3 Preservation of the semantics

It is necessary to check that the semantics of the initial program has not been changed by the transformation. The following results will show that every model of $\mathbb{P}^*$ is also a model of $\mathbb{P}$ and, in addition, the minimal model of $\mathbb{P}^*$ is also the minimal model of $\mathbb{P}$.

Theorem 1 *A model of* $\mathbb{P}^*$ *is also a model of* $\mathbb{P}$.

Proof: It will be sufficient to show that the two transformations T1 and T2 have this property; that is, every model of the output of the rules is also a model of the input of the transformation.

We will give only the prove for transformation T1, since for T2 the idea is similar. Assume that I is a model of the rules

$$\langle A \leftarrow_i A_1, \vartheta\rangle$$
$$\langle A_1 \leftarrow_j \&_j(\mathcal{B}_1, \ldots, \mathcal{B}_n), \top\rangle$$

therefore we have

$$\vartheta \,\dot{\&}_i\, I(A_1) \preceq I(A)$$
$$\hat{I}(\&_j(\mathcal{B}_1, \ldots, \mathcal{B}_n)) \preceq I(A_1)$$

Now, by monotonicity, we have

$$\vartheta \,\dot{\&}_i\, \hat{I}(\&_j(\mathcal{B}_1, \ldots, \mathcal{B}_n)) \preceq I(A)$$

that is, I is a model of $\langle A \leftarrow_i \&_j(\mathcal{B}_1, \ldots, \mathcal{B}_n), \vartheta\rangle$.

The case of an aggregator as the main connective of the body is similar. □

Theorem 2 *The minimal model of $\mathbb{P}^*$ when restricted to the variables in Π is also the minimal model of $\mathbb{P}$.*

Proof: (Sketch) Assume any model I of $\mathbb{P}$, then extend it to Π^* in such a way that it is also a model of $\mathbb{P}^*$, then use minimality on $\mathbb{P}^*$. □

4 Complexity of the transformation

In this section we show that the complexity of the homogenizing procedure is linear.

Theorem 3 *Let $\langle A \leftarrow_i \Theta(\mathcal{B}_1, \ldots, \mathcal{B}_l), \vartheta\rangle$ be a rule with n connectives in the body ($n \geq 1$). Then we have the following affirmations:*

- *The number of homogeneous rules obtained, after applying the procedure is: n if $\Theta = \&_i$ or $\Theta = @$ with $\vartheta = \top$; and $n+1$ otherwise.*
- *The number of transformations obtained, after applying the procedure is: $n-1$ if $\Theta = \&_i$ or $\Theta = @$ with $\vartheta = \top$; and n otherwise.*

Proof: By induction on n: If $n = 1$, we have just two straightforward cases.

Now, we assume the result true for all rule with $k < n$ connectives in the body, and we must prove it for n connectives.

- If $\Theta = \&_i$, or $\Theta = @$ and $\vartheta = \top$, we must apply the transformation $T2$ and we obtain

  $$\langle A \leftarrow_i \Theta(\mathcal{B}_1, \ldots, \mathcal{B}_{k-1}, A_1, \mathcal{B}_{k+1}, \ldots, \mathcal{B}_n), \vartheta\rangle$$
  $$\langle A_1 \leftarrow_j \Theta'(\mathcal{C}_1, \ldots, \mathcal{C}_l), \top\rangle$$

  where $\leftarrow_j$ depends on the connective Θ'. But in both cases, in the body of the second rule, there are i connectives with $i \geq 1$, and in the body of the first rule, there are $n-i < n$ connectives. Thus, we can apply the induction hypothesis and, finally, the number resultant of rules is $i + (n-i) = n$.

- Otherwise, we use $T1$ to obtain

  $$\langle A \leftarrow_i A_1, \vartheta\rangle$$
  $$\langle A_1 \leftarrow_j \Theta(\mathcal{B}_1, \ldots, \mathcal{B}_n), \top\rangle$$

  where if Θ is an aggregator, then $\leftarrow_j$ is the designated implication, and if $\Theta = \&_j$, then $\leftarrow_j$ is its adjoint implication.

  Therefore, similarly to the previous case, the final number of rules is n and the first one is homogeneous, so the final number is $n+1$.

Similarly, we can prove the other affirmation. □

5 Conclusions and future work

A procedure for homogenizing a multi-adjoint program has been introduced. This procedure is used in [3] as a preprocessing step of the implementation of the fixpoint semantics of multi-adjoint programs by using ideas borrowed from neural networks. Future work in this research line is oriented to further developing the use of the neural-like implementation of the multi-adjoint framework in the area of abductive reasoning [2].

References

[1] P. Eklund and F. Klawonn, "Neural fuzzy logic programming," *IEEE Tr. on Neural Networks*, vol. 3, no. 5, pp. 815–818, 1992.

[2] J. Medina, E. Mérida-Casermeiro, and M. Ojeda-Aciego. A neural approach to abductive multi-adjoint reasoning. Lect. Notes in Computer Science 2443, pages 213–222, 2002.

[3] J. Medina, E. Mérida-Casermeiro, and M. Ojeda-Aciego. A neural approach to extended logic programs. Lect. Notes in Computer Science, 2003. To appear.

[4] J. Medina, M. Ojeda-Aciego, and P. Vojtáš. Multi-adjoint logic programming with continuous semantics. Lect. Notes in Artificial Intelligence 2173, pages 351–364, 2001.

[5] D. A. Plaisted and S. Greenbaum, "A structure-preserving clause form translation," *Journal of Symbolic Computation* 2(1):293–304, 1986.