Logic Programs and Connectionist Networks*

Pascal Hitzler[†], Steffen Hölldobler[†], Anthony Karel Seda[‡]

[†]Technische Universität Dresden, Department of Computer Science,
01062 Dresden, Germany
[‡]Department of Mathematics, University College Cork,
Cork, Ireland

Abstract

One facet of the question of integration of Logic and Connectionist Systems, and how these can complement each other, concerns the points of contact, in terms of semantics, between neural networks and logic programs. In this paper, we show that certain semantic operators for propositional logic programs can be computed by feedforward connectionist networks, and that the same semantic operators for first-order normal logic programs can be approximated by feedforward connectionist networks. Turning the networks into recurrent ones allows one also to approximate the models associated with the semantic operators. Our methods depend on a well-known theorem of Funahashi, and necessitate the study of when Funahasi's theorem can be applied, and also the study of what means of approximation are appropriate and significant.

Keywords Logic Programming, Metric Spaces, Connectionist Networks.

1 Introduction

It is widely recognized that Logic and Neural Networks are two rather distinct yet major areas within Computing Science, and that each of them has proved to be especially important in relation to Artificial Intelligence, both in the context of its implementation and in the context of providing it with theoretical foundations. However, in many ways Logic, manifested through Computational Logic or Logic Programming, and Neural Networks

*To appear in the Journal of Applied Logic, Special Edition on Neural-Symbolic Systems. This is a revised and extended treatment of results which to date have appeared only in the workshop paper [HK94] and the conference papers [HS00, HS03a].

are quite complementary. For example, there is a widespread belief that the ability to represent and reason about structured objects and structure-sensitive processes is crucial for rational agents (see, for example, [FP88, New80]), and Computational Logic is well-suited to doing this. On the other hand, rational agents should have additional properties which are not easily found in logic based systems such as, for example, the ability to learn, the ability to adapt to new environments, and the ability to degrade gracefully; these latter properties are typically met by Connectionist Systems or Neural Networks.

For such reasons, there is considerable interest in integrating the Logic based and Neural Network based approaches to Artificial Intelligence with a view to bringing together the advantages to be gained from connectionism and from symbolic AI. However, in attempting to do this, there are considerable obstacles to be overcome. For example, from the computational point of view, most connectionist systems developed so far are propositional in nature. John McCarthy called this a propositional fixation [McC88] in 1988, and not much has changed since then. Although it is known that connectionist systems are Turing-equivalent, we are unaware of any connectionist reasoning system which fully incorporates the power of symbolic computation. Systems like SHRUTI [SA93] or the BUR-calculus [HKW00] allow n-place predicate symbols and a finite set of constants and, thus, are propositional in nature. Systems like CHCL [Höl93] allow a fixed number of first-order clauses, but cannot copy clauses on demand and, thus, the entailment relation is decidable. Connectionist mechanisms for representing terms like holographic reduced representations [Pla91] or recursive auto-associative memories [Pol88] and variations thereof can handle some recursive structures, but as soon as the depth of the represented terms increases, the performance of these methods degrades quickly [McI00]. Furthermore, whilst logic programs have a rather well-developed theory of their semantics, it is not so clear how Neural Networks can be assigned any well-defined meaning which plays an important role comparable with that played by the supported models, the stable model or the well-founded model typically assigned to a logic program to capture its meaning.

It is an important fact that the models just mentioned are fixed points of various operators determined by programs. In particular, the supported models, or Clark completion semantics [Cla78], of a normal logic program P coincide with the fixed points of the immediate consequence operator T_P. Furthermore, the fixed points themselves are frequently found by iterating the corresponding operators.

The previous observation establishes a clear semantical connection between logic programs and neural networks which is the main focus of study in this paper, and it arises because neural networks can be used to compute semantic operators such as T_P. Specifically, in this paper we develop this link between propositional (as well as first-order) logic programs and recursive networks. Our first main observation is that for any given propositional logic program P, one can construct a feedforward connectionist network which can compute the immediate consequence operator T_P. Unfortunately, the methods used in the propositional case do not extend immediately to the first-order case, and our second main observation is that approximation techniques can be used instead to approximate, arbitrarily well, both the semantic operators themselves and also their fixed

points, at least if the feedforward networks are turned into recurrent ones. Our methods here are based on a well-known theorem of Funahashi [Fun89] which shows that every continuous function on the reals can be uniformly approximated by a 3-layer feedforward neural network. However, application of Funahashi's theorem depends on T_P itself being continuous in a precise sense to be defined later. This in turn leads us to study conditions under which T_P meets this criterion, and in doing this we find it convenient to work with quite general semantic operators employing many valued logics. Furthermore, it also raises rather technical questions concerning what are the appropriate approximations to use.

Thus, the overall structure of the paper is as follows. In Section 2, we collect together the basic notions we need concerning logic programs, neural networks, and metric spaces. In Section 3, we establish our claim above that T_P can be computed, for propositional programs P, by feedforward connectionist networks. In Section 4, we take up the issue of extending the results of Section 3 to the first-order case by means of approximation. This involves a fairly detailed study of the (topological) continuity of semantic operators, extending results to be found in [Sed95], before we can ultimately take up the question of applying results such as Funahashi's theorem and discussing measures of approximation appropriate to the study of neural networks. Finally, in Section 5, we present our conclusions and discuss future work. In essence, our techniques and thinking are somewhat in the spirit of dynamical sytems, and provide a link between the areas of logic programming, topology and connectionist systems.

Acknowledgements The authors wish to thank two anonymous referees for comments which helped to improve the presentation of this paper. The last named author wishes to thank the following people and institutions for their support of work related to the results contained in this paper: (i) the members of Professor Steffen Hölldobler's Knowledge Representation and Reasoning Group at TU Dresden, (ii) Deutscher Akademischer Austausch Dienst (DAAD), and (iii) the Boole Centre for Research in Informatics at University College Cork.

2 Basic Notions

In this section, we collect together the basic concepts and notation we need from logic programming, metric spaces and connectionist networks, as can be found, for example, in [Llo88, Wil70, HKP91]. A reader familiar with these notions may skip this section.

2.1 Logic Programs

A (*normal*) *logic program* is a finite set of *clauses* of the form

$$\forall(A \leftarrow L_1 \wedge \cdots \wedge L_n),$$

where $n \in \mathbb{N}$ may differ for each clause, A is an atom in some first-order language $\mathcal{L}$ and $L_1, \ldots, L_n$ are literals, that is, atoms or negated atoms in $\mathcal{L}$. As is customary in logic programming, we will write such a clause in the form

$$A \leftarrow L_1 \wedge \cdots \wedge L_n,$$

in which the universal quantifier is understood. Then A is called the *head* of the clause, each L_i is called a *body literal* of the clause and their conjunction $L_1 \wedge \cdots \wedge L_n$ is called the *body* of the clause. We allow $n = 0$, by an abuse of notation, which indicates that the body is empty; in this case, the clause is called a *unit clause* or a *fact*. We will occasionally use the notation $A \leftarrow \texttt{body}$ for clauses, so that `body` stands for the conjunction of the body literals of the clause. If no negation symbol occurs in a logic program, the program is called a *definite* logic program.

The Herbrand base underlying a given program P will be denoted by B_P, and the set of all Herbrand interpretations by I_P, and we note that the latter can be identified simultaneously with the power set of B_P and with the set $\mathbf{2}^{B_P}$ of all functions mapping B_P into the set $\mathbf{2}$ consisting of two distinct elements. The set $\mathbf{2}$ is usually considered to be the set $\{\mathbf{t}, \mathbf{f}\}$ of truth values. Any interpretation can be extended to literals, clauses and programs in the usual way. A *model* for P is an interpretation which maps P to $\mathbf{t}$. The *immediate consequence operator* (or *single-step operator*) T_P, mapping interpretations to interpretations, is defined as follows. Let I be an interpretation and let A be an atom. Then $T_P(I)(A) = \mathbf{t}$ if and only if there exists a ground instance $A \leftarrow L_1 \wedge \ldots \wedge L_n$ of a clause in P such that $I(L_1 \wedge \ldots \wedge L_n) = \mathbf{t}$. By $\mathrm{ground}(P)$, we will denote the set of all ground instances of clauses in P.

The immediate consequence operator is a convenient tool for capturing the logical meaning, or semantics, of logic programs: an interpretation I is a model for a program P if and only if $T_P(I) \leq I$, that is, if and only if I is a pre-fixed point of T_P, where $\mathbf{2}^{B_P}$ is endowed with the pointwise ordering induced by the unique partial order defined on $\mathbf{2}$ in which $\mathbf{f} < \mathbf{t}$. Fixed points of T_P are called *supported models* for P. They coincide with the models for the so-called *Clark completion* of a program [Cla78] and are considered to be particularly well-suited to capturing the intended meaning of logic programs.

2.2 Metric Spaces and Contraction Mappings

Let X be a non-empty set. A function $d : X \times X \to R$ is called a *metric (on X)*, and the pair (X, d) is called a *metric space*, if the following properties are satisfied.

1. For all $x, y \in X$, we have $d(x, y) \geq 0$ and $d(x, y) = 0$ iff $x = y$.
2. For all $x, y \in X$, we have $d(x, y) = d(y, x)$.
3. For all $x, y, z \in X$, we have $d(x, z) \leq d(x, y) + d(y, z)$.

Let d be a metric defined on a set X. Then a sequence (x_n) in X is said to *converge to* $x \in X$, and x is called the *limit* of (x_n), if, for each $\varepsilon > 0$, there is a natural number

n_0 such that for all $n \geq n_0$ we have $d(x_n, x) < \varepsilon$. Note that the limit of any sequence is unique if it exists. Furthermore, a sequence (x_n) is said to be a *Cauchy sequence* if, for each $\varepsilon > 0$, there is a natural number n_0 such that whenever $m, n \geq n_0$ we have $d(x_m, x_n) < \varepsilon$. It is clear that any sequence which converges is a Cauchy sequence. On the other hand, a metric space (X, d) is called *complete* if every Cauchy sequence in X converges.

Let (X, d) be a metric space. Then a function $f : X \to X$ is called a *contraction mapping* or simply a *contraction* if there exists a real number $\lambda \in [0, 1)$ satisfying $d(f(x), f(y)) \leq \lambda\, d(x, y)$ for all $x, y \in X$. Finally, an element x_0 (of a set X) is called a *fixed point* of a function $f : X \to X$ if, as usual, we have $f(x_0) = x_0$.

One of the main results concerning contraction mappings defined on complete metric spaces is the following well-known theorem.

2.1 Theorem (Banach Contraction Mapping Theorem [Wil70]) Let f be a contraction mapping defined on a complete metric space (X, d). Then f has a unique fixed point $x_0 \in X$. Furthermore, the sequence $x,\ f(x),\ f(f(x)),\ \ldots$ converges to x_0 for any $x \in X$.

If a program P is such that there exists a metric which renders T_P a contraction, then Theorem 2.1 shows that P has a unique supported model. Semantic analysis of logic programs along these general lines was initiated in [Fit94], and has subsequently been studied and generalized by a number of authors. The recent publication [HS03b] contains both a state-of-the-art treatment using this approach and a comprehensive list of references on this topic.

The following definition will be very convenient for our purposes.

2.2 Definition A normal logic program P is called *strongly determined* if there exists a complete metric d on I_P such that T_P is a contraction with respect to d.

It follows from Theorem 2.1 that every strongly determined program has a unique supported model, that is, is *uniquely determined*. Certain well-known classes of programs turn out to contain only strongly determined programs, amongst these are the classes of acyclic and acceptable programs [Bez89, Cav91, AP93, Fit94], which are fundamental in termination analysis under Prolog. More generally, all programs called Φ_ω-accessible in [HS03b] are strongly determined. Indeed, we will take the trouble to define acyclic programs next since we will need this notion in subsequent discussions. To do this, we need first to recall the notion of level mapping, familiar in the context of studies of termination, see [AP93] for example.

A *level mapping* for a program P is a mapping $l : B_P \to \alpha$ for some ordinal α. As usual, we always assume that l has been extended to all literals by setting $l(\neg A) = l(A)$ for each $A \in B_P$. An *ω-level mapping* for P is a level mapping $l : B_P \to \mathbb{N}$.

2.3 Definition A logic program P is called *acyclic* if there exists an ω-level mapping

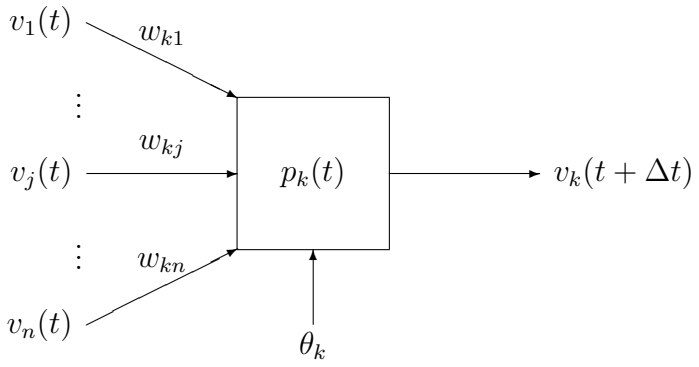

Figure 1: Unit k in a connectionist network.

for P such that for each clause $A \leftarrow L_1 \wedge \cdots \wedge L_n$ in ground(P) we have $l(A) > l(L_i)$ for all $i = 1, \ldots, n$.

2.3 Connectionist Networks

A *connectionist network* is a directed graph. A *unit* k in this graph is characterized, at time t, by its *input vector* $(i_{k1}(t), \ldots, i_{kn_k}(t))$, its *potential* $p_k(t) \in \mathbb{R}$, its *threshold* $\theta_k \in \mathbb{R}$, and its *value* $v_k(t)$. Units are connected via a set of directed and weighted connections. If there is a connection from unit j to unit k, then $w_{kj} \in \mathbb{R}$ denotes the *weight* associated with this connection, and $i_{kj}(t) = w_{kj}v_j(t)$ is the *input* received by k from j at time t. Figure 1 shows a typical unit. The units are updated synchronously. In each update, the potential and value of a unit are computed with respect to an *activation* and an *output function* respectively. All units considered in this paper compute their potential as the weighted sum of their inputs minus their threshold:

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

Having fixed the activation function, we consider three types of units mainly distinguished by their output function. A unit is said to be a *binary threshold unit* if its output function is a threshold function:

$$v_k(t + \Delta t) = \begin{cases} 1 & \text{if } p_k(t) \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

A unit is said to be a *linear unit* if its output function is the identity and its threshold θ is 0. A unit is said to be a *sigmoidal* or *squashing unit* if its output function ϕ is non-decreasing and is such that $\lim_{t\to\infty}(\phi(p_k(t)) = 1$ and $\lim_{t\to-\infty}(\phi(p_k(t)) = 0$. Such functions are called *squashing functions*.

In this paper, we will only consider connectionist networks where the units can be organized in layers. A *layer* is a vector of units. An n*-layer feedforward network* $\mathcal{F}$ consists of the *input* layer, $n-2$ *hidden* layers, and the *output* layer, where $n \geq 2$. Each unit occurring in the i-th layer is connected to each unit occurring in the $(i+1)$-st layer, $1 \leq i < n$. Let r and s be the number of units occurring in the input and output layers, respectively. A connectionist network $\mathcal{F}$ is called a *multilayer feedforward network* if it is an n-layer feedforward network for some n. A multilayer feedforward network $\mathcal{F}$ computes a function $f_{\mathcal{F}} : \mathbb{R}^r \to \mathbb{R}^s$ as follows. The input vector (the argument of $f_{\mathcal{F}}$) is presented to the input layer at time t_0 and propagated through the hidden layers to the output layer. At each time point, all units update their potential and value. At time $t_0 + (n-1)\Delta t$, the output vector (the image under $f_{\mathcal{F}}$ of the input vector) is read off the output layer.

For a 3-layer network with r linear units in the input layer, squashing units in the hidden layer, and a single linear unit in the output layer, the input-output function of the network as described above can thus be obtained as a mapping $f : \mathbb{R}^r \to \mathbb{R}$ with

$$f(x_1, \ldots, x_r) = \sum_j c_j \phi \left(\sum_i w_{ji} x_i - \theta_j \right),$$

where c_j is the weight associated with the connection from the jth unit of the hidden layer to the single unit in the output layer, ϕ is the squashing output function of the units in the hidden layer, w_{ji} is the weight associated with the connection from the ith unit of the input layer to the jth unit of the hidden layer and θ_j is the threshold of the jth unit of the hidden layer.

It is our aim to obtain results on the representation or approximation of consequence operators by input-output functions of 3-layer feedforward networks. Some of our results rest on the following theorem, which is due to Funahashi, see [Fun89].

2.4 Theorem Suppose that $\phi : \mathbb{R} \to \mathbb{R}$ is a non-constant, bounded, monotone increasing and continuous function. Let $K \subseteq \mathbb{R}^n$ be compact, let $f : K \to \mathbb{R}$ be a continuous mapping and let $\varepsilon > 0$. Then there exists a 3-layer feedforward network with squashing function ϕ whose input-output mapping $\bar{f} : K \to \mathbb{R}$ satisfies $\max_{x \in K} d(f(x), \bar{f}(x)) < \varepsilon$, where d is a metric which induces the natural topology[1] on $\mathbb{R}$.

In other words, each continuous function $f : K \to \mathbb{R}$ can be uniformly approximated by input-output functions of 3-layer networks. For our purposes, it will suffice to assume that K is a compact subset of the set of real numbers, so that $n = 1$ in the statement of the theorem.

An n*-layer recurrent network* $\mathcal{N}$ consists of an n-layer feedforward network such that the number of units in the input and output layer are identical. Furthermore, each unit in the k-th position of the output layer is connected with weight 1 to the unit in the k-th position of the input layer, where $1 \leq k \leq N$ and N is the number of units in

[1] For example, $d(x, y) = |x - y|$.

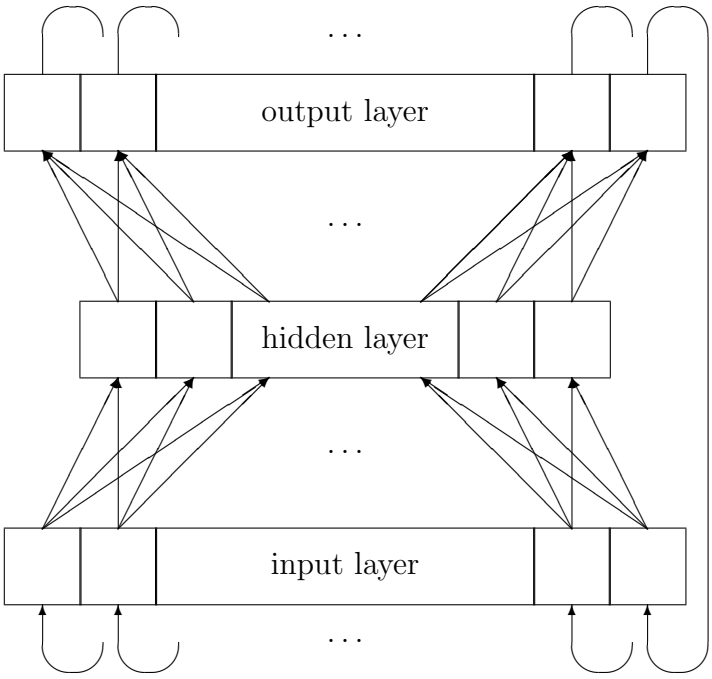

Figure 2: Sketch of a 3-layered recurrent network.

the output (or input) layer. Figure 2 shows a 3-layer recurrent network. The subnetwork consisting of the three layers and the connections between the input and the hidden as well as between the hidden and the output layer is a 3-layer feedforward network called the *kernel* of $\mathcal{N}$.

3 Propositional Logic Programs

> *In this section, we consider the propositional case following [HK94] and show that for each logic program P we can construct a 3-layer feedforward network of binary threshold units computing T_P. Turning such a network into a recurrent one allows one to compute the unique fixed point of T_P provided that P is strongly determined.*

The main question addressed in this section is: can we specify a connectionist network of binary threshold units for a propositional logic program P such that it computes T_P and, if it exists, the least fixed point of T_P? It is well-known that 3-layer feedforward connectionist networks with sigmoidal hidden layer are universal approximators [Fun89, HSW89]. Hence, we expect that recurrent networks with a 3-layer feedforward kernel will do, where the kernel computes T_P and, by the recurrent connections, T_P is iterated.

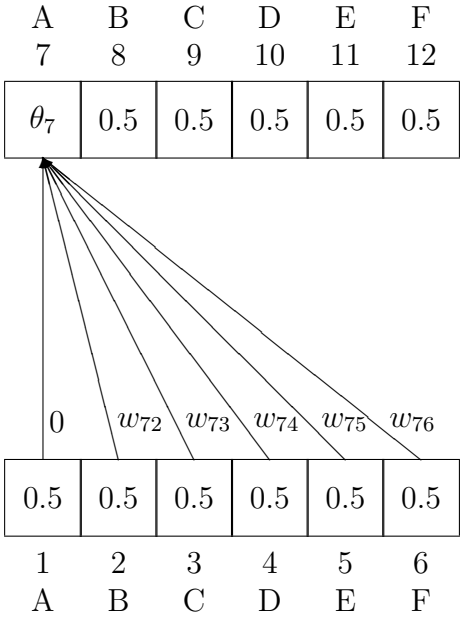

Figure 3: A 2-layer feedforward network of binary threshold units for P_2. The numbers occurring within the units are thresholds. Connections which are not shown have weight 0.

The question addressed in the following subsection is whether or not even simpler networks, viz. recurrent networks with a 2-layer feedforward kernel of binary threshold units will do. Such networks are called *perceptrons* [Ros62]. It is well-known that their computing capabilities are limited to computing solutions for linearly separable problems [MP72].

3.1 Hidden Layers are Needed

Usually, the need for a hidden layer is shown by demonstrating that the exclusive-or cannot be modelled by a feedforward network without hidden layers (see [MP72], for example). A straightforward program to compute the exclusive-or of two propositional atoms A and B such as the program

$$P_1 = \{C \leftarrow A \wedge \neg B,\ C \leftarrow \neg A \wedge B\}$$

is not definite and from this we can only conclude that 2-layer feedforward networks cannot compute T_P for normal P. An even stronger result is the following.

3.1 Proposition 2-layer connectionist networks of binary threshold units cannot compute T_P for definite P.

Proof: Consider the following program

$$P_2 = \{A \leftarrow B,\ A \leftarrow C \wedge D,\ A \leftarrow E \wedge F\}.$$

Let $\mathcal{F}$ be the 2-layer feedforward network of binary threshold units shown in Figure 3 and assume that the weights in $\mathcal{F}$ are selected in such a way that it computes T_{P_2}. Let $w_{ij} = 0$ and $\theta_i = 0.5$ if $i \in [8, 12]$, so that no unit encoding the atoms B to F in the output layer will ever become active and this property is, moreover, independent of the activation pattern of the input layer. Thus, as far as these units are concerned, the network behaves correctly as no atom B to F is evaluated to $\mathbf{t}$ by $T_{P_2}(I)$ for any interpretation I. For unit 7 to behave correctly, we have to find a threshold θ_7 and weights w_{7j}, $1 \leq j \leq 6$, such that

$$T_{P_2}(I)(A) = \mathbf{t} \text{ iff } w_{71}v_1 + w_{72}v_2 + w_{73}v_3 + w_{74}v_4 + w_{75}v_5 + w_{76}v_6 - \theta_7 \geq 0, \qquad (1)$$

where $I = (v_1, \ldots, v_6)$ is the current interpretation, that is, the activation or output pattern of the input layer. Obviously, the output of unit 1 should not influence the potential of unit 7 and hence $w_{71} = 0$. Thus, (1) reduces to

$$T_{P_2}(I)(A) = \mathbf{t} \text{ iff } w_{72}v_2 + w_{73}v_3 + w_{74}v_4 + w_{75}v_5 + w_{76}v_6 - \theta_7 \geq 0. \qquad (2)$$

As the conjunction in the conditions of clauses is commutative, (2) can be transformed to

$$T_{P_2}(I)(A) = \mathbf{t} \text{ iff } w_{72}v_2 + w_{74}v_3 + w_{73}v_4 + w_{75}v_5 + w_{76}v_6 - \theta_7 \geq 0$$

and

$$T_{P_2}(I)(A) = \mathbf{t} \text{ iff } w_{72}v_2 + w_{73}v_3 + w_{74}v_4 + w_{76}v_5 + w_{75}v_6 - \theta_7 \geq 0.$$

Hence, with $w_1 = \frac{1}{2}(w_{73} + w_{74})$ and $w_2 = \frac{1}{2}(w_{75} + w_{76})$ equation (2) becomes

$$T_{P_2}(I)(A) = \mathbf{t} \text{ iff } w_{72}v_2 + w_1(v_3 + v_4) + w_2(v_5 + v_6) - \theta_7 \geq 0. \qquad (3)$$

As the disjunction between clauses is commutative, using an argument similar to that used before we find $w = \frac{1}{3}(w_{72} + w_1 + w_2)$ such that (3) becomes

$$T_{P_2}(I)(A) = \mathbf{t} \text{ iff } w(v_2 + v_3 + v_4 + v_5 + v_6) - \theta_7 \geq 0. \qquad (4)$$

Thus, with $x = \sum_{j=2}^{6} v_j$ we obtain the polynomial $wx - \theta_7$. Now, for $\mathcal{F}$ to compute T_{P_2} the following must hold.

$$\begin{array}{lll} wx - \theta_7 < 0 & \text{if} & x = 0 \quad (v_2 = \ldots = v_6 = 0). \\ wx - \theta_7 \geq 0 & \text{if} & x = 1 \quad (v_2 = 1,\ v_3 = \ldots = v_6 = 0). \\ wx - \theta_7 < 0 & \text{if} & x = 2 \quad (v_2 = v_4 = v_6 = 0,\ v_3 = v_5 = 1). \end{array}$$

However, the first derivative of the polynomial $wx - \theta_7$ cannot change its sign and, consequently, there cannot be weights and thresholds such that the 2-layer feedforward network computes T_{P_2}. ■

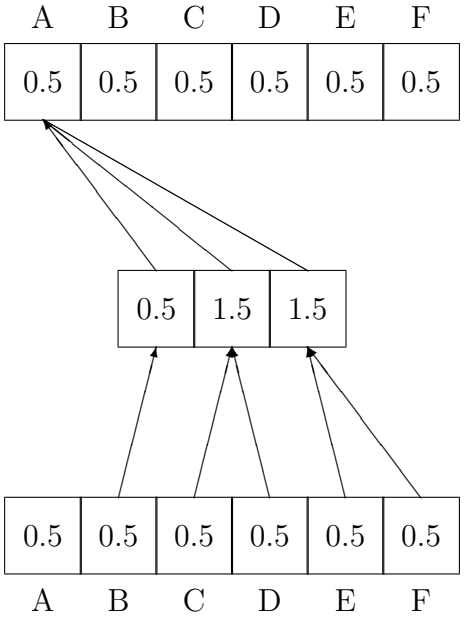

Figure 4: A 3-layer feedforward network of binary threshold units computing T_{P_2}. Only connections with non-zero weights are shown, and these connections have weight 1. The numbers occurring within units denote thresholds.

This result shows the need for hidden layers and it is easy to verify that the 3-layer feedforward network of binary threshold units shown in Figure 4 computes T_{P_2} for the program P_2.

One should observe that each rule R in P_2 is mapped from the input to the output layer through exactly one unit in the hidden layer. The potential of this unit is greater than 0 at $t_0 + \Delta t$ and, thus, the unit becomes active at $t_0 + \Delta t$ if and only if each unit in the input layer representing a condition of R is active at t_0, that is, if and only if each condition of R is assigned $\mathbf{t}$. The potential of the output unit representing A is greater than 0 at $t_0 + 2\Delta t$ and, thus, the unit becomes active at $t_0 + 2\Delta t$ if and only if at least one hidden unit that is connected to A is active at $t_0 + \Delta t$.

Consequently, the number of units in the hidden layer as well as the number of connections between the hidden and the output layer with non-zero weight is equal to the number of clauses in P. Furthermore, the number of connections between the input and the hidden layer with non-zero weight is equal to the number of literals occurring in the conditions of program clauses, and the number of units in the input and output layers is equal to the number of propositional variables occurring in the program. Hence, the size of the network is bounded by the size of the program, and the operator T_P is computed in constant time, viz. in 2 steps.

These construction principles are extended to normal programs in the following subsection.

3.2 Relating Propositional Programs to Networks

3.2 Theorem For each program P, there exists a 3-layer feedforward network computing T_P.

Proof: Let m and n be the number of propositional variables and the number of clauses occurring in P, respectively. Without loss of generality, we may assume that the variables are ordered. The network associated with P can now be constructed by the following *translation algorithm*:

1. The input and output layer is a vector of binary threshold units of length m, where the i-th unit in the input and output layer represents the i-th variable, $1 \leq i \leq m$. The threshold of each unit occurring in the input or output layer is set to 0.5.
2. For each clause of the form $A \leftarrow L_1 \wedge \ldots \wedge L_k$, $k \geq 0$, occurring in P, do the following.
   - 2.1 Add a binary threshold unit c to the hidden layer.
   - 2.2 Connect c to the unit representing A in the output layer with weight 1.
   - 2.3 For each literal L_j, $1 \leq j \leq k$, connect the unit representing L_j in the input layer to c and, if L_j is an atom, then set the weight to 1; otherwise set the weight to -1.
   - 2.4 Set the threshold θ_c of c to $l - 0.5$, where l is the number of positive literals occurring in $L_1 \wedge \ldots \wedge L_k$.

Each interpretation I for P can be represented by a binary vector $(v_1, \ldots, v_m)$. Such an interpretation is given as input to the network by externally activating corresponding units of the input layer at time t_0. It remains to show that $T_P(I)(A) = \mathbf{t}$ if and only if the unit representing A in the output layer becomes active at time $t_0 + 2\Delta t$.

If $T_P(I)(A) = \mathbf{t}$, then there is a clause $A \leftarrow L_1 \wedge \ldots \wedge L_k$ in P such that for all $1 \leq j \leq k$ we have $I(L_j) = \mathbf{t}$. Let c be the unit in the hidden layer associated with this clause according to item 2.1 of the construction. From 2.3 and 2.4 we conclude that c becomes active at time $t_0 + \Delta t$. Consequently, 2.2 and the fact that units occurring in the output layer have a threshold of 0.5 (see 1.) ensure that the unit representing A in the output layer becomes active at time $t_0 + 2\Delta t$.

Conversely, suppose that the unit representing the atom A in the output layer becomes active at time $t_0 + 2\Delta t$. From the construction of the network, we find a unit c in the hidden layer which must have become active at time $t_0 + \Delta t$. This unit is associated with a clause $A \leftarrow L_1 \wedge \ldots \wedge L_k$. If $k = 0$, that is, if the body of the clause is empty, then, according to item 2.4, c has a threshold of -0.5. Furthermore, according to item 2.3, c does not receive any input, that is, $p_c = 0 + 0.5$ and consequently c will always be active. Otherwise, if $k \geq 1$, then c becomes active only if each unit in the input layer representing a positive literal and no unit representing a negative literal in the body of

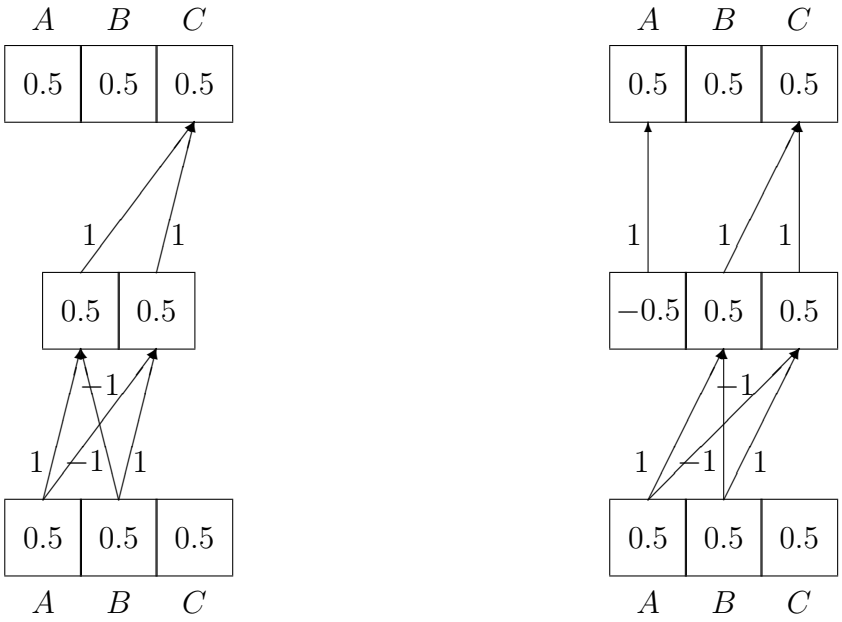

Figure 5: Two 3-layer feedforward networks of binary threshold units computing T_{P_1} and T_{P_3}, respectively. Only connections with non-zero weight are shown. The number occurring within units denote thresholds.

the clause is active at time t_0 (see items 2.3 and 2.4). Hence, we have found a clause $A \leftarrow L_1 \wedge \ldots \wedge L_k$ such that for all $1 \leq j \leq k$ we have $I(L_j) = \mathbf{t}$ and consequently $T_P(I)(A) = \mathbf{t}$. ■

As an example, reconsider

$$P_1 = \{C \leftarrow A \wedge \neg B,\ C \leftarrow \neg A \wedge B\}$$

and extend it to

$$P_3 = \{A,\ C \leftarrow A \wedge \neg B,\ C \leftarrow \neg A \wedge B\}.$$

Their corresponding connectionist networks are shown in Figure 5. One should observe that P_3 exemplifies the representation of unit clauses in 3-layer feedforward networks.[2]

As already mentioned at the end of Subsection 3.1, the number of units and the number of connections in a network $\mathcal{F}$ corresponding to a program P are bounded by $O(m+n)$ and $O(m \times n)$, respectively, where n is the number of clauses and m is the number of propositional variables occurring in P. Furthermore, $T_P(I)$ is computed in 2 steps. As the sequential time to compute $T_P(I)$ is bounded by $O(n \times m)$ (assuming that no literal

[2]We can save the unit in the hidden layer corresponding to the unit clause, if we change the threshold of the unit representing A in the output layer to -0.5.

occurs more than once in the conditions of a clause), the parallel computational model is optimal.[3]

We can now apply the Banach contraction mapping theorem, Theorem 2.1, to obtain the following result.

3.3 Corollary Let P be a strongly determined (propositional) program. Then there exists a 3-layer recurrent network such that each computation starting with an arbitrary initial input converges and yields the unique fixed point of T_P, that is, the unique supported model for P.

Let us mention in passing that a kind of converse of Corollary 3.3 also holds, as follows. Let P be a (propositional) program such that the corresponding network has the property that each computation starting with an arbitrary initial input converges, and in all cases converges to the same state. Then this means that iteration of the T_P-operator exhibits the same behaviour, that is, for each initial interpretation it yields one and the same constant value after a finite number of iterations. By [HS01a, Theorem 2], this suffices to guarantee the existence of a complete metric which renders T_P a contraction. A direct proof of this observation is given in [HK94].

Returning to the programs P_1 and P_3 again, we observe that both programs are strongly determined[4]. Hence, Figure 5 shows the kernels of corresponding recurrent networks which compute the least fixed point of T_{P_1} (the interpretation represented by the vector $(0,0,0)$) and of T_{P_3} (the interpretation represented by the vector $(1,0,1)$).

The time needed by the network to settle down into the unique stable state is equal to the time needed by a sequential machine to compute the least fixed point of T_P in the worst case. As an example, consider the definite program

$$P_4 = \{A_1\} \cup \{A_{i+1} \leftarrow A_i \mid 1 \leq i < n\}.$$

The least fixed point of T_P is the interpretation which evaluates each A_i, $1 \leq i \leq n$, to $\mathbf{t}$. Using the technique described in [DG84] and [Scu90], it can be computed in $O(n)$ steps.[5] Obviously, the parallel computational model needs as many steps. More generally, let P be a definite program containing n clauses. The time needed by the network to settle down into the unique stable state is $3n$ in the worst case and, thus, the time is linear with respect to the number of clauses occurring in the program. This comes as no surprise as it follows from [JL77] that satisfiability of propositional Horn formulae is P-complete and, thus, is unlikely to be in the class NC (see for example [KR90]). On the other hand, consider the program

$$P_5 = \{A_i \mid 1 \leq i \leq n \text{ and } i \text{ even}\} \cup \{A_{i+1} \leftarrow A_i \mid 1 \leq i \leq n \text{ and } i \text{ even}\}.$$

[3] A parallel computational model requiring $p(n)$ processors and $t(n)$ time to solve a problem of size n is *optimal* if $p(n) \times t(n) = O(T(n))$, where $T(n)$ is the sequential time to solve this problem (see for example [KR90]).

[4] They are even acceptable, as can be seen by mapping C to 2, and A as well as B to 1 and considering the model $I(A) = I(C) = \mathbf{t}$ and $I(B) = \mathbf{f}$.

[5] To be precise, the algorithm described in [DG84] needs $O(n)$ time, where n denotes the total number of occurrences of propositional variables in the formula.

The least model mapping each atom to $\mathbf{t}$ is computed in five steps by the recurrent network corresponding to P_5.

3.3 Extensions

In this subsection, various extensions of the basic model developed in Subsection 3.2 are briefly discussed. In particular, we focus on learning, rule extraction and propositional modal logics.

Learning The networks corresponding to logic programs and constructed by the translation algorithm presented in the proof of Theorem 3.2 cannot be trained by the usual learning methods applied to connectionist systems. It was observed in [dGZdC97] (see also [dGZ99, dGBG02]) that results similar to Theorem 3.2 and Corollary 3.3 can be achieved if the binary threshold units occurring in the hidden layer of the feedforward kernels are replaced by sigmoidal units. We omit the technical details here and refer to the above-mentioned literature. Such a move renders the kernels accessible to the backpropagation algorithm, a standard technique for training feedforward networks [RHW86].

Rule Extraction After training a feedforward network with sigmoidal units in the hidden layer, the knowledge encoded in the network is mostly inaccessible to a human without postprocessing. Numerous techniques have been proposed to extract rules from trained feedforward networks (see for example [ADT95] and [dGBG01]). We can now envision a cycle in which a given (preliminary) logic program is translated into a feedforward network, this network is trained by examples using backpropagation, and a new (refined) logic program is extracted from the network after training (see [TS94]). The reference [dGBG02] contains several examples of such cyclic knowledge processing.

Propositional Modal Logics The approach discussed so far has been extended to (*propositional*) *modal programs*, where literals occurring in a clause may be prefixed by the modalities $\Box$ and $\Diamond$, clauses are labelled by the world in which they hold, and a finite set of relations between worlds is given [dGLG02]. It was shown that Theorem 3.2 can be extended to such modal programs in that for each such program there exists a 3-layer connectionist network computing the modal fixed point operator of the given program. The main idea is to construct for each world a 3-layer feedforward network using a variation of the translation algorithm specified in the proof of Theorem 3.2 and then to connect the worlds with respect to the given set of relations between worlds and the usual Kripke semantics of the modalities. It is an interesting open problem to show how to model the temporal aspects of reasoning with respect to modal programs within a connectionist setting other than by just copying the complete network from one point in time to the next one.

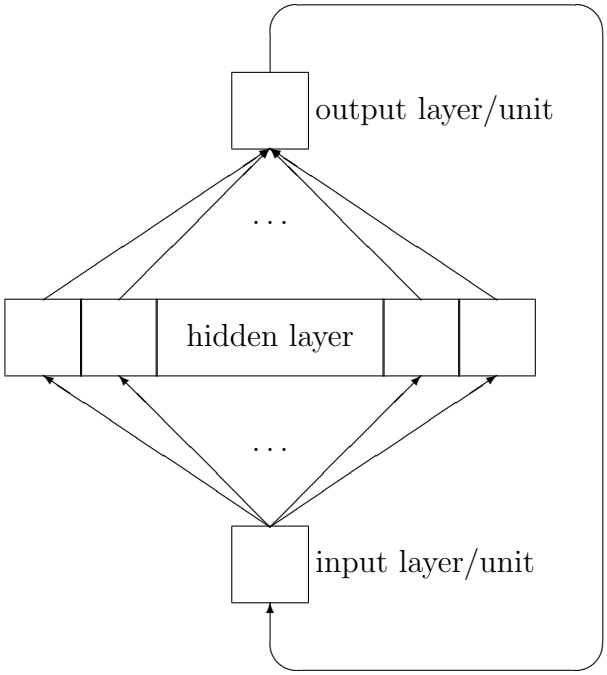

Figure 6: Sketch of a recurrent network for a first-order logic program.

4 First-Order Logic Programs

In this section, we extend the approach presented in Section 3 to the first-order case. In particular, we consider conditions under which semantic operators for first-order logic programs as well as their fixed points can be approximated by connectionist networks.

In the first-order case, (Herbrand) interpretations usually consist of countably many ground atoms. Hence, the simple solution for the propositional case, where each ground atom is represented by a binary threshold unit in the input and the output layer, is no longer feasible. To extend the representational capability of the networks used, binary threshold units are replaced by sigmoidal ones. The values generated by sigmoidal units are real numbers, and we will use real numbers to represent interpretations. In Figure 6, the recurrent nets considered in this section are sketched. This section extends results published in [HKS99] and therefore we review the previous work in the following subsection.

4.1 Previous Work

The reference [HKS99] was concerned with the following problem. Suppose we are given a first-order logic program P together with a continuous consequence operator $T_P : 2^{B_P} \to 2^{B_P}$, where B_P is the Herbrand base of P. We want to know whether or not there exists a class of logic programs such that for each program in this class we can find an invertible mapping $\iota : 2^{B_P} \to \mathbb{R}$ and a function $f_P : \mathbb{R} \to \mathbb{R}$ satisfying the following conditions:

1. $T_P(I) = I'$ implies $f_P(\iota(I)) = \iota(I')$ and $f_P(r) = r'$ implies $T_P(\iota^{-1}(r)) = \iota^{-1}(r')$,
2. T_P is a contraction on 2^{B_P} iff f_P is a contraction on $\mathbb{R}$, and
3. f_P is continuous on $\mathbb{R}$.

The first condition ensures that f_P is a sound and complete encoding of T_P. The second condition ensures that the contraction property, and thus fixed points, are preserved. The third condition ensures that we can apply Theorem 2.4 which then yields a 3-layer feed-forward network with sigmoidal units in the hidden layer approximating f_P arbitrarily well. Moreover, the corresponding recurrent network approximates the least fixed point of T_P arbitrarily well also.

It was shown in [HKS99] that this problem can be solved for the class of acyclic logic programs with injective level mapping. In the following, we will lift some of these observations to a much more general level. In particular, we will show that acyclic programs with injective level mappings represent only a small fraction of the programs for which f_P can be approximated satisfactorily. We will also abstract from the single-step operator and generalize the approach to more general types of semantic operators.

Throughout the rest of the paper, we will make substantial use of elementary notions and results from topology, and our standard background reference to this subject is [Wil70]. Indeed, the results presented subsequently are based on the observation that acyclicity with respect to an injective level mapping is a sufficient, but not necessary, condition for continuity of the single-step operator with respect to a topology which is homeomorphic to the Cantor topology on the real line, namely, the *query* or *atomic topology* studied in [BS89, Sed95] and elsewhere in logic programming. We will therefore start by studying the basic topological facts relevant to our task before turning to the applications we ultimately want to make of these ideas and methods.

4.2 Continuity of Semantic Operators

From now on, we will impose the standing condition on the language $\mathcal{L}$ that it contains at least one constant symbol and at least one function symbol with arity greater than 0. If this is not done, then $\text{ground}(P)$ may be a finite set of ground instances of clauses, and can be treated essentially as a propositional program, for which appropriate methods were laid out in Section 3.

In logic programming semantics, it has turned out to be both useful and convenient to use many-valued logics. Our investigations will therefore begin by studying suitable topologies on spaces of many-valued interpretations. We assume we have given a finite set $\mathcal{T} = \{t_1, \ldots, t_n\}$ of truth values containing at least the two distinguished values t_1 and t_n, which are interpreted as being the truth values for "false", and "true", respectively. We also assume that we have truth tables for the usual connectives $\vee$, $\wedge$, $\leftarrow$, and $\neg$. Given a logic program P, we denote the set of all (Herbrand) *interpretations* or *valuations* in this logic by $I_{P,n}$; thus $I_{P,n}$ is the set $\mathcal{T}^{B_P}$ of all functions $I : B_P \to \mathcal{T}$. If n is clear from the context, we will use the notation I_P instead of $I_{P,n}$ and we note that this usage is consistent with the one given above for $n = 2$. As usual, any interpretation I can be extended, using the truth tables, to give a truth value in $\mathcal{T}$ to any variable-free formula in $\mathcal{L}$.

4.1 Definition Given any logic program P, the *generalized atomic topology* $\mathcal{Q}$ on $I_P = I_{P,n}$ is defined to be the product topology on $\mathcal{T}^{B_P}$, where $\mathcal{T} = \{t_1, \ldots, t_n\}$ is endowed with the discrete topology.

We note that this topology can be defined analogously for the non-Herbrand case. For $n = 2$, the generalized atomic topology $\mathcal{Q}$ specializes to the query topology of [BS89] (in the Herbrand case) and to the atomic topology Q of [Sed95] (in the non-Herbrand case). The following results follow immediately since $\mathcal{Q}$ is a product of the discrete topology on a finite set, and hence is a topology of pointwise convergence.

4.2 Proposition For $A \in B_P$ and t_i a truth value, let $\mathcal{G}(A, t_i) = \{I \in I_{P,n} \mid I(A) = t_i\}$. Then the following hold.

(a) $\mathcal{Q}$ is the topology generated by the subbase $\mathcal{G} = \{\mathcal{G}(A, t_i) \mid A \in B_P, i \in \{1, \ldots, n\}\}$.

(b) A net (I_λ) in I_P converges in $\mathcal{Q}$ to I in I_P if and only if for every $A \in B_P$ there exists some λ_0 such that $I_\lambda(A)$ is constant and equal to $I(A)$ for all $\lambda \geq \lambda_0$.

(c) $\mathcal{Q}$ is a second countable totally disconnected compact Hausdorff topology which is dense in itself. Hence, $\mathcal{Q}$ is metrizable and homeomorphic to the Cantor topology on the unit interval in the real line.

We note that the second countability of $\mathcal{Q}$ rests on the fact that B_P is countable, so that this property does not in general carry over to the non-Herbrand case.

The study of topologies such as $\mathcal{Q}$ comes from our desire to be able to control the iterative behaviour of semantic operators. Topologies which are closely related to order structures, as common in denotational semantics [AJ94], are of limited applicability since non-monotonic operators frequently arise naturally in the logic programming context. See also [Hit01] for a study of these issues.

We proceed next with studying a rather general notion of semantic operator, akin to Fitting's approach in [Fit02], which generalizes standard notions occurring in the literature.

4.3 Definition An operator T on I_P is called a *consequence operator* for P if for every $I \in I_P$ the following condition holds: for every ground clause $A \leftarrow \texttt{body}$ in P, where $T(I)(A) = t_i$, say, and $I(\texttt{body}) = t_j$, say, we have that the truth table for $t_i \leftarrow t_j$ yields the truth value t_n, that is, "true".

It turns out that this notion of consequence operator relates nicely to $\mathcal{Q}$, yielding the following result which was reported in [Hit01, HS01b]. If T is a consequence operator for P and if for any $I \in I_P$ we have that the sequence of iterates $T^m(I)$ converges in $\mathcal{Q}$ to some $M \in I_P$, then M is a model, in a natural sense, for P. Furthermore, continuity of T yields the desirable property that M is a fixed point of T.

Intuitively, consequence operators should propagate "truth" along the implication symbols occurring in the program. From this point of view, we would like the outcome of the truth value of such a propagation to be dependent only on the relevant clause bodies. The next definition captures this intuition.

4.4 Definition Let $A \in B_P$ and denote by $\mathcal{B}_A$ the set of all body atoms of clauses with head A that occur in ground(P). A consequence operator T is called (P-)*local* if for every $A \in B_P$ and any two interpretations $I, K \in I_P$ which agree on all atoms in $\mathcal{B}_A$, we have $T(I)(A) = T(K)(A)$.

It is our desire to study continuity in $\mathcal{Q}$ of local consequence operators. Since $\mathcal{Q}$ is a product topology, it is reasonable to expect that finiteness conditions will be involved, and indeed conditions which ensure finiteness in the sense of Definition 4.5 below, due to [Sed95], have made their appearance in this context.

4.5 Definition Let C be a clause in P and let $A \in B_P$ be such that A coincides with the head of C. The clause C is said to be of *finite type relative to* A if C has only finitely many different ground instances with head A. The program P will be said to be of *finite type relative to* A if each clause in P is of finite type relative to A, that is, if the set of all clauses in ground(P) with head A is finite. Finally, P will be said to be of *finite type* if P is of finite type relative to A for every $A \in B_P$.

A *local variable* is a variable which appears in a clause body but not in the corresponding head. Local variables appear naturally in practical logic programs, but their occurrence is awkward from the point of view of denotational semantics, especially if they occur in negated body literals since this leads to the so-called floundering problem, see [Llo88].

It is easy to see that, in the context of Herbrand-interpretations, and if function symbols are present, then the absence of local variables is equivalent to a program being of finite type.

4.6 Proposition Let P be a logic program of finite type and let T be a local consequence operator for P. Then T is continuous in $\mathcal{Q}$.

Proof: Let $I \in I_P$ be an interpretation and let $G_2 = \mathcal{G}(A, t_i)$ be a subbasic neighbourhood of $T(I)$ in $\mathcal{Q}$, and note that G_2 is the set of all $K \in I_P$ such that $K(A) = t_i$.

We need to find a neighbourhood G_1 of I such that $T(G_1) \subseteq G_2$. Since P is of finite type, the set $\mathcal{B}_A$ is finite. Hence, the set $G_1 = \bigcap_{B \in \mathcal{B}_A} \mathcal{G}(B, I(B))$ is a finite intersection of open sets and is therefore open. Since each $K \in G_1$ agrees with I on $\mathcal{B}_A$, we obtain $T(K)(A) = T(I)(A) = t_i$ for each $K \in G_1$ by locality of T. Hence, $T(G_1) \subseteq G_2$. ■

Now, if P is not of finite type, but we can ensure by some other property of P that the possibly infinite intersection $\bigcap_{B \in \mathcal{B}_A} \mathcal{G}(B, I(B))$ is open, then the above proof will carry over to programs which are not of finite type. Alternatively, we would like to be able to disregard the infinite intersection entirely under conditions which ensure that we have to consider finite intersections only, as in the case of a program of finite type. The following definition is, therefore, quite a natural one to make.

4.7 Definition Let P be a logic program and let T be a consequence operator on I_P. We say that T is (P-)*locally finite for* $A \in B_P$ *and* $I \in I_P$ if there exists a finite subset $S = S(A, I) \subseteq \mathcal{B}_A$ such that we have $T(J)(A) = T(I)(A)$ for all $J \in I_P$ which agree with I on S. We say that T is (P-)*locally finite* if it is locally finite for all $A \in B_P$ and all $I \in I_P$.

It is easy to see that a locally finite consequence operator is local. Conversely, a local consequence operator for a program of finite type is locally finite. This follows from the observation that, for a program of finite type, the sets $\mathcal{B}_A$, for any $A \in B_P$, are finite. But a much stronger result holds.

4.8 Theorem A local consequence operator is locally finite if and only if it is continuous in $\mathcal{Q}$.

Proof: Let T be a locally finite consequence operator, let $I \in I_P$, let $A \in B_P$, and let $G_2 = \mathcal{G}(A, T(I)(A))$ be a subbasic neighbourhood of $T(I)$ in $\mathcal{Q}$. Since T is locally finite, there is a finite set $S \subseteq \mathcal{B}_A$ such that $T(J)(A) = T(I)(A)$ for all $J \in \bigcap_{B \in S} \mathcal{G}(B, I(B))$. By finiteness of S, the set $\bigcap_{B \in S} \mathcal{G}(B, I(B))$ is open, and this suffices for continuity of T.

For the converse, assume that T is continuous in $\mathcal{Q}$ and let $A \in B_P$ and $I \in I_P$ be chosen arbitrarily. Then $G_2 = \mathcal{G}(A, T(I)(A))$ is a subbasic open set, so that, by continuity of T, there exists a basic open set $G_1 = \mathcal{G}(B_1, I(B_1)) \cap \cdots \cap \mathcal{G}(B_k, I(B_k))$ with $T(G_1) \subseteq G_2$. In other words, we have $T(J)(A) = T(I)(A)$ for each $J \in \bigcap_{B \in S'} \mathcal{G}(B, I(B))$, where $S' = \{B_1, \ldots, B_k\}$ is a finite set. Since T is local, the value of $T(J)(A)$ depends only on the values $J(A)$ of atoms $A \in \mathcal{B}_A$. So, if we set $S = S' \cap \mathcal{B}_A$, then $T(J)(A) = T(I)(A)$ for all $J \in \bigcap_{B \in S} \mathcal{G}(B, I(B))$ which is to say that T is locally finite for A and I. Since A and I were chosen arbitrarily, we obtain that T is locally finite. ■

The following corollary was communicated to us by Howard A. Blair in the two-valued case.

4.9 Corollary Let P be a program, let T be a local consequence operator and let l be

an injective ω-level mapping for P with the following property: for each $A \in B_P$ there exists an $n_A \in \mathbb{N}$ such that $l(B) < n_A$ for all $B \in \mathcal{B}_A$. Then T is continuous in $\mathcal{Q}$.

Proof: It follows easily from the given conditions that $\mathcal{B}_A$ is finite for all $A \in B_P$, which implies that T is locally finite. ■

We next take a short detour from our discussion of continuity to study the weaker notion of measurability [Bar66] for consequence operators. For a collection M of subsets of a set X, we denote by $\sigma(M)$ the smallest σ-algebra containing M, called the σ-algebra *generated by* M. Recall that a function $f : X \to X$ is measurable with respect to $\sigma(M)$ if and only if $f^{-1}(A) \in \sigma(M)$ for each $A \in M$. If β is the subbase of a topology τ and β is countable, then $\sigma(\beta) = \sigma(\tau)$. It turns out that local consequence operators are always measurable with respect to the σ-algebra generated by a generalized atomic topology.

4.10 Theorem Local consequence operators are measurable with respect to $\sigma(\mathcal{G}) = \sigma(\mathcal{Q})$.

Proof: Let T be a local consequence operator. We need to show that, for each subbasic set $\mathcal{G}(A, t_i)$, we have $T^{-1}(\mathcal{G}(A, t_i)) \in \sigma(\mathcal{G})$.

Let $A \in B_P$ and let $t \in \mathcal{T}$ both be chosen arbitrarily. Let F be the set of all functions from $\mathcal{B}_A$ to $\mathcal{T}$, and note that F is countable since $\mathcal{B}_A$ is countable and $\mathcal{T}$ is finite. Let F' be the subset of F which contains all functions f with the following property: whenever an interpretation I agrees with f on $\mathcal{B}_A$, then $T(I)(A) = t$. Then, $\bigcap_{B \in \mathcal{B}_A} \mathcal{G}(B, f(B)) \in T^{-1}(\mathcal{G}(A, t))$ for each $f \in F'$.

We obtain by locality of T that, whenever I is an interpretation for which $T(I)(A) = t$, there exists a function $f_I \in F'$ such that f_I and I agree on $\mathcal{B}_A$, and this yields $T^{-1}(\mathcal{G}(A, t)) = \bigcup_{f_I \in F'} \bigcap_{B \in \mathcal{B}_A} \mathcal{G}(B, I(B))$. Since F' and $\mathcal{B}_A$ are countable, the set on the right hand side of this last equality is measurable, as required. ■

We turn now to the study of the continuity of a particular operator introduced by Fitting [Fit02] to logic programming semantics. To this end, we associate a set P^* with each logic program P by the following construction. Let $A \in B_P$. If A occurs as the head of some unit clause $A \leftarrow$ in ground(P), then replace it by the clause $A \leftarrow t_n$, where by a slight abuse of notation we interpret t_n to be an additional atom which we adjoin to the language $\mathcal{L}$ and always evaluate to $t_n \in \mathcal{T}$, that is, it evaluates to "true". If A does not occur in the head of any clause in ground(P), then add the clause $A \leftarrow t_0$, where t_0 is interpreted as an additional atom which again we adjoin to $\mathcal{L}$ and always evaluate to $t_0 \in \mathcal{T}$, that is, it evaluates to "false". The resulting (ground) program, which results from ground(P) by the changes just given with respect to every $A \in B_P$, will be denoted by P'. Now let P^* be the set of all *pseudo clauses* determined by P', that is, the set of all formulae of the form $A \leftarrow C_1 \vee C_2 \vee \ldots$, where the C_i are exactly the bodies of the clauses in P' with head A. We call A the *head* and $B_A = C_1 \vee C_2 \vee \ldots$ the *body* of such a pseudo clause, and we note that each $A \in B_P$ occurs in the head of exactly

one pseudo clause in P^*. Bodies of pseudo clauses are possibly infinite disjunctions, but this will not pose any particular difficulty with respect to the logics which we are going to discuss. We note that a program P is of finite type if and only if all bodies of all pseudo clauses in P^* are finite.

Now, if we are given (suitable) truth tables for negation, conjunction and disjunction, we are able to evaluate the truth values of bodies of pseudo clauses relative to given interpretations.

4.11 Definition Let P be a logic program. Define the mapping $F_P : I_{P,n} \to I_{P,n}$ relative to a given (suitable) logic with n truth values by $F_P(I) = J$, where J assigns to each $A \in B_P$ the truth value $I(B_A)$.

We call operators which satisfy Definition 4.11 *Fitting operators*. If we impose the mild assumption that $t_j \leftarrow t_j$ evaluates to "true" for every j with respect to the underlying logic, then we easily obtain that every Fitting operator is a local consequence operator. This will always be the case in what follows in this paper.

The virtue of Definition 4.11, due to Fitting [Fit02], lies in the fact that several operators known from the theory of logic programming can be derived from it in a very concise way, and we refer to [Fit02, DMT00] for a discussion of these matters, see also [HS01b]. We will now investigate some of these operators in the light of Theorem 4.8. In the following, we will denote the "true" truth value by $\mathbf{t}$ and the "false" truth value by $\mathbf{f}$.

If the chosen logic is classical two-valued logic, then the corresponding Fitting operator is the *single-step* or *immediate consequence operator* T_P (for a given program P). Now, if $T_P(I)(A) = \mathbf{t}$, then there exists a clause $A \leftarrow \texttt{body}$ in ground(P) such that $I(\texttt{body})$ is true, and we obtain $T_P(J)(A) = \mathbf{t}$ whenever $J(\texttt{body}) = \mathbf{t}$. The observation that bodies of clauses are finite conjunctions leads us to conclude the following lemma.

4.12 Lemma If $T_P(I)(A)$ is true, then T_P is locally finite for A and I. Furthermore, T_P is continuous if and only if it is locally finite for all A and I with $T_P(I)(A) = \mathbf{f}$.

A body $\bigvee C_i$ of a pseudo clause is false if and only if all C_i are false. Since T_P is a Fitting operator, we obtain $T_P(I)(A) = \mathbf{f}$ if and only if all C_i are false. If we require T_P to be locally finite for A and I, then there must be a finite set $S \subseteq \mathcal{B}_A$ such that any $J \in I_P$ which agrees with I on S renders all C_i false. These observations now easily yield the following theorem from [Sed95].

4.13 Theorem Let P be a normal logic program. Then T_P is continuous if and only if, for each $I \in I_P$ and for each $A \in B_P$ with $T_P(I)(A) = \mathbf{f}$, either there is no clause in P with head A or there exists a finite set $S(I, A) = \{A_1, \ldots, A_k, B_1, \ldots, B_{k'}\} \subseteq \mathcal{B}_A$ with the following properties:

(i) $A_1, \ldots, A_k$ are true in I and $B_1, \ldots, B_{k'}$ are false in I.

(ii) Given any clause C with head A, at least one $\neg A_i$ or at least one B_j occurs in the body of C.

p	q	$p \wedge q$	$p \vee q$	$\neg p$
t	t	t	t	f
t	u	u	t	f
t	f	f	t	f
u	t	u	t	u
u	u	u	u	u
u	f	f	u	u
f	t	f	t	t
f	u	f	u	t
f	f	f	f	t

Table 1: Connectives for Kleene's strong three-valued logic.

In the case of Kleene's strong three-valued logic, with set of truth values $\mathcal{T} = \{t, u, f\}$ and logical connectives as in Table 1, the associated Fitting operator was introduced in [Fit85] and is denoted by Φ_P, for a given program P. As in the case of classical two-valued logic, we obtain the following lemma.

4.14 Lemma If $\Phi_P(I)(A) = t$, then Φ_P is locally finite for A and I. Furthermore, Φ_P is continuous if and only if it is locally finite for all A and I with $\Phi_P(I)(A) \in \{u, f\}$.

Obtaining a theorem analogous to Theorem 4.13 is now straightforward, but tedious, and we omit the details. Similar considerations apply to the operator Ψ on Belnap's four-valued logic [Fit02] and to the operators from [HS99].

We mention in passing the non-monotonic Gelfond-Lifschitz operator [GL88] in classical two-valued logic, whose fixed points yield the stable models of the program in question. It turns out that this operator is not a consequence operator in the sense discussed in this paper, and attempts to characterize continuity of it will involve different methods (by means of the results from [Wen02], for example).

4.3 Approximation by Artificial Neural Networks

We have now finished our general preparations and continue next with our main task, namely, the study of the representability of logic programs by means of connectionist networks. We recall that the Cantor set $\mathcal{C}$ is a compact subset of the real line, and the topology which $\mathcal{C}$ inherits as a subspace of $\mathbb{R}$ coincides with the Cantor topology on $\mathcal{C}$. Also, the Cantor space $\mathcal{C}$ is homeomorphic to $I_{P,n}$ when the latter is endowed with a generalized atomic topology $\mathcal{Q}$. Hence, if a consequence operator T is continuous in $\mathcal{Q}$, we can identify it with a mapping $\iota(T) : x \mapsto \iota(T(\iota^{-1}(x)))$ on $\mathcal{C}$ which is continuous in the subspace topology of $\mathcal{C}$ in $\mathbb{R}$, as follows.

4.15 Theorem Let P be a program, let T be a consequence operator which is locally finite and let ι be a homeomorphism from $(I_{P,n}, \mathcal{Q})$ to $\mathcal{C}$. Then T (more precisely

$\iota(T)$) can be uniformly approximated by input-output mappings of 3-layer feedforward networks.

Proof: Under the conditions stated in the theorem, the operator T is continuous in $\mathcal{Q}$. Using the homeomorphism ι, the resulting function $\iota(T)$ is continuous on the Cantor set $\mathcal{C}$, which is a compact subset of $\mathbb{R}$. Applying Theorem 2.4, $\iota(T)$ can be uniformly approximated by input-output functions of 3-layer feedforward networks. ■

The restriction to programs with continuous consequence operator is not entirely satisfactory. There is another approximation theorem, due to [HSW89], which requires only measurability of the functions in question.

4.16 Theorem Suppose that ϕ is a monotone increasing function from $\mathbb{R}$ onto $(0,1)$. Let $f : \mathbb{R}^r \to \mathbb{R}$ be a Borel-measurable function and let μ be a probability Borel-measure on $\mathbb{R}^r$. Then, given any $\varepsilon > 0$, there exists a 3-layer feedforward network with squashing function ϕ whose input-output function $\bar{f} : \mathbb{R}^r \to \mathbb{R}$ satisfies

$$\varrho_\mu(f, \bar{f}) = \inf\{\delta > 0 : \mu\{x : |f(x) - \bar{f}(x)| > \delta\} < \delta\} < \varepsilon.$$

In other words, the class of functions computed by 3-layer feedforward neural nets is dense in the set of all Borel measurable functions $f : \mathbb{R}^r \to \mathbb{R}$ relative to the metric ϱ_μ defined in Theorem 4.16.

By means of Theorem 4.10, we can now view a local consequence operator T as a measurable function $\iota(T)$ on $\mathcal{C}$ by identifying $I_{P,n}$ with $\mathcal{C}$ via a homeomorphism ι. Since $\mathcal{C}$ is measurable as a subset of the real line, this operator can be extended[6] to a measurable function on $\mathbb{R}$ and we obtain the following result.

4.17 Theorem Given any program P with local consequence operator T, the operator T (more precisely $\iota(T)$) can be approximated in the manner of Theorem 4.16 by input-output mappings of 3-layer feedforward networks.

This result is somewhat unsatisfactory since the approximation stated in Theorem 4.16 is only *almost everywhere*, that is, pointwise with the exception of a set of measure zero. The Cantor set is, however, a set of measure zero. We can strenghen the result a bit by giving an explicit construction for the two-valued case. We define a sequence (T_n) of measurable functions on $\mathbb{R}$ as follows, where $l(x) = \max\{y \in \mathcal{C} : y \leq x\}$ and $u(x) = \min\{y \in \mathcal{C} : y \geq x\}$ for each $x \in [0,1] \setminus \mathcal{C}$:

[6]For example, as a function $T : \mathbb{R} \to \mathbb{R}$ with $T(x) = \iota(T_P(\iota^{-1}(x)))$ if $x \in \mathcal{C}$ and $T(x) = 0$ otherwise.

$$T_0(x) = \begin{cases} \iota(T_P)(x) & \text{if } x \in \mathcal{C} \\ \iota(T_P)(0) & \text{if } x < 0 \\ \iota(T_P)(1) & \text{if } x > 1 \\ 0 & \text{otherwise} \end{cases}$$

$$T_1(x) = \begin{cases} \iota(T_P)(l(x)) + \frac{\iota(T_P)(u(x)) - \iota(T_P)(l(x))}{u(x) - l(x)} & \text{if } x \in [3^{-1}, 2 \cdot 3^{-1}] \\ 0 & \text{otherwise} \end{cases}$$

$$T_i(x) = \begin{cases} \iota(T_P)(l(x)) + \frac{\iota(T_P)(u(x)) - \iota(T_P)(l(x))}{u(x) - l(x)}(x - l(x)) & \text{if } x \in \bigcup_{k=1}^{2 \cdot 3^{i-2}} [(2k-1)3^{-i}, 2k \cdot 3^{-i}] \\ 0 & \text{otherwise} \end{cases}$$

for $i \geq 2$.

We define the function $T : \mathbb{R} \to \mathbb{R}$ by $T(x) = \sup_i T_i(x)$ and obtain $T(x) = \iota(T_P(x))$ for all $x \in \mathcal{C}$ and $T(\iota(I)) = \iota(T_P(I))$ for all $I \in I_P$. Since all the functions T_i, for $i \geq 1$, are piecewise linear and therefore measurable, the function T is also measurable. Intuitively, T is obtained by a kind of linear interpolation.

If $i : B_P \to \mathbb{N}$ is a bijective mapping, then we can obtain a homeomorphism $\iota : I_P \to \mathcal{C}$ from i as follows: we identify $I \in I_P$ with $x \in \mathcal{C}$ where x written in ternary form has 2 as its $i(A)$ th digit (after the decimal point) if $A \in I$, and 0 as its $i(A)$ th digit if $A \notin I$. If $I \in I_P$ is finite or cofinite[7], then the sequence of digits of $\iota(I)$ in ternary form is eventually constant 0 (if I finite) or eventually constant 2 (if I cofinite). Thus, each such interpretation is the endpoint of a linear piece of one of the functions T_i, and therefore of T.

4.18 Corollary Given any normal logic program P, its single-step operator T_P (more precisely $\iota(T_P)$) can be approximated by input-output mappings of 3-layer feedforward networks in the following sense: for every $\varepsilon > 0$ and for every $I \in I_P$ which is either finite or cofinite, there exist a 3-layer feedforward network with input-output function f and $x \in [0, 1]$ with $|x - \iota(I)| < \varepsilon$ such that $|\iota(T_P(I)) - f(x)| < \varepsilon$.

Proof: We use a homeomorphism ι which is obtained from a bijective mapping $i : B_P \to \mathbb{N}$ as in the paragraph preceeding the Corollary. We can assume that the measure μ from Theorem 4.16 has the property that $\mu\{[x, x+\varepsilon]\} \leq \varepsilon$ for each $x \in \mathbb{R}$. Let $\varepsilon > 0$ and $I \in I_P$ be finite or cofinite. Then by construction of T, there exists an interval $[\iota(I), \iota(I) + \delta]$ with $\delta < \frac{\varepsilon}{2}$ (or analogously $[\iota(I) - \delta, \iota(I)]$) such that T is linear on $[\iota(I), \iota(I) + \delta]$ and $|T(\iota(I)) - T(x)| < \frac{\varepsilon}{2}$ for all $x \in [\iota(I), \iota(I) + \delta]$. By Theorem 4.16 and the previous paragraph, there exists a 3-layer feedforward network with input-output function f such that $\varrho_\mu(T, f) < \delta$, that is, $\mu\{x : |T(x) - f(x)| > \delta\} < \delta$. By our condition on μ, there is $x \in [\iota(I), \iota(I)+\delta]$ with $|T(x)-f(x)| \leq \delta < \frac{\varepsilon}{2}$. We can conclude that $|\iota(T_P(I)) - f(x)| = |T(\iota(I)) - f(x)| \leq |T(\iota(I)) - T(x)| + |T(x) - f(x)| < \varepsilon$, as required. ■

[7] $I \in I_P$ is cofinite if $B_P \setminus I$ is finite.

It would be of interest to strengthen this approximation for sets other than the finite and cofinite elements of I_P, although it is interesting to note that the finite interpretations correspond to compact elements in the sense of domain theory, see [AJ94].

We want to return now to the case discussed earlier in Theorem 4.15. In Section 3, and also in [HKS99], the following recurrent neural network architecture was considered: we assume that the number of output and input units is equal and that, after each propagation through the network, the output values are fed back without changes into input values. For the case which we consider, it will again be sufficient to suppose that the input layer consists of one unit only, so that the architecture can be depicted as in Figure 6.

We will show in the following that iterates of locally finite local consequence operators can be approximated arbitrarily closely by iterates of suitably chosen networks. This is in fact a consequence of the uniform approximation obtained from Theorem 2.4 and the compactness of the unit interval.

Let P be a logic program, let T be a locally finite local consequence operator for P and let $\iota : I_P \to \mathcal{C}$ be a homeomorphism. Let F be a continuous extension of $\iota(T)$ onto the unit interval $[0,1]$ in the reals, let d be the natural metric on $\mathbb{R}$, and let $\varepsilon > 0$. By Theorem 4.15, there exists a 3-layer feedforward network with input-output mapping f such that $\max_{x\in[0,1]} d(f(x), F(x)) < \varepsilon$. Since $[0,1]$ is compact and F is continuous, we obtain that F is Lipschitz-continuous, that is, there exists $\lambda \geq 0$ such that for all $x, y \in [0,1]$ we have $d(F(x), F(y)) \leq \lambda d(x, y)$. For $x, y \in [0,1]$ we therefore obtain

$$d(f(x), F(y)) \leq d(f(x), F(x)) + d(F(x), F(y)) \leq \varepsilon + \lambda d(x, y). \tag{5}$$

Now let $x \in [0,1]$ be arbitrarily chosen. By Equation (5) we obtain

$$d(f^2(x), F^2(x)) \leq \varepsilon + \lambda d(f(x), F(x)) \leq \varepsilon + \lambda\varepsilon. \tag{6}$$

Inductively, we can prove that for all $n \in \mathbb{N}$ we have

$$d(f^n(x), F^n(x)) \leq \varepsilon + \lambda\varepsilon + \cdots + \lambda^{n-1}\varepsilon = \varepsilon \left(\sum_{i=0}^{n-1} \lambda^i \right) = \varepsilon \frac{1-\lambda^n}{1-\lambda}. \tag{7}$$

Thus, we obtain the following bound on the error produced by the recurrent network after n iterations.

4.19 Theorem With the notation and hypotheses above, for any $I \in I_P$ and any $n \in \mathbb{N}$ we have

$$|f^n(\iota(I)) - \iota(T^n(I))| \leq \varepsilon \frac{1-\lambda^n}{1-\lambda}.$$

Proof: Note that $\iota(T^n(I)) = F^n(\iota(I))$, and the assertion follows from Equation (7) since d is the natural metric on $\mathbb{R}$. ■

We derive a few corollaries from this result.

4.20 Corollary If F is a contraction on $[0,1]$, so that $\lambda < 1$, then $\big(F^k(\iota(I))\big)$ converges for every I to the unique fixed point x of F and there exists $m \in \mathbb{N}$ such that for all $n \geq m$ we have

$$|f^n(\iota(I)) - x| \leq \varepsilon \frac{1}{1-\lambda}.$$

Proof: The convergence follows from the Banach contraction mapping theorem. The inequality follows immediately from Theorem 4.19 using the well-known expression for limits of geometric series. ■

If F is a contraction on $[0,1]$, then T is a contraction on the complete subspace $\mathcal{C}$, and also has a fixed point M with $\iota(M) = x$. However, it seems difficult to guarantee the hypothesis of Corollary 4.20, although in [HKS99] a similar result for acyclic programs with injective level mappings in classical logic was achieved. The following result may be more promising.

4.21 Corollary If, for some $I \in I_P$, $T^n(I)$ converges in $\mathcal{Q}$ to a fixed point M of T, then, for every $\delta > 0$, there exists a network with input-output function f and some $n \in \mathbb{N}$ such that $|f^n(\iota(I)) - \iota(M)| < \delta$.

Proof: The hypothesis implies that $F^n(\iota(I))$ converges to $\iota(M)$ in the natural metric on $\mathbb{R}$. Given $\delta > 0$, there exists $n \in \mathbb{N}$ such that $|F^m(\iota(I)) - \iota(M)| < \frac{\delta}{2}$ for all $m \geq n$. Since F is fixed, we know the value of λ. Now, by the approximation results above, we choose a network with input-output function f such that $\varepsilon \frac{1-\lambda^n}{1-\lambda} < \frac{\delta}{2}$. Then using Theorem 4.19 and the triangle inequality we obtain

$$\begin{aligned} |f^n(\iota(I)) - \iota(M)| &\leq |f^n(\iota(I)) - F^n(\iota(I))| + |F^n(\iota(I)) - \iota(M)| \\ &< 2 \cdot \frac{\delta}{2} \leq \delta. \end{aligned}$$

■

We close by describing a class of programs for which the additional hypothesis from Corollary 4.21 is satisfied. The result is well-known for the case of classical two-valued logic and the immediate consequence operator. So, let P be acyclic with level mapping l, and let T be a local consequence operator for P. We define a mapping $d : I_P \times I_P \to \mathbb{R}$ by $d(I,J) = 2^{-n}$, where n is least such that I and J differ on some atom A with $l(A) = n$. It is easily verified that d is a complete metric on I_P, see [Fit94].

4.22 Proposition With the stated hypotheses, T is a contraction with respect to d.

Proof: Suppose $d(I,J) = 2^{-n}$. Then I and J coincide on all atoms of level less than n. Now let $A \in B_P$ with $l(A) = n$. Then by acyclicity of P we have that all atoms in $\mathcal{B}_A$ are of level less than n, and by locality of T we have that $T(I)(A) = T(J)(A)$. So $d(T(I), T(J)) \leq 2^{-(n+1)}$. ■

We obtain finally the following theorem.

4.23 Theorem Let P be an acyclic program and let T be a local consequence operator for P. Then, for any $I \in I_P$, we have that $T^n(I)$ converges in $\mathcal{Q}$ to the unique fixed point M of T.

Proof: By Proposition 4.22 and the fact that d is a complete metric, we can apply the Banach contraction mapping theorem to obtain the convergence of $T^n(I)$ in d to a unique fixed point M of T. By definition of d, the convergence of the sequence of interpretations $T^n(I)$ to M must be pointwise, hence is also convergence in $\mathcal{Q}$. ■

Theorem 4.23 is remarkable since the existence of a fixed point of the semantic operator can be guaranteed without any particular knowledge about the underlying multi-valued logic.

5 Conclusions and Further Work

In considering the integration of Logic and Connectionist Systems, we have taken the natural point of contact between them provided by the immediate consequence operator T_P, associated with a normal logic program P, and the issue of its computation by means of neural networks. In so far as one may identify two logic programs with the same immediate consequence operator (subsumption equivalence), this provides a sort of semantics for a neural network which computes T_P, namely, the supported model semantics of P.

A number of questions arise out of these considerations, and we close by briefly mentioning a few of them, as follows. First, there is the question of giving explicit constructions of networks for approximating T_P in case that T_P is continuous, and this point is considered in [BH03]. A question which is also related to the results given in [BH03] is that of providing good bounds on Lipschitz constants for f_P, and this issue appears to be central to actually giving constructions of approximating networks. Another natural question concerns carrying over the programme given here for the supported model semantics of a normal logic program to the stable model semantics [GL88] and the well-founded semantics [vGRS91], and one possible means of doing this is provided by the results of [Wen02]. From the connectionist point of view, the main open question is how to build a connectionist network given a first-order logic program. Ideally, assuming that this is done, we would then like to apply known connectionist learning techniques, in particular backpropagation, to such networks and, after training, extract a refined set of first-order clauses from the network. Finally, there is the purely mathematical question of what mathematical notions of approximation are useful and appropriate. Here we have discussed two well-known ones: uniform approximation on compacta, and a notion of approximation closely related to convergence in measure. However, others may prove to be significant, and this is a problem still to be investigated.

References

[ADT95] Robert Andrews, Joachim Diederich, and Alan B. Tickle. A survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge–Based Systems*, 8(6), 1995.

[AJ94] Samson Abramsky and Achim Jung. Domain theory. In Samson Abramsky, Dov Gabbay, and Thomas S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 3. Clarendon, Oxford, 1994.

[AP93] Krzysztof R. Apt and Dino Pedreschi. Reasoning about termination of pure Prolog programs. *Information and Computation*, 106:109–157, 1993.

[Bar66] Robert G. Bartle. *The Elements of Integration.* John Wiley & Sons, New York, 1966.

[Bez89] Marc Bezem. Characterizing termination of logic programs with level mappings. In Ewing L. Lusk and Ross A. Overbeek, editors, *Proceedings of the North American Conference on Logic Programming*, pages 69–80. MIT Press, Cambridge, MA, 1989.

[BH03] Sebastian Bader and Pascal Hitzler. Logic programs, iterated function systems, and recurrent radial basis function networks. Technical Report WV–03–11, Knowledge Representation and Reasoning Group, Artificial Intelligence Institute, Department of Computer Science, Dresden University of Technology, Dresden, Germany, 2003. To appear in this volume.

[BS89] Aida Batarekh and V.S. Subrahmanian. Topological model set deformations in logic programming. *Fundamenta Informaticae*, 12:357–400, 1989.

[Cav91] Lawrence Cavedon. Acyclic programs and the completeness of SLDNF-resolution. *Theoretical Computer Science*, 86:81–92, 1991.

[Cla78] Keith L. Clark. Negation as failure. In Hervé Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

[DG84] William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *The Journal of Logic Programming*, 1(3):267–284, 1984.

[dGBG01] Artur S. d'Avila Garcez, Krysia Broda, and Dov M. Gabbay. Symbolic knowledge extraction from trained neural networks: A sound approach. *Artificial Intelligence*, 125:155–207, 2001.

[dGBG02] Artur S. d'Avila Garcez, Krysia B. Broda, and Dov M. Gabbay. *Neural-Symbolic Learning Systems — Foundations and Applications.* Perspectives in Neural Computing. Springer, Berlin, 2002.

[dGLG02] Artur S. d'Avila Garcez, Luís C. Lamb, and Dov M. Gabbay. A connectionist inductive learning system for modal logic programming. In *Proceedings of the IEEE International Conference on Neural Information Processing ICONIP'02, Singapore*, 2002.

[dGZ99] Artur S. d'Avila Garcez and Gerson Zaverucha. The connectionist inductive learning and logic programming system. *Applied Intelligence, Special Issue on Neural networks and Structured Knowledge*, 11(1):59–77, 1999.

[dGZdC97] Artur S. d'Avila Garcez, Gerson Zaverucha, and Luis A. V. de Carvalho. Logical inference and inductive learning in artificial neural networks. In Christoph Hermann, Frank Reine, and Antje Strohmaier, editors, *Knowledge Representation in Neural networks*, pages 33–46. Logos Verlag, Berlin, 1997.

[DMT00] Marc Denecker, V. Wiktor Marek, and Miroslaw Truszczynski. Approximating operators, stable operators, well-founded fixpoints and applications in non-monotonic reasoning. In Jack Minker, editor, *Logic-based Artificial Intelligence*, chapter 6, pages 127–144. Kluwer Academic Publishers, Boston, 2000.

[Fit85] Melvin Fitting. A Kripke-Kleene semantics for general logic programs. *The Journal of Logic Programming*, 2:295–312, 1985.

[Fit94] Melvin Fitting. Metric methods: Three examples and a theorem. *The Journal of Logic Programming*, 21(3):113–127, 1994.

[Fit02] Melvin Fitting. Fixpoint semantics for logic programming — A survey. *Theoretical Computer Science*, 278(1–2):25–51, 2002.

[FP88] Jerry A. Fodor and Zenon W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. In Steven Pinker and Jacques Mehler, editors, *Connections and Symbols*, pages 3–71. MIT Press, 1988.

[Fun89] Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183–192, 1989.

[GL88] 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.

[Hit01] Pascal Hitzler. *Generalized Metrics and Topology in Logic Programming Semantics*. PhD thesis, Department of Mathematics, National University of Ireland, University College Cork, 2001.

[HK94] Steffen Hölldobler and Yvonne Kalinke. Towards a massively parallel computational model for logic programming. In *Proceedings ECAI94 Workshop*

on Combining Symbolic and Connectionist Processing, pages 68–77. ECCAI, 1994.

[HKP91] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley Publishing Company, 1991.

[HKS99] Steffen Hölldobler, Yvonne Kalinke, and Hans-Peter Störr. Approximating the semantics of logic programs by recurrent neural networks. *Applied Intelligence*, 11:45–58, 1999.

[HKW00] Steffen Hölldobler, Yvonne Kalinke, and Jörg Wunderlich. A recursive neural network for reflexive reasoning. In Stefan Wermter and Ron Sun, editors, *Hybrid Neural Symbolic Integration*, number 1778 in Lecture Notes in Artificial Intelligence, pages 46–62. Springer, 2000.

[Höl93] Steffen Hölldobler. Automated inferencing and connectionist models. Technical Report AIDA–93–06, Intellektik, Informatik, TH Darmstadt, 1993. (Postdoctoral Thesis).

[HS99] Pascal Hitzler and Anthony K. Seda. Characterizations of classes of programs by three-valued operators. In Michael Gelfond, Nicola Leone, and Gerald Pfeifer, editors, *Logic Programming and Non-monotonic Reasoning, Proceedings of the 5th International Conference on Logic Programming and Non-Monotonic Reasoning, LPNMR'99, El Paso, Texas, USA*, volume 1730 of *Lecture Notes in Artificial Intelligence*, pages 357–371. Springer, Berlin, 1999.

[HS00] Pascal Hitzler and Anthony K. Seda. A note on relationships between logic programs and neural networks. In Paul Gibson and David Sinclair, editors, *Proceedings of the Fourth Irish Workshop on Formal Methods, IWFM'00*, Electronic Workshops in Computing (eWiC). British Computer Society, 2000.

[HS01a] Pascal Hitzler and Anthony K. Seda. A "converse" of the Banach contraction mapping theorem. *Journal of Electrical Engineering*, 52(10/s):3–6, 2001. Proceedings of the 3rd Slovakian Student Conference in Applied Mathematics, SCAM2001, Bratislava. Slovak Academy of Sciences.

[HS01b] Pascal Hitzler and Anthony K. Seda. Semantic operators and fixed-point theory in logic programming. In *Proceedings of the joint IIIS & IEEE meeting of the 5th World Multiconference on Systemics, Cybernetics and Informatics, SCI2001 and the 7th International Conference on Information Systems Analysis and Synthesis, ISAS2001, Orlando, Florida, USA*. International Institute of Informatics and Systemics: IIIS, 2001.

[HS03a] Pascal Hitzler and Anthony K. Seda. Continuity of semantic operators in logic programming and their approximation by artificial neural networks. In Andreas Günter, Rudolf Krause, and Bernd Neumann, editors, *Proceedings of*

the 26th German Conference on Artificial Intelligence, KI2003, volume 2821 of *Lecture Notes in Artificial Intelligence*, pages 105–119. Springer, 2003.

[HS03b] Pascal Hitzler and Anthony K. Seda. Generalized metrics and uniquely determined logic programs. *Theoretical Computer Science*, 305(1–3):187–219, 2003.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[JL77] Neil D. Jones and William T. Laaser. Complete problems for deterministic sequential time. *Theoretical Computer Science*, 3:105–117, 1977.

[KR90] Richard M. Karp and Vijaya Ramachandran. Parallel algorithms for shared-memory machines. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 17, pages 869–941. Elsevier Science Publishers B.V., New York, 1990.

[Llo88] John W. Lloyd. *Foundations of Logic Programming*. Springer, Berlin, 1988.

[McC88] John McCarthy. Epistomological challenges for connectionism. *Behavioral and Brain Sciences*, 11:44, 1988. (Commentary on [Smo88]).

[McI00] Yvonne McIntyre. *Modellgenerierung mit konnektionistischen Systemen*. PhD thesis, TU Dresden, Fakultät Informatik, 2000.

[MP72] Marvin L. Minsky and Seymour Papert. *Perceptrons*. MIT Press, 1972.

[New80] Allen Newell. Physical symbol systems. *Cognitive Science*, 4:135–183, 1980.

[Pla91] Tony A. Plate. Holographic reduced representations. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 30–35, 1991.

[Pol88] Jordan B. Pollack. Recursive auto-associative memory: Devising compositional distributed representations. In *Proceedings of the 10th Annual Conference of the Cognitive Science Society*, pages 33–39, 1988.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing*. MIT Press, 1986.

[Ros62] Frank Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Machines*. Spartan Books, Washington, 1962.

[SA93] Lokendra Shastri and Venkat Ajjanagadde. From associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences*, 16(3):417–494, September 1993.

[Scu90] Maria G. Scutellà. A note on Dowling and Gallier's top-down algorithm for propositional Horn satisfiability. *The Journal of Logic Programming*, 8:265–273, 1990.

[Sed95] Anthony K. Seda. Topology and the semantics of logic programs. *Fundamenta Informaticae*, 24(4):359–386, 1995.

[Smo88] Paul Smolensky. On the proper treatment of connectionism. *Behavioral and Brain Sciences*, 11:1–74, 1988.

[TS94] Geoffrey G. Towell and Jude W. Shavlik. Knowledge-based artificial neural networks. *Artificial Intelligence*, 70(1–2):119–165, 1994.

[vGRS91] Allen van Gelder, Kenneth A. Ross, and John S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

[Wen02] Matthias Wendt. Unfolding the well-founded semantics. *Journal of Electrical Engineering, Slovak Academy of Sciences*, 53(12/s):56–59, 2002. (Proceedings of the 4th Slovakian Student Conference in Applied Mathematics, Bratislava, April 2002).

[Wil70] Stephen Willard. *General Topology.* Addison-Wesley, Reading, MA, 1970.