A Connectionist Inductive Learning System for Modal Logic Programming

Artur S. d'Avila Garcez[δ], Luís C. Lamb[λ] and Dov M. Gabbay[γ]

[δ]Dept. of Computing, Imperial College, London, SW72BZ, UK
aag@doc.ic.ac.uk
[λ]Faculdade de Informática, PUCRS, Porto Alegre, RS, 90450-030, Brazil
lamb@inf.pucrs.br
[γ]Dept. of Computer Science, King's College, London WC2R 2LS, UK
dg@dcs.kcl.ac.uk

Abstract. Neural-Symbolic integration has become a very active research area in the last decade. In this paper, we present a new massively parallel model for modal logic. We do so by extending the language of Modal Prolog [32, 37] to allow modal operators in the head of the clauses. We then use an ensemble of *C-IL^{2}P* neural networks [14, 15] to encode the extended modal theory (and its relations), and show that the ensemble computes a fixpoint semantics of the extended theory. An immediate result of our approach is the ability to perform learning from examples efficiently using each network of the ensemble. Therefore, one can adapt the extended *C-IL^{2}P* system by training possible world representations.
Keywords: Neural-Symbolic Integration, Artificial Neural Networks, Modal Logic, Change of Representation, Learning from Structured Data.

1 Introduction

Neural-Symbolic integration concerns the utilization of problem-specific symbolic knowledge within the neurocomputing paradigm. In spite of the progress in the last decade, important open problems remain to be solved. In particular, neural systems have not been shown to fully represent and learn predicate logic [11]. We believe that a promising approach to unravel this problem is to investigate ways of representing the necessity and possibility operators of modal logic [9, 16] and the modal acessibility relation in artificial neural networks.

In this paper, we present a new massively parallel model for propositional modal logic, contributing towards the representation of quantification in neural networks. We do so by extending the language of Modal Prolog [32, 37] with the necessity and possibility operators. We then set up an ensemble of *Connectionist Inductive Learning and Logic Programming (C-IL^{2}P)* networks [14, 15], each network being an extension of Holldobler and Kalinke's parallel model for logic programming [24], to compute a fixpoint semantics of a given modal theory.

As pointed out in [4], most ILP systems, including FOIL [33], GOLEM [31] and PROGOL [30], apply a covering algorithm in order to generate hypothesis. Such systems "have difficulties in learning multiple predicates and recursive definitions, and will have unnecessarily long clauses for hypothesis" [4], in contrast with most Neural-Symbolic Learning Systems [11, 14]. Examples of Neural-Symbolic Learning Systems for first order logic include [4] and [7], which use, respectively, a Cascade ARTMAP [38] and a Radial-basis Function (RBF) network [22]. Differently, *C-IL^{2}P* networks use *Backpropagation* [35], the neural learning algorithm most successfully applied in real-world problems such as DNA sequence analysis and pattern recognition. In [5], a single *C-IL^{2}P* network has been used in conjunction with LINUS [26]. In this paper, an ensemble of *C-IL^{2}P* networks is used, allowing efficient learning with examples by training a number of possible world representations. In addition, as modal logics have been the subject of intense investigation in knowledge representation, this work is contribution towards the learning of modally defined aspects of reasoning using neural networks [16].

In Section 2, we briefly present the basic concepts of modal logic and artificial neural networks used throughout this paper. In Section 3, we present a Modalities Algorithm that translates extended modal programs into artificial neural networks. The network obtained is an ensemble of simple *C-IL^{2}P* networks, each representing a given possible world. We then show that the network computes a fixpoint semantics of the given modal theory, thus proving the correctness of the Modalities Algorithm. In Section 4, we conclude and discuss directions for future work.

2 Preliminaries

In this section, we present some basic concepts of Modal Logic and Artificial Neural Networks that will be used throughout the paper. We also extend Modal Prolog [32, 37] to allow modalities $\Box$ and $\Diamond$ in the head of the clauses and default negation $\sim$ in the body of the clauses [10]. Finally, we define a fixpoint semantics for such extension.

2.1 Modal Logic and Extended Modal Programs

Modal logic began with the analysis of concepts such as necessity and possibility under a philosophical perspective [25, 27]. A main feature of modal logics is the use of *possible world semantics* (proposed by Kripke and Hintikka) which has significantly contributed to the development of new non-classical logics, many of which have had a great impact in computing science [1, 29]. A proposition is necessary in a world if it is true in all worlds which are possible in relation to that world, whereas it is possible in a world if it is true in at least one world which is possible in relation to that same world.

Modal logic was found to be appropriate to study mathematical necessity (in the logic of provability), time, knowledge, belief, obligation and other concepts and modalities. In artificial intelligence and computing, modal logics are among the most employed formalisms to analyse and represent multi-agent systems and concurrency properties [16]. The basic definitions about modal logics that we shall use in this paper are as follows. As usual, we assume that any clause is ground over a finite domain.

Definition 1. *A* modal atom *is of the form MA where $M \in \{\Box, \Diamond\}$ and A is an atom. A* modal literal *is of the form ML where L is a literal.*

Definition 2. *A* modal prolog program *is a finite set of clauses of the form $MA_1, ..., MA_n \rightarrow A$.*

We define *extended modal programs* as modal prolog programs extended with modalities $\Box$ and $\Diamond$ in the head of clauses, and default negation $\sim$ in the body of clauses. In addition, each clause is labelled by the possible world in which they hold, similarly to Gabbay's Labelled Deductive Systems [20].

Definition 3. *An* extended modal program *is a finite set of clauses C of the form $\omega_i : ML_1, ..., ML_n \rightarrow MA$, where ω_i is a label representing a world in which the associated clause holds, and a finite set of relations $\mathcal{R}(\omega_i, \omega_j)$ between worlds ω_i and ω_j in C.*

For example: $\mathcal{P} = \{\omega_1 : r \rightarrow \Box q, \omega_1 : \Diamond s \rightarrow r, \omega_2 : s, \omega_3 : q \rightarrow \Diamond p, \mathcal{R}(\omega_1, \omega_2), \mathcal{R}(\omega_1, \omega_3)\}$ is an extended modal program. The $\Box$ and $\Diamond$ modalities will have the following interpretation.

Definition 4. *(*Kripke Models for Modal Logic*) Let $\mathcal{L}$ be a modal language. A* model *for $\mathcal{L}$ is a tuple $\mathcal{M} = \langle \Omega, \mathcal{R}, v \rangle$ where Ω is a set of possible worlds, v is a mapping that assigns to each propositional letter of $\mathcal{L}$ a subset of Ω, and $\mathcal{R}$ is a binary relation over Ω, such that: (i) $(\mathcal{M}, \omega) \models \Box\alpha$ iff for all $\omega_1 \in \Omega$, if $\mathcal{R}(\omega, \omega_1)$ then $(\mathcal{M}, \omega_1) \models \alpha$, and (ii) $(\mathcal{M}, \omega) \models \Diamond\alpha$ iff there exists a ω_1 such that $\mathcal{R}(\omega, \omega_1)$ and $(\mathcal{M}, \omega_1) \models \alpha$.*

A variety of proof procedures for modal logics has been developed over the years, e.g., [18, 36]. In some of these, formulas are labelled by the worlds in which they hold, thus facilitating the modal reasoning process (see [36] for a discussion on this topic). In the natural deduction-style rules below, the notation $\omega : \alpha$ represents that the formula α holds at the possible world ω. Moreover, the explicit reference to the accessibility relation also helps in deriving what formula holds in the worlds which are related by $\mathcal{R}$. The rules we shall represent using *C-IL^{2}P* are similar to the ones presented in [36], which we reproduce below.

The $\Diamond E$ rule can be seen (informally) as a *skolemization* of the existential quantifier over possible worlds, which is semantically implied by the formula $\Diamond\alpha$ in the premise. The term $f_\alpha(\omega)$ defines a particular possible world uniquely associated with the formula α, and inferred to be accessible from the possible world ω (i.e., $\mathcal{R}(\omega, f_\alpha(\omega))$). In the $\Box I$ rule, the temporary assumption should be read as "given an arbitrary accessible world $g_\alpha(\omega)$". The rule of $\Diamond I$ represents that if we have a relation $\mathcal{R}(\omega_1, \omega_2)$, and if α holds at ω_2 then it must be the case that α holds at ω_1 The rule $\Box E$ represents that if $\Box\alpha$ holds at a world ω_1, and ω_1 is related to ω_2, then we can infer that α holds at ω_2.

Table 1. Rules for modality operators

$\dfrac{[R(\omega, g_\alpha(\omega))] \ldots g_\alpha(\omega) : \alpha}{\omega : \Box\alpha}\ \Box I$	$\dfrac{\omega_1 : \Box\alpha, R(\omega_1, \omega_2)}{\omega_2 : \alpha}\ \Box E$
$\dfrac{\omega : \Diamond\alpha}{f_\alpha(\omega) : \alpha, R(\omega, f_\alpha(\omega))}\ \Diamond E$	$\dfrac{\omega_2 : \alpha, R(\omega_1, \omega_2)}{\omega_1 : \Diamond\alpha}\ \Diamond I$

In what follows, we define a model-theoretic semantics for extended modal programs. When computing the semantics of the program, we have to consider both the fixpoint of a particular world, and the fixpoint of the program as a whole. When computing the fixpoint in each world, we have to consider the consequences derived locally and the consequences derived from the interaction between worlds. Locally, fixpoints are computed as in the stable model semantics of logic programming, by simply renaming each modal literal ML_i by a new literal L_j not in the language $\mathcal{L}$, and applying the Gelfond-Lifschitz Transformation [8] to it. When considering interacting worlds, there are two cases to be addressed, according to the $\Box I$ and $\Diamond I$ rules in Table 1, together with the acessibility relation $\mathcal{R}$, which might render additional consequences in each world.

Definition 5. (Modal Immediate Consequence Operator) *Let $\mathcal{P} = \{\mathcal{P}_1, ..., \mathcal{P}_k\}$ be an extended modal program, where each $\mathcal{P}_i$ is the set of modal clauses that hold in a world ω_i ($1 \leq i \leq k$). Let $B_\mathcal{P}$ be the Herbrand base of $\mathcal{P}$ and I be a Herbrand interpretation for $\mathcal{P}_i$. The mapping $MT_{\mathcal{P}_i} : 2^{B_\mathcal{P}} \rightarrow 2^{B_\mathcal{P}}$ in ω_i is defined as follows: $MT_{\mathcal{P}_i}(I) = \{MA \in B_\mathcal{P} \mid$ either (i) or (ii) or (iii) below holds$\}$. (i) $ML_1, ..., ML_n \rightarrow MA$ is a clause in $\mathcal{P}_i$ and $\{ML_1, ..., ML_n\} \subseteq I$; (ii) $M = \Diamond$ and there exists a world ω_j such that $\mathcal{R}(\omega_i, \omega_j)$, $ML_1, ..., ML_m \rightarrow A$ is a clause in $\mathcal{P}_j$ and $\{ML_1, ..., ML_m\} \subseteq J$, where J is a Herbrand interpretation for $\mathcal{P}_j$; (iii) $M = \Box$ and for each world ω_j such that $\mathcal{R}(\omega_i, \omega_j)$, $ML_1, ..., ML_o \rightarrow A$ is a clause in $\mathcal{P}_j$ and $\{ML_1, ..., ML_o\} \subseteq K$, where K is a Herbrand interpretation for $\mathcal{P}_j$.*

Definition 6. (Global Modal Immediate Consequence Operator) *Let $\mathcal{P} = \{\mathcal{P}_1, ..., \mathcal{P}_k\}$ be an extended modal program. Let $B_\mathcal{P}$ be the Herbrand base of $\mathcal{P}$ and I_i be a Herbrand interpretation for $\mathcal{P}_i$ ($1 \leq i \leq k$). The mapping $MT_\mathcal{P} : 2^{B_\mathcal{P}} \rightarrow 2^{B_\mathcal{P}}$ is defined as follows: $MT_\mathcal{P}(I_1, ..., I_k) = \bigcup_{j=1}^{k}\{MT_{\mathcal{P}_j}\}$.*

In the case of definite extended modal programs, by renaming each modal atom MA_i by a new atom A_j, we can apply the following result of Ramanujam [34], regarding the fixpoint semantics of distributed definite logic programs.

Theorem 1. (Minimal Model of Distributed Programs *[34]) For each distributed definite logic program $\mathcal{P}$, the function $MT_\mathcal{P}$ has a unique fixpoint. The se-*

quence of all $MT_{\mathcal{P}}^{m}(I_1, ..., I_k), m \in \mathbb{N}$, *converges to this fixpoint* $MT_{\mathcal{P}}^{\varpi}(I_1, ..., I_k)$, *for each* $I_i \subseteq 2^{B_{\mathcal{P}}}$.

In order to provide a fixpoint semantics for extended modal programs, we have to extend the definition of *acceptable* programs [2, 3].

Definition 7. *(*Level Mapping*) Let* $\mathcal{P}$ *be a general logic program. A* level mapping *for* $\mathcal{P}$ *is a function* $| \; | : B_{\mathcal{P}} \to \mathbb{N}$ *from ground atoms to natural numbers. For* $A \in B_{\mathcal{P}}$, $|A|$ *is called the* level *of* A *and* $|\sim A| = |A|$.

Definition 8. *(*Acceptable Programs*) Let* $\mathcal{P}$ *be a program,* $| \; |$ *a level mapping for* $\mathcal{P}$*, and* $\mathbf{I}$ *a model of* $\mathcal{P}$*.* $\mathcal{P}$ *is called* acceptable *w. r. t* $| \; |$ *and* $\mathbf{I}$ *if for every clause* $L_1, ..., L_k \to A$ *in* $\mathcal{P}$ *the following implication holds. If* $\mathbf{I} \models \bigwedge_{j=1}^{i-1} L_j$ *then* $|A| > |L_j|$ *for* $1 \leq i \leq k$.

Theorem 2. *(*Minimal Model of Acceptable Programs *[17]) For each acceptable program* $\mathcal{P}$*, the function* $T_{\mathcal{P}}$ *has a unique fixpoint*[1]*. The sequence of all* $T_{\mathcal{P}}^{m}(I), m \in \mathbb{N}$, *converges to this fixpoint* $T_{\mathcal{P}}^{\varpi}(I)$ *(which is identical to the* stable model *of* $\mathcal{P}$ *[21])*[2]*, for each* $I \subseteq B_{\mathcal{P}}$.

Definition 9. *(*Acceptable Extended Modal Programs*) An extended modal program* $\mathcal{P}$ *is acceptable iff the program* $\mathcal{P}'$*, obtained by renaming each modal literal* ML_i *in* $\mathcal{P}$ *by a new literal* L_j *not in the language* $\mathcal{L}$ *is acceptable.*

Following [6], one can construct the semantics of extended modal programs by considering extended modal ground formulas in order to compute the fixpoint. As a result, one can associate with every extended modal program a modal ground program (the modal closure of the program) so that both programs have the same models. Hence, the classical results about the fixpoint semantics of logic programming can be applied directly to the modal ground closure of a program. Thus, Theorem 3, below, follows directly from Theorems 2 and 1.

Theorem 3. *(*Minimal Model of Acceptable Extended Modal Programs*) For each acceptable extended modal program* $\mathcal{P}$*, the function* $MT_{\mathcal{P}}$ *has a unique fixpoint. The sequence of all* $MT_{\mathcal{P}}^{m}(I_1, ..., I_k), m \in \mathbb{N}$, *converges to this fixpoint* $MT_{\mathcal{P}}^{\varpi}(I_1, ..., I_k)$*, for each* $I_i \subseteq 2^{B_{\mathcal{P}}}$.

Finally, note that in the above semantics, the choice of an arbitrary world for $\diamond$ elimination (made before the computation of $MT_{\mathcal{P}}$) may lead to different fixpoints of a given extended modal program. Such a choice is similar to the approach adopted by Gabbay in [19], in which one chooses a point in the future for execution and then backtracks if judged necessary (and at all possible).

[1] The mapping $T_{\mathcal{P}}$ is defined as follows: Let I be a Herbrand *interpretation*, then $T_{\mathcal{P}}(I) = \{A_0 \mid L_1, ..., L_n \to A_0$ is a clause in $\mathcal{P}$ and $\{L_1, ..., L_n\} \subseteq I\}$.

[2] A Herbrand interpretation I of a general logic program $\mathcal{P}$ is called stable iff $T_{\mathcal{P}_I}(I) = I$, where $\mathcal{P}_I$ is the definite program obtained from applying the Gelfond-Lifschitz Transformation over $\mathcal{P}$.

2.2 Artificial Neural Networks

An artificial neural network is a directed graph. A unit in this graph is characterised, at time t, by its input vector $I_i(t)$, its input potential $U_i(t)$, its activation state $A_i(t)$, and its output $O_i(t)$. The units (neurons) of the network are interconnected via a set of directed and weighted connections. If there is a connection from unit i to unit j, then $W_{ji} \in \Re$ denotes the *weight* associated with such a connection.

We start by characterising the neuron's *functionality* (see Figure 1). The *activation state* of a neuron i at time t ($A_i(t)$) is a bounded real or integer number. The output of neuron i at time t ($O_i(t)$) is given by the *output rule* f_i, such that $O_i(t) = f_i(A_i(t))$. The input potential of neuron i at time t ($U_i(t)$) is obtained by applying the *propagation rule* of neuron i (g_i) such that $U_i(t) = g_i(I_i(t), W_i)$, where $I_i(t)$ contains the input signals $(x_1(t), x_2(t), ..., x_n(t))$ to neuron i at time t, and W_i denotes the weight vector $(W_{i1}, W_{i2}, ..., W_{in})$ to neuron i. In addition, θ_i (an extra weight with input always fixed at 1) is known as the *threshold* of neuron i. Finally, the neuron's new activation state $A_i(t + \Delta t)$ is given by its *activation rule* h_i, which is a function of the neuron's current activation state and input potential, i.e. $A_i(t + \Delta t) = h_i(A_i(t), U_i(t))$, and the neuron's new output value $O_i(t + \Delta t) = f_i(A_i(t + \Delta t))$.

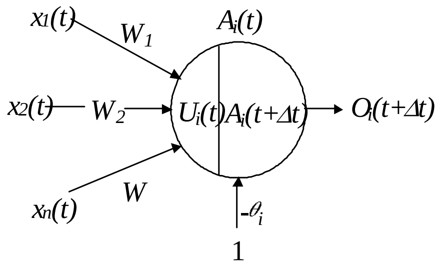

Fig. 1. The Processing Unit or Neuron.

In general, h_i does not depend on the previous activation state of the unit, that is, $A_i(t + \Delta t) = h_i(U_i(t))$, the propagation rule g_i is a weighted sum, such that $U_i(t) = \sum_j W_{ij} x_j(t)$, and the output rule f_i is given by the identity function, i.e. $O_i(t) = A_i(t)$.

The units of a neural network can be organised in layers. A *n-layer feedforward network* N is an acyclic graph. N consists of a sequence of layers and connections between successive layers, containing one input layer, $n - 2$ hidden layers and one output layer, where $n \geq 2$. When $n = 3$, we say that N is a *single hidden layer network*. When each unit occurring in the *i-th* layer is connected to each unit occurring in the $i + 1$-*st* layer, we say that N is a *fully-connected network* (see Figure 2).

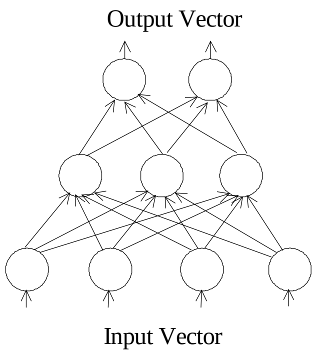

Fig. 2. A typical feedforward Neural Network.

The most interesting properties of a neural network do not arise from the functionality of each neuron, but from the collective effect resulting from the interconnection of units. Let r and s be the number of units occurring in the input and output layer, respectively. A multilayer feedforward network N computes a function $f : \Re^r \longrightarrow \Re^s$ as follows. The input vector is presented to the input layer at time t_1 and propagated through the hidden layers to the output layer. At each time point, all units update their input potential and activation state synchronously. At time t_n the output vector is read off the output layer. In addition, most neural models have a *learning rule*, responsible for changing the weights of the network so that it learns to approximate f given a number of *training examples* (input vectors and their respective target output vectors).

In this paper, we concentrate on single hidden layer feedforward networks, since they can approximate any (Borel) measurable function arbitrarily well, regardless of the dimension of the input space [12]. In this sense, single hidden layer networks are *approximators* of virtually any function of interest. We also use *bipolar* semi-linear activation functions $h(x) = \frac{2}{1+e^{-\beta x}} - 1$ with inputs in $\{-1, 1\}$, and the *Backpropagation* learning algorithm to perform training from examples (see [35]).

3 Connectionist Modal Logic

In this section, we extend the *C-IL*^{2}*P* system to represent modal theories by using ensembles of *C-IL*^{2}*P* networks. In [24], it is shown that the semantics of any first order acyclic logic program [28] can be approximated by a single hidden layer recurrent neural network (although no translation algorithm is presented). Since this is precisely the kind of network used by *C-IL*^{2}*P*, ensembles of *C-IL*^{2}*P* networks can enhance the expressive power of the system, yet maintaining the simplicity needed for performing inductive learning efficiently. In this section, we also present an efficient translation algorithm from extended modal programs to artificial neural networks.

We start with a simple example. It briefly illustrates how an ensemble of *C-IL^{2}P* networks can be used for modelling non-classical reasoning with modal logic. Input and output neurons may represent literals $\Box L$, $\Diamond L$ and L.

Example 1. Figure 3 shows an ensemble of three *C-IL^{2}P* networks $(\omega_1, \omega_2, \omega_3)$, which might "communicate" in many different ways. If we look at ω_1, ω_2 and ω_3 as *possible worlds*, we might be able to incorporate modalities in the language of *C-IL^{2}P*. For example, (i) "If $\omega_1 : \Box A$ then $\omega_2 : A$" could be communicated from ω_1 to ω_2 by connecting $\Box A$ in the output layer of ω_1 to A in the output layer of ω_2 such that, whenever $\Box A$ is activated in ω_1, A is activated in ω_2. In addition, analogously to the feedback of *C-IL^{2}P* networks, we could have feedback between ensembles of *C-IL^{2}P* networks. For example, (ii) "If $(\omega_2 : A) \vee (\omega_3 : A)$ Then $\omega_1 : \Diamond A$" could be implemented by connecting output neurons A of ω_2 and ω_3 into output neuron $\Diamond A$ of ω_1, through two hidden neurons (say, h_1 and h_2) in ω_1 such that $\omega_1 : h_1 \vee h_2 \rightarrow \Diamond A$. Examples (i) and (ii) simulate, in a finite universe, the rules of $\Box$ *Elimination* and $\Diamond$ *Introduction* (see Table 1).

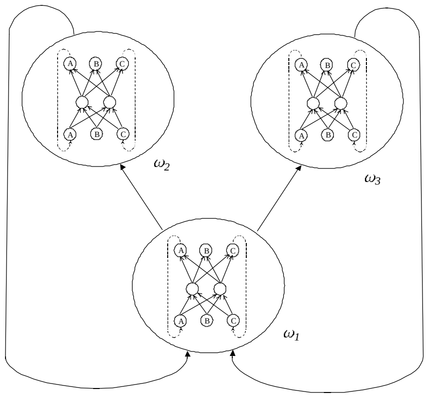

Fig. 3. An ensemble of *C-IL^{2}P* networks for modeling uncertainty by using modalities and possible worlds.

Due to the simplicity of each *C-IL^{2}P* network, e.g. ω_1, performing inductive learning within each *possible world* is straightforward. The main problem to be tackled when it comes to learning in the new neural model, therefore, is how to learn or set up the connections that establish the necessary communication between networks, e.g., ω_1 and ω_2. In the case of modal logic, such connections are defined analogously to the modal rules of natural deduction (Table 1). The *Modalities Algorithm* presented in Section 3.2 implements those rules, but first we recall how the *C-IL^{2}P* sytem works.

3.1 The *C-IL^{2}P* System

C-IL^{2}P [14, 15] is a massively parallel computational model based on a feedforward artificial neural network that integrates inductive learning from examples and background knowledge with deductive learning from logic programming. Following [23] (see also [24]), a *Translation Algorithm* maps a general logic program $\mathcal{P}$ into a single hidden layer neural network $\mathcal{N}$ such that $\mathcal{N}$ computes the least fixpoint of $\mathcal{P}$. This provides a massively parallel model for computing the stable model semantics of $\mathcal{P}$ [28]. In addition, $\mathcal{N}$ can be trained with examples using *Backpropagation* [35], having $\mathcal{P}$ as background knowledge. The knowledge acquired by training can then be extracted [13], closing the learning cycle (as in [39]).

Let us exemplify how *C-IL^{2}P's Translation Algorithm* works. Each rule (r_l) of $\mathcal{P}$ is mapped from the input layer to the output layer of $\mathcal{N}$ through one neuron (N_l) in the single hidden layer of $\mathcal{N}$. Intuitively, the *Translation Algorithm* from $\mathcal{P}$ to $\mathcal{N}$ has to implement the following conditions: (**C1**) The input potential of a hidden neuron (N_l) can only exceed N_l's threshold (θ_l), activating N_l, when all the positive antecedents of r_l are assigned the truth-value *true* while all the negative antecedents of r_l are assigned *false*; and (**C2**) The input potential of an output neuron (A) can only exceed A's threshold (θ_A), activating A, when at least one hidden neuron N_l that is connected to A is activated.

Example 2. Consider the logic program $\mathcal{P} = \{BC \sim D \rightarrow A; EF \rightarrow A; \rightarrow B\}$. The *Translation Algorithm* derives the network $\mathcal{N}$ of Figure 4, setting weights ($W's$) and thresholds ($\theta's$) in such a way that conditions (**C1**) and (**C2**) above are satisfied. Note that, if $\mathcal{N}$ ought to be fully-connected, any other link (not shown in Figure 4) should receive weight zero initially.

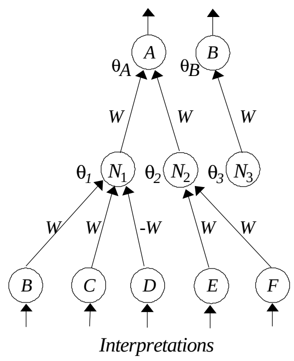

Fig. 4. Sketch of a neural network for the above logic program $\mathcal{P}$.

Note that, in Example 2, each input and output neuron of $\mathcal{N}$ is associated with an atom of $\mathcal{P}$. As a result, each input and output vector of $\mathcal{N}$ can be associated with an interpretation for $\mathcal{P}$. Note also that each hidden neuron N_l corresponds to a rule r_l of $\mathcal{P}$. In order to compute the stable models of $\mathcal{P}$, output neuron B should feed input neuron B such that $\mathcal{N}$ is used to iterate $T_{\mathcal{P}}$, the fixpoint operator of $\mathcal{P}$. $\mathcal{N}$ will eventually converge to a stable state which is identical to the answer set of $\mathcal{P}$ [15].

Notation : Given a general logic program $\mathcal{P}$, let q denote the number of rules r_l ($1 \leq l \leq q$) occurring in $\mathcal{P}$; η, the number of literals occurring in $\mathcal{P}$; A_{min}, the minimum activation for a neuron to be considered "active" (or $true$), $A_{min} \in (0, 1)$; A_{max}, the maximum activation for a neuron to be considered "not active" (or $false$), $A_{max} \in (-1, 0)$; $h(x) = \frac{2}{1+e^{-\beta x}} - 1$, the bipolar semi-linear activation function[3]; $g(x) = x$, the standard linear activation function; $s(x) = y$, the standard nonlinear activation function ($y = 1$ if $x > 0$; and $y = 0$ otherwise), also known as the step function; W (resp. $-W$), the weight of connections associated with positive (resp. negative) literals; θ_l, the threshold of hidden neuron N_l associated with rule r_l; θ_A, the threshold of output neuron A, where A is the head of rule r_l; k_l, the number of literals in the body of rule r_l; p_l, the number of positive literals in the body of rule r_l; n_l, the number of negative literals in the body of rule r_l; μ_l, the number of rules in $\mathcal{P}$ with the same atom in the head, for each rule r_l; $MAX_{r_l}(k_l, \mu_l)$, the greater element among k_l and μ_l for rule r_l; and $MAX_{\mathcal{P}}(k_1, ..., k_q, \mu_1, ..., \mu_q)$, the greatest element among all k's and μ's of $\mathcal{P}$. We also use $\overrightarrow{k}$ as a shorthand for $(k_1, ..., k_q)$, and $\overrightarrow{\mu}$ as a shorthand for $(\mu_1, ..., \mu_q)$.

For instance, for the program $\mathcal{P}$ of Example 2, $q = 3$, $\eta = 6$, $k_1 = 3$, $k_2 = 2$, $k_3 = 0$, $p_1 = 2$, $p_2 = 2$, $p_3 = 0$, $n_1 = 1$, $n_2 = 0$, $n_3 = 0$, $\mu_1 = 2$, $\mu_2 = 2$, $\mu_3 = 1$, $MAX_{r_1}(k_1, \mu_1) = 3$, $MAX_{r_2}(k_2, \mu_2) = 2$, $MAX_{r_3}(k_3, \mu_3) = 1$, and $MAX_{\mathcal{P}}(k_1, k_2, k_3, \mu_1, \mu_2, \mu_3) = 3$.

In the *Translation Algorithm* below, we define A_{min}, W, θ_l, and θ_A such that conditions (**C1**) and (**C2**) above are satisfied. Given a general logic program $\mathcal{P}$, consider that the literals of $\mathcal{P}$ are numbered from 1 to η such that the input and output layers of $\mathcal{N}$ are vectors of length η, where the i-th neuron represents the i-th literal of $\mathcal{P}$. We assume, for mathematical convenience and without loss of generality, that $A_{max} = -A_{min}$. We start by calculating $MAX_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu})$ of $\mathcal{P}$ and A_{min} such that:

$$A_{min} > \frac{MAX_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu}) - 1}{MAX_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu}) + 1}$$

– **Translation Algorithm:**

1. Calculate the value of W such that the following is satisfied:

$$W \geq \frac{2}{\beta} \cdot \frac{ln\,(1 + A_{\min}) - ln\,(1 - A_{\min})}{MAX_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu})\,(A_{\min} - 1) + A_{\min} + 1};$$

[3] We use the bipolar semi-linear activation function for convenience. Any monotonically crescent activation function could have been used here.

2. For each rule r_l of $\mathcal{P}$ of the form $L_1, ..., L_k \to A$ $(k \geq 0)$:
   (a) Add a neuron N_l to the hidden layer of $\mathcal{N}$;
   (b) Connect each neuron L_i $(1 \leq i \leq k)$ in the input layer to the neuron N_l in the hidden layer. If L_i is a positive literal then set the connection weight to W; otherwise, set the connection weight to $-W$;
   (c) Connect the neuron N_l in the hidden layer to the neuron A in the output layer and set the connection weight to W;
   (d) Define the threshold (θ_l) of the neuron N_l in the hidden layer as:
   $$\theta_l = \frac{(1 + A_{\min})(k_l - 1)}{2} W$$
   (e) Define the threshold (θ_A) of the neuron A in the output layer as:
   $$\theta_A = \frac{(1 + A_{\min})(1 - \mu_l)}{2} W$$
3. Set $g(x)$ as the activation function of the neurons in the input layer of $\mathcal{N}$. In this way, the activation of the neurons in the input layer of $\mathcal{N}$, given by each input vector $\mathbf{i}$, will represent an interpretation for $\mathcal{P}$.
4. Set $h(x)$ as the activation function of the neurons in the hidden and output layers of $\mathcal{N}$. In this way, a gradient descent learning algorithm, such as *backpropagation*, can be applied on $\mathcal{N}$ efficiently.
5. If $\mathcal{N}$ ought to be fully-connected, set all other connections to zero.

Theorem 4. *[14, 15] For each propositional general logic program $\mathcal{P}$, there exists a feedforward artificial neural network $\mathcal{N}$ with exactly one hidden layer and semi-linear neurons such that $\mathcal{N}$ computes the fixpoint operator $\mathrm{T}_{\mathcal{P}}$ of $\mathcal{P}$.*

3.2 The Modal *C-IL²P* System

In this section, we extend the *C-IL²P* system to deal with modalities. We use the *Translation Algorithm* presented in Section 3.1 for creating each network of the ensemble, and the following *Modalities Algorithm* for interconnecting the different networks. The *Modalities Algorithm* translates natural deduction rules into the networks. Intuitively, the accessibility relation is represented in the metalevel by connections between (sub)networks, as depicted in Figure 5, where $\mathcal{R}(\omega_1,\omega_2)$ and $\mathcal{R}(\omega_1,\omega_3)$. Connections from ω_1 to ω_2 and ω_3 represent either $\Box$E or $\Diamond$E (Table 1). Connections from ω_2 and ω_3 to ω_1 represent either $\Box$I or $\Diamond$I.

Let $\mathcal{P}$ be an extended modal program with clauses of the form $\omega_i : ML_1, ..., ML_k \to MA$, where each L_j is a literal, A is an atom and $M \in \{\Box, \Diamond\}$, $1 \leq i \leq n$, $0 \leq j \leq k$. As in the case of individual *C-IL²P* networks, we start by calculating $MAX_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu}, n)$ of $\mathcal{P}$ and A_{min} such that:

$$A_{min} > \frac{MAX_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu}, n) - 1}{MAX_{\mathcal{P}}(\overrightarrow{k}, \overrightarrow{\mu}, n) + 1}$$

which, now, also considers the number n of networks (possible worlds) in the ensemble.

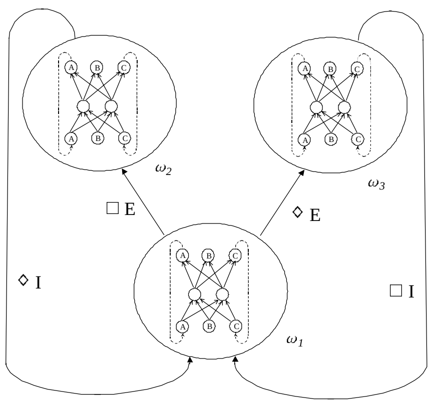

Fig. 5. An ensemble of networks that represents modalities.

– **Modalities Algorithm**

1. Let $\mathcal{P}_i \subseteq \mathcal{P}$ be the set of clauses labelled by ω_i in $\mathcal{P}$. Let $\mathcal{N}_i$ be the neural network that denotes $\mathcal{P}_i$. Let $W^M \in \Re$ be such that $W^M > h^{-1}(A_{\min}) + \mu_l W + \theta_A$, where μ_l, W and θ_A are obtained from *C-IL²P's Translation Algorithm*[4].
2. For each $\mathcal{P}_i$ do:
   (a) Rename each ML_j in $\mathcal{P}_i$ by a new literal not occuring in $\mathcal{P}$ of the form $L_j^\Box$ if $M = \Box$, or $L_j^\Diamond$ if $M = \Diamond$;[5]
   (b) Call *C-IL²P's Translation Algorithm;*
3. For each output neuron $L_j^\Diamond$ in $\mathcal{N}_i$, do:
   (a) Add a hidden neuron L_j^M to an arbitrary $\mathcal{N}_k$ $(0 \le k \le n)$ such that $\mathcal{R}(\omega_i, \omega_k)$;
   (b) Set the step function $s(x)$ as the activation function of L_j^M;
   (c) Connect $L_j^\Diamond$ in $\mathcal{N}_i$ to L_j^M and set the connection weight to 1;
   (d) Set the threshold θ^M of L_j^M such that $-1 < \theta^M < A_{min}$;
   (e) Connect L_j^M to L_j in $\mathcal{N}_k$ and set the connection weight to W^M.
4. For each output neuron $L_j^\Box$ in $\mathcal{N}_i$, do:
   (a) Add a hidden neuron L_j^M to each $\mathcal{N}_k$ $(0 \le k \le n)$ such that $\mathcal{R}(\omega_i, \omega_k)$;
   (b) Set the step function $s(x)$ as the activation function of L_j^M;
   (c) Connect $L_j^\Box$ in $\mathcal{N}_i$ to L_j^M and set the connection weight to 1;
   (d) Set the threshold θ^M of L_j^M such that $-1 < \theta^M < A_{min}$;
   (e) Connect L_j^M to L_j in $\mathcal{N}_k$ and set the connection weight to W^M.

[4] Recall that μ_l is the number of connections to output neuron l.

[5] This allows us to treat each ML_j as a literal and apply the *Translation Algorithm* directly to $\mathcal{P}_i$, by labelling neurons as $\Box L_j$, $\Diamond L_j$, or L_j.

5. For each output neuron L_j in $\mathcal{N}_k$ such that $\mathcal{R}(\omega_i, \omega_k)$ $(0 \le i \le m)$, do:
   (a) Add a hidden neuron $L_j^{\vee}$ to $\mathcal{N}_i$;
   (b) Set the step function $s(x)$ as the activation function of $L_j^{\vee}$;
   (c) For each output neuron $L_j^{\diamond}$ in $\mathcal{N}_i$, do:
      i. Connect L_j in $\mathcal{N}_k$ to $L_j^{\vee}$ and set the connection weight to 1;
      ii. Set the threshold $\theta^{\vee}$ of $L_j^{\vee}$ such that $-nA_{min} < \theta^{\vee} < A_{min} - (n-1)$;
      iii. Connect $L_j^{\vee}$ to $L_j^{\diamond}$ in $\mathcal{N}_i$ and set the connection weight to W^M.
6. For each output neuron L_j in $\mathcal{N}_k$ such that $\mathcal{R}(\omega_i, \omega_k)$ $(0 \le i \le m)$, do:
   (a) Add a hidden neuron $L_j^{\wedge}$ to $\mathcal{N}_i$;
   (b) Set the step function $s(x)$ as the activation function of $L_j^{\wedge}$;
   (c) For each output neuron $L_j^{\Box}$ in $\mathcal{N}_i$, do:
      i. Connect L_j in $\mathcal{N}_k$ to $L_j^{\wedge}$ and set the connection weight to 1;
      ii. Set the threshold $\theta^{\wedge}$ of $L_j^{\wedge}$ such that $n - (1 + A_{min}) < \theta^{\wedge} < nA_{min}$;
      iii. Connect $L_j^{\wedge}$ to $L_j^{\Box}$ in $\mathcal{N}_i$ and set the connection weight to W^M.

Let us now illustrate the use of the *Modalities Algorithm* with the following example.

Example 3. Let $\mathcal{P} = \{\omega_1 : r \rightarrow \Box q,\ \omega_1 : \diamond s \rightarrow r,\ \omega_2 : s,\ \omega_3 : q \rightarrow \diamond p,\ \mathcal{R}(\omega_1,\omega_2),\ \mathcal{R}(\omega_1,\omega_3)\}$. We start by applying *C-IL^{2}P's Translation Algorithm,* which creates three neural networks to represent the worlds ω_1, ω_2, and ω_3 (see Figure 6). Then, we apply the *Modalities Algorithm.* Hidden neurons labelled by $\{M, \vee, \wedge\}$ are created using the *Modalities Algorithm.* The remaining neurons are all created using the *Translation Algorithm.* For the sake of clarity, unconnected input and output neurons are not shown in Figure 6.

Taking $\mathcal{N}_1$ (which represents ω_1), output neurons $L_j^{\diamond}$ should be connected to output neurons L_j in an arbitrary network $\mathcal{N}_i$ (which represents ω_i) to which $\mathcal{N}_1$ is related. For example, taking $\mathcal{N}_i = \mathcal{N}_2$, $\diamond s$ in $\mathcal{N}_1$ is connected to s in $\mathcal{N}_2$.

Then, output neurons $L_j^{\Box}$ should be connected to output neurons L_j in every network $\mathcal{N}_i$ to which $\mathcal{N}_1$ is related. For example, $\Box q$ in $\mathcal{N}_1$ is connected to q in both $\mathcal{N}_2$ and $\mathcal{N}_3$.

Now, taking $\mathcal{N}_2$, output neurons L_j need to be connected to output neurons $L_j^{\diamond}$ and $L_j^{\Box}$ in every world $\mathcal{N}_j$ related to $\mathcal{N}_2$. For example, s in $\mathcal{N}_2$ is connected to $\diamond s$ in $\mathcal{N}_1$ via the hidden neuron denoted by $\vee$ in Figure 6, while q in $\mathcal{N}_2$ is connected to $\Box q$ in $\mathcal{N}_1$ via the hidden neuron denoted by $\wedge$. Similarly, q in $\mathcal{N}_3$ is connected to $\Box q$ in $\mathcal{N}_1$ via $\wedge$. The algorithm terminates when all output neurons have been connected.

We are now in a position to show that the ensemble of neural networks $\mathcal{N}$ obtained from the above *Modalities Algorithm* is equivalent to the original extended modal program $\mathcal{P}$, in the sense that $\mathcal{N}$ computes the *modal immediate consequence operator $MT_{\mathcal{P}}$* of $\mathcal{P}$ (see Definition 5).

Theorem 5. *For any extended modal program $\mathcal{P}$ there exists an ensemble of feedforward neural networks $\mathcal{N}$ with a single hidden layer and semi-linear neurons, such that $\mathcal{N}$ computes the modal fixpoint operator $MT_{\mathcal{P}}$ of $\mathcal{P}$.*

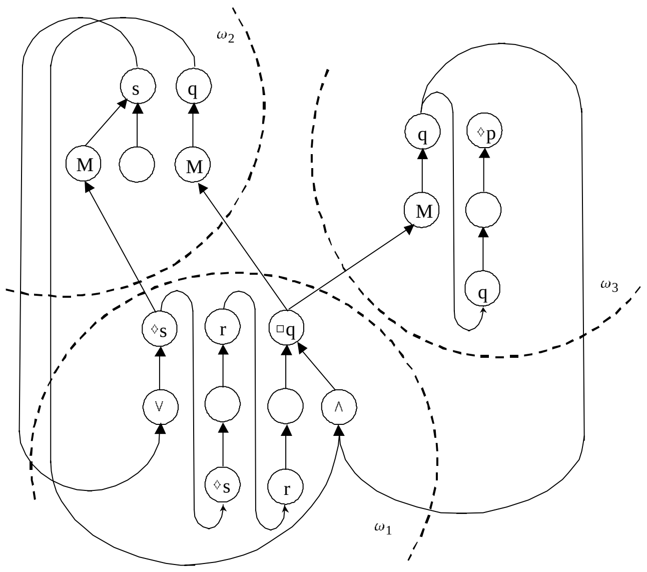

Fig. 6. The ensemble of networks $\{\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3\}$ that represents $\mathcal{P}$.

Proof. We have to show that there exists $W > 0$ such that the network $\mathcal{N}$, obtained by the above Modalities Algorithm, *computes $MT_{\mathcal{P}}$. Throughout, we assume that $\mathcal{N}_i$ and $\mathcal{N}_j$ are two arbitrary sub-networks of $\mathcal{N}$, representing possible worlds ω_i and ω_j, respectively, such that $\mathcal{R}(\omega_i, \omega_j)$. We distinguish two cases: (a) rules with modalities $\Box$ and $\Diamond$ in the head, and (b) rules with no modalities in the head.*

(a) Firstly, note that rules with $\Box$ in the head must satisfy $\Box E$, while rules with $\Diamond$ in the head must satisfy $\Diamond E$ in Table 1. Given input vectors **i** *and* **j** *to $\mathcal{N}_i$ and $\mathcal{N}_j$, respectively, each neuron A in the output layer of $\mathcal{N}_j$ is active ($A > A_{min}$) if and only if: (i) there exists a clause of $\mathcal{P}_j$ of the form $ML_1, ..., ML_k \rightarrow A$ s.t. $ML_1, ..., ML_k$ are satisfied by interpretation* **j**, *or (ii) there exists a clause of $\mathcal{P}_i$ of the form $ML_1, ..., ML_k \rightarrow \Box A$ s.t. $ML_1, ..., ML_k$ are satisfied by interpretation* **i**, *or even (iii) there exists a clause of $\mathcal{P}_i$ of the form $ML_1, ..., ML_k \rightarrow \Diamond A$ s.t. $ML_1, ..., ML_k$ are satisfied by interpretation* **i**, *and the* Modalities Algorithm *(step 3a) has selected $\mathcal{N}_j$ as the arbitrary network $\mathcal{N}_k$.*

($\leftarrow$) (i) results directly from Theorem 4. (ii) and (iii) share the same proof, as follows: from Theorem 4, we know that if $ML_1, ..., ML_k$ are satisfied by interpretation **i** *then MA is active in $\mathcal{N}_i$ (recall, $M \in \{\Box, \Diamond\}$). Hence, we only need to show that MA in $\mathcal{N}_i$ activates A in $\mathcal{N}_j$. From the* Modalities Algorithm, *A^M is a non-linear hidden neuron in $\mathcal{N}_j$. Thus, if MA is active ($MA > A_{min}$) then A^M presents activation 1. As a result, the minimum activation of A is $h(W^M - \mu_A W - \theta_A)$. Now, since $W^M > h^{-1}(A_{min}) + \mu_A W + \theta_A$, we have*

$h(W^M - \mu_A W - \theta_A) > A_{min}$ and, therefore, A is active ($A > A_{min}$). ($\rightarrow$) Directly from the Modalities Algorithm, *since A^M is a non-linear neuron, it contributes with zero to the input potential of A in $\mathcal{N}_j$ when MA is not active in $\mathcal{N}_i$. In this case, the behaviour of A in $\mathcal{N}_j$ is not affected by $\mathcal{N}_i$. Now, from Theorem 4, $\mathcal{N}_j$ computes the fixpoint operator $T_{\mathcal{P}_j}$ of $\mathcal{P}_j$. Thus, if $ML_1, ..., ML_k$ is not satisfied by* $\mathbf{j}$ *then A is not active in $\mathcal{N}_j$.*

(b) Rules with no modalities must satisfy $\Box I$ and $\Diamond I$ in Table 1. Given input vectors $\mathbf{i}$ *and* $\mathbf{j}$ *to $\mathcal{N}_i$ and $\mathcal{N}_j$, respectively, each neuron $\Box A$ in the output layer of $\mathcal{N}_i$ is active ($\Box A > A_{min}$) if and only if: (i) there exists a clause of $\mathcal{P}_i$ of the form $ML_1, ..., ML_k \rightarrow \Box A$ s.t. $ML_1, ..., ML_k$ are satisfied by interpretation* $\mathbf{i}$, *or (ii) for all $\mathcal{N}_j$, there exists a clause of $\mathcal{P}_j$ of the form $ML_1, ..., ML_k \rightarrow A$ s.t. $ML_1, ..., ML_k$ are satisfied by interpretation* $\mathbf{j}$. *Each neuron $\Diamond A$ in the output layer of $\mathcal{N}_i$ is active ($\Diamond A > A_{min}$) if and only if: (iii) there exists a clause of $\mathcal{P}_i$ of the form $ML_1, ..., ML_k \rightarrow \Diamond A$ s.t. $ML_1, ..., ML_k$ are satisfied by interpretation* $\mathbf{i}$, *or (iv) there exists a clause of $\mathcal{P}_j$ of the form $ML_1, ..., ML_k \rightarrow A$ s.t. $ML_1, ..., ML_k$ are satisfied by interpretation* $\mathbf{j}$.

($\leftarrow$) (i) and (iii) result directly from Theorem 4. (ii) and (iv) are proved in what follows: from Theorem 4, we know that if $ML_1, ..., ML_k$ are satisfied by interpretation $\mathbf{j}$ *then A is active in $\mathcal{N}_j$. (ii) We need to show that if A is active in every network $\mathcal{N}_j$ $(0 \leq j \leq n)$ to which $\mathcal{N}_i$ relates to, $\Box A$ is active in $\mathcal{N}_i$. From the* Modalities Algorithm, *$A^\wedge$ is a non-linear hidden neuron in $\mathcal{N}_i$. If A is active ($A > A_{min}$) in $\mathcal{N}_j$, the minimum input potential of $A^\wedge$ is $nA_{min} - \theta^\wedge$. Now, since $\theta^\wedge < nA_{min}$ (*Modalities Algorithm, *step 6(c)ii), the minimum input potential of $A^\wedge$ is greater than zero and, therefore, $A^\wedge$ presents activation* 1. *(iv) We need to show that if A is active in at least one network $\mathcal{N}_j$ $(0 \leq j \leq n)$ to which $\mathcal{N}_i$ relates to, $\Diamond A$ is active in $\mathcal{N}_i$. From the* Modalities Algorithm, *$A^\vee$ is a non-linear hidden neuron in $\mathcal{N}_i$. If A is active ($A > A_{min}$) in $\mathcal{N}_j$, the minimum input potential of $A^\vee$ is $A_{min} - \theta^\vee$. Now, since $\theta^\vee < A_{min} - (n-1)$ (*Modalities Algorithm, *step 5(c)ii), and $n \geqslant 1$, the minimum input potential of $A^\vee$ is greater than zero and, therefore, $A^\vee$ presents activation* 1. *Finally, if $A^\wedge$ presents activation* 1, *the minimum activation of $\Box A$ is $h(W^M - \mu_{\Box A} W - \theta_{\Box A})$, and, exactly as in item (a) above, $\Box A$ is active in $\mathcal{N}_i$. Similarly, if $A^\vee$ presents activation* 1, *the minimum activation of $\Diamond A$ is $h(W^M - \mu_{\Diamond A} W - \theta_{\Diamond A})$, and, exactly as in item (a) above, $\Diamond A$ is active in $\mathcal{N}_i$.*

($\rightarrow$) Again, (i) and (iii) result directly from Theorem 4. (ii) and (iv) are proved below: (ii) We need to show that if $\Box A$ is not active in $\mathcal{N}_i$ then at least one A is not active in $\mathcal{N}_j$ to which $\mathcal{N}_i$ relates to $(0 \leq j \leq n)$. *If $\Box A$ is not active, $A^\wedge$ presents activation* 0. *In the worst case, A is active in $n-1$ networks with maximum activation (*1.0*), and not active in a single network with minimum activation ($-A_{min}$). In this case, the input potential of $A^\wedge$ is $n - 1 - A_{min} - \theta^\wedge$. Now, since $\theta^\wedge > n - (1 + A_{min})$ (*Modalities Algorithm, *step 6(c)ii), the maximum input potential of $A^\wedge$ is smaller than zero and, therefore, $A^\wedge$ presents activation* 0. *(iv) We need to show that if $\Diamond A$ is not active in $\mathcal{N}_i$ then A is not active in any network $\mathcal{N}_j$ to which $\mathcal{N}_i$ relates to* $(0 \leq j \leq n)$. *If $\Diamond A$ is not active, $A^\vee$ presents activation* 0. *In the worst case, A presents activation $-A_{min}$*

*in all $\mathcal{N}_j$ networks. In this case, the input potential of $A^\vee$ is $-nA_{min} - \theta^\vee$. Now, since $\theta^\vee > -nA_{min}$ (*Modalities Algorithm*, step 5(c)ii), the maximum input potential of $A^\vee$ is smaller than zero and, therefore, $A^\vee$ presents activation* 0*. Finally, from Theorem 4, if $A^\wedge$ and $A^\vee$ present activation* 0*, $\mathcal{N}_i$ computes the fixpoint operator $MT_{\mathcal{P}_i}$ of $\mathcal{P}_i$. This completes the proof.* ■

Now, if each network $\mathcal{N}_i$ in the ensemble is transformed into a *partially recurrent network* $\mathcal{N}_i^r$ by linking the neurons in the output layer to the corresponding neurons in the input layer, the ensemble can be used to compute the extended modal program in parallel. For example, in Figure 6, if we connect output neurons $\Diamond s$ and r to input neurons $\Diamond s$ and r, respectively, in $\mathcal{N}_1$, and output neuron q to input neuron q in $\mathcal{N}_3$, the ensemble computes $\{\Diamond s, r, \Box q\}$ in ω_1, $\{s, q\}$ in ω_2, and $\{q, \Diamond s\}$ in ω_3. As expected, these are some of the logical consequences of the original program $\mathcal{P}$ given in Example 3. Although the computation is done in parallel in $\mathcal{N}$, following it by starting from facts (such as s in ω_2) would help in verifying this.

Corollary 1. *Let $\mathcal{P}$ be an acceptable extended modal program. There exists an ensemble of recurrent neural networks $\mathcal{N}^r$ with semi-linear neurons such that, starting from an arbitrary initial input, $\mathcal{N}^r$ converges to a stable state and yields the unique fixpoint ($MT_{\mathcal{P}}^{\varpi}(I)$) of $MT_{\mathcal{P}}$.*

Proof. Assume that $\mathcal{P}$ is an acceptable program. By Theorem 5, $\mathcal{N}^r$ computes $MT_{\mathcal{P}}$. Recurrently connected, $\mathcal{N}^r$ computes the upwards powers ($T_{\mathcal{P}}^m(I)$) of $T_{\mathcal{P}}$. Finally, by Theorem 3, $\mathcal{N}^r$ computes the unique fixpoint ($MT_{\mathcal{P}}^{\varpi}(I)$) of $MT_{\mathcal{P}}$. ■

Hence, in order to use $\mathcal{N}$ as a massively parallel model for modal logic, we simply have to recurrently connect $\mathcal{N}$ with fixed-weight links $W_r = 1$.

4 Conclusions and Future Work

In this paper, we have presented a new massively parallel model for modal logic, thus contributing towards the representation of quantification in neural networks. We have defined an extension of the language Modal Prolog [32, 37], which allows modal operators in the head of clauses. We then presented an algorithm to translate the modal theory into an ensemble of *C-IL²P* neural networks [14, 15], and showed that the ensemble computes a fixpoint semantics of the theory. As a result, the ensemble can be seen as a new massively parallel model for modal logic. In addition, since each *C-IL²P* network can be trained efficiently using the Backpropagation learning algorithm [35], one can adapt the *C-IL²P* ensemble by training possible world representations from examples in each network.

Our next step is to perform experiments on learning possible world representations in the *C-IL²P* ensemble. This would lead us to another interesting avenue of research, namely, rule extraction from neural networks ensembles, which would need to consider extraction methods for more expressive knowledge representation formalisms [13]. In addition, extensions of the basic modal *C-IL²P* ensemble presented in this paper include the study of how to represent properties of other modal logics, e.g., S4 and S5, and of inference and learning of fragments of first order logic. Finally, the addition of modalities to the *C-IL^{2}P* system leads us towards richer distributed knowledge representation and learning mechanisms [16], with a broader range of potential applications, including practical reasoning and learning in a multiagent environment.

References

1. G. Antoniou. *Nonmonotonic Reasoning.* MIT Press, Cambridge, MA, 1997.
2. K. R. Apt and N. Bol. Logic programming and negation: a survey. *Journal of Logic Programming*, 19-20:9–71, 1994.
3. K. R. Apt and D. Pedreschi. Reasoning about termination of pure prolog programs. *Information and Computation*, 106:109–157, 1993.
4. R. Basilio, G. Zaverucha, and V. Barbosa. Learning logic programs with neural networks. In *Inductive Logic Programming.* Springer LNAI 2157, 2001.
5. R. Basilio, G. Zaverucha, and A. S. d'Avila Garcez. Inducing relational concepts with neural networks via the LINUS system. In *Proceedings of the Fifth International Conference on Neural Information Processing ICONIP98*, pages 1507–1510, 1998.
6. M. Baudinet. Temporal logic programming is complete and expressive. In *Proceedings of ACM Symposium on Principles of Programming Languages*, pages 267–280, Austin, Texas, 1989.
7. M. Botta, A. Giordana, and R. Piola. FONN: Combining first order logic with connectionist learning. In *Proceedings of International Conference on Machine Learning ICML97*, pages 46–56. Morgan-Kaufmann, 1997.
8. G. Brewka and T. Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109:297–356, 1999.
9. A. Chagrov and M. Zakharyaschev. *Modal Logic.* Clarendon Press, Oxford, 1997.
10. K. L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322, Plenum Press, New York, 1978.
11. I. Cloete and J. M. Zurada, editors. *Knowledge-Based Neurocomputing.* The MIT Press, 2000.
12. G. Cybenco. Approximation by superposition of sigmoidal functions. In *Mathematics of Control, Signals and Systems 2*, pages 303–314. 1989.
13. A. S. d'Avila Garcez, K. Broda, and D. M. Gabbay. Symbolic knowledge extraction from trained neural networks: A sound approach. *Artificial Intelligence*, 125:155–207, 2001.
14. A. S. d'Avila Garcez, K. Broda, and D. M. Gabbay. *Neural-Symbolic Learning Systems: Foundations and Applications.* Perspectives in Neural Computing. Springer-Verlag, 2002.
15. A. S. d'Avila Garcez and G. Zaverucha. The connectionist inductive learning and logic programming system. *Applied Intelligence Journal, Special Issue on Neural Networks and Structured Knowledge*, 11(1):59–77, 1999.
16. R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge.* MIT Press, 1995.
17. M. Fitting. Metric methods: Three examples and a theorem. *Journal of Logic Programming*, 21:113–127, 1994.

18. Melvin Fitting. *Proof methods for modal and intuitionistic logics.* Reidel, Dordrecht, 1983.
19. D. M. Gabbay. The declarative past and imperative future. In H. Barringer, editor, *Proceedings of the Colloquium on Temporal Logic and Specifications*, LNCS 398. Springer-Verlag, 1989.
20. D. M. Gabbay. *Labelled Deductive Systems*, volume 1. Clarendom Press, Oxford, 1996. Oxford Logic Guides, Vol. 33.
21. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of the fifty Logic Programming Symposium.* MIT Press, 1988.
22. S. Haykin. *Neural Networks: A Comprehensive Foundation.* Prentice Hall, 1999.
23. S. Holldobler and Y. Kalinke. Toward a new massively parallel computational model for logic programming. In *Proceedings of the Workshop on Combining Symbolic and Connectionist Processing, ECAI 94*, 1994.
24. S. Holldobler, Y. Kalinke, and H. P. Storr. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence Journal, Special Issue on Neural Networks and Structured Knowledge*, 11(1):45–58, 1999.
25. G. E. Hughes and M. J. Cresswell. *A new introduction to modal logic.* Routledge, London and New York, 1996.
26. N. Lavrac and S. Dzeroski. *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood Series in Artificial Intelligence, 1994.
27. C. Lewis. *A Survey of Symbolic Logic.* University of California Press, Berkeley, 1918.
28. J. W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1987.
29. R. Moore. Semantical considerations on nonmonotonic logic. *Artificial Intelligence*, 25(1):75–94, 1985.
30. S. Muggleton. Inverse entailment and progol. *New Generation Computing*, 13:245–286, 1995.
31. S. Muggleton and C. Feng. Efficient induction of logic programs. In *Inductive Logic Programming*, pages 453–472. Academic press, 1992.
32. Mehmet A. Orgun and Wanli Ma. An overview of temporal and modal logic programming. In *Proceedings of International Conference on Temporal Logic, ICTL'94*, LNAI 827, pages 445–479. Springer.
33. J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.
34. R. Ramanujam. Semantics of distributed definite clause programs. *Theoretical Computer Science*, 68:203–220, 1989.
35. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press, 1986.
36. Alessandra M. Russo. Generalising propositional modal logic using labelled deductive systems. In F. Baader and K. U. Schulz, editors, *Frontiers of Combining Systems, Applied Logic Series (APLS),*, volume 3, pages 57–74. Kluwer, 1996.
37. Yasubumi Sakakibara. Programming in modal logic: An extension of PROLOG based on modal logic. In *Logic Programming 86*, pages 81–91. Springer LNCS 264, 1986.
38. A. H. Tan. Cascade artmap: Integrating neural computation and symbolic knowledge processing. *IEEE Transactions on Neural Networks*, 8(2):237–250, 1997.
39. G. G. Towell and J. W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1):119–165, 1994.