# A Neural Implementation of Multi-Adjoint Logic Programming

J. Medina, E. Mérida-Casermeiro, M. Ojeda-Aciego

*Dept. Matemática Aplicada. Universidad de Málaga*
{merida,jmedina,aciego}@ctima.uma.es [1]

**Abstract**

We present a neural net based implementation of propositional $[0,1]$-valued multi-adjoint logic programming. The implementation needs some preprocessing of the initial program to transform it in a *homogeneous* program; then, transformation rules carry programs into neural networks, where truth-values of rules relate to output of neurons, truth-values of facts represent input, and network functions are determined by a set of general operators; the output of the net being the values of propositional variables under its minimal model.

## 1 Introduction

Fuzzy logic is a powerful mathematical tool for dealing with modeling and control aspects of complex processes, as well as with uncertain, incomplete and/or inconsistent information. On the other hand, neural networks have a massively parallel architecture-based dynamics which are inspired by the structure of human brain, adaptation capabilities, and fault tolerance.

The main advantages of fuzzy logic systems are the capability to express non-linear 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. The main advantage of neural networks, on the other hand, is the inherent learning capability, which enables the networks to adaptively improve their performance. The key properties of neuro-fuzzy networks are the accurate learning and adaptive capabilities of the neural networks, together with the generalization and fast learning capabilities of fuzzy logic systems.

[1] Partially supported by Spanish DGI project BFM2000-1054-C02-02.

Numerous examples of synergistic fuzzy neural network models have been proposed in the literature [3,6,11,17]: the fusion problem of neural networks and fuzzy logic is addressed mainly through two types of methods: one is to substitute membership functions or fuzzy operations for active functions of neurons; another way is the introduction of the fuzzified input data. Most models of those integrated networks are just a regular fuzzy neural network [3] with real number inputs and fuzzy weighting factors, which is a typical fuzzy logic controller implemented by neural networks.

In this work, we introduce a hybrid approach to handling uncertainty, which is *expressed* in the rich language of multi-adjoint logic but is *implemented* by using ideas borrowed from the world of neural networks.

The handling of uncertainty inside our logic model is based on the use of a generalised set of truth-values, usually a (finite or infinite) subset of the real unit interval $[0, 1]$, instead of the Boolean constants $\{v, f\}$, in this respect it is an approach to fuzzy logic programming which, in particular, extends [22]. On the other hand, multi-adjoint logic programming [14] is a generalization of residuated logic programming [4] in that several different implications are allowed in the same program. In this respect, as far as we are concerned, the only existing approach which allows for more than one type of implication in the programs is [1], although there are considerable differences in the underlying logic used: their logic of bunched implications was developed as an attempt to obtain a direct decomposition of implication in absence of structural rules in linear logic, similar to the splitting of conjunction into an additive and a multiplicative part, whereas multi-adjoint logic builds on the framework of fuzzy and residuated logic, as a means to facilitate the task of the specification.

Considering several implications in the same program is interesting because it provides a more flexible framework for the specification of problems, for instance, in situations in which connectives are built from the users preferences. In these contexts, it is likely that knowledge is described by a many-valued logic program where connectives have many-valued truth functions and, perhaps, aggregation operators (such as arithmetic mean or weighted sum) where different implications could be needed for different purposes, and different aggregators are defined for different users, depending on their preferences.

The long-term goal of our research is to develop a multi-adjoint approach to abductive logic programming. In this paper, following ideas in [16], we present a neural-like implementation of multi-adjoint logic programming with the advantage that, at least potentially, we can calculate in parallel the answer for any query.

An important point of the implementation is a preprocessing of the initial program to transform it into a *homogeneous* program; the ideas under this definition are based on [6]. For simplicity in the presentation we will only describe the implementation for $[0, 1]$-valued programs, although the general framework of multi-adjoint logic is lattice-valued.

The structure of the paper is as follows: In Section 2, the preliminary definitions are introduced, together with the syntax and semantics of multi-adjoint logic programs; in Section 3, the translation from multi-adjoint programs into homogeneous programs is given, the preservation of the semantics is proved, and the complexity of the translation is studied; then, in Section 4, the model of neural network is described; it is in Section 5 where the translation from homogeneous programs to neural net is given. The paper finishes with some examples in Section 6, and then some conclusions are drawn.

## 2 Preliminary definitions

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; a preliminar version was presented in [14], where models of these programs were proved to be post-fixpoints of the immediate consequences operator, which turned out to be monotonic under very general hypotheses. In addition, the continuity of the immediate consequences operator was studied, and some sufficient conditions for its continuity were obtained. A procedural semantics, under these conditions, for multi-adjoint logic programs, together its completeness result was given in [15].

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 [14].

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

**Definition 1** *Let $\langle L, \preceq\rangle$ be a lattice. A* multi-adjoint lattice *$\mathcal{L}$ is a tuple $(L, \preceq, \leftarrow_1, \&_1, \dots, \leftarrow_n, \&_n)$ 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 $i = 1, \dots, n$;*
*(3) $(\leftarrow_i, \&_i)$ is an* adjoint pair *in $\langle L, \preceq\rangle$ for $i = 1, \dots, n$; 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*

$(x \mathbin{\&_i} z) \preceq y$ *holds.*

The need of the monotonicity of operators $\leftarrow_i$ and $\&_i$ is clear, if they are to be interpreted as generalised implications and conjunctions. The third property in the definition, corresponds to the categorical adjointness; but can be adequately interpreted in terms of multiple-valued inference as asserting that the truth-value of $y \leftarrow_i z$ is the maximal $x$ satisfying $x \mathbin{\&_i} z \preceq y$, and also the validity of the following generalised modus ponens rule [8,7]:

> If $x$ is a lower bound of $\psi \leftarrow_i \varphi$, and $z$ is a lower bound of $\varphi$ then a lower bound $y$ of $\psi$ is $x \mathbin{\&_i} z$.

Although, the multi-adjoint paradigm was developed for multi-adjoint lattices, for the sake of simplicity, in this specific implementation we will restrict our attention to $[0, 1]$ with its standard ordering $\leq$.

**Example 1** *The three pairs of connectives* $(\leftarrow_P, \&_P)$, $(\leftarrow_G, \&_G)$, $(\leftarrow_L, \&_L)$ *given below are adjoint pairs on the real unit interval, which are called, respectively,* product*,* Gödel *and* Łukasiewicz *connectives:*

$$x \leftarrow_P y = \begin{cases} x/y & \textit{if } y > x \\ 1 & \textit{otherwise} \end{cases}; \qquad x \mathbin{\&_P} y = x \cdot y$$

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

$$x \leftarrow_L y = \min(1 - y + x, 1); \qquad x \mathbin{\&_L} y = \max(0, x + y - 1)$$

**Definition 2** *A* multi-adjoint program *is a set of weighted rules* $\langle F, \vartheta \rangle$ *satisfying the following conditions:*

*(1)* *$F$ is a formula of the form $A \leftarrow_i \mathcal{B}$ where $A$ is a propositional symbol called the* head *of the rule, and $\mathcal{B}$ is a well-formed formula built from propositional symbols $B_1, \dots, B_n$ ($n \geq 0$) by the use of monotone operators, which is called the* body *formula.*
*(2)* *The* weight *$\vartheta$ is an element (a truth-value) of* $[0, 1]$.

Facts *are rules with body*[2] 1 *and a* query *(or* goal*) is a propositional symbol intended as a question $?A$ prompting the system.*

Regarding the implementation as a neural network, to be introduced later, it will be useful to introduce the *homogeneous rules*, in order to provide a simpler and standard representation for any multi-adjoint program.

[2] It is also customary to use write $\top$ instead of 1, and even not to write any body.

**Definition 3** *A weighted rule is said to be* homogeneous *if it has one of the following forms:*

- $\langle A \leftarrow_i \&_i(B_1, \ldots B_n), \vartheta\rangle$
- $\langle A \leftarrow_i @(B_1, \ldots, B_n), 1\rangle$
- $\langle A \leftarrow_i B_1, \vartheta\rangle$

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

In this definition of homogeneous rules, we consider the last form to be different from the first, although formally the latter is a particular case of the former. The reason to do so is related to the number of steps of computation generated by each of them; this fact will be clarified in the following section.

The homogeneous 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 $\&_i$ are used. The purpose of the next section is to give a procedure for translating a multi-adjoint logic program into one containing only homogeneous rules.

Once we have given the syntax of our programs, the semantics is given below.

**Definition 4** *An* interpretation *is a mapping $I$ from the set of propositional symbols $\Pi$ to the lattice $\langle[0, 1], \leq\rangle$.*

Note that each of these interpretations can be uniquely extended to the whole set of formulas, and this extension is noted as $\hat{I}$. The set of all the interpretations is denoted $\mathcal{I}_{\mathfrak{L}}$.

The ordering $\leq$ of the truth-values $L$ can be easily extended to $\mathcal{I}_{\mathfrak{L}}$, which also inherits the structure of complete lattice and is denoted $\sqsubseteq$. The minimum element of the lattice $\mathcal{I}_{\mathfrak{L}}$, which assigns 0 to any propositional symbol, will be denoted $\triangle$.

**Definition 5**

*(1) An interpretation $I \in \mathcal{I}_{\mathfrak{L}}$* satisfies *$\langle A \leftarrow_i \mathcal{B}, \vartheta\rangle$ if and only if $\vartheta \leq \hat{I}\,(A \leftarrow_i \mathcal{B})$.*
*(2) 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 $I$.*
*(3) An element $\lambda \in L$ is a* correct answer *for a program $\mathbb{P}$ and a query $?A$ if for any interpretation $I \in \mathcal{I}_{\mathfrak{L}}$ which is a model of $\mathbb{P}$ we have $\lambda \leq I(A)$.*

The operational approach to multi-adjoint logic programs used in this paper will be based on the fixpoint semantics provided by the immediate consequences operator, given by van Emden and Kowalski [21], which can be generalised to the framework of multi-adjoint logic programs by means of the adjoint property, as shown below:

**Definition 6** *Let $\mathbb{P}$ be a multi-adjoint program. The* immediate consequences operator*, $T_{\mathbb{P}}\colon \mathcal{I}_{\mathfrak{L}} \to \mathcal{I}_{\mathfrak{L}}$, maps interpretations to interpretations, and for $I \in \mathcal{I}_{\mathfrak{L}}$ and $A \in \Pi$ is defined by*

$$T_{\mathbb{P}}(I)(A) = \sup\Big\{\vartheta \,\&_i\, \hat{I}(\mathcal{B}) \mid \langle A \leftarrow_i \mathcal{B}, \vartheta\rangle \in \mathbb{P}\Big\}$$

As usual, it is possible to characterise the semantics of a multi-adjoint logic program by the post-fixpoints of $T_{\mathbb{P}}$; that is, an interpretation $I$ is a model of a multi-adjoint logic program $\mathbb{P}$ iff $T_{\mathbb{P}}(I) \sqsubseteq I$. The $T_{\mathbb{P}}$ operator is proved to be monotonic and continuous under very general hypotheses, see [15], and it is remarkable that these results are true even for non-commutative and non-associative conjunctors, although these facts will not be stressed here.

Regarding continuity, the result below was proved in [14].

**Theorem 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.*

Once we know that $T_{\mathbb{P}}$ can be continuous under very general hypotheses, then the least model can be reached in at most countably many iterations beginning with the least interpretation, that is, the least model is $T_{\mathbb{P}} \uparrow \omega(\triangle)$.

## 3 Obtaining a homogeneous program

In order to provide a simpler and standard neural network representation, in this section we present a translation method for transforming a multi-adjoint program into another consisting solely of homogeneous rules,.

### *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, and the second one handles the subcomponents of the body.

$T1$. A weighted rule $\langle A \leftarrow_i \&_j(\mathcal{B}_1, \dots, \mathcal{B}_n), \vartheta\rangle$, in which the implication of the rule and the conjunction in the body do not form an adjoint pair, is substituted by the following pair of formulas:

$$\langle A \leftarrow_i A_1, \vartheta\rangle$$
$$\langle A_1 \leftarrow_j \&_j(\mathcal{B}_1,\dots,\mathcal{B}_n), 1\rangle$$

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

For the case $\langle A \leftarrow_i @(\mathcal{B}_1,\dots,\mathcal{B}_n), \vartheta\rangle$ in which the main connective of the body of the rule happens to be an aggregator, and $\vartheta \neq 1$ the transformation is similar:

$$\langle A \leftarrow_i A_1, \vartheta\rangle$$
$$\langle A_1 \leftarrow_i @(\mathcal{B}_1,\dots,\mathcal{B}_n), 1\rangle$$

where $A_1$ is a fresh propositional symbol.

*T*2. A weighted rule $\langle A \leftarrow_i \Theta(\mathcal{B}_1,\dots,\mathcal{B}_n), \vartheta\rangle$, where $\Theta$ is either $\&_i$ or an aggregator, and a component $\mathcal{B}_k$ is assumed to be either of the form $\&_j(\mathcal{C}_1,\dots,\mathcal{C}_l)$ or $@(\mathcal{C}_1,\dots,\mathcal{C}_l)$, is substituted by the following pair of formulas in either case:

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

or

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

where $A_1$ is a fresh propositional symbol.

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 above are presented in the examples below, in which we are using the more standard infix notation for the conjunctions in the body of the rules:

**Example 2** *Consider $\langle A \leftarrow_P (B_1 \&_P B_2) \&_G B_3, \vartheta\rangle$, which is a T1-rule, for the main connective in the body is not the adjoint conjunctor to the implication. A first step of the previous algorithm gives:*

$$\langle A \leftarrow_P A_1, \vartheta\rangle \quad \textit{Homogeneous}$$
$$\langle A_1 \leftarrow_G (B_1 \&_P B_2) \&_G B_3, 1\rangle$$

```
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-rules nor T2-rules exist
  end
```

Fig. 1. Pseudo-code for translating into a homogeneous program.

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

$$\begin{array}{rl}
\langle A \leftarrow_P A_1, \vartheta \rangle & \textit{Homogeneous} \\
\langle A_1 \leftarrow_G A_2 \,\&_G\, B_3, 1 \rangle & \textit{Homogeneous} \\
\langle A_2 \leftarrow_P B_1 \,\&_P\, B_2, 1 \rangle & \textit{Homogeneous}
\end{array}$$

**Example 3** *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 the following result*

$$\begin{array}{l}
\langle A \leftarrow_P A_1 \,\&_P\, @(B_3, B_4), \vartheta \rangle \\
\qquad\quad \langle A_1 \leftarrow_G B_1 \,\&_G\, B_2, 1 \rangle \;\; \textit{Homogeneous}
\end{array}$$

*The procedure continues with the first rule above*

$$\begin{array}{rl}
\langle A \leftarrow_P A_1 \,\&_P\, A_2, \vartheta \rangle & \textit{Homogeneous} \\
\langle A_2 \leftarrow @(B_3, B_4), 1 \rangle & \textit{Homogeneous} \\
\langle A_1 \leftarrow_G B_1 \,\&_G\, B_2, 1 \rangle & \textit{Homogeneous}
\end{array}$$

The idea of including new symbols and definitions (we will use this term in the sequel) for these symbols is a reformulation and adaptation of the technique introduced originally in the context of automated deduction in [19]. 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

$$\langle A \leftarrow 1, \vartheta_j \rangle \quad j \in \{1, \dots, l\}$$

then, the following fact is substituted for the previous ones

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

the computed truth-value for $A$ will be denoted $\vartheta_A$.

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 $\Pi$ is the set of propositional symbols occurring in $\mathbb{P}$, then the set of propositional symbols occurring in $\mathbb{P}^*$ is denoted $\Pi^*$; 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 2** *Every model of $\mathbb{P}^*$ when restricted to variables in $\Pi$ 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.

T1. Assume that $I$ is a model of the rules

$$\langle A \leftarrow_i A_1, \vartheta \rangle \quad \text{and} \quad \langle A_1 \leftarrow_j \&_j(\mathcal{B}_1, \dots, \mathcal{B}_n), 1 \rangle$$

therefore we have

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

Now, by monotonicity, we have

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

which means that $I$ 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.

T2. Now, assume that $I$ is a model of the pair of rules

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

then we have

$$\begin{aligned} &\vartheta \,\&_i\, \hat{I}(\Theta(\mathcal{B}_1, \ldots, \mathcal{B}_{k-1}, A_1, \mathcal{B}_{k+1}, \ldots, \mathcal{B}_n)) \preceq I(A) \\ &\qquad\qquad\qquad\qquad \hat{I}(\&_j(\mathcal{C}_1, \ldots, \mathcal{C}_l)) \preceq \hat{I}(A_1) \end{aligned}$$

Now, by monotonicity, recalling that $\mathcal{B}_k$ was assumed to be $\&_j(\mathcal{C}_1, \ldots, \mathcal{C}_l)$, and the definition of $\hat{I}$ we have

$$\vartheta \,\&_i\, \hat{I}(\Theta(\mathcal{B}_1, \ldots, \mathcal{B}_{k-1}, \mathcal{B}_k, \mathcal{B}_{k+1}, \ldots, \mathcal{B}_n)) \preceq I(A)$$

that is, $I$ is a model of

$$\langle A \leftarrow_i \Theta(\mathcal{B}_1, \ldots, \mathcal{B}_{k-1}, \mathcal{B}_k, \mathcal{B}_{k+1}, \ldots, \mathcal{B}_n), \vartheta\rangle$$

In the case of an aggregator, the proof is similar. □

**Theorem 3** *The minimal model of* $\mathbb{P}^*$ *when restricted to the variables in* $\Pi$ *is also the minimal model of* $\mathbb{P}$.

**PROOF.**

The structure of the proof is as follows: Assume any model $I$ of $\mathbb{P}$, then extend it to $\Pi^*$ in such a way that it is also a model of $\mathbb{P}^*$, then use minimality on $\mathbb{P}^*$.

Let $M^*$ be the minimal model of $\mathbb{P}^*$, and let us denote by $M$ its restriction to $\mathbb{P}$. By the previous theorem we have that $M$ is also a model of $\mathbb{P}$, so let us prove that it is minimal.

The intuition after the definition of the extension is the following: for a given rule the transformations T1 and T2 introduce a finite number of fresh propositional symbols and *definitions* for these symbols. The procedure ends when the new symbol get defined completely in terms of propositional symbols from $\Pi$. This feature allows for extending any model $I$ to these new symbols in a recursive manner. For the sake of readability consider that the new symbols are denoted by $A_i$.

Given a model $I$ of $\mathbb{P}$, consider a definition $\langle A_k \leftarrow_j \mathcal{B}, 1\rangle$, which makes sense because there is only one rule headed with $A_i$ for each fresh symbol $A_i$ introduced in the program:

(1) If all the propositional symbols in $\mathcal{B}$ are in $\Pi$, define:

$$I^*(A_k) = \hat{I}(\mathcal{B})$$

(2) If the body contains some defined symbol, define

$$I^*(A_k) = \widehat{I^*}(\mathcal{B})$$

Clearly, this extension $I^*$ is also a model of $\mathbb{P}^*$, therefore the minimal model $M^*$ of $\mathbb{P}^*$ satisfies $M^* \sqsubseteq I^*$. Now, by restricting the domain of the models to $\Pi$ we obtain $M \sqsubseteq I$. Therefore, $M$ is the minimal model of $\mathbb{P}$. $\square$

## *3.4 Complexity*

In this section it is shown that the complexity of the algorithm for transforming a multi-adjoint program into a homogeneous one is linear on the size of the program.

**Theorem 4** *Let $\langle A \leftarrow_i \Theta(\mathcal{B}_1, \dots, \mathcal{B}_l), \vartheta\rangle$ 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 either $\Theta = \&_i$ or $\Theta = @$ with $\vartheta = 1$; and $n+1$ otherwise.*
- *The number of transformations obtained after applying the procedure is $n-1$ if either $\Theta = \&_i$ or $\Theta = @$ with $\vartheta = 1$; and $n$ otherwise.*

**PROOF.** By induction in the number $n$ of connectives in the body.

If $n = 1$, we must consider two cases:

- If $\Theta = \&_i$, or $\Theta = @$ and $\vartheta = 1$, the rule is homogeneous, so the final number of rules is 1.
- Otherwise, we apply only the transformation $T1$, and we obtain the rules

$$\langle A \leftarrow_i A_1, \vartheta\rangle \quad \text{and} \quad \langle A_1 \leftarrow_j \Theta(\mathcal{B}_1, \dots, \mathcal{B}_n), 1\rangle$$

where $\leftarrow_j$ depends on $\Theta$. But these are two homogeneous rules, because in the body there is only one connective. Thus, the final number of rules is 2.

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

- If $\Theta = \&_i$, or $\Theta = @$ and $\vartheta = 1$, we must apply the transformation $T2$ and we obtain the rules

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

  where $\leftarrow_j$ depends on the connective $\Theta'$. But in both cases, in the body of the second rule, there are $c$ connectives with $c \geq 1$, and in the body of the first rule, there are $n - c < n$ connectives. Thus, we can apply the induction hypothesis and, finally, the number resultant of rules is $c + (n - c) = n$.
- Otherwise, we use the transformation $T1$ and we have the rules

  $$\langle A \leftarrow_i A_1, \vartheta\rangle \quad \text{and} \quad \langle A_1 \leftarrow_j \Theta(\mathcal{B}_1, \dots, \mathcal{B}_n), 1\rangle$$

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

  Consequently, by a similar reasoning to the previous case, the final number of rules is $n$ and the first rule is homogeneous, so the final number of rules is $n + 1$.

  Similarly, we can prove the other statement. □

In the next section we present a model of network which allows to evaluate the $T_{\mathbb{P}}$ operator and, therefore, by iteration will be able to approximate the actual values of the least model up to any prescribed precision.

## 4 On the type of the network

Using neural networks in the context of logic programming is not a completely novel idea; for instance, in [6] it is shown how fuzzy logic programs can be transformed into neural nets, where adaptations of uncertainties in the knowledge base increase the reliability of the program and are carried out automatically.

Regarding the approximation of the semantics of logic programs, in [9] the fixpoint of the $T_{\mathbb{P}}$ operator for a certain class of classical propositional logic programs (called acyclic logic programs) is constructed by using a 3-layered recurrent neural network, as a means of providing a massively parallel computational model for logic programming; this result is later extended in [10] to deal with the first order case.

Our approach somehow tries to join the two approaches above, and it is interesting since our logic is much richer than classical or the usual versions of fuzzy logic in the literature, although we only consider the propositional case. Although there are some results regarding the expressive power of feed-forward multilayered neural nets, such as Kurková's theorem [12], the structure of our net is not described as an $n$-layered network, instead a more straightforward approach is used.

Before describing the model of neural net chosen to implement the immediate consequences operator $T_{\mathbb{P}}$ for multi-adjoint logic programming, some considerations are needed:

Firstly, the set of operators to be implemented will consist of the three most important adjoint pairs on the real unit interval: product $(\&_P, \leftarrow_P)$, Gödel $(\&_G, \leftarrow_G)$ and Łukasiewicz $(\&_L, \leftarrow_L)$, defined in Example 1. Regarding the selection of operators implemented, just recall that every continuous triangular norm (or $t$-norm), which is the type of conjunctor more commonly used in the context of fuzzy reasoning, is expressible as an ordinal sum of these three basic conjunctors [7]. Regarding the aggregation operators, we will implement a family of weighted sums, which are denoted $@_{(n_1,\dots,n_m)}$ and defined as follows:

$$@_{(n_1,\dots,n_m)}(p_1,\dots,p_m) = \frac{n_1p_1 + \cdots + n_mp_m}{n_1 + \cdots + n_m}$$

Now, we can properly begin the description: A neural network will be considered in which each process unit is associated to either a propositional symbol of the initial program or an homogeneous rule of the transformed program. The state of the $i$-th neuron in the instant $t$ is expressed by its output $S_i(t)$. Therefore, the state of the network can be expressed by means of a state vector $\vec{S}(t)$, whose components are the output of the neurons forming the network, and its initial state is:

$$S_A(0) = \begin{cases} \vartheta_A & \text{if } \langle A \leftarrow \top, \vartheta_A\rangle \in \mathbb{P}, \\ 0 & \text{otherwise.} \end{cases}$$

where $S_A(0)$ denotes the component associated to a propositional symbol $A$.

Regarding the user interface, there are two types of neurons, visible or hidden, the output of the visible neurons is part of the overall output of the net, and the output of the hidden neurons is only used as input values for other neurons. Visible neurons are associated to propositional symbols of the initial program and the hidden neurons are those associated to definitions and homogeneous rules. This is because we are interested in the values of the minimal model, that is the consequence operator $T_{\mathbb{P}}^{\omega}(\Delta)$, about each propositional symbol of the initial program.

The connection between neurons is denoted by a matrix of weights $W$, in which $w_{kj}$ indicates the existence or absence of connection between unit $k$ and unit $j$; if the neuron represents a weighted sum, then the matrix of weights also represents the weights associated to any of the inputs. The weights of the connections related to neuron $i$ (that is, the $i$-th row of the matrix $W$) are represented by $\vec{w}_{i\bullet}$, and are allocated in an internal vector register of the neuron, which can be seen as a distributed information system.

The initial truth-value of the propositional symbol or homogeneous rule $v_i$ is loaded in the internal register, together with a signal $m_i$ for distinguishing whether the neuron is associated to either a fact or a rule; in the latter case, information about the type of operator is also included. Therefore, we have two vectors: one storing the truth-values $\vec{v}$ of atoms and homogeneous rules, and another $\vec{m}$ storing the type of the neurons in the net.

The signal $m_i$ indicates the functioning mode of the neuron. If $m_i = 1$, then the neuron is assumed to be associated to a propositional symbol, and its next state is the maximum value among all the operators involved in its input, its previous state, and the initial truth-value $v_i$. More precisely:

$$S_i(t+1) = \max\left\{v_i, \max_k\{S_k(t) \mid w_{ik} > 0\}\right\}$$

When a neuron is associated to the product, Gödel, or Łukasiewicz implication, respectively, then the signal $m_i$ is set to 2, 3, and 4, respectively. Its input is formed by the external value $v_i$ of the rule, and the outputs of the neurons associated to the body of the implication.

The output of the neuron mimics the behaviour of the implication in terms of the adjoint property when a rule of type $m_i$ has been used; specifically, the output in the next instant will be:

- Product implication, $m_i = 2$

$$S_i(t+1) = v_i \prod_{k/w_{ik}>0} S_k(t)$$

- Gödel implication, $m_i = 3$

$$S_i(t+1) = \min\left\{v_i, \min_k\{S_k(t) \mid w_{ik} > 0\}\right\}$$

- Łukasiewicz implication, $m_i = 4$

$$S_i(t+1) = \max\{v_i + \sum_{k/w_{ik}>0} (S_k(t) - 1), 0\}$$

Neurons associated to aggregation operators have signal $t_i = 5$, and its output is

$$S_i(t+1) = \sum_{k/w_{ik}>0} w'_{ik} S_k(t)$$

where

$$w'_{ik} = \frac{w_{ik}}{\sum\limits_{r/w_{ir}>0} w_{ir}}$$

It is important to note that the neurons's output is never decreasing, as shown in the following theorem.

**Theorem 5** *Operators $S_i$ are non-decreasing for all $i$.*

**PROOF.** We proceed by induction.

For $t = 0$, if $m_i \neq 1$ we have that $S_i(0) = 0 \leq S_i(1)$.

If $m_i = 1$, the neuron is related to a propositional symbol $A$, so we have that

$$S_i(0) = \begin{cases} \vartheta_A & \text{if } \langle A \leftarrow \top, \vartheta_A \rangle \in \mathbb{P}, \\ 0 & \text{otherwise.} \end{cases}$$

thus, $S_i(0) \leq S_i(1) = \max\{\vartheta_A, \max_k\{S_k(0) \mid w_{ik} > 0\}\}$.

Assume $S_i(t-1) \leq S_i(t)$ for all $i$ as the induction hypothesis. We have several cases, depending on $m_i$:

If $m_i = 1$, the neuron corresponds to a fact, then:

$$\begin{aligned} S_i(t+1) &= \max\left\{v_i, \max_k\{S_k(t) \mid w_{ik} > 0\}\right\} \\ &\overset{(*)}{\geq} \max\left\{v_i, \max_k\{S_k(t-1) \mid w_{ik} > 0\}\right\} = S_i(t) \end{aligned}$$

where $(*)$ is due to the induction hypothesis to neurons $k$.

If $m_i = 2$, the neuron corresponds to product implication, then:

$$S_i(t+1) = v_i \prod_{k/w_{ik}>0} S_k(t) \overset{(*)}{\geq} v_i \prod_{k/w_{ik}>0} S_k(t-1) = S_i(t)$$

where $(*)$ is due to the induction hypothesis and that the product in $[0,1]$ is monotonic.

The proof of the cases $m_i = 3, 4, 5$ is similar. $\square$

The input of the initial values to the neurons is governed by an external reset signal $r$, common to all the neurons. The user is allowed to modify both the values of the internal registers of the neurons and their state vector $\vec{S}(t)$. A more formal description of this reset signal is given below:

$r = 1$. The initial truth-value $v_i$, the type of formula $m_i$, and the $i$-th row of the matrix of weights $\vec{w_{i\bullet}}$ are set in the internal registers. This allows to reinitialise the network when a new problem has to be studied.

$r = 0$. The neurons evolve with the usual dynamics and are only affected by the state vector of the net $\vec{S}(t)$. The value $m_i$, set in their internal register, selects the function which is activated in the neuron. By using a delay, the output of the activated function is compared with the previous value of the neuron.

Once the corresponding values for both the registers and the initial state of the net have been loaded, the signal $r$ is set to 0.

Figure 2 shows a generic neuron; the main inputs are the vector $\vec{S}$ of outputs of all the neurons in the net, the initial truth value of the neuron $v_i$ and the signal $m_i$ that establishes the type of rule the neuron is associated with. The other signals allow to set some parameters of the problem into the neuron, so $r$ is a reset signal that introduces the external values of $\vec{w_{i\bullet}}$ as the new parameters of the neuron and $x$ as its new state. The $i$-th neuron receives the output of the rest of neurons in the previous step $\vec{S(t)}$.

## 5 Representing a homogenous program by a neural net

As we have observed in the previous section, each propositional symbol of the initial program $\mathbb{P}$, and the new rules of the homogeneous program $\mathbb{P}^*$ are represented by a neuron in the net. The different types of neuron are described below:

(1) **A propositional symbol:** Its type is $m_i = 1$. The initial truth-value $v_i$ is set either to 0 (by default) or to the truth-value $\vartheta_A$ if $A$ is a fact.

The $i$-th row of the matrix of weights have all components set to 0 but those corresponding to rules whose head is the given propositional symbol, in which case have value 1.

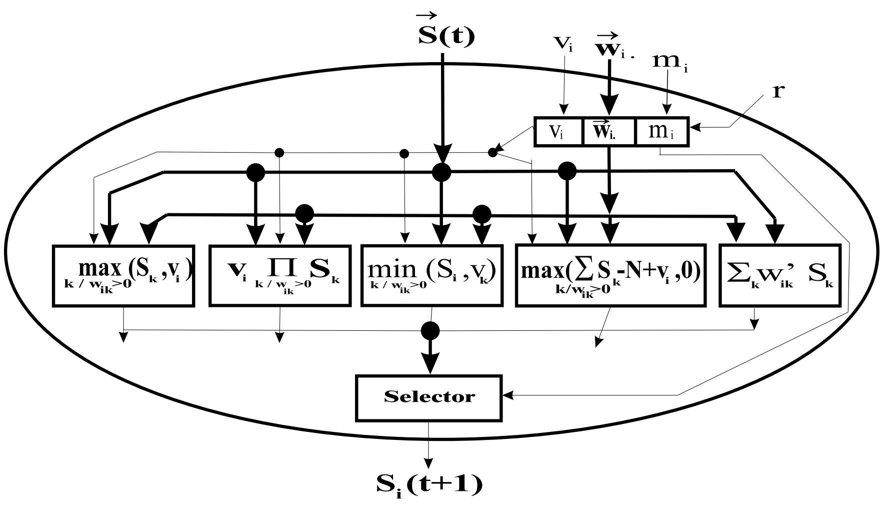
$\vec{S}(t)$
$v_i$
$\vec{w}_{i\cdot}$
$m_i$
r
$v_i$ | $\vec{w}_{i\cdot}$ | $m_i$
$\max_{k/w_{ik}>0}(S_k, v_i)$
$v_i \prod_{k/w_{ik}>0} S_k$
$\min_{k/w_{ik}>0}(S_i, v_k)$
$\max(\sum_{k/w_{ik}>0} S_k - N + v_i, 0)$
$\sum_k w'_{ik} S_k$
Selector
$S_i(t+1)$

Fig. 2. A generic neuron

(2) **Product:** These neurons correspond to a homogeneous product rule. Its internal registers are $m_i = 2$, $v_i$ uses the truth-value of the rule in $v_i$, and the corresponding row in the matrix of weights is fixed with all components 0, except those assigned to propositional symbols involved in the body, which are set to 1.

**Example 4** *Consider the program below, consisting of two facts and one rule*

$$\langle p \leftarrow \top, 0.2\rangle \qquad \langle q \leftarrow \top, 0.8\rangle \qquad \langle p \leftarrow_P q, 0.5\rangle$$

*we would use a net with three neurons, the first and second to represent the facts $p$ and $q$, and the third to represent $p \leftarrow_P q$. Thus, the registers of the associated neurons are initialized as follows:*

- *For neuron 1: $v_1 = 0.2$, $m_1 = 1$, $w_{1\bullet} = (0,0,1)$, since the head of the rule represented by the third neuron is $p$.*
- *For neuron 2: $v_2 = 0.8$, $m_2 = 1$, $w_{2\bullet} = (0,0,0)$, because there is no rule with head $q$.*
- *For neuron 3: $v_3 = 0.5$, $m_3 = 2$, $w_{3\bullet} = (0,1,0)$, for the body of the rule depends on $q$, represented by the second neuron.*

**Example 5** *Given the homogeneous rule*

$$\langle p \leftarrow_P (q \,\&_P\, r \,\&_P\, s),\ 0.7\rangle$$

*there are four neurons to represent each propositional symbol. The fifth neuron would have its internal registers initialized as follows:*

*$v_5 = 0.7$, $m_5 = 2$ and $w_{5\bullet} = (0,1,1,1,0)$, assuming the ordering of neurons corresponding to propositional symbols $p, q, r, s$ respectively. Moreover $w_{1\bullet} = (0,0,0,0,1)$, since the head of the rule is the propositional symbol $p$.*

(3) **Gödel:** The only difference with the previous case is that $m_i = 3$.

(4) **Łukasiewicz:** The only difference with the previous case is that $m_i = 4$.

**Example 6** *For the program with three facts and two rules*

$$\langle p \leftarrow \top, 0.7\rangle \qquad \langle r \leftarrow \top, 0.5\rangle \qquad \langle s \leftarrow \top, 0.6\rangle$$

$$\langle p \leftarrow_G (q \,\&_G\, r \,\&_G\, s), 0.8\rangle \qquad \langle q \leftarrow_L (p \,\&_L\, r \,\&_L\, s), 0.7\rangle$$

*we should use a net with six neurons, the first four corresponding to the propositional symbols $p, q, r$ and $s$, whereas the other two ones correspond to the rules. As the initial vector of truth-values we have $v = (0.7, 0, 0.5, 0.6, 0.8, 0.7)$, and the vector of types (indicating the interpretation of the neurons) is $\vec{m} = (1, 1, 1, 1, 3, 4)$. Finally, the weights matrix $W$ is set to*

$$W = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 1 & 1 & 1 & \cdot & \cdot \\ 1 & \cdot & 1 & 1 & \cdot & \cdot \end{pmatrix}$$

*where the dots indicate a zero value.*

(5) **Weighted sums:** These neurons are related to rules of the type:

$$\langle p \leftarrow @_{(n_1, n_2, \ldots, n_k)}(q_1, q_2, \ldots, q_k), 1\rangle$$

So the truth-value is always one and it is unimportant which type of implication is used since all of them assign the same value to the head of the rule.

The register of this type of neurons is set with truth-value $v_i = 1$, its type is $m_i = 5$, and the vector $w_{i\bullet}$ indicates the weights ($w_{ij} \geq 0$) of the rest of neurons on the output of the weighted sum. The normalization process for the weighted sum is done internally by the neuron using the values in the vector $w_{i\bullet}$ calculating $w'_{ij}$ as $w'_{ij} = \dfrac{w_{ij}}{\sum_j w_{ij}}$.

**Example 7** *Consider the non homogeneous rule:*

$$\langle p \leftarrow_P @_{(3,7)}(q, r)\,, 0.5\rangle$$

*which is transformed into*

$$\langle h \leftarrow @_{(3,7)}(q, r), 1\rangle \qquad \langle p \leftarrow_P h, 0.5\rangle$$

*we have three neurons associated, respectively, to the propositional symbols $p$, $q$ and $r$, we need two extra neurons, one to represent the first rule, whose registers are initialized as $v_4 = 1$, $m_4 = 5$ and $w_{4\bullet} = (0, 3, 7, 0, 0)$, and another one to represent the product implication, with $v_5 = 0.5$, $m_5 = 2$ and $w_{5\bullet} = (0, 0, 0, 1, 0)$. Moreover, since the symbol $p$ is the* *head of the rule corresponding to the fifth neuron, we must set* $w_{15} = 1$, *resulting in :*

$$W = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 3 & 7 & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot \end{pmatrix} \quad \textit{and } \vec{m} = (1, 1, 1, 5, 2).$$

At this stage, no learning algorithm is used because the net just described completely specifies the behaviour of the immediate consequences operator $T_{\mathbb{P}}$. The use of the adaptive capabilities of the networks are used in the search for explanations of a set of given observations in the context of abductive logic programming, as introduced in [13], but this is not in the scope of this paper.

## *5.1 Relating the net and $T_{\mathbb{P}}$*

In this section we will relate the behavior of the components of the state vector with the immediate consequence operator. Firstly, we need to introduce some notation to group the set of rules in $\mathbb{P}$ depending on the signal $m_i$:

- $\mathbb{P}_P$ is the set of product homogeneous rules,
- $\mathbb{P}_G$ is the set of Gödel homogeneous rules,
- $\mathbb{P}_L$ is the set of Łukasiewicz homogeneous rules
- $\mathbb{P}_@$ is the set of aggregator homogeneous rules

**Theorem 6** *Given a homogeneous program $\mathbb{P}$ and a symbol $A$, then*

$$T_{\mathbb{P}}{}^n(\vartriangle)(A) = S_A(2n-2) \quad \textit{for} \quad n \geq 1.$$

**PROOF.** By induction on $n$:

For $n = 1$ is trivially true,

$$T^1_{\mathbb{P}}(\vartriangle)(A) = \sup\{\vartheta \,\&_i \vartriangle(\mathcal{B}) \mid \langle A \leftarrow_i \mathcal{B}, \vartheta\rangle \in \mathbb{P}\} = \vartheta_A = S_A(0)$$

Note that sometimes we are abusing the notation, in that in the expression $S_*$ the star indicates either an index of an existing neuron or the formula attached to the neuron. However, the context and the symbols used are enough to avoid any ambiguity.

Now, assume that $T^n_{\mathbb{P}}(\triangle)(A) = S_A(2n-2)$ for all propositional symbol $A$. We have to consider the different types of rules in $\mathbb{P}$: we will prove that for each weighted rule $j = \langle A \leftarrow_i \mathcal{B}, \vartheta\rangle \in \mathbb{P}_i$ we have $\vartheta \,\&_i\, \widehat{T^n_{\mathbb{P}}(\triangle)}(\mathcal{B}) = S_j(2n-1)$.

Let us consider the case of product rules. In this case $\mathcal{B} = B_1 \,\&_P \ldots \&_P\, B_l$ and thus:

$$\begin{aligned}
\vartheta \,\&_P\, \widehat{T^n_{\mathbb{P}}(\triangle)}(\mathcal{B}) &= \vartheta \,\&_P\, \widehat{T^n_{\mathbb{P}}(\triangle)}(B_1 \,\&_P \ldots \&_P\, B_l)\\
&= \vartheta \,\&_P\, T^n_{\mathbb{P}}(\triangle)(B_1) \,\&_P \ldots \&_P\, T^n_{\mathbb{P}}(\triangle)(B_l)\\
&= \vartheta \,\&_P\, S_{B_1}(2n-2) \,\&_P \ldots \&_P\, S_{B_l}(2n-2)\\
&= S_j(2n-1)
\end{aligned}$$

where the last equality is due to the definition and the induction hypothesis. For the case of Gödel and Łukasiewicz rules the proof follows similarly.

In the case of aggregator-based rules, given a rule of the form $j = \langle A \leftarrow @(B_1, \ldots, B_l), \top\rangle$ we have:

$$\begin{aligned}
\widehat{T^n_{\mathbb{P}}(\triangle)}(\mathcal{B}) &= \widehat{T^n_{\mathbb{P}}(\triangle)}(@(B_1, \ldots, B_l))\\
&= @(T^n_{\mathbb{P}}(\triangle)(B_1), \ldots, T^n_{\mathbb{P}}(\triangle)(B_l))\\
&\overset{(*)}{=} @(S_{B_1}(2n-2), \ldots, S_{B_l}(2n-2))\\
&= S_j(2n-1)
\end{aligned}$$

where the last equality is due to the definition and $(*)$ by induction hypothesis.

Thus, we have

$$\begin{aligned}
T^{n+1}_{\mathbb{P}}(\triangle)(A) &= \sup\Big\{\vartheta \,\&_i\, \widehat{T^n_{\mathbb{P}}(\triangle)}(\mathcal{B}) \mid A \overset{\vartheta}{\leftarrow}_i \mathcal{B} \in \mathbb{P}\Big\}\\
&= \sup\Big\{\vartheta_A, \sup\{\vartheta \,\&_P\, \widehat{T^n_{\mathbb{P}}(\triangle)}(\mathcal{B}) \mid A \overset{\vartheta}{\leftarrow}_P \mathcal{B} \in \mathbb{P}_P\},\\
&\qquad \sup\{\vartheta \,\&_G\, \widehat{T^n_{\mathbb{P}}(\triangle)}(\mathcal{B}) \mid A \overset{\vartheta}{\leftarrow}_G \mathcal{B} \in \mathbb{P}_G\},\\
&\qquad \sup\{\vartheta \,\&_L\, \widehat{T^n_{\mathbb{P}}(\triangle)}(\mathcal{B}) \mid A \overset{\vartheta}{\leftarrow}_L \mathcal{B} \in \mathbb{P}_L\},\\
&\qquad \sup\{\widehat{T^n_{\mathbb{P}}(\triangle)}(\mathcal{B}) \mid A \overset{\top}{\leftarrow} \mathcal{B} \in \mathbb{P}_@\}\Big\}\\
&\overset{(*)}{=} \sup\Big\{\vartheta_A, \sup\{S_{A\leftarrow_P\mathcal{B}}(2n-1) \mid A \overset{\vartheta}{\leftarrow}_P \mathcal{B} \in \mathbb{P}_P\},\\
&\qquad \sup\{S_{A\leftarrow_G\mathcal{B}}(2n-1) \mid A \overset{\vartheta}{\leftarrow}_G \mathcal{B} \in \mathbb{P}_G\},\\
&\qquad \sup\{S_{A\leftarrow_L\mathcal{B}}(2n-1) \mid A \overset{\vartheta}{\leftarrow}_L \mathcal{B} \in \mathbb{P}_L\},\\
&\qquad \sup\{S_{A\leftarrow\mathcal{B}}(2n-1) \mid A \overset{\vartheta}{\leftarrow} \mathcal{B} \in \mathbb{P}_@\}\Big\}\\
&= S_A(2n)
\end{aligned}$$

where the equality $(*)$ follows from the previous result, and the last equality is due to definition of $S_A(n)$. $\square$

A number of simulations have been obtained through a MATLAB implementation in a conventional sequential computer. A high level description of the implementation is given below:

## *5.2 Implementation*

(1) **Initialize** the network is with the appropriate values of $\vec{v}$, $\vec{m}$, $W$ and, in addition, a tolerance value *tol* to be used as a stop criterion. The output $S_i(t)$ of the neurons associated to facts (which are propositional variables, so $m_i = 1$) are initialized with its truth-value $v_i$.

(2) **Repeat** until the following stop criterion is fulfilled $\|\vec{S}(t) - \vec{S}(t-1)\|_2 < tol$, where $\|\cdot\|$ denotes euclidean distance.

Update all the states $S_i$ of the neurons of the network (in parallel):

(a) If $m_i = 1$, then:

(i) Find the neurons $j$ (if any) which operate on the body of the rule $i$. These neurons form the set $J_i = \{j \mid W_{i,j} = 1\}$.

(ii) Then, update the state of neuron $i$ as follows:

$$S_i(t) = \begin{cases} \max\{v_i, \max_J S_j(t-1)\} & \text{if } J_i \neq \varnothing \\ v_i & \text{otherwise} \end{cases}$$

(b) If $m_i = 2, 3$, then:

(i) Find the neurons $j$ (if any) which operate on the neuron $i$, that is, construct the set $J_i = \{j \mid W_{i,j} = 1\}$.

(ii) Then, update the state of neuron $i$ as follows:

$$S_i(t) = \begin{cases} v_i \prod_{J_i} S_j(t-1) & \text{if } t_i = 2 \\ \min\{v_i, \min_J S_j(t-1)\} & \text{if } t_i = 3 \end{cases}$$

Note that when $m_i = 2$ the neuron corresponds to a product implication and when $m_i = 3$ to a Gödel implication.

(c) If $m_i = 4$ the neuron represents a Łukasiewicz implication, then:

(i) Find the set $J_i = \{j \mid W_{i,j} = 1\}$, and let $N_i$ be its cardinal.

(ii) Then, update the state of neuron $i$ as follows:

$$S_i(t) = \max\{v_i + \sum_J S_j(t-1) - N_i, 0\}$$

(d) If $m_i = 5$, then the neuron corresponds to an aggregator, and its update follows a different pattern:

(i) Determine the set $K_i = \{j \mid W_{i,j} > 0\}$ and calculate $sum = \sum_{K_i} W_{i,j}$

(ii) Update the neuron as follows:

$$S_i(t) = \frac{1}{sum} \sum_{K_i} W_{i,j} * S_j(t-1)$$

**Until** the stop criterion $\|\vec{S}(t) - \vec{S}(t-1)\|_2 < tol$ is fulfilled.

Regarding the analysis of the convergence of the network, from Thm. 6, this is the case if the immediate consequences operator reaches the fixpoint after a finite number of steps. Otherwise, because of the monotonicity of $T_{\mathbb{P}}$, it is the case that the net always obtains an approximation to the fixed point up to any level of precision.

Obtaining the *exact* fixed point of the $T_{\mathbb{P}}$ operator is possible for special classes of programs without aggregation operators, for instance in [18] termination in finitely many steps is proved for homogeneous programs with only one conjunction connective and without aggregators.

## 6 Examples

A number of programs have been carried out with the implementation. We present two toy examples:

**Example 8** *Consider the program with facts* $\langle o \leftarrow \top, 0.2\rangle$, $\langle w \leftarrow \top, 0.2\rangle$, $\langle r \leftarrow \top, 0.5\rangle$ *and rules*

$$\begin{array}{cc}
\langle h \leftarrow_G (r \,\&_P\, o)\ ,\ 0.9\rangle & \langle v \leftarrow_G @_{(1,2)}(o,w)\ ,\ 0.8\rangle \\
\langle n \leftarrow_P r\ ,\ 0.8\rangle & \langle n \leftarrow_P w\ ,\ 0.9\rangle \\
\langle w \leftarrow_P v\ ,\ 0.75\rangle &
\end{array}$$

*As there are non-homogeneous rules, the rules of the program are transformed as follows*

$$\begin{array}{cc}
\langle i \leftarrow r \,\&_P\, o\ ,\ 1\rangle & \langle j \leftarrow @_{(1,2)}(o,w)\ ,\ 1\rangle \\
\langle h \leftarrow_G i\ ,\ 0.9\rangle & \langle v \leftarrow_G j\ ,\ 0.8\rangle \\
\langle n \leftarrow_P r\ ,\ 0.8\rangle & \langle n \leftarrow_P w\ ,\ 0.9\rangle \\
\langle w \leftarrow_P v\ ,\ 0.75\rangle &
\end{array}$$

*Therefore, we will need 13 neurons (7 hidden ones) associated to* $h, n, o, r, v, w$ *and the homogeneous rules.*

*The initial values of the registers $\vec{m}, \vec{v}$ and $W$ are:*

- $\vec{m} = (1,1,1,1,1,1,2,5,3,3,2,2,2)$
- $\vec{v} = (0,0,0.2,0.5,0,0.2,1,1,0.9,0.8,0.8,0.9,0.75)$
- *The matrix $W$ is defined as*

$$W = \begin{pmatrix}
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\
\cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & 1 & \cdot & \cdot & 2 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{pmatrix}$$

*After five iterations, the net gets a stable state:*

$$\vec{S} = (0.1,0.4,0.2,0.5,0.2,0.2,0.1,0.2,0.1,0.2,0.4,0.18,0.15)$$

*with the following values for the propositional symbols: $v_h = 0.1$; $v_n = 0.4$; $v_o = 0.2$; $v_r = 0.5$; $v_v = 0.2$; $v_w = 0.2$.*

**Example 9** *Consider the program with rules*

$$\begin{array}{cc}
\langle p \leftarrow_G @_{(1,2,3)}(q,r,s)\ ,\ 0.8\rangle & \langle q \leftarrow_P (t \,\&_L\, u)\ ,\ 0.6\rangle \\
\langle t \leftarrow_P (v \,\&_G\, u)\ ,\ 0.5\rangle & \langle v \leftarrow_P u\ ,\ 0.8\rangle
\end{array}$$

*and facts $\langle u \leftarrow \top, 0.75\rangle$, $\langle r \leftarrow \top, 0.7\rangle$, $\langle s \leftarrow \top, 0.6\rangle$.*

*Firstly, we will transform the rules of the program into an homogeneous one,* *as follows*

$$\begin{array}{cc}
\langle h \leftarrow_P @_{(1,2,3)}(q,r,s)\ ,\ 1\rangle & \langle i \leftarrow_L t \,\&_L\, u\ ,\ 1\rangle \\
\langle j \leftarrow_G v \,\&_G\, u\ ,\ 1\rangle & \langle p \leftarrow_G h\ ,\ 0.8\rangle \\
\langle q \leftarrow_P i\ ,\ 0.6\rangle & \langle t \leftarrow_P j\ ,\ 0.5\rangle \\
\langle v \leftarrow_P u\ ,\ 0.8\rangle &
\end{array}$$

*The net will consist of 14 neurons, to represent the initial propositional symbols* $p$, $q$, $r$, $s$, $t$, $u$, $v$ *together with the rules of the homogeneous program.*

*The initial values of the net are:*

- $\vec{m} = (1,1,1,1,1,1,1,1,5,4,3,3,2,2,2)$
- $\vec{v} = (0,0,0.7,0.6,0,0.75,0,1,1,1,0.8,0.6,0.5,0.8)$
- *The matrix* $W$ *is defined as*

$$W = \begin{pmatrix}
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \\
\cdot & 1 & 2 & 3 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{pmatrix}$$

*After running the net, its state vector get stabilized at*

$$\vec{S} = (0.5383, 0.03, 0.7, 0.6, 0.3, 0.75, 0.6, 0.5383, 0.05, 0.6, 0.5383, 0.03, 0.3, 0.6)$$

*where the last seven components correspond to hidden neurons, the first ones are interpreted as the obtained truth-value for* $p$, $q$, $r$, $s$, $t$, $u$ *and* $v$ *by means for* $T_{\mathbb{P}}^{\omega}(\triangle)$.

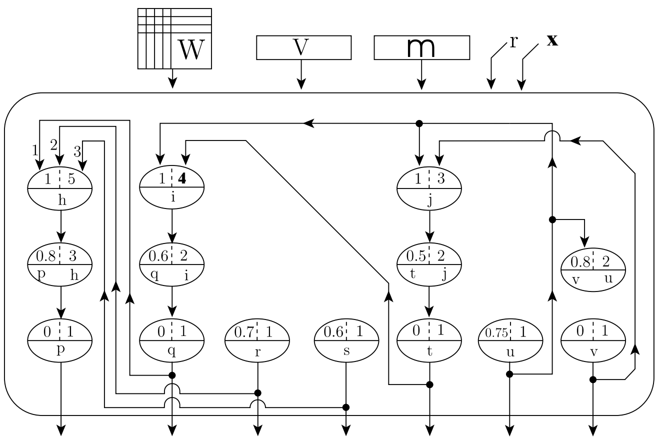

Fig. 3. Neural network for Example 9.

Figure 3 represents a possible network that is able to solve Example 9, so it has 14 neurons. Seven of them, associated to the symbols $p$, $q$, $r$, $s$, $t$, $u$ and $v$, whose register $v$ has been initialized with either 0 or the value for facts, the register $m$ is 1 for all of them and the weight vectors have all components 0, excepting for neurons associated to $p$, $q$, $t$ and $v$ whose components 11, 12, 13 and 14 have the value 1 respectively, that is $w_{1,11} = w_{2,12} = w_{5,13} = w_{7,14} = 1$.

The other neurons are hidden, the initial truth value of the neuron associated to the rule $\langle h \leftarrow_P @_{(1,2,3)}(q, r, s), 1\rangle$ is 1, and its type is 5, the weights of connections are in this case 1, 2, 3 (this operator is the only one with weight different from 1).The rest of neurons are associated to Łukasiewicz, Gödel, or product implications, so their types are 4, 3 or 2, whereas their truth values are those given by the program.

## 7 Conclusions and future work

A neural-like model has been introduced, which implements the procedural semantics recently given to multi-adjoint logic programming. This way, it is possible to obtain the computed truth-values of all propositional symbols involved in the program in a parallel way. This model can easily be modified to add new types of fuzzy rules.

We consider only the three most important adjoint pairs in the unit interval (product, Gödel, and Łukasiewicz) and weighted sums. As future work, we aim at developing a general neural network for t-norms, by the introduction of an ordinal sum unit in order to combine the basic conjunctors.

Regarding termination, the net always obtains a fixed point of $T_{\mathbb{P}}$ up to any level of precision. For special classes of programs it is also possible to obtain the *exact* fixed point of the $T_{\mathbb{P}}$ operator, as shown in [18]. Initial unpublished work extending the ideas in [5] shows that this kind of result can be extended to any multi-adjoint homogeneous program. This seems to be a new result, since the works [9,10] assume acyclicity while resorting to more standard units.

Concerning learning aspects, our approach being more complex than usual systems, it seems likely that some ideas from hybrid type networks should have to be taken into account, for instance reinforcement for fuzzy control like systems [2]. As current and future work, we are extending the framework by adding learning capabilities to the net, so that it will be able to adapt the truth-values of the rules in a given program to fit a number of observations. Following this idea, a neural net implementation for abductive multi-adjoint logic programming, already sketched in [13], is planned. Another interesting line of (more theoretically oriented) future work is to study the problem of whether ordinary feed forward networks can be captured by multi-adjoint LP.

*Acknowledgements*

We thank P. Eklund for providing interesting comments on previous versions of this work.

## References

[1] P. A. Armelín and D. J. Pym. Bunched Logic Programming. *Lecture Notes in Computer Science* 2083:289–304, 2001.

[2] H.R. Berenji and P. Khedkar. Learning and tuning fuzzy logic controllers through reinforcements. *IEEE Tr. on Neural Networks*, 3(5):724–740, 1992.

[3] J.J. Buckley and Y. Hayashi. Fuzzy neural networks : a survey. *Fuzzy Sets and Systems*, 66:1–13, 1994.

[4] 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.

[5] C.V. Damásio and M. Ojeda-Aciego. On termination of a tabulation procedure for residuated logic programming. In *6th Intl Workshop on Termination, WST'03*, pages 40–43, 2003.

[6] P. Eklund and F. Klawonn. Neural fuzzy logic programming. *IEEE Tr. on Neural Networks*, 3(5):815–818, 1992.

[7] S. Gottwald. *A treatise on many-valued logics.* Research Studies Press, 2001.

[8] P. Hájek. *Metamathematics of Fuzzy Logic.* Trends in Logic. Kluwer Academic, 1998.

[9] S. Hölldobler and Y. Kalinke. Towards a new massively parallel computational model for logic programming. In *ECAI'94 workshop on Combining Symbolic and Connectioninst Processing*, pages 68–77, 1994.

[10] S. Hölldobler, Y. Kalinke, and H.-P. Störr. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence*, 11(1):45–58, 1999.

[11] N. Kasabov. *Neuro-fuzzy techniques for Intelligent Information Processing.* Physica Verlag, 1999.

[12] V. Kurková. Kolmogorov's theorem and multilayer neural networks, *Neural Networks* 5: 501-506, 1992.

[13] J. Medina, E. Mérida-Casermeiro, and M. Ojeda-Aciego. A neural approach to abductive multi-adjoint reasoning. In *AI - Methodologies, Systems, Applications. AIMSA'02*, pages 213–222, Lect. Notes in Computer Science 2443, 2002.

[14] 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.

[15] 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.

[16] E. Mérida-Casermeiro, G. Galán-Marín, and J. Muñoz Pérez. An efficient multivalued Hopfield network for the traveling salesman problem. *Neural Processing Letters*, 14:203–216, 2001.

[17] S. Mitra and Y. Hayashi. Neuro-fuzzy rule generation: Survey in soft computing framework. *IEEE Tr. Neural Networks*, 11:748–768, 2000.

[18] L. Paulík. Best Possible Answer is Computable for Fuzzy SLD-Resolution In *Proceedings of Gödel'96*: Logical Foundations of Mathematics, Computer Science and Physics; Kurt Gödel's Legacy, pages 257–266, 1997.

[19] D. A. Plaisted and S. Greenbaum. A structure-preserving clause form translation. *Journal of Symbolic Computation*, 2(3):293–304, 1986.

[20] D. J. Pym. Logic Programming with Bunched Implications. *Electronic Notes in Theoretical Computer Science* 17. Didier Galmiche (ed.). Elsevier, 2000.

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

[22] P. Vojtáš. Fuzzy logic programming. *Fuzzy Sets and Systems*, 124(3):361–370, 2001.