Computation of Normal Logic Programs by Fibring Neural Networks

Vladimir Komendantsky[1] and Anthony Seda[2]

[1] Boole Centre for Research in Informatics, University College Cork, Cork, Ireland
Postal address: BCRI, 20 South Bank, Crosses Green, Cork, Ireland
v.komendantsky@bcri.ucc.ie
[2] Department of Mathematics, University College Cork, Cork, Ireland
a.seda@ucc.ie* **

Abstract. In this paper, we develop a theory of the integration of fibring neural networks (a generalization of conventional neural networks) into model-theoretic semantics for logic programming. We present some ideas and results about the approximate computation by fibring neural networks of semantic immediate consequence operators T_P and $\mathfrak{T}_P$, where $\mathfrak{T}_P$ denotes a generalization of T_P relative to a many-valued logic analogous to Kleene's strong logic. We establish a minimal-fixed-point semantics for normal logic programs somewhat analogous to the least-fixed-point semantics for definite logic programs. We argue that the class of logic programs for which the approximation by fibring neural networks may be employed to compute minimal fixed points of T_P and of $\mathfrak{T}_P$ is the class of normal programs. Our theorems on the approximation of T_P and $\mathfrak{T}_P$ for normal programs extend recent results on approximation of these operators for definite programs by conventional neural networks.
Key words: logic programs, fibring neural networks, immediate consequence operators, least-fixed-point semantics, Kleene's strong logic.

1 Introduction

The central problem we address here is the integration of logic-based systems and connectionist systems. In brief, this problem is concerned with how one may give a logical interpretation of neural networks (NN, for short), how one may interpret logical reasoning within neural networks, and how one may combine the advantages of each within a single system, see [8].

In this paper, we study generalized neural networks called fibring neural networks (fNNs, for short). These were first introduced in [3] where the definition allows one to treat subnetworks as separate neurons, but in fact we employ fNN as discussed in [1] since the fNNs of the latter paper are more flexible and more suited to our purposes. Indeed, the class of such fNNs may be viewed as a further generalization of the fNNs of [3]. We propose algebraic definitions of NNs as well as of fNNs in a procedural rather than a functional way, and we believe our

* The authors thank the Boole Centre for Research in Informatics (BCRI) at University College Cork for substantial support in presenting this paper. They also thank two anonymous referees whose valuable remarks led to a considerable improvement in the paper.
** To appear in the Proceedings of the Seventh International Workshop on First-Order Theorem Proving (FTP'05), Koblenz, Germany, September 14 - 17, 2005.

treatment of neural networks is well-suited to the task of presenting the proofs of our theorems. The alternative geometic way of describing NNs by means of diagrams (employed in [3, 1]) could be modelled by extracting diagrams from the algebraic notation. On the other hand, one might find the converse extraction difficult, or even impossible, especially in cases where fNNs are concerned, because fNNs commonly have a complicated structure and would therefore require large complicated diagrams. A fibring NN is not a conventional NN (we add the term "conventional" to distinguish between NNs and fNNs). In fact, fibring NNs are more expressive and more powerful then conventional NNs, and for example the fNNs from [3] (and therefore, the fNNs from [1]) can compute polynomial functions over an unbounded domain unlike the case of conventional NNs.

This paper is concerned with the (approximate) computation of the fixed points of semantic operators determined by logic programs, a problem which has been considered by a number of authors, see for example [1, 8, 9, 2]. Specifically, we show how one may use fNNs to compute the minimal fixed points of immediate consequence operators determined by normal logic programs, and this is important in that the minimal-fixed-point semantics can be taken to capture the meaning of first-order normal logic programs.

The overall structure of the paper is as follows. In 2.1, we give preliminary definitions of logic programs with particular attention paid to model-theoretic semantics of logic programming in the form due to van Emden and Kowalski, see [10]. Our notation differs slightly from that to be found in the common reference [10], and is close to Miller's notation [11] as employed in a number of papers concerned with proof-theoretic treatments of logic programming. In 2.2, we define some notions concerning metric spaces and topology [15] used in the semantics of logic programming as discussed in [12]. Next, in 2.3, we give our definitions of neural networks. In 3.1 and in 3.2, we give an overview of recent results on the approximate computation by NNs of semantic operators T_P and $\mathfrak{T}_P$ for definite programs. In 3.3, we develop a many-valued generalization of the stable model semantics for logic programming originally due to Gelfond and Lifschitz [5]. Also in 3.3, the minimal-fixed-point semantics is established as our adaptation of the least-fixed-point semantics of definite programs to the more general class of normal programs. This approach to semantics is somewhat original in that it comprises an application of certain key structures from the semantics, where Scott continuity holds [10], to the analysis of normal programs, where Scott continuity does not hold, as also well-known [10]. Finally, in 3.4 and in 3.5 we present our approach to the approximate computation of immediate consequence operators T_P and $\mathfrak{T}_P$ for normal first-order programs using the ideas of many-valued stable model semantics. Indeed, in 3.4 and 3.5 we clarify the intended meaning of the phrase "computation of normal logic programs" appearing in the title of our paper. In fact, by this term we mean the approximate

computation (up to arbitrarily small error) of the set of minimal fixed points of an immediate consequence operator for a normal logic program.

2 Preliminaries

2.1 Logic Programs

Let the variable A range over the atomic formulae, or atoms, in some fixed first-order signature Σ. In this section, we specify classes of formulae called logic programs, clauses, and bodies in two modes: normal and definite. We let P range over logic programs, we let D range over clauses, and we let G range over bodies of clauses (sometimes also called goals). When convenient, we will allow the use of subscripts on any of these syntactic variables.

Definition 1. *A* normal logic program P *is given by the following recursive assignments:*

$$P ::= D \mid (P_1 \wedge P_2) \ , \tag{1}$$

$$D ::= A \mid (A \leftarrow G) \ , \tag{2}$$

$$G ::= A \mid (\neg A) \mid (G_1 \wedge G_2) \ . \tag{3}$$

Note that P may contain free object variables. These free variables are all thought of as universally quantified, and we omit the quantifiers, as is normally done.

All such normal programs P form the language $\mathcal{L}^{\text{norm}}$. (We will omit brackets where possible.) Alternatively, P can be viewed as a finite set of clauses written in the form $A \leftarrow L_1, \ldots, L_n$, where A is an atomic formula called the *head* of the clause, and $L_1, \ldots, L_n$ denotes a conjunction of *literals* L_i (atoms or negated atoms), for $1 \leq i \leq n$, called the *body* of the clause. The case $n \geq 0$ is allowed and interpreted to mean empty body, that is, the unit clause or fact A. Note that in the present context the original Definition 1 is required, and not its alternative version, because we employ different connectives other than classical conjunction in clause bodies due to our many-valued treatments of logic programs, cf. Definition 5.

Definition 2. *A logic program P is called* definite *if it satisfies (1), (2), and the following assignment:*

$$G ::= A \mid (G_1 \wedge G_2) \ . \tag{4}$$

Thus, each literal L_i in each clause is an atom. We denote the language consisting of all definite logic programs by $\mathcal{L}^{\text{def}}$. (Recall that we work over a fixed first-order signature Σ.)

Remark 1. As can be easily seen, $\mathcal{L}^{\text{def}}$ may be obtained from $\mathcal{L}^{\text{norm}}$ by removing all programs containing occurrences of $\neg$ in their clauses. This observation applies to further definitions for normal programs: to obtain the definite version we just drop $\neg$ from the language.

We let B_P denote the Herbrand base of the program P, namely, the set of all variable free, or ground, instances of atoms formed from the symbols in the signature Σ, and we let ground(P) denote the set of all ground instances of clauses of P. We allow interpretations for logic programs to be many-valued in a specific sense, rather then just two-valued, and we make the following definition.

Definition 3. *An* algebra $\mathcal{A}$ of truth values *is a triple* $(\mathcal{V}, \mathcal{V}', \mathcal{O})$*, where* $\mathcal{V}$ *is a set of truth values,* $\mathcal{V}' \subset \mathcal{V}$ *is a set of designated truth values, and* $\mathcal{O}$ *is a set of operators such that, for all* $o \in \mathcal{O}$*,* $o : \mathcal{O}^n \to \mathcal{O}$ *for some* $n \in \mathbb{N}, n \geq 1$*.*

Remark 2. In the present paper, we always assume that $\mathcal{V}$ is finite, and that $\mathcal{O} = \{\wedge, \vee, \neg\}$, where the operators $\wedge$ and $\vee$ are associative, commutative, idempotent, and distributive with respect to each other, and $\neg$ is a self-dual operator, that is, $\neg(\neg v) = v$ for all $v \in \mathcal{V}$. Additionally, we restrict the class of possible operators to those which satisfy De Morgan's law: $(\neg v_1 \wedge \neg v_2) = \neg(v_1 \vee v_2)$. In other words, $(\mathcal{V}, \wedge, \vee, \neg)$ is a complete distributive lattice with self-dual pseudocomplements (cf. [6], Chapter II, §6), where the lattice ordering is given by $v_1 \leq v_2$ if and only if $v_1 \vee v_2 = v_2$. We use the same symbols for denoting algebraic operators and logical connectives whenever no ambiguity results.

Definition 4. *By an* (Herbrand) interpretation I *for a normal logic program* P *in an algebra* $\mathcal{A} = (\mathcal{V}, \mathcal{V}', \mathcal{O})$ *we mean a mapping* $I : B_P \to \mathcal{V}$ *which assigns to each ground atom* A *in* B_P *a truth value in* $\mathcal{V}$*. We denote by* $I_{P,\mathcal{A}}$ *the set of all interpretations* $I : B_P \to \mathcal{V}$*. We extend an interpretation* I *to normal clauses and to sets of normal clauses by means of the usual inductive definitions. Finally, we call such an interpretation* I *an* (Herbrand) model for P *if, for all clauses* D *in* P*, we have* $I(D) \in \mathcal{V}'$*.*

Let us further note that $I_{P,\mathcal{A}}$ forms a complete lattice whose ordering is given by $I_1 \leq I_2$ if and only if $I_1(A) \leq I_2(A)$ for all $A \in B_P$.

In order to introduce the general immediate consequence operator $\mathfrak{T}_P$, we first specify a language of normal logic pseudo-programs $\mathcal{L}_\psi^{\text{norm}}$, and then define a translation function from the language of programs into the language of pseudo-programs.

Definition 5. *A* normal logic pseudo-program P *is given by the following assignments:*

$$P ::= D \mid (P_1 \wedge P_2) \ , \tag{5}$$
$$D ::= (A \leftarrow \mathbf{0}) \mid (A \leftarrow C) \ , \tag{6}$$
$$C ::= \mathbf{1} \mid G \mid (C_1 \vee C_2) \ , \tag{7}$$
$$G ::= A \mid (\neg A) \mid (G_1 \wedge G_2) \ , \tag{8}$$

where **1** *and* **0** *are nullary connectives, and D in (6) ranges over the class of so called* normal pseudo-clauses. *We allow P, D, and C in this definition to be countably infinite.*

Additionally, for any interpretation I, we require that $I(\mathbf{1}) \in \mathcal{V}'$, and that $I(\mathbf{0}) = \neg I(\mathbf{1})$. The translation function $\psi : \mathcal{L}^{\mathrm{norm}} \to \mathcal{L}^{\mathrm{norm}}_{\psi}$ is defined as follows. Given a (countable) conjunction of normal clauses $\mathcal{P}$, first, put in $\psi(\mathcal{P})$ all clauses in $\mathcal{P}$ whose bodies are non-empty; second, for each unit clause A in $\mathcal{P}$, put $A \leftarrow \mathbf{1}$ in $\psi(\mathcal{P})$, and, for each atom A which occurs in $\mathcal{P}$ but not in the head of any clause in $\mathcal{P}$, put $A \leftarrow \mathbf{0}$ in $\psi(\mathcal{P})$. Finally, whenever there are several clauses $A \leftarrow C_1$, $A \leftarrow C_2$, ... in $\psi(\mathcal{P})$, replace them in $\psi(\mathcal{P})$ with the pseudo-clause $A \leftarrow C_1 \vee C_2 \vee \ldots$. Note that $A \leftarrow C_1 \vee C_2 \vee \ldots$ may be written as $A \leftarrow \bigvee_i C_i$.

Definition 6 (cf. [13]). *Let P be a normal logic program and let $\mathcal{A}$ be a truth-value algebra. We define $\mathfrak{T}_{P,\mathcal{A}} : I_{P,\mathcal{A}} \to I_{P,\mathcal{A}}$ as follows. For any $I \in I_{P,\mathcal{A}}$ and $A \in B_P$, we set*

$$\mathfrak{T}_{P,\mathcal{A}}(I)(A) = I\,(\textstyle\bigvee_i C_i),$$

where $A \leftarrow \bigvee_i C_i$ is the unique pseudo-clause in $\psi(\mathrm{ground}(P))$ whose head is A.

In fact, we will often use the classical two-valued immediate consequence operator T_P which can be viewed as a particular case of $\mathfrak{T}_{P,\mathcal{A}}$.

Definition 7 (cf. [10]). *Let P be a normal logic program, and let $\mathcal{B}$ be the Boolean algebra $(\{0,1\}, \{1\}, \{\wedge, \vee, \neg\})$. We define $T_P : I_{P,\mathcal{B}} \to I_{P,\mathcal{B}}$ as follows. For any $I \in I_{P,\mathcal{B}}$ and $A \in B_P$, we set $T_P(I)(A) = 1$ if and only if either there is a clause $A \leftarrow G$ in* ground(P) *such that $I(G) = 1$ or there is a unit clause $A \leftarrow$; otherwise $T_P(I)(A) = 0$.*

Definition 8. *A* level mapping *for a normal logic program P is a function $\lambda : B_P \to \mathbb{N}$. If $\lambda(A) = k$, for $k \in \mathbb{N}$, we say that the level of A is k.*

We assume that, given P and $n \in \mathbb{N}$, we can effectively find the set of all $A \in B_P$ such that $\lambda(A) = k$. Such a mapping always exists for any first-order language $\mathcal{L}$, and, for example, λ may be defined by taking $\lambda(A)$ to be the number of pairs of brackets used in forming A by means of the usual definitions of term and of atomic formula in $\mathcal{L}$.

2.2 Metrics

Below we introduce background material on metrics. This material is essential for understanding Theorems 2, 3, 6, and 7. We place it here in order to provide a basis for further notation. A *metric* on a set X is a mapping $d : X \times X \to \mathbb{R}$ such that $d(x, y) = 0$ iff $x = y$, $d(x, y) = d(y, x)$, and $d(x, y) \leq d(x, z) + d(z, y)$. The pair (X, d) is called a *metric space*.

Definition 9 ([4, 13]). *Consider a normal logic program P, an algebra $\mathcal{A}$, and a level mapping λ for P. We define the (ultra)metric $\delta : I_{P,\mathcal{A}} \times I_{P,\mathcal{A}} \to \mathbb{R}$ as follows: given $I, J \in I_{P,\mathcal{A}}$, if $I = J$ we set $\delta(I, J) = 0$, and if $I \neq J$ we set $\delta(I, J) = 2^{-k}$, where k is such that I and J differ on some ground atom of level k and agree on all atoms of level less than k relative to the level mapping λ.*

A topological space X is a *Hausdorff space* if and only if whenever x and y are distinct points of X, there are disjoint open sets U and V in X with $x \in U$ and $y \in V$. A space X is *compact* if and only if each open cover of X has a finite subcover. The set $I_{P,\mathcal{A}}$ can be viewed as the product space $\mathcal{V}^{B_P}$, where $\mathcal{V}$ has the discrete topology, and the product space is then a compact Hausdorff space. This topology on $I_{P,\mathcal{A}}$ is called the *Cantor topology*. The following fact is known from the topological semantics of logic programming.

Proposition 1 ([13]). *For any level mapping λ, the metric δ generates the Cantor topology on $I_{P,\mathcal{A}}$, and $(I_{P,\mathcal{A}}, \delta)$ is a compact metric space homeomorphic to the Cantor set in the closed unit interval in $\mathbb{R}$.*

Definition 10 (cf. [13]). *Let (X, d) be a metric space, where d is a bounded metric, and let $T_1, T_2 : X \to X$. We define $\rho(T_1, T_2) = \sup_{x \in X} d(T_1(x), T_2(x))$.*

2.3 Neural Networks

We give next a quite general definition of neural networks in the form in which we use them, see also [7, 8, 2, 14] and the references there.

Definition 11. *Let $l, m \in \mathbb{N}$ with $l, m \geq 1$. A* conventional neural network C *is a system $(\mathcal{N}, \mathcal{N}^{\text{in}}, \mathcal{N}^{\text{out}}, \mathcal{C}, \Phi, \Theta, \mathcal{W})$ consisting of a set $\mathcal{N} = \{n_1, n_2, \ldots, n_l\}$ of* neurons *(or nodes, commonly called* vertices *in graph theory); a set $\mathcal{N}^{\text{in}} \subseteq \mathcal{N}$ of* input neurons*; a set $\mathcal{N}^{\text{out}} \subseteq \mathcal{N}$ of* output neurons*; a set $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$ of* connections *(or* edges*, in graph theory); a set $\Phi = \{\phi_1, \phi_2, \ldots, \phi_l\}$ of* signal functions *associated with the neurons, where $\phi_i : \mathbb{R} \to \mathbb{R}$ for $1 \leq i \leq l$; a set $\Theta = \{\theta_1, \theta_2, \ldots, \theta_l\}$ of* thresholds*, where $\theta_i \in \mathbb{R}$ for $1 \leq i \leq l$; and a set $\mathcal{W} = \{w_1, w_2, \ldots, w_m\}$ of* weights *associated with the connections, where $w_i \in \mathbb{R}$ for $1 \leq i \leq m$. Each connection c_i, for $1 \leq i \leq m$, is associated with an ordered pair $\langle n_j, n_k \rangle$ of neurons, for some $1 \leq j, k \leq l$. The first of these is the* source *$n_j = \text{src}(c_i)$ of the connection, and the second is the* target *$n_k = \text{trg}(c_i)$. For $1 \leq i, j \leq m$, all $c_i, c_j \in \mathcal{C}$ satisfy $\text{src}(c_i) \neq \text{src}(c_j)$ or $\text{trg}(c_i) \neq \text{trg}(c_j)$ whenever $i \neq j$.*

Consider a discrete timescale consisting of equidistant moments of time $t = 0, 1, 2, \ldots$. In the general case, for a neuron n_i, the *activation potential* π_i at the time t is given by

$$\pi_i(0) := 0, \qquad \pi_i(t) := \left(\sum_{j \in D_i} w_{i,j} \sigma_j(t) \right) - \theta_i + \iota_i(t) \quad \text{if } t > 0, \tag{9}$$

where D_i is the set of indices of all those neurons n_j which serve as sources for the connections whose target is n_i; $w_{i,j}$ is the weight of the connection from n_j to n_i at time t; $\sigma_j(t)$ is the output signal emitted by the neuron n_j at the moment of time t; and $\iota_i(t)$ is an external input signal received by n_i at the moment of time t (we set $\iota_i(t) = 0$ if $n_i \notin \mathcal{N}^{\text{in}}$). The *output signal* σ_i of a neuron n_i at time t is computed by evaluating the corresponding signal function and taking the previous time moment's activation potential of that neuron as the argument, thus

$$\sigma_i(0) := 0 \ , \qquad \sigma_i(t) := \phi_i(\pi_i(t-1)) \quad \text{if } t > 0. \tag{10}$$

Let $\mathsf{C} = (\mathcal{N}, \mathcal{N}^{\text{in}}, \mathcal{N}^{\text{out}}, \mathcal{C}, \Phi, \Theta, \mathcal{W})$ be a conventional NN with the property that there is a partition $\mathcal{N}'_1, \mathcal{N}'_2, \ldots, \mathcal{N}'_k$ of its set of neurons satisfying the properties $\mathcal{N}'_1 = \mathcal{N}^{\text{in}}$, and $\mathcal{N}'_k = \mathcal{N}^{\text{out}}$. We call C a *k-layer feedforward NN* if, for $1 \leq i \leq m$, $c_i \in \mathcal{C}$ is such that $\text{src}(c_i) \in \mathcal{N}'_j$ and $\text{trg}(c_i) \in \mathcal{N}'_{j+1}$, for some j with $1 \leq j \leq k-1$. The following fact is well-known.

Theorem 1 ([9]). *For each propositional normal logic program P, it is possible to construct a 3-layer feedforward NN which computes T_P.*

The calculation of fixed points of immediate consequence operators for logic programs is usually done by using 3-layer feedforward NNs with the same number of neurons in $\mathcal{N}^{\text{in}}$ and in $\mathcal{N}^{\text{out}}$, and by adding to the set of connections $\mathcal{C}$, for each $1 \leq j \leq |\mathcal{N}^{\text{out}}| = |\mathcal{N}^{\text{in}}|$, a connection c_j with $\text{src}(c_j) = n_j^{\text{out}}$ and $\text{trg}(c_j) = n_j^{\text{in}}$, where $n_j^{\text{out}} \in \mathcal{N}^{\text{out}}$ and $n_j^{\text{in}} \in \mathcal{N}^{\text{in}}$. Such NNs are called *3-layer feedforward recurrent NNs*.

Remark 3. It is worth noting that there is an uncountable number of different NNs $\mathsf{C} = (\mathcal{N}, \mathcal{N}^{\text{in}}, \mathcal{N}^{\text{out}}, \mathcal{C}, \Phi, \Theta, \mathcal{W})$, for any fixed sets $\mathcal{N}, \mathcal{N}^{\text{in}}, \mathcal{N}^{\text{out}}, \mathcal{C}$, Φ, and Θ. By convention in neural network theory, in certain cases two NNs which are identical to each other except in their sets of weights are treated as the same NN if one of these sets of weights is obtained from the other by some number of applications of a learning algorithm. Though we do not define the general notion of learning and do not use it in this paper, we have a learning-like process in the definition of a fibring NN, as follows.

Definition 12. *Let C_1 and C_2 be two conventional neural networks. A function $\gamma_i : \mathbb{N} \times \mathbb{R}^{m_2+1} \to \mathbb{R}^{m_2}$ such that $\gamma_i : (t, \pi_{i,1}(t), \mathcal{W}_2) \mapsto \mathcal{W}_2^+$, for $t = 1, 2, \ldots$, is called a* fibring function from C_1 to C_2*, denoted by $\gamma_i : \mathsf{C}_1 \bowtie \mathsf{C}_2$, where m_2 is the number of connections in C_2; $\pi_{i,1}(t)$ is the activation potential of the neuron n_i in C_1 at time t; $\mathcal{W}_2$ is the set of weights for C_2 at time t; and $\mathcal{W}_2^+$ is the new set of weights for C_2 at time $t+1$.*

Thus, in this definition, we substitute the set of weights $\mathcal{W}_2$ in C_2 with a new set $\mathcal{W}_2^+$ and still treat the resulting network to be C_2 by the convention of Remark 3.

Definition 13. *Let* C_1 *and* C_2 *be two conventional neural networks. We say that* C_2 is fibred into C_1 with the fibring function γ_i *if* γ_i *is a fibring function from* C_1 *to* C_2*. We call the system* $\mathsf{F} = (\mathsf{C}_1, \mathsf{C}_2, \gamma_i)$ *a* fibring neural network *(or fNN). In the general case, we will call a fibring neural network a system* $(\mathsf{C}_1, \mathsf{C}_2, \gamma_{i_1}, \ldots, \mathsf{C}_{2k-1}, \mathsf{C}_{2k}, \gamma_{i_{2k-1}})$ *where* $\mathsf{C}_1, \ldots, \mathsf{C}_{2k}$ *are conventional neural networks, and* $\gamma_{i_{2j-1}} : \mathsf{C}_{2j-1} \bowtie \mathsf{C}_{2j}$ *for* $1 \leq j \leq k$*.*

Remark 4. Note that we adopt the following convention concerning the priority in which fibring functions are to be evaluated in order to avoid ambiguity. If in an fNN at some time t there is a connection whose weight w gets changed by fibring functions $\gamma_{i_1}, \ldots, \gamma_{i_k}$, for $k \in \mathbb{N}$, then: (1) first, amongst all the γ_{i_j} such that $\gamma_{i_j}(t, \pi_{i_j,1}(t), w)$ is constant, with value w^+ say, for varying w, only that fibring function with minimum value w^+ is applied; (2) second, the γ_{i_j} such that $\gamma_{i_1}, \ldots, \gamma_{i_k}$ is non-constant for varying w are applied in arbitrary order; (3) no other fibring functions are applied.

3 Integration of fNNs into Fixed-Point Semantics

3.1 Computation by NNs of T_P for Definite Programs

We begin describing our approach to the integration of fNNs into least-fixed-point semantics by defining the basic case of the integration, namely, the case of definite programs.

Definition 14 ([13]). *Let* C *be a 3-layer feedforward NN with* m *units in the input layer, and* k *units in the output layer. We consider the* input-output mapping f_{C} of C *as a mapping* $f_{\mathsf{C}} : I_{P,\mathcal{A}} \to I_{P,\mathcal{A}}$ *as follows. Given* $I \in I_{P,\mathcal{A}}$*, we present the vector* $(I(A_{i_1}), \ldots, I(A_{i_m}))$ *to the input layer at time* $t := t'$*. After propagation through the network, at time* $t := t' + 3$ *we determine* $f_{\mathsf{C}}(I)$ *by taking the value of* $f_{\mathsf{C}}(I)(A_{o_j})$ *to be the value in the* j*-th unit in the output layer,* $j = 1, \ldots, k$*, and by taking all other values of* $f_{\mathsf{C}}(I)(A_{o_j})$ *to be a fixed value* v *such that* $v = \neg v'$*, for some* $v' \in \mathcal{V}'$*.*

Definition 15 ([13]). *Suppose that* $\mathcal{M}$ *is a fixed point of* $\mathfrak{T}_P$*. We say that a* family $\mathbf{C} = \{\mathsf{C}_i \mid i \in \mathcal{I}\}$ of 3-layer feedforward recurrent NNs C_i computes $\mathcal{M}$ *if there exists* $I \in I_{P,\mathcal{A}}$ *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* $\delta(f^m_{\mathsf{C}_i}(I), \mathcal{M}) < \varepsilon$ *where* $f^m_{\mathsf{C}_i}(I)$ *denotes the* m*-th iterate of* f_{C_i} *applied to* I*.*

Remark 5 ([13]). Suppose that $\mathbf{C} = \{\mathsf{C}_i \mid i \in I\}$ computes $\mathcal{M}$, as just defined. Taking $\varepsilon = \frac{1}{n}$, for $n = 1, 2, \ldots$, and applying the definition to each of these values of ε in turn, we obtain a sequence of elements i_n of $\mathcal{I}$ and a sequence of natural numbers m_{i_n} such that for all $m \geq m_{i_n}$ we have $\delta(f^m_{\mathsf{C}_{i_n}}(I), \mathcal{M}) < \frac{1}{n}$. Write m_n for $m_{\mathsf{C}_{i_n}}$ and, given any $\varepsilon > 0$, choose n_0 so large that $\frac{1}{n_0} < \varepsilon$. Then we see that

we have a sequence (C_n) of elements of $\mathbf{C}$ which satisfies the property that once a level of approximation is reached, all subsequent approximations are at least as good.

Theorem 2 ([13]). *Let P be an arbitrary definite program, let I denote the least fixed point of T_P and suppose that we are given $\varepsilon > 0$. Then there exists a program $\overline{P} = \overline{P}(\varepsilon)$ which is a finite subset of* ground(P) *such that $\delta(\overline{I}, I) < \varepsilon$, where $\overline{I}$ denotes the least fixed point of $T_{\overline{P}}$. Therefore, the family $\{\mathsf{C}_n \mid n \in \mathbb{N}\}$ computes I, where C_n denotes the neural network obtained by applying the algorithm of Theorem 1 to $\overline{P}(2^{-n})$, for $n = 1, 2, 3, \ldots$. Furthermore, the sequence (C_n) has the property stated in Remark 5.*

3.2 Computation by NNs of $\mathfrak{T}_{P,\mathcal{A}}$ for Definite Programs

Definition 16 ([13]). *We say that a* family $\mathbf{C} = \{\mathsf{C}_i \mid i \in \mathcal{I}\}$ of 3-layer feedforward recurrent NNs computes $\mathfrak{T}_{P,\mathcal{A}}$ *if, given any $\varepsilon > 0$, there is an index $i \in \mathcal{I}$ such that $\rho(f_{\mathsf{C}_i}, \mathfrak{T}_{P,\mathcal{A}}) < \varepsilon$.*

Using Definition 16, we can prove the following theorem by using the fact that any interpretation $I : B_P \to \mathcal{V}$, for a finite $\mathcal{V}$, can be embedded into $\mathbb{R}$ by means of expansions of decimal type with base $2k-1$, where k is the number of truth values in $\mathcal{V}$, using only the k even numbers $0, 2, \ldots, 2k-2$ in the expansion.

Theorem 3 ([13]). *There is a family $\mathbf{C} = \{\mathsf{C}_i \mid i \in \mathcal{I}\}$ of 3-layer feedforward recurrent NNs which computes $\mathfrak{T}_{P,\mathcal{A}}$ if and only if $\mathfrak{T}_{P,\mathcal{A}}$ is continuous in the Cantor topology on $I_{P,\mathcal{A}}$.*

3.3 Stable Models

There are two distinctive features of a normal logic program P which makes it hard to analyse the denotational meaning of such a program. These are the discontinuity and non-monotonicity of the immediate consequence operator associated with the program. The consequences of these "bad" features are, respectively, inaccessibility of the least fixed point of the associated immediate consequence operator in ω steps, and even the absence of unique fixed points of this operator in the general case. Despite these consequences, the underlying operator might still have minimal fixed points. These minimal fixed points are contained in the set of all stable models for P, as defined in [5]. We recall next the definition of stable models, modifying it slightly to suit our notation.

Definition 17. *Let P be a normal logic program. For a set $\mathcal{M} = \{A_1, A_2, \ldots\} \subseteq B_P$ of atoms, let $P_{\mathcal{M}}$ be the set of clauses obtained from* ground(P) *by deleting, for $1 \leq i \leq |\mathcal{M}|$, each clause $A \leftarrow C$ that has $\neg A_i$ in C, where $A_i \in \mathcal{M}$, and then deleting all negative literals in the bodies of the remaining clauses. If a*

minimal Herbrand model for $P_{\mathcal{M}}$ coincides with $\mathcal{M}$, then we say that $\mathcal{M}$ is a stable model *for P. Such models can also be described as the fixed points of the operator S_P defined by the condition: for any set $\mathcal{M} \subseteq B_P$, $S_P(\mathcal{M}) = T_{P_{\mathcal{M}}}$.*

Note that the program $P_{\mathcal{M}}$ in Definition 17 is definite whether or not P is definite. Now we give a many-valued (Kleene's strong) version of Definition 17.

Definition 18. *Let P be a normal logic program. For any interpretation $I \in I_{P,\mathcal{A}}$, let P_I be the set of clauses obtained from* ground(P) *by deleting, for $1 \leq i \leq |I|$, each clause $A \leftarrow C$ that has $\neg A_i$ in C with $I(A_i) \in \mathcal{V}'$, and then deleting all negative literals in the bodies of the remaining clauses. If a minimal Herbrand model for P_I in the algebra $\mathcal{A}$ coincides with I, then we say that I is a* stable (many-valued) model for P with respect to $\mathcal{A}$. *Such models can also be described as the fixed points of the operator $\mathfrak{S}_{P,\mathcal{A}}$ defined by the condition: for any interpretation $I \in I_{P,\mathcal{A}}$, $\mathfrak{S}_{P,\mathcal{A}}(I) = \mathfrak{T}_{P_I,\mathcal{A}}$.*

Again, P_I is always a definite program as in Definition 17. As a generalization of Theorem 1 from [5], we have the following result.

Theorem 4. *Any stable many-valued model for a normal program P in an algebra $\mathcal{A}$ is a minimal Herbrand model for P in $\mathcal{A}$.*

Proof. Consider a stable many-valued model I. First, we must show that I is a model for P in $\mathcal{A}$. Let D be a clause from ground(P). If the body of D contains a literal $\neg A$ such that $I(A) \in \mathcal{V}'$, then $I(D) \in \mathcal{V}'$. If the body of D does not contain such a literal, consider the clause D_1 obtained from D by deleting all negative literals from its body. Since D_1 is one of the clauses of P_I, and I is the minimal model for P_I, it is clear that $I(D_1) \in \mathcal{V}'$. At the same time, $I(D_1) \leq I(D)$, and thus $I(D) \in \mathcal{V}'$. To show that I is minimal, let I_1 be a model for P such that $I_1 \leq I$. We will show that I_1 is also a model for P_I. Consider any clause D_1 from ground(P_I). It is obtained from some clause D in ground(P) by deleting all negative literals from its body, and, in every such literal $\neg A$, $I(A) \notin \mathcal{V}'$. To show that $I_1(D_1) \in \mathcal{V}'$, observe that (1) $I_1(D) \in \mathcal{V}'$ (since I_1 is a model for P); (2) every negative literal $\neg A$ in the body of D is such that $I_1(A) \notin \mathcal{V}'$ (since $I(A) \notin \mathcal{V}'$ and $I_1 \leq I$); and (3) D_1 is obtained from D by removing these negative literals. Since I is the minimal model for P_I, $I_1 = I$. □

We base our subsequent results on the following theorem about the relationship between the set of all minimal fixed points of the immediate consequence operator $\mathfrak{T}_{P,\mathcal{A}}$ and the set of all stable models for P.

Theorem 5. *Let P be a normal program. An interpretation I is a minimal fixed point of $\mathfrak{T}_{P,\mathcal{A}}$ if and only if I is a stable model for P in $\mathcal{A}$ and $\mathfrak{T}_{P,\mathcal{A}}(I) = I$.*

Proof. ($\Rightarrow$) If I is a minimal fixed point of $\mathfrak{T}_{P,\mathcal{A}}$ then, by the definition of a minimal fixed point [10], $\mathfrak{T}_{P,\mathcal{A}}(I) = I$. Let I be a non-stable model for P. Then

the minimal Herbrand model for P_I (as obtained by the algorithm described in Definition 18) does not coincide with I, that is $\mathfrak{T}_{P_I,\mathcal{A}}(I) \neq I$. Since, by simple lattice-theoretical arguments, $\mathfrak{T}_{P_I,\mathcal{A}} \leq \mathfrak{T}_{P,\mathcal{A}}$, the inequality $\mathfrak{T}_{P_I,\mathcal{A}}(I) \neq I$ entails $\mathfrak{T}_{P,\mathcal{A}}(I) \neq I$. This contradiction shows that I is a stable model for P.
($\Leftarrow$) Follows from Definition 18, Theorem 4, and the fact that any minimal fixed point of $\mathfrak{T}_{P,\mathcal{A}}$ is a minimal Herbrand model for P in $\mathcal{A}$. □

Corollary 1. *Let P be a normal program. The set of minimal fixed points of $\mathfrak{T}_{P,\mathcal{A}}$ is contained in or is equal to the set of stable models for P in $\mathcal{A}$.*

The following two results are the obvious restrictions of, respectively, Theorem 5 and Corollary 1 to the case of interpretations in the two-element Boolean algebra $\mathcal{B}$ given in Definition 7.

Corollary 2. *Let P be a normal program. An interpretation I is a minimal fixed point of T_P if and only if I is a stable model for P in $\mathcal{B}$ and $T_P(I) = I$.*

Corollary 3. *Let P be a normal program. The set of minimal fixed points of T_P is contained in or is equal to the set of stable models for P in $\mathcal{B}$.*

3.4 Computation by fNNs of Minimal Fixed Points of T_P for Normal Programs

Speaking informally, the idea behind the approximate computation of minimal fixed points of immediate consequence operators for a normal program P is the following. Given the set $\mathbf{M}$ of stable models for P (either two-valued or many-valued in our restricted sense), we want to know which of these stable models are also minimal fixed points of an immediate consequence operator for P. For this purpose, we construct $|\mathbf{M}|$ feedforward NNs computing the corresponding ground definite programs P_M, for each $M \in \mathbf{M}$. To each such NN, we connect the NN computing the corresponding ground instance of the original program P. At time t', we present input vectors, interpretations for M, to the subnetwork calculating P_M, for each $M \in \mathbf{M}$. Once we have reached the fixed point of the underlying function computed by the NN for P_M (let us call this subnetwork C_1), we present (at the time $t' + 3l$, for $l = 1, 2, \ldots$, by means of a suitable set of fibring functions which fibre input and output neurons of C_1 into a set of additional subnetworks having 4 layers and just 4 neurons for determining the fixed point of f_{C_1}) the output vector of this NN to its associated NN (call this subnetwork C_2) computing the corresponding ground instance of the normal program. The criterion for determining a fixed point of the function computed by C_1 is receipt of the input signal 1 by the neuron in the fourth layer in each of the additional NNs. Then, at the time $t' + 3l + 4$, we read out the output vector of C_2 and compare it with the input vector (given at the time $t' + 3l$; again, a similar construction with fibred subnetworks is used for determining

fixed points of f_{C_2}). If these vectors are equal, the fNN composed of C_1, C_2, and the additional constructions, outputs the right answer, the output vector of C_2, otherwise the output of C_2 is not the required answer. The stable model M is a minimal fixed point of T_P if and only if the corresponding fNN outputs positive answer. Now let us now present a more detailed description of this process.

Let F be an fNN with m_1 input units, and $2m_2$ output units, where $m_1, m_2 \in \mathbb{N}$, and $m_1 \leq m_2$. We consider the *input-output mapping* f_{F} *of* F as a mapping $f_{\mathsf{F}} : I_{P,\mathcal{A}} \to I_{P,\mathcal{A}}$ as follows. Given $I \in I_{P,\mathcal{A}}$, we input the vector $(I(A_{i_1}), \ldots, I(A_{i_{m_1}}))$ to the input neurons at time $t := t'$. After propagation through the network, at time $t := t' + 3l + 4$, where $l = 1, 2, \ldots$, we determine $f_{\mathsf{F}}(I)$ by taking the value of $f_{\mathsf{F}}(I)(A_{o_j})$ to be the output signal of the j-th output unit, $j = 1, \ldots, m_2$, and by taking the values of $f_{\mathsf{F}}(I)(A_k)$, for $A_k \notin \{A_{o_j} \mid 1 \leq j \leq m_2\}$, to be equal to a fixed value v such that $v = \neg v'$, for some $v' \in \mathcal{V}'$. For the case when $\mathcal{A}$ is $\mathcal{B}$, v is 0.

Definition 19. *Suppose that* $\mathbf{M}$ *is the set of all minimal fixed points of* T_P. *We say that a* family $\mathbf{F} = \{\mathsf{F}_i \mid i \in \mathcal{I}\}$ of fNNs F_i computes $\mathbf{M}$ *if there exists* $I \in I_{P,\mathcal{A}}$ *such that the following holds for any* $M \in \mathbf{M}$*: given any* $\varepsilon > 0$*, there is an index* $i \in \mathcal{I}$ *such that we have* $\delta(f_{\mathsf{F}_i}(I), M) < \varepsilon$.

Theorem 6. *Let* P *be an arbitrary normal program, let* $\mathbf{M}$ *denote the set of all minimal fixed points of* T_P*, choose any* $M \in \mathbf{M}$ *and suppose that we are given* $\varepsilon > 0$. *Then there exists a set of programs* $\overline{\mathbf{P}} = \overline{\mathbf{P}}(\varepsilon)$ *which is a set of finite subsets of* ground(P) *such that, for every* $\overline{P} \in \overline{\mathbf{P}}$ *and, for a minimal fixed point* M *of* T_P*, there is a minimal fixed point* $\overline{M}$ *of* $T_{\overline{P}}$ *such that* $\delta(\overline{M}, M) < \varepsilon$. *Therefore, there exists a family* $\mathbf{F} = \{\mathsf{F}_{n,k} \mid n, k \in \mathbb{N}\}$ *of fNNs which computes* $\mathbf{M}$.

Proof. The existence of a set $\overline{\mathbf{P}}(\varepsilon)$ of programs is proved by applying Theorem 2 and Theorem 5 to every element of $\mathbf{M}$ and by taking into account the fact that all the $\overline{P}$ are sets of definite clauses; thus each $\overline{P}$ has a unique least fixed point. This allows one to calculate these least fixed points by means of Theorem 2. Let $\overline{\mathbf{M}'}$ be a set of stable models for $\overline{P}$. The family $\mathbf{F}$ is constructed as a family of fNNs, where each n-th subfamily $\{\mathsf{F}_{n,k} \mid 0 \leq k \leq |\overline{\mathbf{M}'}|\}$ consists of $|\overline{\mathbf{M}'}|$ pairwise different fNNs $\mathsf{F}_{n,k}$ of the kind $\mathsf{F}_{n,k} =$

$$(\mathsf{C}_{1+2}, \mathsf{C}_{g_1}, \gamma_{i_1}; \ldots; \mathsf{C}_{1+2}, \mathsf{C}_{g_m}, \gamma_{i_{2m}}; \mathsf{C}_{g_1}, \mathsf{C}_{g_{m_1+1}}, \gamma_{i_{2m+1}}; \ldots; \mathsf{C}_{g_{m_1}}, \mathsf{C}_{g_{2m_1+1}}, \gamma_{i_{2m+m_1}}) \ ,$$

for each $\overline{M'} \in \overline{\mathbf{M}'}$, where

1. C_1 is a recurrent 3-layer feedforward NN computing $\overline{P}_{\overline{M'}}$ constructed according to Theorem 1 with m_1 neurons in the input layer, and with added output neurons which represent atoms of $\overline{P}$ which are absent in $\overline{P}_{\overline{M'}}$ to the output layer (these neurons are sending the constant value 0);

2. C_2 is a non-recurrent 3-layer feedforward NN computing $\overline{P}$ with m_2 neurons in the input layer, and with an added fourth layer where the neurons perform identity mapping on their activation potentials, that is, this added layer simply retransmits output signals received from the third layer, and thus C_2 does indeed consists of four layers;
3. C_{1+2} is a 7-layer NN composed by connecting output neurons of C_1 to the input neurons of C_2 given that the connected pairs of neurons represent the same ground atoms;
4. $m = m_1 + m_2$;
5. $\mathsf{C}_{g_1}, \ldots, \mathsf{C}_{g_m}$ are pairwise different additional 4-layer networks each with no input neurons, m_1 without output neurons, m_2 with output neurons, and each with a single neuron per layer: the first neuron outputs a constant signal 1 starting from the time t', the other neurons retransmit their activation potentials, and the weights are initially set to the value 1;
6. $i_1, \ldots, i_{m_1}$ in the subscripts of the fibring functions are pairwise different numbers of input nodes in C_1; $i_{m_1+1}, \ldots, i_{2m_1}$ are pairwise different numbers of output nodes in C_1; $i_{2m_1+1}, \ldots, i_{2m_1+m_2}$ are pairwise different numbers of input nodes in C_2; $i_{2m_1+m_2+1}, \ldots, i_{2m}$ are pairwise different numbers of output nodes in C_2; additionally, if $\gamma_{i_{2j-1}} : \mathsf{C}_1 \bowtie \mathsf{C}_k$, for $1 \leq k \leq m$, then also $\gamma_{i_{2j}} : \mathsf{C}_1 \bowtie \mathsf{C}_k$, and if $\gamma_{i_{2j-1}} : \mathsf{C}_2 \bowtie \mathsf{C}_k$, for $1 \leq k \leq m$, then also $\gamma_{i_{2j}} : \mathsf{C}_2 \bowtie \mathsf{C}_k$; moreover, each of the above mentioned nodes in C_1 and in C_2 fibres exactly one additional NN, and, for each additional NN C_k, if it is fibred by some neuron representing an atom A, then no neuron representing an atom other than A may fibre C_k;
7. $i_{2m+1}, \ldots, i_{2m+m_1}$ are numbers of neurons in the last layer of the corresponding additional NNs $\mathsf{C}_{g_1}, \ldots, \mathsf{C}_{g_{m_1}}$;
8. for $1 \leq j \leq m_1$, if C_{g_j} is fibred by a neuron in C_1 representing an atom A, then $\mathsf{C}_{g_{m_1+j}}$ is fibred by a neuron in C_2 representing the same atom A;
9. the fibring functions are as follows:
   (a) let $1 \leq j \leq m$, let t denote time, and let $q > 0$ be some fixed value, then $\gamma_{i_{2j-1}} : \mathsf{C}_{1+2} \bowtie \mathsf{C}_{g_j}$ is such that if $\mathcal{W} = \{w_1, w_2, w_3\}$ is a set of weights of C_{g_j} (which connect, respectively, the 1st layer to the 2nd, etc.) then γ_{i_j} maps w_1 to $\pi_{i_j}(t) + q$, and does not change w_2 and w_3;
   (b) next, let j, t, and q be as above, then $\gamma_{i_{2j}} : \mathsf{C}_{1+2} \bowtie \mathsf{C}_{g_j}$ is such that if $\mathcal{W} = \{w_1, w_2, w_3\}$ is a set of weights of C_{g_j}, then γ_{i_j} does not change w_1 and w_2, and maps w_3 to $1/(\pi_{i_j}(t) + q)$;
   (c) let $1 \leq j \leq m_1$, and let t denote time, then $\gamma_{i_{2m+j}} : \mathsf{C}_{g_j} \bowtie \mathsf{C}_{g_{m_1+j}}$ is such that if $\mathcal{W} = \{w_1, w_2, w_3\}$ is a set of weights of $\mathsf{C}_{g_{m_1+j}}$, then $\gamma_{i_{2m+j}}$ maps w_1 to $w_1 \cdot (-1)$ if $\pi_{i_{2m+j}}(t) \neq 1$; otherwise it does not change w_1; and in any case it does not change either w_2 or w_3.

Using the fNN $\mathsf{F}_{n,k}$ constructed according to the rules above, each time when the least fixed point $\overline{M}$ has been calculated by the corresponding C_1 at time

$t' + 3l$, for $l = 1, 2, \ldots$, the NN C_2 is applied one more time to the output of C_1. If $f_{\mathsf{C}_2}(\overline{M}) = \overline{M}$ at time $t' + 3l + 4$, that is, if each of the NNs $\mathsf{C}_{g_{m_1+1}}, \ldots, \mathsf{C}_{g_m}$ outputs 1 at time $t' + 3l + 4$, then $\overline{M}$ is a minimal fixed point of $T_{\overline{P}}$, and, as guaranteed by Theorem 2, $\delta(\overline{M}, M) < \varepsilon$, where M is a minimal fixed point of T_P. □

3.5 Computation by fNNs of Minimal Fixed Points of $\mathfrak{T}_P$ for Normal Programs

Definition 20. *Given an arbitrary normal program P and a truth-value algebra $\mathcal{A}$ as defined in Remark 2, we say that a* family $\mathbf{F} = \{\mathsf{F}_i \mid i \in \mathcal{I}\}$ of fNNs computes the set of all minimal fixed points $\mathbf{M}$ of $\mathfrak{T}_{P,\mathcal{A}}$ *if, for any $M \in \mathbf{M}$ and any $\varepsilon > 0$, there is an index $i \in \mathcal{I}$ such that $\rho(f_{\mathsf{F}_i}, \mathfrak{S}_{P,\mathcal{A}}(M)) = \rho(f_{\mathsf{F}_i}, \mathfrak{T}_{P_M,\mathcal{A}}) < \varepsilon$.*

Theorem 7. *For an arbitrary normal logic program P and a truth-value algebra $\mathcal{A}$ satisfying the restrictions of Remark 2, there is a family $\mathbf{F} = \{\mathsf{F}_{n,k} \mid n, k \in \mathbb{N}\}$ of fNNs which computes the set of all minimal fixed points $\mathbf{M}$ of $\mathfrak{T}_{P,\mathcal{A}}$.*

Proof (sketch). By analogy with Theorem 3, we again use the fact that any interpretation $I : B_P \to \mathcal{V}$, for a finite $\mathcal{V}$, is embeddable into $\mathbb{R}$ by means of expansions of decimal type with base $2k - 1$. Also, we use the fact that the functions which are computed by $\mathsf{F}_{i,j}$ are all continuous (due to the construction of $\overline{P}$ which contains definite pseudo-clauses only, cf. Theorem 6). Thus, the proof essentially follows the proof of Theorem 3. □

4 Conclusions and Further Work

We have presented an approach to the approximate computation of normal first-order logic programs by means of fibring neural networks. Our approach is based on the definite programs produced by means of the Gelfond–Lifschitz (stable model) transformation [5], and the approximate computation of these definite programs by means of conventional neural networks [13]. This is satisfactory in that we can approximate not just a single step of an immediate consequence operator, but the class of minimal fixed points of this operator for a given normal program, and this class can be taken to be the denotational meaning of the program. One can see several ways of solving the problem of approximate computation of first-order normal logic programs: (1) our fNN-based approximation using the stable model transformation of normal programs into finite sets of definite programs; (2) other possible fNN-based approximations using, say, the well-founded model; (3) NN-based approximation using ideas similar to those employing definite programs in [13] applied to wider classes of logic programs (for a treatment of covered programs, see [2]). It would be of interest to compare these approaches in the future.

References

1. Sebastian Bader, Arthur d'Avila Garcez, and Pascal Hitzler. Computing first-order logic programs by fibring artificial neural networks. In *Proceedings of the 18th International FLAIRS Conference*, Clearwater Beach, Florida, May 2005. To appear.
2. Sebastian Bader, Pascal Hitzler, and Andreas Witzel. Integrating first-order logic programs and connectionist systems — a constructive approach. In *Proceedings of IJCAI NeSy'05*, Edinburgh, August 2005.
3. Artur S. d'Avila Garcez and Dov M. Gabbay. Fibring neural networks. In *Proceedings of 19th National Conference on Artificial Intelligence (AAAI 04)*. AAAI Press, July 2004.
4. Melvin Fitting. Metric methods: Three examples and a theorem. *The Journal of Logic Programming*, 21(3):113–127, 1994.
5. Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth A. Bowen, editors, *Logic Programming. Proceedings of the 5th International Conference and Symposium on Logic Programming*, pages 1070–1080. MIT Press, 1988.
6. George Grätzer. *General lattice theory*. Birkhauser-Verlag, Basel, 2nd edition, 1998.
7. Michael J. Healy and Thomas P. Caudell. Neural networks, knowledge, and cognition: a mathematical semantic model based upon category theory. Technical Report EECE-TR-04-020, Department of Electrical and Computer Engineering, School of Engineering, University of New Mexico, 2004.
8. Pascal Hitzler, Steffen Hölldobler, and Anthony K. Seda. Logic programs and connectionist networks. *Journal of Applied Logic*, 2(3):245–272, 2004.
9. Steffen Hölldobler and Yvonne Kalinke. Towards a new massively parallel computational model for logic programming. In *Proceedings ECAI94 Workshop on Combining Symbolic and Connectionist Processing*, pages 68–77. ECCAI, 1994.
10. John W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 1988.
11. Dale Miller. A logical analysis of modules in logic programming. *Journal of Logic Programming*, 1989.
12. Anthony K. Seda. Quasi-metrics and the semantics of logic programs. *Fundamenta Informaticae*, 29(1):97–117, 1997.
13. Anthony 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.
14. Anthony K. Seda and Máire Lane. On approximation in the the integration of connectionist and logic-based systems. In L. Li and K.K. Yen, editors, *Proceedings of the third international conference on information*, pages 297–300, November 2004.
15. Stephen Willard. *General Topology*. Addison-Wesley, Reading, MA, 1970.