Logic Programs with Uncertainty: Neural Computation and Automated Reasoning

Ekaterina Komendantskaya[1] and Anthony Seda[2]

[1] Department of Mathematics, University College Cork, Cork, Ireland
e.komendantskaya@mars.ucc.ie
[2] Department of Mathematics, University College Cork, Cork, Ireland
a.seda@ucc.ie * **

Abstract. Bilattice-based annotated logic programs (BAPs) form a very general class of programs which can handle uncertainty and conflicting information. We use BAPs to integrate two alternative paradigms of computation: specifically, we build learning artificial neural networks which can model iterations of the semantic operator associated with each BAP and introduce sound and complete SLD-resolution for this class of programs.
Key words: Logic programs, artificial neural networks, SLD-resolution

1 Introduction

The problem of reasoning with uncertainty and conflicting sources of information has been a subject of research for quite a long time, see [1, 5, 6, 9], for example. In [7], we defined very general annotated (first-order) logic programs (BAPs) based on infinite bilattices. These logic programs can process information about facts whilst incorporating conflicting or incomplete information about them. The semantic operator $\mathcal{T}_P$ defined in [7] for BAPs reflects some remarkable properties of the least Herbrand model for BAPs. In this paper, we show that the computation of $\mathcal{T}_P$ by connectionist neural networks requires the introduction of learning functions into the structure of the networks. In this sense, we believe that BAPs provide a suitable formalism for integrating the pure logical deduction of connectionism and the properties of spontaneous learning manifested by artificial neural networks (ANNs) thought of as nature-inspired models of computation. We propose an SLD-resolution for BAPs which is the first sound and complete proof procedure we know of for logic programs based on infinite (bi)lattices.

The structure of the paper is as follows. In §2, we summarise all the results obtained in [7] in relation to the computation of the least Herbrand model for BAPs. In §3, we build learning ANNs which are able to compute the least Herbrand model for BAPs and prove this fact. In particular, we describe in §3 how

* The authors thank the Boole Centre for Research in Informatics (BCRI) at University College Cork for substantial support in the preparation of this paper.
** The authors are grateful to three anonymous referees for their useful suggestions concerning a preliminary version of the paper. We also thank D. Woods for providing us with interesting examples of reasoning with uncertainty in complexity theory.

first-order fragments of BAPs can be approximated by ANNs. In §4, we introduce sound and complete SLD-resolution for BAPs. In §5, we conclude by giving a summary of our results.

2 Bilattice-Based Logic Programming

In this section, we survey some basic definitions and results obtained in [7]. We use the well-known definition of bilattices due to Ginsberg, see [1].

Definition 1. *A* bilattice $\mathbf{B}$ *is a sextuple* $(\mathbf{B}, \vee, \wedge, \oplus, \otimes, \neg)$ *such that* $(\mathbf{B}, \vee, \wedge)$ *and* $(\mathbf{B}, \oplus, \otimes)$ *are both complete lattices, and* $\neg : \mathbf{B} \to \mathbf{B}$ *is a mapping satisfying the following three properties:* $\neg^2 = Id_{\mathbf{B}}$, $\neg$ *is a dual lattice homomorphism from* $(\mathbf{B}, \vee, \wedge)$ *to* $(\mathbf{B}, \wedge, \vee)$, *and* $\neg$ *is the identity mapping on* $(\mathbf{B}, \oplus, \otimes)$.

We use here the fact that each distributive bilattice can be regarded as a product of two lattices, see [1]. Therefore, we consider only logic programs over distributive bilattices and regard the underlying bilattice of any program as a product of two lattices. Moreover, we always treat each bilattice we work with as isomorphic to some subset of $\mathbf{B} = L_1 \times L_2 = ([0,1], \leq) \times ([0,1], \leq)$, where $[0,1]$ is the unit interval of reals with the linear ordering defined on it.[3] Throughout this paper, we use $\mathbf{B}$ to denote the underlying bilattice of a given language.

In fact, we can use bilattice structures to formalize hypothetical and uncertain reasoning of the sort humans beings are capable of carrying out.

Example 1. Hypothetical reasoning is natural when, for example, scientists face an unsolvable problem which is, however, very important for their subject. Consider, for example the "P $\neq$ NP?" problem. Imagine two bright scientists Dr. N and Dr. M employed by a university to solve it. The scientists consider some related problems which may lead to the final proof of "P $\neq$ NP"; for example, "NP $\neq$ coNP?" and "NC$\neq$ NP?". After a while, both scientists give proofs (both are very long and need to be checked by someone): Dr. M has proven that "NP$\neq$ coNP", and Dr. N has proven that "NP=coNP". If a two-valued logic program receives the data, it will report a contradiction. Humans might, for example, make the following conclusions. If "NP$\neq$ coNP" and then "NP=coNP" were proven, than no conclusions yet can be made about the "P $\neq$ NP" problem. If next Dr. M reports that "P $\neq$ NP" and Dr. N reports that "P=NP", it will be possible to derive the evidence foe and against the statement "It is proven that "NC$\neq$ NP".

We need to introduce some formalism to reason in situations of the sort described in Example 1. We define an annotated bilattice-based language $\mathcal{L}$

[3] Elements of such a bilattice are pairs: the first element of each pair denotes evidence for a fact, and the second element denotes evidence against it. Thus, $\langle 1, 0\rangle$ is the analogue of "truth" and is maximal with respect to the truth ordering, while $\langle 1, 1\rangle$ may be seen as "contradiction" (or "both") and is maximal with respect to the knowledge ordering.

to consist of individual variables, constants, functions and predicate symbols together with annotation terms which can consist of variables, constants and/or functions over a bilattice. We allow six connectives and two quantifiers, as follows: $\oplus, \otimes, \vee, \wedge, \neg, \sim, \Sigma, \Pi$.

An *annotated formula* is defined inductively as follows: if R is an n-ary predicate symbol, $t_1, \ldots, t_n$ are terms, and (μ, ν) is an annotation term, then $R(t_1, \ldots, t_n) : (\mu, \nu)$ is an *annotated formula* (called an *annotated atom*). Annotated atoms can be combined to form complex formulae using the connectives and quantifiers.

A *bilattice-based annotated logic program (BAP)* P consists of a finite set of (annotated) *program clauses* of the form

$$A : (\mu, \nu) \leftarrow L_1 : (\mu_1, \nu_1), \ldots, L_n : (\mu_n, \nu_n),$$

where $A : (\mu, \nu)$ denotes an annotated atom called the *head* of the clause, and $L_1 : (\mu_1, \nu_1), \ldots, L_n : (\mu_n, \nu_n)$ denotes $L_1 : (\mu_1, \nu_1) \otimes \ldots \otimes L_n : (\mu_n, \nu_n)$ and is called the *body* of the clause; each $L_i : (\mu_i, \nu_i)$ is an annotated literal called an *annotated body literal* of the clause. Individual and annotation variables in the body are thought of as being existentially quantified using Σ.

In [7], we showed how the remaining connectives $\oplus, \vee, \wedge$ can be introduced into BAPs. The definitions of the terms *unit clause*, *program goal* and *pre-interpretation* are standard, see [8].

Let D, v, and J denote respectively a domain of (pre-)interpretation, a variable assignment and a pre-interpretation for a given language, see [8]. An interpretation I for $\mathcal{L}$ consists of J together with the following mappings. The first mapping $\mathcal{I}$ assigns $|R|_{\mathcal{I},v} : D^n \longrightarrow \mathbf{B}$ to each n-ary predicate symbol R in $\mathcal{L}$. Further, for each element $\langle \alpha, \beta \rangle$ of $\mathbf{B}$, we define a mapping $\chi_{\langle \alpha,\beta \rangle} : \mathbf{B} \longrightarrow \mathbf{B}$, where $\chi_{\langle \alpha,\beta \rangle}(\langle \alpha', \beta' \rangle) = \langle 1, 0 \rangle$ if $\langle \alpha, \beta \rangle \leq_k \langle \alpha', \beta' \rangle$ and $\chi_{\langle \alpha,\beta \rangle}(\langle \alpha', \beta' \rangle) = \langle 0, 1 \rangle$ otherwise. The mapping χ is used to evaluate annotated formulae. Thus, if F is an annotated atom $R(t_1, \ldots, t_n) : (\mu, \nu)$, then the value of F is given by $I(F) = \chi_{\langle \mu,\nu \rangle}(|R|_{\mathcal{I},v}(|t_1|_v, \ldots, |t_n|_v))$. Furthermore, using χ we can proceed to give interpretation to complex annotated formulae in the standard way, see [7] (or [6] for lattice-based interpretations of annotated logic programs). All the connectives of the language are put into correspondence with bilattice operations, and in particular quantifiers correspond to infinite bilattice operations. We call the composition of the two mappings $\mathcal{I}$ and χ an *interpretation* for the bilattice-based annotated language $\mathcal{L}$ and for simplicity of notation denote it by I. Indeed, the interpretations of BAPs possess some remarkable properties which make the study of BAPs worthwhile, as follows.

Proposition 1. *[7]*

1. *Let F be a formula, and fix the value $I(F)$. If $I(F : (\alpha, \beta)) = \langle 1, 0 \rangle$, then $I(F : (\alpha', \beta')) = \langle 1, 0 \rangle$ for all $\langle \alpha', \beta' \rangle \leq_k \langle \alpha, \beta \rangle$.*
2. *$I(F_1 : (\mu_1, \nu_1) \otimes \ldots \otimes F_k : (\mu_k, \nu_k)) = \langle 1, 0 \rangle \iff I(F_1 : (\mu_1, \nu_1) \oplus \ldots \oplus F_k : (\mu_k, \nu_k)) = \langle 1, 0 \rangle \iff I(F_1 : (\mu_1, \nu_1) \wedge \ldots \wedge F_k : (\mu_k, \nu_k)) = \langle 1, 0 \rangle \iff$ each $I(F_i : (\mu_i, \nu_i)) = \langle 1, 0 \rangle$, where $i \in \{1, \ldots, k\}$.*

3. *If* $I(F_1 : (\mu_1, \nu_1) \odot \ldots \odot F_k : (\mu_k, \nu_k)) = \langle 1, 0\rangle$, *then* $I((F_1 \odot \ldots \odot F_k) : ((\mu_1, \nu_1) \odot \ldots \odot (\mu_k, \nu_k))) = \langle 1, 0\rangle$, *where* $\odot$ *is any one of the connectives* $\otimes, \oplus, \wedge$.
4. *For every formula* F, $I(F : (0, 0)) = \langle 1, 0\rangle$.

These properties influence models for BAPs. In particular, we introduced in [7] a semantic operator which shows how all the logical consequences of each program can be computed.

Let I be an interpretation for $\mathcal{L}$ and let F be a closed annotated formula of $\mathcal{L}$. Then I is a *model* for F if $I(F) = \langle 1, 0\rangle$. We say that I is a model for a set S of annotated formulae if I is a model for each annotated formula of S. We say that F *is a logical consequence of* S if, for every interpretation I of $\mathcal{L}$, I is a model for S implies I is a model for F.

Let B_P and U_P denote an annotation Herbrand base respectively Herbrand universe for a program P, see [7] for further explanations. An *annotation Herbrand interpretation* HI *for* P consists of the Herbrand pre-interpretation HJ (see [8]) with domain HD of $\mathcal{L}$ together with the following: for each n-ary predicate symbol in $\mathcal{L}$, the assignment of a mapping from $U_{\mathcal{L}}^n$ into $\mathbf{B}$. In common with conventional logic programming, each Herbrand interpretation HI for P can be identified with the subset $\{R(t_1, \ldots, t_k) : (\alpha, \beta) \in B_P | R(t_1, \ldots, t_k) : (\alpha, \beta)$ receives the value $\langle 1, 0\rangle$ with respect to $HI\}$ of B_P it determines, where $R(t_1, \ldots, t_k) : (\alpha, \beta)$ denotes a typical element of B_P. This set constitutes an *annotation Herbrand model for* P. Finally, we let $\mathrm{HI}_{P,\mathbf{B}}$ denote the set of all annotation Herbrand interpretations for P.

In [7], we introduced a semantic operator $\mathcal{T}_P$ for BAPs, proved its continuity and showed that it computes the least Herbrand model for a given BAP. Indeed, we define $\mathcal{T}_P$ next.

Definition 2. *We define the mapping* $\mathcal{T}_P : \mathrm{HI}_{P,\mathbf{B}} \rightarrow \mathrm{HI}_{P,\mathbf{B}}$ *as follows:* $\mathcal{T}_P(\mathrm{HI})$ *denotes the set of all* $A : (\mu, \nu) \in B_P$ *such that either*

1. *There is a strictly ground instance of a clause* $A : (\mu, \nu) \leftarrow L_1 : (\mu_1, \nu_1), \ldots, L_n : (\mu_n, \nu_n)$ *in* P *such that there exist annotations* $(\mu'_1, \nu'_1), \ldots, (\mu'_n, \nu'_n)$ *satisfying* $\{L_1 : (\mu'_1, \nu'_1), \ldots, L_n : (\mu'_n, \nu'_n)\} \subseteq \mathrm{HI}$, *and one of the following conditions holds for each* (μ'_i, ν'_i)*:*
   (a) $(\mu'_i, \nu'_i) \geq_k (\mu_i, \nu_i)$,
   (b) $(\mu'_i, \nu'_i) \geq_k \oplus_{j \in J_i} (\mu_j, \nu_j)$, *where* J_i *is the finite set of those indices such that* $L_j = L_i$

or

2. *there are annotated strictly ground atoms* $A : (\mu^*_1, \nu^*_1), \ldots, A : (\mu^*_k, \nu^*_k) \in \mathrm{HI}$ *such that* $\langle \mu, \nu\rangle \leq_k \langle \mu^*_1, \nu^*_1\rangle \oplus \ldots \oplus \langle \mu^*_k, \nu^*_k\rangle$.[4]

Semantic operators defined for many logic programs as in the papers of Fitting and Van Emden (and other authors) use only some form of item 1.*a* from Definition 2. However, this condition is not sufficient for computation of the Herbrand models for (bi)lattice-based logic programs.

[4] Note that whenever $F : (\mu, \nu) \in HI$ and $(\mu', \nu') \leq_k (\mu, \nu)$, then $F : (\mu', \nu') \in HI$. Also, for each formula F, $F : (0, 0) \in HI$.

Example 2. Consider the logic program: $B : (0,1) \leftarrow$, $B : (1,0) \leftarrow$, $A : (0,0) \leftarrow B : (1,1)$, $C : (1,1) \leftarrow A : (1,0), A : (0,1)$. We can regard this program as formalizing Example 1. Let B stand for "NP $\neq$ coNP", A stand for "P $\neq$ NP" and C stand for "It is proven that "NC $\neq$ NP", annotations $(0,0), (0,1), (1,0), (1,1)$ express respectively "no proof/refutation is given", "proven", "proven the opposite" and "contradictory, or proven both the statement and the opposite". The least fixed point of $\mathcal{T}_P$ is $\mathcal{T}_P \uparrow 3 = \{B : (0,1), B : (1,0), B : (1,1), A : (0,0), C : (1,1)\}$, precisely the conclusions we mentioned in Example 1. However, the item 1.*a* (corresponding to the classical semantic operator) would allow us to compute only $\mathcal{T}_P \uparrow 1 = \{B : (0,1), B : (1,0)\}$, that is, only explicit consequences of a program, which then leads to a contradiction in the two-valued case. In the same way, the properties stated in Proposition 1 suggest that there can be some implicit logical consequences which can be derived if we take into consideration the underlying (bi)lattice structure of the program.

3 Neural Networks for Reasoning with Uncertainty

3.1 Connectionist Networks: Some Basic Definitions

In this subsection, we follow closely [3] and [4]. A *connectionist network* is a directed graph. A *unit* k in this graph is characterized, at time t, by its *input vector* $(i_{k1}(t), \ldots i_{kn_k}(t))$, its potential $p_k(t) \in \mathbb{R}$, its *threshold* $\Theta_k \in \mathbb{R}$, and its *value* $v_k(t)$. Units are connected via a set of directed and weighted connections. If there is a connection from unit j to unit k, then $w_{kj} \in \mathbb{R}$ denotes the *weight* associated with this connection, and $i_{kj}(t) = w_{kj}v_j(t)$ is the *input* received by k from j at time t. The units are updated synchronously. In each update, the potential and value of a unit are computed with respect to an *activation* and an *output function* respectively. All units considered in this paper compute their potential as the weighted sum of their inputs minus their threshold:

$$p_k(t) = \left(\sum_{j=1}^{n_k} w_{kj} v_j(t) \right) - \Theta_k.$$

The units are updated synchronously, time becomes $t + \Delta t$, and the output value for k, $v_k(t + \Delta t)$ is calculated from $p_k(t)$ by means of a given *output function* ψ, that is, $v_k(t + \Delta t) = \psi(p_k(t))$. The output function ψ we use in this paper is the binary threshold function H, that is, $v_k(t + \Delta t) = H(p_k(t))$, where $H(p_k(t)) = 1$ if $p_k(t) > 0$ and 0 otherwise. Units of this type are called *binary threshold units.*

In this paper, we will only consider connectionist networks where the units can be organized in layers. A *layer* is a vector of units. An n-layer *feedforward network* $\mathcal{F}$ consists of the *input* layer, $n-2$ *hidden* layers, and the *output* layer, where $n \geq 2$. Each unit occurring in the i-th layer is connected to each unit occurring in the $(i+1)$-st layer, $1 \leq i < n$. Let r and s be the number of units occurring in the input and output layers, respectively. A connectionist network $\mathcal{F}$ is called a *multilayer feedforward network* if it is an n-layer feedforward network

for some n. A miltilayer feedforward network $\mathcal{F}$ computes a function $f_{\mathcal{F}} : \mathbb{R}^r \rightarrow \mathbb{R}^s$ as follows. The input vector (the argument of $f_{\mathcal{F}}$) is presented to the input layer at time t_0 and propagated through the hidden layer to the output layer. At each time point, all units update their potential and value. At time $t_0+(n-1)\Delta t$, the output vector (the image under $f_{\mathcal{F}}$ of the input layer) is read off the output layer.

3.2 Neural Networks and Propositional BAPs.

Hölldobler et al. defined in [4] ANNs which are capable of computing the immediate consequence operator T_P for classical propositional logic programs. However, these ANNs cannot "learn" new information, that is, they cannot change their weights either with supervision or without it.

We extend this approach to learning ANNs which can compute logical consequences of BAPs. This will allow us to introduce hypothetical and uncertain reasoning into the framework of neural-symbolic computation. Bilattice-based logic programs can work with conflicting sources of information and inconsistent databases. Therefore, ANNs corresponding to these logic programs should reflect this facility as well, and this is why we introduce some forms of learning into ANNs. These forms of leaning can be seen as corresponding to a sort of unsupervised Hebbian learning, which is commonly used in the context of ANNs. The general idea behind Hebbian learning is that positively correlated activities of two neurons strengthen the weight of the connection between them and that uncorrelated or negatively correlated activities weaken the weight of the connection (the latter form is known as Anti-Hebbian learning).

The general conventional definition of Hebbian learning is given as follows, see [2] for example. Let k and j denote two neurons and w_{kj} denote a weight of the connection from j to k. We denote the value of j at time t as $v_j(t)$ and the potential of k at time t as $p_k(t)$. Then the rate of change in the weight between j and k is expressed in the form

$$\Delta w_{kj}(t) = F(v_j(t), p_k(t)),$$

where F is some function. As a special case of this formula, it is common to write

$$\Delta w_{kj}(t) = \eta(v_j(t))(p_k(t)),$$

where η is a constant that determines the *rate of learning* and is positive in case of Hebbian learning and negative in case of Anti-Hebbian learning. In this section, we will compare the two learning functions we introduce with this conventional definition of Hebbian learning.

First, we prove a theorem establishing a relationship between learning ANNs and bilattice-based annotated logic programs with no function symbols occurring either in the predicate symbols or in the annotations. (Since the Herbrand base for these programs is finite, they can equivalently be seen as propositional bilattice-based logic programs with no functions allowed in the annotations.) In the next subsection, we will extend the result to first-order BAPs with functions in individual and annotation terms.

Theorem 1. *For each function-free BAP P, there exists a 3-layer feedforward learning ANN which computes $\mathcal{T}_P$.*

Proof. Let m and n be the number of strictly ground annotated atoms from the annotation Herbrand base $\mathcal{B}_P$ and the number of clauses occurring in P respectively. Without loss of generality, we may assume that the annotated atoms are ordered. The network associated with P can now be constructed by the following translation algorithm.

1. The input and output layers are vectors of binary threshold units of length k, where the i-th unit in the input and output layers represents the i-th strictly ground annotated atom, $1 \leq k \leq m$. The threshold of each unit occurring in the input or output layer is set to 0.5.
2. For each clause of the form $A : (\alpha, \beta) \leftarrow B_1 : (\alpha_1, \beta_1), \ldots, B_m : (\alpha_m, \beta_m)$, $m \geq 0$, in P do the following.
   2.1 Add a binary threshold unit c to the hidden layer.
   2.2 Connect c to the unit representing $A : (\alpha, \beta)$ in the output layer with weight 1. We will call connections of this type 1-*connections*.
   2.3 For each atom $B_j : (\alpha_j, \beta_j)$ in the input layer, connect the unit representing $B_j : (\alpha_j, \beta_j)$ to c and set the weight to 1. (We will call these connections 1-connections also.)
   2.4 Set the threshold θ_c of c to $l - 0.5$, where l is the number of atoms in $B_1 : (\alpha_1, \beta_1), \ldots, B_m : (\alpha_m, \beta_m)$.
   2.5 If an input unit representing $B : (\alpha, \beta)$ is connected to a hidden unit c, connect each of the input units representing annotated atoms $B_i : (\alpha_i, \beta_i), \ldots, B_j : (\alpha_j, \beta_j)$, where $(B_i = B), \ldots, (B_j = B)$, to c. These connections will be called $\otimes$-*connections*. The weights of these connections will depend on a learning function. If the function is inactive, set the weight of each $\otimes$-*connection* to 0.
3. If there are units representing atoms of the form $B_i : (\alpha_i, \beta_i), \ldots, B_j : (\alpha_j, \beta_j)$, where $B_i = \ldots = B_j$ in input and output layers, correlate them as follows. For each $B_i : (\alpha_i, \beta_i)$, connect the unit representing $B_i : (\alpha_i, \beta_i)$ in the input layer to each of the units representing $B_i : (\alpha_i, \beta_i), \ldots, B_j : (\alpha_j, \beta_j)$ in the output layer. These connections will be called the $\oplus$-*connections*. If an $\oplus$-connection is set between two atoms with different annotations, we consider them as being connected via hidden units with thresholds 0. If an $\oplus$-connection is set between input and output units representing the same annotated atom $B : (\alpha, \beta)$, we set the threshold of the hidden unit connecting them to -0.5, and we will call them $\oplus$-hidden units, so as to distinguish the hidden units of this type. The weights of all these $\oplus$-connections will depend on a learning function. If the function is inactive, set the weight of each $\oplus$-*connection* to 0.
4. Set all the weights which are not covered by these rules to 0.

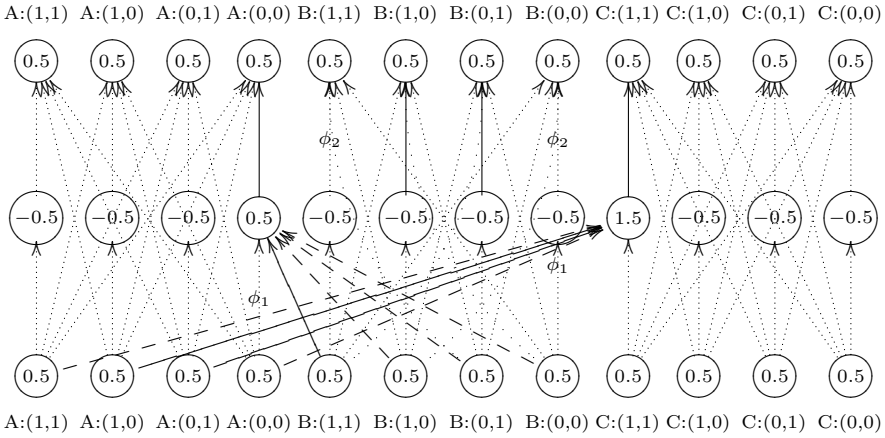
Allow two learning functions to be embedded into the $\otimes$ -connections and the $\oplus$ -connections. We let v_i denote the value of the neuron representing $B_i : (\alpha_i, \beta_i)$ and p_c denote the potential of the unit c.

Let a unit representing $B_i : (\alpha_i, \beta_i)$ in the input layer be denoted by i. If i is connected to a hidden unit c via an $\otimes$ -connection, then a learning function ϕ_1 is associated to this connection. We let $\phi_1 = \Delta w_{ci}(t) = (v_i(t))(-p_c(t) + 0.5)$ become active and change the weight of the $\otimes$-connection from i to c at time t if units representing atoms $B_j : (\alpha_j, \beta_j), \ldots, B_k : (\alpha_k, \beta_k)$ $(B_i = B_j = \ldots = B_k)$ became activated at time $t - \Delta t$, they are connected to c via 1-connections and $\langle \alpha_i, \beta_i \rangle \geq_k (\langle \alpha_j, \beta_j \rangle \otimes \ldots \otimes \langle \alpha_k, \beta_k \rangle)$.

Function ϕ_2 is embedded only into connections of type $\oplus$, namely, into $\oplus$-connections between hidden and output layers. Let o be an output unit representing an annotated atom $B_i : (\alpha_i, \beta_i)$. Activate $\phi_2 = \Delta w_{oc}(t) = (v_c(t))(p_o(t)+1.5)$ at time t if it is embedded into an $\oplus$-connection from the $\oplus$- hidden unit c to o and there are output units representing annotated atoms $B_j : (\alpha_j, \beta_j), \ldots, B_k : (\alpha_k, \beta_k)$, where $(B_i = B_j), \ldots, (B_i = B_k)$, which are connected to the unit o via $\oplus$-connections, these output units became activated at time $t - 2\Delta t$ and $\langle \alpha_i, \beta_i \rangle \leq_k (\langle \alpha_j, \beta_j \rangle \oplus \ldots \oplus \langle \alpha_k, \beta_k \rangle)$.

Each interpretation I for P can be represented by a binary vector $(v_1, \ldots, v_m)$. Such an interpretation is given as an input to the network by externally activating corresponding units of the input layer at time t_0. It remains to show that $A : (\alpha, \beta) \in \mathcal{T}_P \uparrow n$ for some n if and only if the unit representing $A : (\alpha, \beta)$ becomes active at time $t_0 + 2\Delta t$, for some Δt. The proof that this is so proceeds by routine induction.

Example 3. The following diagram displays the neural network which computes $\mathcal{T}_P \uparrow 3$ from Example 2. Without functions ϕ_1, ϕ_2 the ANN will compute only $\mathcal{T}_P \uparrow 1 = \{B : (0,1), B : (1,0)\}$, explicit logical consequences of the program. Note that arrows $\longrightarrow$, $- - \succ$, $\cdots\succ$ denote respectively 1-connections, $\otimes$-connections and $\oplus$-connections, and we have marked by ϕ_1, ϕ_2 the connections which are activated by the learning functions.[5]



[5] According to the conventional definition of feedforward ANNs, each output neuron denoting some atom is in turn connected to the input neuron which denotes the same atom via a 1-connection and thus forms a loop. We do not draw these connections here.

We can make several conclusions from the construction of Theorem 1.

- Neurons representing annotated atoms with identical first-order (or propositional) components are joined into multineurons in which neurons are correlated using $\oplus$- and $\otimes$-connections.
- The learning function ϕ_2 roughly corresponds to Hebbian learning, with the rate of learning $\eta_2 = 1$, the learning function ϕ_1 corresponds to Anti-Hebbian learning with the rate of learning 1, and we can regard η_1 as negative because the factor p_c in the formula for ϕ_1 is multiplied by (-1).
- The main problem Hebbian learning causes is that the weights of connections with embedded learning functions tend to grow exponentially, which cannot fit the model of biological neurons. This is why traditionally some functions are introduced to bound the growth. In the ANNs we have built some of the weights may grow with iterations, but the growth will be very slow, because of the activation functions, namely, binary threshold functions, used in the computation of each v_i.

3.3 Neural Networks and First-Order BAPs

Since ANNs were proven to compute least fixed points of the semantic operator defined for propositional logic programs, many attempts have been made to extend this result to first-order logic programs. See, for example, [3], [10]. We extend here the result obtained by Seda [10] for two-valued first-order logic programs and their representations by ANNs to first-order BAPs.

Let $l : B_P \to \mathbb{N}$ be a *level mapping* with the property that, given $n \in \mathbb{N}$, we can effectively find the set of all $A : (\mu, \nu) \in B_P$ such that $l(A : (\mu, \nu)) = n$. The following definition is due to Fitting, and for further explanation see [10] or [7].

Definition 3. *Let* $\mathrm{HI}_{P,\mathbf{B}}$ *be the set of all interpretations* $B_P \to \mathbf{B}$*. We define the ultrametric* $d : \mathrm{HI}_{P,\mathbf{B}} \times \mathrm{HI}_{P,\mathbf{B}} \to \mathbb{R}$ *as follows: if* $\mathrm{HI}_1 = \mathrm{HI}_2$*, we set* $d(\mathrm{HI}_1, \mathrm{HI}_2) = 0$*, and if* $\mathrm{HI}_1 \neq \mathrm{HI}_2$*, we set* $d(\mathrm{HI}_1, \mathrm{HI}_2) = 2^{-N}$*, where* N *is such that* HI_1 *and* HI_2 *differ on some ground atom of level* N *and agree on all atoms of level less then* N*.*

Fix an interpretation HI from elements of the Herbrand base for a given program P to the set of values $\{\langle 1, 0\rangle, \langle 0, 1\rangle\}$. We assume further that $\langle 1, 0\rangle$ is encoded by 1 and $\langle 0, 1\rangle$ is encoded by 0. Let HI_P denote the set of all such interpretations, and take the semantic operator $\mathcal{T}_P$ as in Definition 2. Let $\mathcal{F}$ denote a 3-layer feedforward learning ANN with m units in the input and output layers. The input-output mapping $f_{\mathcal{F}}$ is a mapping $f_{\mathcal{F}} : HI_P \to HI_P$ defined as follows. Given $HI \in HI_P$, we present the vector $(HI(B_1 : (\alpha_1, \beta_1)), \ldots, HI(B_m : (\alpha_m, \beta_m)))$, to the input layer; after propagation through the network, we determine $f_{\mathcal{F}}(HI)$ by taking the value of $f_{\mathcal{F}}(HI)(A_j : (\alpha_j, \beta_j))$ to be the value in the jth unit in the output layer, $j = 1, \ldots, m$, and taking all other values of $f_{\mathcal{F}}(HI)(A_j : (\alpha_j, \beta_j))$ to be 0.

Suppose that M is a fixed point of $\mathcal{T}_P$. Following [10], we say that a family $\mathcal{F} = \{\mathcal{F}_i : i \in \mathcal{I}\}$ of 3-layer feedforward learning network $\mathcal{F}_i$ computes M if

there exists $HI \in HI_P$ such that the following holds: given any $\varepsilon > 0$, there is an index $i \in \mathcal{I}$ and a natural number m_i such that for all $m \geq m_i$ we have $d(f_i^m(HI), M) < \varepsilon$, where f_i denotes $f_{\mathcal{F}_i}$ and $f_i^m(HI)$ denotes the mth iterate of f_i applied to HI.

Theorem 2. *Let P be an arbitrary annotated program, let* HI *denote the least fixed point of $\mathcal{T}_P$ and suppose that we are given $\varepsilon > 0$. Then there exists a finite program $\overline{P} = \overline{P}(\varepsilon)$ (a finite subset of ground(P)) such that $d(\overline{\mathrm{HI}}, \mathrm{HI}) < \varepsilon$, where $\overline{\mathrm{HI}}$ denotes the least fixed point of $\mathcal{T}_{\overline{P}}$. Therefore, the family $\{\mathcal{F}_n | n \in \mathbb{N}\}$ computes* HI*, where $\mathcal{F}_n$ denotes the neural network obtained by applying the algorithm of Theorem 1 to P_n, and P_n denotes $\overline{P}(\varepsilon)$ with ε taken as 2^{-n} for $n = 1, 2, 3, \ldots$.*

This theorem clearly contains two results corresponding to the two separate statements made in it. The first concerns finite approximation of $\mathcal{T}_P$, and is a straightforward generalization of a theorem established in [10]. The second is an immediate consequence of the first conclusion and Theorem 1. Thus, we have shown that the learning ANNs we have built can approximate the least fixed point of the semantic operator defined for first-order BAPs.

4 SLD-Resolution for BAPs

We propose a sound and complete proof procedure for BAPs as an alternative computational paradigm to ANNs. It can be particularly useful for programs whose annotation Herbrand Base is infinite, because in this case it may be problematical to build ANNs approximating the least fixed point of $\mathcal{T}_P$. As far as we know, this is the first sound and complete proof procedure for first-order infinitely interpreted (bi)lattice-based annotated logic programs. Compare, for example, our results with those obtained for constrained resolution for GAPs, which was shown to be incomplete, see [6], or with sound and complete (SLD)-resolutions for finitely-interpreted annotated logic programs (these logic programs do not contain annotation variables and annotation functions), see, for example, [5, 9]. We proceed with the definition of our proof procedure for BAPs.

We adopt the following terminology. Let P be a BAP and let G be a goal $\leftarrow A_1 : (\mu_1, \nu_1), \ldots, A_k : (\mu_k, \nu_k)$. An *answer* for $P \cup \{G\}$ is a substitution $\theta\lambda$ for individual and annotation variables of G. We say that $\theta\lambda$ is a *correct answer* for $P \cup \{G\}$ if $\Pi((A_1 : (\mu_1, \nu_1), \ldots, A_k : (\mu_k, \nu_k))\theta\lambda)$ is a logical consequence of P.

Definition 4 (SLD-derivation). *Let G_i be the annotated goal $\leftarrow A_1 : (\mu_1, \nu_1), \ldots, A_k : (\mu_k, \nu_k)$, and let $C, C_1^*, \ldots, C_l^*$ be the annotated clauses $A : (\mu, \nu) \leftarrow B_1 : (\mu_1', \nu_1'), \ldots, B_q : (\mu_q', \nu_q')$, $A_1^* : (\mu_1^*, \nu_1^*) \leftarrow body_1^*, \ldots, A_l^* : (\mu_l^*, \nu_l^*) \leftarrow body_l^*$. Then the set of goals $G_{i+1}^1, \ldots, G_{i+1}^m$ is* derived *from G_i and C (and $C_1^*, \ldots, C_l^*$) using mgu[6] $\theta\lambda$ if the following conditions hold.*

1. *$A_m : (\mu_m, \nu_m)$ is an annotated atom, called the selected atom, in G.*

[6] Throughout this section, *mgu* stands for "most general unifier".

2. *θ is an* mgu *of A_m and A, and one of the following conditions holds:*
   (a) *λ is an* mgu *of (μ_m, ν_m) and (μ, ν);*
   (b) *$(\mu_m, \nu_m)\lambda$ and $(\mu, \nu)\lambda$ are constants and $(\mu, \nu)\lambda \geq_k (\mu_m, \nu_m)\lambda$;*
   (c) *there are clauses $C_1^*, \ldots, C_l^*$ of the form $A_1^* : (\mu_1^*, \nu_1^*) \leftarrow body_1^*, \ldots, A_l^* : (\mu_l^*, \nu_l^*) \leftarrow body_l^*$ in P, such that θ is an* mgu *of A, A_m and $A_1^*, \ldots, A_l^*$, λ is an* mgu *of (μ_m, ν_m), (μ, ν) and $(\mu_1^*, \nu_1^*), \ldots, (\mu_l^*, \nu_l^*)$ or $(\mu_m, \nu_m)\lambda$, $(\mu, \nu)\lambda$ and $(\mu^*, \nu^*)\lambda, \ldots, (\mu_l^*, \nu_l^*)\lambda$ are constants such that $(\mu_m, \nu_m)\lambda \leq_k ((\mu, \nu)\lambda \oplus (\mu_1^*, \nu_1^*)\lambda \oplus \ldots \oplus (\mu_l^*, \nu_l^*)\lambda)$.*
3. *in case 2(a), 2(b), $G_{i+1} = (\leftarrow A_1 : (\mu_1, \nu_1), \ldots, A_{m-1} : (\mu_{m-1}, \nu_{m-1}), B_1 : (\mu_1', \nu_1'), \ldots, B_q : (\mu_q', \nu_q'), A_{m+1} : (\mu_{m+1}, \nu_{m+1}), \ldots, A_k : (\mu_k, \nu_k))\theta\lambda$.*
4. *in case 2(c), $G_{i+1} = (\leftarrow A_1 : (\mu_1, \nu_1), \ldots, A_{m-1} : (\mu_{m-1}, \nu_{m-1}), B_1 : (\mu_1', \nu_1'), \ldots, B_q : (\mu_q', \nu_q'), body_1^*, \ldots, body_l^*, A_{m+1} : (\mu_{m+1}, \nu_{m+1}), \ldots, A_k : (\mu_k, \nu_k))\theta\lambda$.*
   In this case, G_{i+1} is said to be derived from G_i, C and $C_1^*, \ldots, C_l^*$ using $\theta\lambda$.
5. *The goals $G_{i+1}^1, \ldots, G_{i+1}^m$ can be obtained using the following rules: in case there are atomic formulae $F_i : (\mu_i, \nu_i), F_{i+1} : (\mu_{i+1}, \nu_{i+1}), \ldots, F_j : (\mu_j, \nu_j)$ in G_i such that $F_i\theta = F_{i+1}\theta = \ldots = F_j\theta$, form the next goal $G_{i+1}^1 = F_i\theta : ((\mu_i, \nu_i) \otimes (\mu_{i+1}, \nu_{i+1})), \ldots, F_j : (\mu_j, \nu_j)$, then $G_{i+1}^2 = F_i : (\mu_i, \nu_i), F_i\theta : ((\mu_{i+1}, \nu_{i+1}) \otimes (\mu_{i+2}, \nu_{i+2})), \ldots, F_j : (\mu_j, \nu_j)$ and so on for all possible combinations of these replacements. Form the set of goals $G_{i+1}^1, \ldots, G_{i+1}^m$, which is always finite and can be effectively enumerated by, for example, enumerating goals according to their leftmost replacements and then according to the number of replacements.*
6. *Whenever a goal G_j^i contains a formula of the form $F : (0, 0)$, then remove $F : (0, 0)$ from the goal and form the next goal G_{j+1}^i.*

Definition 5. *Suppose that P is a BAP and G_0 is a goal. An* SLD-derivation *of $P \cup \{G_0\}$ consists of a sequence $G_0, G_1^i, G_2^j \ldots$ of BAP goals, a sequence $C_1, C_2, \ldots$ of BAP clauses and a sequence $\theta_1\lambda_1, \theta_2\lambda_2, \ldots$ of* mgu*s such that each G_{i+1}^k is derived from G_i^j and C_{i+1} using $\theta_{i+1}\lambda_{i+1}$.*

An SLD-refutation *of $P \cup \{G_0\}$ is a finite SLD-derivation of $P \cup \{G\}$ which has the empty clause $\square$ as the last goal of the derivation. If $G_n^i = \square$, we say that the refutation has length n.*

The *success set* of P is the set of all $A : (\mu, \nu) \in B_P$ such that $P \cup \{\sim A\}$ has an SLD-refutation.

Theorem 3 (Soundness and completeness of SLD-resolution). *The success set of P is equal to its least annotation Herbrand model. Alternatively, soundness and completeness can be stated as follows. Every computed answer for $P \cup \{G\}$ is a correct answer for $P \cup \{G\}$, and for every correct answer $\theta\lambda$ for $P \cup \{G\}$, there exist a computed answer $\theta^*\lambda^*$ for $P \cup \{G\}$ and substitutions φ, ψ such that $\theta = \theta^*\varphi$ and $\lambda = \lambda^*\psi$.*

5 Conclusions and Further Work

We have shown that the logical consequences of the BAPs introduced in [7] can be computed by artificial neural networks with learning functions. Certain

constructions in the ANNs we have built for BAPs appear to be novel, and the question concerning the relationship between quantitative (bi)lattice-based logic programming and learning neural networks is itself quite novel. The BAPs were shown to be a very general formalism for reasoning about uncertainty and conflicting sources of information. In [7], we showed that implication-based logic programs á la van Emden and the annotation-free bilattice-based logic programs of [1] can be translated into the language of BAPs, and iterations of the semantic operators usually associated with these logic programs were shown to correspond to iterations of $\mathcal{T}_P$. These results extend bounds for further implementation of the ANNs we have introduced in the paper. The sound and complete SLD-resolution we have introduced for BAPs will serve as a complementary technique when working with BAPs having infinite annotation Herbrand base.

References

1. M. C. Fitting. Bilattices in logic programming. In G. Epstein, editor, *The twentieth International Symposium on Multiple-Valued Logic*, pages 238–246. IEEE, 1990.
2. S. Haykin. *Neural Networks. A comprehensive Foundation.* Macmillan College Publishing Company, 1994.
3. P. Hitzler, S. Hölldobler, and A. K. Seda. Logic programs and connectionist networks. *Journal of Applied Logic*, 2(3):245–272, 2004.
4. S. Hölldobler, Y. Kalinke, and H. P. Storr. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence*, 11:45–58, 1999.
5. M. Kifer and E. L. Lozinskii. Ri: A logic for reasoning with inconsistency. In *Proceedings of the 4th IEEE Symposium on Logic in Computer Science (LICS)*, pages 253–262, Asilomar, 1989. IEEE Computer Press.
6. M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *Journal of logic programming*, 12:335–367, 1991.
7. E. Komendantskaya, A. K. Seda, and V. Komendantsky. On approximation of the semantic operators determined by bilattice-based logic programs. In *Proceedings of the Seventh International Workshop on First-Order Theorem Proving (FTP'05)*, pages 112–130, Koblenz, Germany, September 15–17 2005.
8. J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 2nd edition, 1987.
9. J. J. Lu, N. V. Murray, and E. Rosental. Deduction and search strategies for regular multiple-valued logics. *Journal of Multiple-valued logic and soft computing*, 11:375–406, 2005.
10. A. K. Seda. On the integration of connectionist and logic-based systems. In T. Hurley, M. Mac an Airchinnigh, M. Schellekens, A. K. Seda, and G. Strong, editors, *Proceedings of MFCSIT2004, Trinity College Dublin, July, 2004*, Electronic Notes in Theoretical Computer Science. Elsevier, 2005. To appear.