Logic Programs, Iterated Function Systems, and Recurrent Radial Basis Function Networks

Sebastian Bader and Pascal Hitzler*

Artificial Intelligence Institute
Department of Computer Science
Dresden University of Technology

s.bader@gmx.net, phitzler@inf.tu-dresden.de
www.wv.inf.tu-dresden.de/∼{borstel,pascal}

Abstract

Graphs of the single-step operator for first-order logic programs — displayed in the real plane — exhibit self-similar structures known from topological dynamics, i.e. they appear to be *fractals*, or more precisely, attractors of iterated function systems. We show that this observation can be made mathematically precise. In particular, we give conditions which ensure that those graphs coincide with attractors of suitably chosen iterated function systems, and conditions which allow the approximation of such graphs by iterated function systems or by fractal interpolation. Since iterated function systems can easily be encoded using recurrent radial basis function networks, we eventually obtain connectionist systems which approximate logic programs in the presence of function symbols.

*Corresponding author

Contents

1 Introduction

Intelligent systems based on logic programming on the one hand, and on artificial neural networks (sometimes called connectionist sytems) on the other, differ substantially. Logic programs are highly recursive and well understood from the perspective of knowledge representation: The underlying language is that of first-order logic, which is symbolic in nature and makes it easy to encode problem specifications directly as programs. The success of artificial neural networks lies in the fact that they can be trained using raw data, and in some problem domains the generalization from the raw data made during the learning process turns out to be highly adequate for the problem at hand, even if the training data contains some noise. Successful architectures, however, often do not use recursive (or recurrent) structures. Furthermore, the knowledge encoded by a trained neural network is only very implicitly represented, and no satisfactory methods for extracting this knowledge in symbolic form are currently known.

It would be very desirable to combine the robust neural networking machinery with symbolic knowledge representation and reasoning paradigms like logic programming in such a way that the strenghts of either paradigm will be retained. Current state-of-the-art research, however, fails by far to achieve this ultimate goal. As one of the main obstacles to be overcome we perceive the question how symbolic knowledge can be encoded by artificial neural networks: Satisfactory answers to this will naturally lead the way to knowledge extraction algorithms and to hybrid neural-symbolic systems.

Earlier attempts to integrate logic and connectionist systems have mainly been restricted to propositional logic, or to first-order logic without function symbols. They go back to the pioneering work by McCulloch and Pitts [35], and have led to a number of systems developed in the 80s and 90s, including Towell and Shavlik's KBANN [44], Shastri's SHRUTI [43], the work by Pinkas [37], Hölldobler [27], and d'Avila Garcez et al. [12, 14], to mention a few, and we refer to [10, 13, 17] for comprehensive literature overviews.

Without the restriction to the finite case (including propositional logic and first-order logic without function symbols), the task becomes much harder due to the fact that the underlying language is infinite but shall be encoded using networks with a finite number of nodes. The sole approach known to us for overcoming this problem (apart from work on recursive autoassociative memory, RAAM, initiated by Pollack [38], which concerns the learning of recursive terms over a first-order language) is based on a proposal by Hölldobler et al. [30], spelled out first for the propositional case in [29], and reported also in [21]. It is based on the idea that logic programs can be represented — at least up to subsumption equivalence [33] — by their associated single-step or immediate consequence operators. Such an operator can then be mapped to a function on the real numbers, which can under certain conditions in turn be encoded or approximated e.g. by feedforward networks with sigmoidal activation functions using an approximation theorem due to Funahashi [16].

While contemplating this approach, we plotted graphs of resulting real-valued functions and found that in *all* cases these plots showed self-similar structures as known from topological dynamics. To be more precise, they looked like *fractals* in the sense of attractors of iterated function systems [3], see Figure 1 and Figure 7 on page 10 for examples. While the general observation that logic programming is linked to topological dynamics and chaos theory is not new (see the work by Blair et al. [8, 9]), the strikingly self-similar representation in the Euclidean plane offers a setting for developing real-valued iterated function systems for representing logic programs, with the concrete goal of in turn converting these into recurrent neural networks, thus obtaining connectionist representations of logic programs.

In this paper we substantiate formally the fact that these plots can indeed be obtained as attractors of iterated function systems, and give concrete representations of such systems. More generally, we give necessary and sufficient conditions under which graphs of single-step operators in the Euclidean plane arise as attractors of certain iterated function systems. We will give algorithms for constructing iterated function systems and fractal interpolation systems for approximating graphs of single-step operators. We will finally use our results for constructing recurrent radial basis function networks which approximate graphs of single-step operators.

The paper is structured as follows. In Section 2, we will introduce basic notions concerning logic programs and iterated function systems which we will need throughout the paper.

In Section 3, we show that graphs of logic programs can be obtained as attractors of iterated function systems. In particular, in Theorem 3.2 we will give necessary and sufficient conditions under which this is possible. Building on this, in Theorem 3.4 we will show that these conditions are satisfied whenever the embedded single-step operator is Lipschitz continuous with respect to the natural metric on the real numbers. The section closes with a concrete construction of an iterated function system and two detailed examples.

In Section 4 we shift our attention to the task of approximating logic programs — via their single-step operators — by means of fractal interpolation. More precisely, in Theorem 4.6 we show that programs with Lipschitz continuous single-step operator can be approximated uniformly by this method.

In Section 5 we will use our insights in order to show how logic programs can be represented or approximated by recurrent radial basis function networks.

The paper closes with a discussion of related and further work.

Most of the new results in this paper are discussed in more detail in [2].

Acknowledgement. We benefitted substantially from discussions with Steffen Hölldobler and his support of the project. The comments of three anonymous referees were highly appreciated and led to improvements of the presentation of our results. We are very grateful that Howard Blair spotted a mistake in an earlier version of this paper, which we were now able to remove.

```
n(0).
n(s(X)) :- n(X).
```

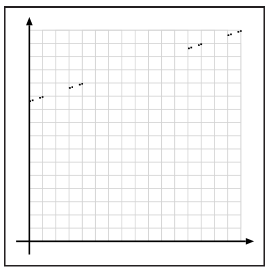

Figure 1: The graph of a real valued single-step operator.

2 Preliminaries

We will now shortly review and introduce terminology and notation from logic programming and iterated function systems, which we will use throughout. It will be helpful if the reader is familiar with these areas, but we will make an attempt to keep the paper self-contained in this respect, with terminology essentially following [32] respectively [3]. In some places we will have to assume basic knowledge of set-theoretic topology, our general reference being [45]. For Section 5, some familiarity with radial basis function networks (e.g. [7, Chapter 5]) will be helpful.

2.1 Logic Programs

A (*normal*) *logic program* is a finite set of universally quantified *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 a 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

$$A \leftarrow L_1, \ldots, L_n,$$

in which the universal quantifier is understood, or even as

$$A\texttt{:-}L_1, \ldots, L_n$$

following Prolog notation. Then A is called the *head* of the clause, each L_i is called a *body literal* of the clause and their conjunction $L_1, \ldots, L_n$ 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*. The *Herbrand base* underlying a given program $\mathcal{P}$ is defined as the set of all ground instances of atoms over $\mathcal{L}$ and will be denoted by $B_\mathcal{P}$. Figure 2 shows an example of a logic program and the corresponding Herbrand base. Subsets of the Herbrand base are called (*Herbrand*) *interpretations* of $\mathcal{P}$, and we can think of such a set as containing those atoms which are "true" under the interpretation. The set $I_\mathcal{P}$ of all interpretations of a program $\mathcal{P}$ can be identified with the power set of $B_\mathcal{P}$.

In this paper, we will not make use of any procedural aspects concerning logic programs. Indeed, logic programs are being used for many different purposes in computer science, e.g. as the language underlying Prolog [32], as languages for non-monotonic reasoning [31, 34], for machine learning [36], etc, and the respective computational mechanisms differ substantially. Common to all these paradigms, however, is that logic programs are accepted as a convenient tool for knowledge representation in logical form. The knowledge represented by a logic program $\mathcal{P}$ can essentially be captured by the *immediate consequence* or *single-step operator* $T_\mathcal{P}$, which is defined as a mapping on $I_\mathcal{P}$ where for any $I \in I_\mathcal{P}$ we have that $T_\mathcal{P}(I)$ is the set of all $A \in B_\mathcal{P}$ for which there exists a ground instance $A \leftarrow A_1, \ldots, A_m, \neg B_1, \ldots, \neg B_n$ of a clause in $\mathcal{P}$ such that for all i we have $A_i \in I$ and for all j we have $B_j \notin I$.

A *level mapping* for a program $\mathcal{P}$ is a mapping $|\cdot| : B_\mathcal{P} \to \mathbb{N}$, and with a slight abuse of notation we set $|\neg A| = |A|$ for each $A \in B_\mathcal{P}$. Figure 2 shows a simple logic program, the corresponding Herbrand base $B_\mathcal{P}$, and a possible level mapping. Level mappings can be used for describing dependencies between atoms in a program, and they have been studied in logic programming for many different purposes, e.g. for termination analysis under Prolog [1, 6], or for giving uniform descriptions of different non-monotonic semantics [20, 25, 26]. For our investigations, we can restrict our attention to *injective* level mappings, which can simply be understood as enumerations of the Herbrand base. The latter perspective was employed e.g. in [8]. It makes no essential difference, and we choose to stick with the more general notion of level mapping, and will explicitly require injectivity when needed.

Fitting [15] has used level mappings in order to define metrics on spaces of interpretations, an approach which was further extended in [19, 24]. Recall that a *metric* over a set X is a mapping $d : X \times X \to \mathbb{R}$ satisfying (i) $d(x, y) = 0$ iff $x = y$, (ii) $d(x, y) = d(y, x)$, and (iii) $d(x, z) \leq d(x, y) + d(y, z)$ for all $x, y, z \in X$. The pair (X, d) is then called a *metric space*. A metric is called an *ultrametric* if it satisfies the stronger requirement (iii') $d(x, z) \leq \max\{d(x, y), d(y, z)\}$ for all $x, y, z \in X$. On the real numbers, the function $d(x, y) = |x - y|$ is a metric and is called the *natural metric* on $\mathbb{R}$. A sequence $(x_n)_{n\in\mathbb{N}}$ in some metric space (X, d) *converges to* (or *has limit*) x, written $\lim x_n = x$, if for all $\varepsilon > 0$ there is some $n_0 \in \mathbb{N}$ such that $d(x_n, x) < \varepsilon$ for all $n \geq n_0$. A *Cauchy sequence* in a metric space (X, d) is a sequence (x_n) such that for each $\varepsilon > 0$ there exists $n_0 \in \mathbb{N}$ such that for all $m, n \geq n_0$ we have $d(x_m, x_n) < \varepsilon$. Converging sequences are always Cauchy sequences. A metric space in which every Cauchy sequence converges is called *complete*.

The following definition is a slight generalization of one given in [15].

2.1 Definition Let $\mathcal{P}$ be a logic program, $2 \leq B \in \mathbb{N}$, and let $|\cdot|$ be a level mapping for $\mathcal{P}$. For $I, J \in I_\mathcal{P}$ define

$$d_B(I, J) = \begin{cases} 0 & \text{if } I = J, \\ B^{-n} & \text{if } I \text{ and } J \text{ differ on some atom } A \text{ with } |A| = n, \\ & \text{but agree on all atoms with a level smaller than } n. \end{cases}$$

It is easily verified that $(I_\mathcal{P}, d_B)$ is a complete metric space, indeed an ultrametric space. If $|\cdot|$ is injective — or more generally, if for each $n \in \mathbb{N}$ the set of all atoms with level n is finite — then the metric d_B, for any B, induces a topology on $I_\mathcal{P}$ which is known as the *query* [5] or *atomic* [41] topology Q. If furthermore the language underlying $\mathcal{P}$ contains at least one function symbol of arity at least 1, then $(I_\mathcal{P}, Q)$ is homeomorphic, i.e. topologically equivalent, to the Cantor space in the unit interval on the real line [41], which we will discuss further in Example 2.5.

A logic program $\mathcal{P}$ is *acyclic* [6, 11] if there exists a level mapping $|\cdot|$ for $\mathcal{P}$ such that for each ground instance $A \leftarrow L_1, \ldots, L_n$ of a clause in $\mathcal{P}$ we have that $|A| > |L_i|$ for all $i = 1, \ldots, n$. In this case the operator $T_\mathcal{P}$ is a contraction on $(I_\mathcal{P}, d_B)$ with contractivity factor B^{-1}, i.e. it satisfies $d_B(T_\mathcal{P}(I), T_\mathcal{P}(J)) \leq B^{-1} d_B(I, J)$ for all $I, J \in I_\mathcal{P}$ [15, 24].

2.2 Iterated Function Systems

Iterated function systems originate from the study of chaos theory and self-similar structures and they have found applications e.g. in image compression. An excellent introduction to the field is [3], and we follow its notation, as already mentioned. We will later

$\mathcal{P}$:	$B_{\mathcal{P}}$:	$\lvert\cdot\rvert$:
`n(0).` `n(s(X)) :- n(X).`	$n(0), n(s(0)), n(s(s(0))),$ $n(s(s(s(0)))), \ldots$	$\lvert n(s^x(0))\rvert = x + 1$

Figure 2: A logic program, the corresponding Herbrand base, and a level mapping

make use of the fact that real-valued iterated function systems can easily be encoded using recurrent neural networks, a point to which we will return in Section 5.

Recall that a function $f : X \to X$ on a metric space (X, d) is *continuous* if for all $\varepsilon > 0$ there exists $\delta > 0$ such that $d(f(x), f(y)) < \varepsilon$ whenever $d(x, y) < \delta$. A *Lipschitz continuous function* is a mapping $f : X \to X$ for which there exists a real number $\lambda \geq 0$, called a *Lipschitz constant* for f, such that $d(f(x), f(y)) \leq \lambda d(x, y)$ for all $x, y \in X$. *Contraction mappings* are exactly those Lipschitz continuous functions which have a Lipschitz constant (called *contractivity factor*) less than 1. Every contraction is Lipschitz continuous, and every Lipschitz continuous function is continuous. The importance of contractions lies in the fact that every contraction f on a complete metric space (X, d) has a unique fixed point x, which can be obtained as $\lim f^n(y)$, for all $y \in X$, where $f^n(y)$ denotes the n-th iteration of the function f on the point y. This fact is well-known as the *Banach contraction mapping theorem*.

2.2 Definition A (hyperbolic) *iterated function system (IFS)* $((X, d), \Omega)$ is a pair consisting of a complete metric space (X, d) and a finite set $\Omega = \{\omega_1, \ldots, \omega_n\}$ of contraction mappings $\omega_i : X \to X$.

The idea behind iterated function systems is to lift the set Ω to be a contraction mapping on a space of certain subsets of X. More precisely, we consider *compact* subsets of X, which can be characterized as follows: $A \subseteq X$ is *compact* if for every (possibly infinite) collection of sets $B_{\varepsilon_i}(x_i) = \{y \mid d(x_i, y) < \varepsilon_i\}$ with $A \subseteq \bigcup_{i\in I} B_{\varepsilon_i}(x_i)$ there exists a finite selection $\{i_1, \ldots, i_n\} \subseteq I$ with $A \subseteq \bigcup_{k=1}^{n} B_{\varepsilon_{i_k}}(x_{i_k})$.

Given (X, d), we define $\mathcal{H}(X)$ to be the set of all non-empty compact subsets of X, and define the *Hausdorff distance* on $\mathcal{H}(X)$ as follows.

2.3 Definition Let (X, d) be a complete metric space, $x \in X$ and $A, B \in \mathcal{H}(X)$. Then $d(x, B) = \min\{d(x, y) \mid y \in B\}$ is called the distance between the point x and the set B. The distance from A to B is then defined as $d(A, B) = \max\{d(a, B) \mid a \in A\}$. Finally, the *Hausdorff distance* h_d between A and B is defined as $h_d(A, B) = \max\{d(A, B), d(B, A)\}$.

The resulting *Hausdorff space* $(\mathcal{H}(X), h_d)$ is a complete metric space. A continuous mapping $f : X \to X$ can be extended to a function on $\mathcal{H}(X)$ in the usual way, i.e. by setting $f(A) = \{f(a) \mid a \in A\}$ (recalling that any continuous image of a compact set is compact). Given an IFS consisting of a metric space (X, d) and a set Ω of contractions, we identify Ω with a function on $\mathcal{H}(X)$ defined by $\Omega(A) = \bigcup_i w_i(A)$. The function Ω thus defined is a contractive mapping on $\mathcal{H}(X)$, and by the Banach contraction mapping theorem we can conclude that Ω has a unique fixed point $A \in \mathcal{H}(X)$, which hence obeys $A = \Omega(A)$ and can be obtained from any $B \in \mathcal{H}(X)$ as $A = \lim_{n\to\infty} \Omega^n(B)$, the limit being taken with respect to h_d. The fixed point $A \in \mathcal{H}(X)$ is called the *attractor* of the IFS $((X, d), \Omega)$.

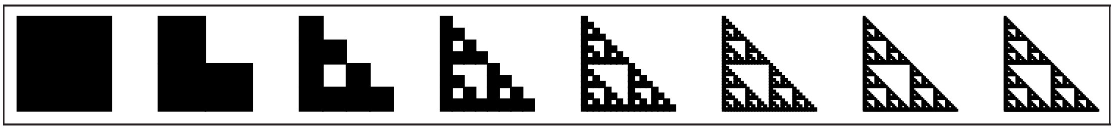

Figure 3: The first iterations for the production of the Sierpinski triangle.

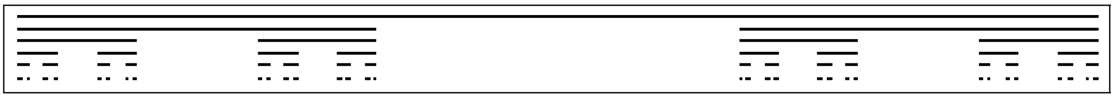

Figure 4: The first iterations for the production of the Cantor set.

2.4 Example Figure 3 depicts part of the iterative process leading to an attractor (starting from a square), in this case the so-called *Sierpinski triangle*. It is produced by an IFS consisting of the following three mappings on the space $(\mathbb{R}^2, d_2)$, where d_2 denotes the Euclidean metric on $\mathbb{R}^2$.

$$\omega_1\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} .5 & 0 \\ 0 & .5 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \end{pmatrix} \qquad \omega_2\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} .5 & 0 \\ 0 & .5 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} .5 \\ 0 \end{pmatrix} \qquad \omega_3\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} .5 & 0 \\ 0 & .5 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ .5 \end{pmatrix}$$

2.5 Example As a second example we give representations of Cantor space as compact subsets of the real numbers. The underlying space thus consists of the real numbers with the natural metric. As contractions, we choose

$$\omega_1 : \mathbb{R} \to \mathbb{R} : x \mapsto \frac{1}{B}x$$
$$\omega_2 : \mathbb{R} \to \mathbb{R} : x \mapsto \frac{1}{B}x + a,$$

where $B > 2$ is a positive integer and a is chosen such that the images of the unit interval under ω_1 and ω_2 do not have more than a single point in common, but are both contained in the unit interval. The corresponding iterates of the unit interval are depicted in Figure 4 for the values $B = 3$ and $a = \frac{2}{3}$. The subsets of the unit interval which can occur as attractors for different parameters are all homeomorphic, i.e. topologically equivalent, and also homeomorphic to the Cantor space and to $(I_\mathcal{P}, Q)$, if the Herbrand base $B_\mathcal{P}$ of the program $\mathcal{P}$ is countably infinite.

Some further examples of attractors of iterated function systems are depicted in Figure 5, defined on the real plane. The projections of the attractors to the x-coordinate are homeomorphic to the Cantor space.

3 Logic Programs as Iterated Function Systems

In this section we show how logic programs can be represented by iterated function systems. We will review an embedding introduced by Hölldobler et al. in [30], which can be used to embed the graph of the single-step operator into the real plane. Plots of these graphs exhibit self-similar structures, i.e. they look like attractors of iterated function systems. We will provide a way to transform logic programs into iterated function systems such that the graph of the program coincides with the attractor of the IFS, or can at least be approximated by it.

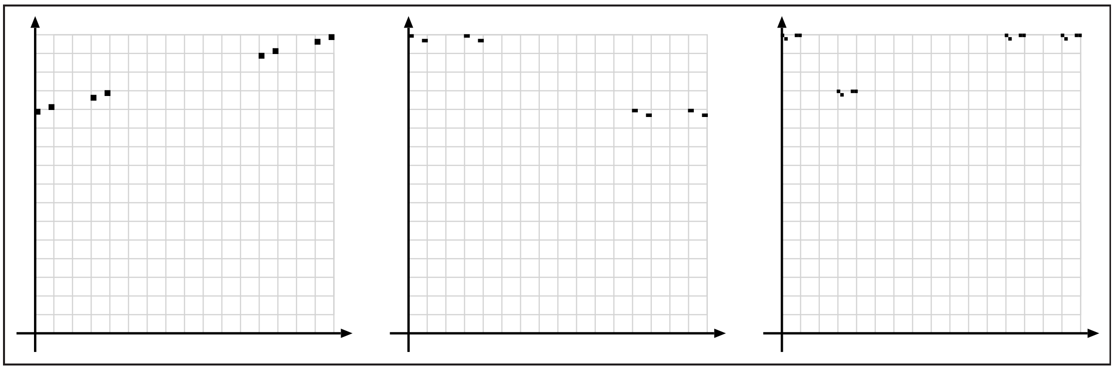

Figure 5: Some attractors of iterated function systems.

$$\begin{array}{ccc} I \in I_{\mathcal{P}} & \xrightarrow{T_{\mathcal{P}}} & I' \in I_{\mathcal{P}} \\ R\Big\downarrow\Big\uparrow R^{-1} & & R\Big\downarrow\Big\uparrow R^{-1} \\ i \in D_{\mathsf{f}} & \xrightarrow{f_{\mathcal{P}}} & i' \in D_{\mathsf{f}} \end{array}$$

Figure 6: The relation between $T_{\mathcal{P}}$ and $f_{\mathcal{P}}$.

3.1 Definition Let $\mathcal{P}$ be a logic program, $|\cdot| : B_{\mathcal{P}} \to \mathbb{N}$ be an injective level mapping and let $B \in \mathbb{N}$, with $B > 2$. Then the mapping R assigns a unique real number $R(I)$ to every interpretation $I \in I_{\mathcal{P}}$ by

$$R\, : I_{\mathcal{P}} \to \mathbb{R} : I \mapsto \sum_{A \in I} B^{-|A|}.$$

The range $\{R(I) \mid I \in B_{\mathcal{P}}\} \subseteq \mathbb{R}$ of the mapping R will be denoted by D_{f} and the maximal value, which always exists, by $R_{\mathsf{m}} = R(B_{\mathcal{P}}) = \lim_{n\to\infty} \sum_{i=1}^{n}(B^{-i}) = \frac{1}{B-1}$. Without loss of generality we will treat R as a (bijective) function from $I_{\mathcal{P}}$ to D_{f}.

The probably most obvious base $B = 2$ does not create a valid embedding: Let $B = 2$, and $\mathcal{P}$ and $|\cdot|$ be defined as in Figure 2. Let $I = \{n(0)\}$ and $J = B_{\mathcal{P}} \setminus \{n(0)\}$. It follows that $R(I) = B^{-1} = \frac{1}{2}$ and $R(J) = R_{\mathsf{m}} - B^{-1} = \frac{1}{B-1} - B^{-1} = \frac{1}{2}$, so the resulting function R is not injective. This is due to the fact that the numbers $0.111\ldots$ and $0.0111\ldots$ coincide in the number system with base 2. But for all $B > 2$ the mapping R is injective, if the level-mapping is injective. Furthermore, it can be shown that R is a homeomorphism (a bijective mapping which preserves topological structure in both directions) from $(I_{\mathcal{P}}, Q)$ to D_{f} and that D_{f} is compact.

By means of the mapping R we can now embed $T_{\mathcal{P}}$ into $\mathbb{R}$ as shown in Figure 6, i.e. for a given logic program $\mathcal{P}$ the function $R(T_{\mathcal{P}}) = f_{\mathcal{P}}$ is defined by

$$f_{\mathcal{P}} : D_{\mathsf{f}} \to D_{\mathsf{f}} : r \mapsto R(T_{\mathcal{P}}(R^{-1}(r))),$$

and its graph $F_{\mathcal{P}}$ is

$$F_{\mathcal{P}} = \{(R(I), R(T_{\mathcal{P}}(I))) \mid I \in I_{\mathcal{P}}\} = \{(x, f_{\mathcal{P}}(x)) \mid x \in D_{\mathsf{f}}\}.$$

Figure 7 shows some (embedded) graphs of logic programs. Note the similarity to the plots shown in Figure 5. Indeed we have noticed that all plots of graphs obtained by the method described above showed self-similar structures, thus appearing to be attractors of iterated function systems on the real plane.

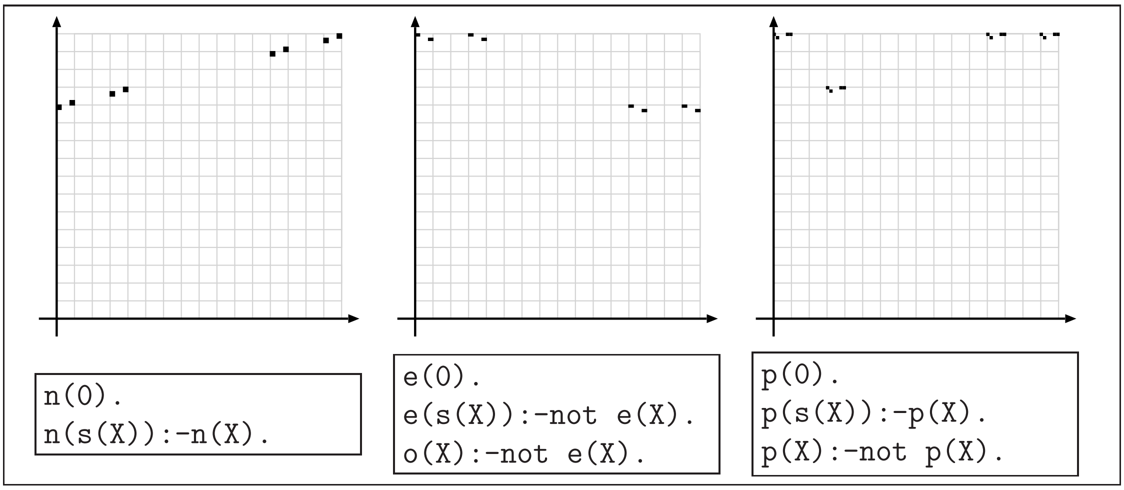

Figure 7: Some graphs of logic programs.

3.1 Representation of Logic Programs by Iterated Function Systems

We have just discussed that logic programs and iterated function systems create similar graphs. In this section we will link both by giving necessary and sufficient conditions under which the graph of a logic program is the attractor of a hyperbolic iterated function system. Since the iterated function systems shall approximate graphs in $\mathbb{R}^2$, they must be defined on that space. Therefore, we will focus on the space $(\mathbb{R}^2, d_2)$, where d_2 denotes the usual 2-dimensional Euclidean metric, i.e. $d_2((x_1, y_1), (x_2, y_2)) = \sqrt{|x_1 - x_2|^2 + |y_1 - y_2|^2}$, which is complete. For any function f on $\mathbb{R}^2$ we denote its coordinate functions by f^{x} and f^{y}, i.e. we have $f(a) = (f^{\mathsf{x}}(a), f^{\mathsf{y}}(a))$ for all $a \in \mathbb{R}^2$. Furthermore, let $\pi_{\mathsf{x}}(\cdot)$ denote the projection to the x-axis. The natural metric on $\mathbb{R}$ is denoted by d_1, i.e. $d_1(x, y) = |x - y|$ for all $x, y \in \mathbb{R}$.

The following theorem gives necessary and sufficient conditions for exact representability by an iterated function system.

3.2 Theorem (First Representation Theorem) Let $\mathcal{P}$ be a logic program, let $f_{\mathcal{P}}$ be the embedded $T_{\mathcal{P}}$-operator with graph $F_{\mathcal{P}}$, and let D_{f} be the range of the mapping R, as introduced earlier. Let $((\mathbb{R}^2, d_2), \Omega)$ be a (hyperbolic) iterated function system and let A be its uniquely determined attractor. Then the graph $F_{\mathcal{P}}$ coincides with the attractor A, i.e. $F_{\mathcal{P}} = A$, if and only if $\pi_{\mathsf{x}}(A) = D_{\mathsf{f}}$ and $f_{\mathcal{P}}(\omega_i^{\mathsf{x}}(a)) = \omega_i^{\mathsf{y}}(a)$ hold for all $a \in F_{\mathcal{P}}$ and all $\omega_i \in \Omega$.

Proof The proof is divided into two parts. First, we will show that $F_{\mathcal{P}} = A$ if $f_{\mathcal{P}}(\omega_i^{\mathsf{x}}(a)) = \omega_i^{\mathsf{y}}(a)$ and $\pi_{\mathsf{x}}(A) = D_{\mathsf{f}}$, and then the converse.

(i) To show the equivalence of $F_{\mathcal{P}}$ and A, we need to show that $F_{\mathcal{P}} \subseteq A$ and $A \subseteq F_{\mathcal{P}}$. **(i.a)** From $\Omega(A) = A$ and $\pi_{\mathsf{x}}(A) = \pi_{\mathsf{x}}(F_{\mathcal{P}}) = D_{\mathsf{f}}$ we know that for each $a \in F_{\mathcal{P}}$ there must be an $a' \in A$ and an $\omega_i \in \Omega$ such that $\pi_{\mathsf{x}}(a) = \omega_i^{\mathsf{x}}(a')$. Using $f_{\mathcal{P}}(\omega_i^{\mathsf{x}}(a')) = \omega_i^{\mathsf{y}}(a')$ and the definition of $f_{\mathcal{P}}$ we know that $\omega_i^{\mathsf{y}}(a') = f_{\mathcal{P}}(\pi_{\mathsf{x}}(a)) = \pi_{\mathsf{y}}(a)$. So we can conclude that $(\pi_{\mathsf{x}}(a), \pi_{\mathsf{y}}(a)) = (\omega_i^{\mathsf{x}}(a'), \omega_i^{\mathsf{y}}(a'))$, i.e. $a = \omega_i(a')$. Since $a' \in A$, hence $\omega_i(a') \in A$, it follows that $a \in A$ and finally $F_{\mathcal{P}} \subseteq A$. **(i.b)** From $f_{\mathcal{P}}(\omega_i^{\mathsf{x}}(a)) = \omega_i^{\mathsf{y}}(a)$ we can conclude that $(\omega_i^{\mathsf{x}}(a), \omega_i^{\mathsf{y}}(a)) \in F_{\mathcal{P}}$, i.e. $\omega_i(a) \in F_{\mathcal{P}}$. Knowing that this equation holds for all $a \in F_{\mathcal{P}}$ and all $\omega_i \in \Omega$ we obtain $\omega_i(F_{\mathcal{P}}) \subseteq F_{\mathcal{P}}$ and finally $\Omega(F_{\mathcal{P}}) \subseteq F_{\mathcal{P}}$. Hence $F_{\mathcal{P}} = A$.

(ii) Since $F_{\mathcal{P}} = A$ and $\pi_{\mathsf{x}}(F_{\mathcal{P}}) = D_{\mathsf{f}}$ we immediately obtain $\pi_{\mathsf{x}}(A) = D_{\mathsf{f}}$. Furthermore, we know that $F_{\mathcal{P}} = \Omega(F_{\mathcal{P}})$ and hence that $\Omega(a) \subseteq F_{\mathcal{P}}$ holds for all $a \in F_{\mathcal{P}}$. So we can conclude that for all $a \in F_{\mathcal{P}}$ and all $\omega_i \in \Omega$ there is an $a' \in F_{\mathcal{P}}$ such that $(\omega_i^{\mathsf{x}}(a), \omega_i^{\mathsf{y}}(a)) = a'$ holds. By the definition of $F_{\mathcal{P}}$we know that $a' = (x', f_{\mathcal{P}}(x'))$, hence $\omega_i^{\mathsf{x}}(a) = x'$ and $\omega_i^{\mathsf{y}}(a) = f_{\mathcal{P}}(x')$. If follows that $f_{\mathcal{P}}(\omega_i^{\mathsf{x}}(a)) = \omega_i^{\mathsf{y}}(a)$ holds for all $a \in F_{\mathcal{P}}$ and all $\omega_i \in \Omega$ if $F_{\mathcal{P}} = A$. □

The proof of Theorem 3.2 does not make use of the fact that the function $f_{\mathcal{P}}$ (the graph of which is represented by an IFS) comes from the single-step operator of a logic program. Indeed it holds for all functions defined on D_{f} and is easily generalized to functions on other compact subsets of the reals. In particular, we note that it does not restrict the class of programs covered. We will use Theorem 3.2 for establishing a stronger result for logic programs whose embedded single-step operator $R(T_P)$ is Lipschitz continuous. Before we do so, however, we need to have a closer look at the set D_{f}. So assume that a base B is fixed, thus the mapping R is determined and in turn also D_{f} as the range of R. It is our desire to characterize D_{f} as the attractor of an IFS. Now define

$$\Omega_1^{\mathsf{x}} = \left\{ x \mapsto \frac{1}{B}x + 0; x \mapsto \frac{1}{B}x + \frac{1}{B} \right\}.$$

For all $n > 1$ we define recursively

$$\begin{aligned} \Omega_n^{\mathsf{x}} &= \left\{ f \circ g \mid f \in \Omega_1^{\mathsf{x}} \text{ and } g \in \Omega_{n-1}^{\mathsf{x}} \right\} \\ &= \left\{ x \mapsto \frac{1}{B}\omega(x) + 0 \mid \omega \in \Omega_{n-1}^{\mathsf{x}} \right\} \cup \left\{ x \mapsto \frac{1}{B}\omega(x) + \frac{1}{B} \mid \omega \in \Omega_{n-1}^{\mathsf{x}} \right\}. \end{aligned}$$

Note that every mapping $\omega_i^{\mathsf{x}} \in \Omega_P^{\mathsf{x}}$ is of the form $\omega_i^{\mathsf{x}} = \frac{1}{B^P}x + d_i^{\mathsf{x}}$, where d_i^{x} depends on the application of either the first or second mapping from Ω_1^{x} during the construction, i.e. d_i^{x} can be written as $\sum_{j=1}^{P} a_j \cdot B^{-j}$, where $a_j \in \{0, 1\}$. In particular we have that for each d_i^{x} there exists an interpretation I_i with $R(I_i) = d_i^{\mathsf{x}}$. More precisely, I_i consists of all those atoms A with $|A| \leq P$ such that $d_i^{\mathsf{x}} = \sum_{A \in I_i} B^{-|A|}$, and by injectivity of $|\cdot|$ the interpretation I_i is indeed uniquely determined by this equation.

The proof of the following lemma is straightforward.

3.3 Lemma For any $P \geq 1$ we have that D_{f} is the attractor of the IFS $((\mathbb{R}, d), \Omega_P^{\mathsf{x}})$.

We are now ready to establish the promised second representation result. Even though it does not define a convenient way to construct an IFS, it explains why the plotted graphs of the programs are self-similar.

3.4 Theorem (Second Representation Theorem) Let $\mathcal{P}$ be a logic program. Let $f_{\mathcal{P}}$ be the embedded $T_{\mathcal{P}}$-operator using base $B > 2$, and let $F_{\mathcal{P}}$ be its graph. Furthermore assume that $f_{\mathcal{P}}$ is Lipschitz continuous. Then there exists an IFS on $(\mathbb{R}^2, d_2)$ with attractor $F_{\mathcal{P}}$.

Proof We prove this theorem by applying Theorem 3.2, i.e. we will show that under the stated hypotheses there is a hyperbolic IFS $((\mathbb{R}^2, d_2), \Omega)$ such that $\pi_{\mathsf{x}}(A) = D_{\mathsf{f}}$ and $f_{\mathcal{P}}(\omega_i^{\mathsf{x}}(a)) = \omega_i^{\mathsf{y}}(a)$ hold for all $a \in F_{\mathcal{P}}$ and all $\omega_i \in \Omega$.

By Lemma 3.3 we know that for each $P \geq 1$ there is a set Ω_P^{x} consisting of contractive mappings $\omega_i^{\mathsf{x}} : \mathbb{R} \to \mathbb{R}$ with contractivity factor $\frac{1}{B^P}$ and such that D_{f} is the attractor of the IFS $((\mathbb{R}, d_1), \Omega_P^{\mathsf{x}})$. For every $\omega_i^{\mathsf{x}} \in \Omega_P^{\mathsf{x}}$ we now define a mapping $\omega_i^{\mathsf{y}} : \mathbb{R} \to \mathbb{R}$ by $\omega_i^{\mathsf{y}}(x) = f_{\mathcal{P}}(\omega_i^{\mathsf{x}}(x))$. It remains to show that $((\mathbb{R}^2, d_2), \Omega)$ with $\Omega = \{(\omega_i^{\mathsf{x}} \circ \pi_{\mathsf{x}}, \omega_i^{\mathsf{y}} \circ \pi_{\mathsf{x}}) \mid \omega_i^{\mathsf{x}} \in \Omega_P^{\mathsf{x}}\}$ is a hyperbolic IFS for some suitably chosen $P \geq 1$, and for this it suffices to show that every $\omega_i = (\omega_i^{\mathsf{x}}, \omega_i^{\mathsf{y}}) \in \Omega$ is a contraction on $(\mathbb{R}^2, d_2)$.

Since $f_{\mathcal{P}}$ is Lipschitz continuous, there is a constant L with $d_1(f_{\mathcal{P}}(x), f_{\mathcal{P}}(y)) \leq L \cdot d_1(x, y)$ for all $x, y \in D_{\mathsf{f}}$. Taking this and the contractivity of ω_i^{x} into account we obtain for all $a, b \in \mathbb{R}^2$

$$\begin{aligned} d_2(\omega_i(a), \omega_i(b))^2 &= d_1(\omega_i^{\mathsf{x}}(\pi_{\mathsf{x}}(a)), \omega_i^{\mathsf{x}}(\pi_{\mathsf{x}}(b)))^2 + d_1(\omega_i^{\mathsf{y}}(\pi_{\mathsf{x}}(a)), \omega_i^{\mathsf{y}}(\pi_{\mathsf{x}}(b)))^2 \\ &\leq B^{-2P} \cdot |\pi_{\mathsf{x}}(a) - \pi_{\mathsf{x}}(b)|^2 + L^2 B^{-2P} |\pi_{\mathsf{x}}(a) - \pi_{\mathsf{x}}(b)|^2 \\ &\leq \frac{L^2 + 1}{B^{2P}} \cdot \left|\pi_{\mathsf{x}}(a) - \pi_{\mathsf{x}}(b)\right|^2. \end{aligned}$$

Since π_{x} is continuous with Lipschitz constant 1 we obtain

$$d_2(\omega_i(a), \omega_i(b)) \leq \sqrt{\frac{L^2 + 1}{B^{2P}}} \cdot d_2(a, b).$$

We see now that it is possible to choose P such that ω_i is a contraction, and Theorem 3.2 is applicable. □

Before we move on, let us dwell a bit on the implications of Theorem 3.4 and also on some questions it raises. We require $f_{\mathcal{P}}$ to be Lipschitz continuous, which implies that $f_{\mathcal{P}}$ is continuous on the Cantor set as a subspace of $\mathbb{R}$, and hence that $T_{\mathcal{P}}$ is continuous with respect to the Cantor topology Q on $I_{\mathcal{P}}$. The latter notion is well-understood (see [21, 41]). For example, it turns out that programs without local variables (called *covered programs* in [8]) have continuous single-step operators, where a local variable is a variable which occurs in some body literal of a program clause but not in its corresponding head.

The exact relationships between covered programs, continuity of the single-step operator in Q, Lipschitz continuity with respect to a metric generating Q, and Lipschitz continuity of the embeded single-step operator with respect to the natural metric on $\mathbb{R}$ remain to be studied, and these matters appear to be not straightforward. What we can say at this stage is that if $T_{\mathcal{P}}$ is continuous in Q then $f_{\mathcal{P}}$ is continuous on the Cantor space (because the latter is homeomorphic to $(I_{\mathcal{P}}, Q)$), and since the Cantor space is compact, we obtain that $f_{\mathcal{P}}$ must be uniformly continuous, which is stronger than continuity, but strictly weaker than Lipschitz continuity. The interested reader will also be able to verify that the single-step operator of the covered program

```
p(X) :- p(f(X,X))
```

is not Lipschitz continuous with respect to any metric based on an injective level mapping as in Definition 2.1. We owe this example to Howard Blair.

Programs which are acyclic with respect to an injective level mapping also have continuous single-step operators, which is easily seen by observing that such programs cannot contain any local variables — or by considering the remark made earlier that for such programs the single-step operator is a contraction with respect to a metric which generates Q. Furthermore, it was shown in [30] that for base $B = 4$ (and hence for all larger bases) for the embedding R, the resulting embedded function $f_{\mathcal{P}} = R(T_{\mathcal{P}})$ is a contraction on a subset of $\mathbb{R}$, hence is Lipschitz continuous. If $\mathcal{P}$ is a program for which ground($\mathcal{P}$) is finite (and hence $B_\mathcal{P}$ is finite), then D_f is a finite subset of $\mathbb{R}$, and hence $f_\mathcal{P}$ is trivially Lipschitz continuous as a function on a subset of $\mathbb{R}$. We can thus state the following corollary.

3.5 Corollary For programs which are acyclic with respect to an injective level mapping, and for programs for which ground($\mathcal{P}$) is finite, there exists an IFS in the form given in the proof of Theorem 3.4 with attractor $F_\mathcal{P}$.

Corollary 3.5 gives a formal, albeit not satisfactory, explanation for the observation which started our investigations: In order to obtain approximate plots of the graph of some $f_\mathcal{P}$, we restricted ourselves to plotting the graph corresponding to a *finite*, though large, subprogram of ground($\mathcal{P}$).

As yet, we know of no general method for obtaining Lipschitz constants of $f_\mathcal{P}$, or even for showing whether it is Lipschitz continuous at all. In the light of Theorem 3.4 and other results which we will discuss in the sequel, and also by considering our remarks made earlier on the unclear relations between different notions of continuity for single-step operators, we feel that investigations into these matter will have to be made in order to obtain satisfactory constructions of iterated function systems — or of connectionist systems — for representing logic programs in our approach.

3.2 Worked Examples

Although Theorem 3.4 covers a wide range of programs, it is unsatisfactory since it does not provide a convenient way of constructing the iterated function system. Indeed, the IFS obtained in the proof of the theorem does involve the single-step operator for the calculation of the functions ω_i^y. In this section, we provide a simple but reasonable form of iterated function system which avoids this drawback, and show in detail that it covers some example programs. The same form of IFS will also be used in later parts of the paper.

3.6 Definition Let the natural numbers $B > 2$ (hence also the mapping R) and $P \geq 1$ be fixed, and let $\mathcal{P}$ be a program. Then we associate with $\mathcal{P}$ the IFS $((\mathbb{R}^2, d_2), \Omega)$, where the $\omega_i \in \Omega$ are defined as $\omega_i : \mathbb{R}^2 \to \mathbb{R}^2 : (x, y) \mapsto (\omega_i^\mathsf{x}(x), \omega_i^\mathsf{y}(y))$ with ω_i^x and ω_i^y being

$$\begin{aligned}\omega_i^\mathsf{x} &: \mathbb{R} \to \mathbb{R} : x \mapsto \frac{1}{B^P}x + d_i^\mathsf{x}, \qquad \text{and}\\ \omega_i^\mathsf{y} &: \mathbb{R} \to \mathbb{R} : y \mapsto \frac{1}{B^P}y + f_\mathcal{P}(d_i^\mathsf{x}) - \frac{f_\mathcal{P}(0)}{B^P}.\end{aligned}$$

The parameter i, in this case, ranges from 1 to 2^P, and the ω_i^x and $d_i^\mathsf{x} \in D_\mathsf{f}$ are exactly as in the IFS $((\mathbb{R}, d), \Omega_\mathcal{P}^\mathsf{x})$ from Lemma 3.3. For convenience, we call such a resulting IFS *linear* and use the notation $\text{IFS}_\mathcal{P}^l$ when we are referring to it. Note that whenever B, P, and $\mathcal{P}$ are fixed, then the corresponding $\text{IFS}_\mathcal{P}^l$ is uniquely determined, so that our notation is sound.

We consider the base B fixed in the sequel. The parameter P, which we call *periodicity*, will usually depend on the program $\mathcal{P}$. How to construct an $\text{IFS}_\mathcal{P}^l$ from a fixed B is also depicted in Figure 8.

Algorithm 3.7 (Construction of $\text{IFS}^l_{\mathcal{P}}$ for a given program $\mathcal{P}$)
Let $\mathcal{P}$ be a logic program and $f_{\mathcal{P}}$ be its embedded $T_{\mathcal{P}}$-operator.

1. Choose a natural number (the *periodicity*) $P > 0$.
2. Compute $\Omega^{\mathsf{x}}_{\mathcal{P}}$ as explained in Section 3.1.
3. Construct for each $\omega^{\mathsf{x}}_i \in \Omega^{\mathsf{x}}_{\mathcal{P}}$ the corresponding $\omega^{\mathsf{y}}_i : y \mapsto \frac{1}{B^P} y + f_{\mathcal{P}}(d^{\mathsf{x}}_i) - \frac{f_{\mathcal{P}}(0)}{B^P}$.
4. Return the set $\Omega = \{\omega_i = (\omega^{\mathsf{x}}_i, \omega^{\mathsf{y}}_i)\}$ as mappings for the $\text{IFS}^l_{\mathcal{P}} = \{(\mathbb{R}^2, d_2), \Omega\}$.

Figure 8: Constructing linear iterated function systems.

Before we explain the intuition behind Definition 3.6 we need to introduce a new operator denoted $\dot{\div}$, which takes as arguments an interpretation and a natural number, and returns an interpretation. This operator defines a kind of shift operation on interpretations.

3.8 Definition Let $\mathcal{P}$ be a logic program, $I \in I_{\mathcal{P}}$, $P \in \mathbb{N}$, and $|\cdot|$ be an injective level mapping for $\mathcal{P}$. Then define

$$\tfrac{I}{\dot{}P} = \{A \mid \text{ there exists } A' \in I \text{ with } |A'| + P = |A|\}.$$

We call $\dot{\div}$ the *right-shift operator*.

3.9 Proposition $\frac{I}{\dot{}P} = R^{-1}\left(\frac{R(I)}{B^P}\right)$ holds for all $I \in I_{\mathcal{P}}$ and all $P \in \mathbb{N}$.

Proof The equation follows immediately from the definition of R since

$$\tfrac{I}{\dot{}P} = R^{-1}\left(\sum_{A \in I} B^{-(|A|+P)}\right) = R^{-1}\left(\frac{\sum_{A\in I} B^{-|A|}}{B^P}\right) = R^{-1}\left(\frac{R(I)}{B^P}\right).$$

□

We have already observed in Section 3.1 that for each d^{x}_i occuring in Definition 3.6 there exists some $I_i \in I_{\mathcal{P}}$ with $R(I_i) = d^{\mathsf{x}}_i$. Using Proposition 3.9 we can therefore carry over the functions ω^{x}_i to $I_{\mathcal{P}}$, as follows.

$$\begin{aligned} \omega^{\mathsf{x}}_i : \mathbb{R} \to \mathbb{R} : x &\mapsto \frac{x}{B^P} + d^{\mathsf{x}}_i \\ R^{-1}(\omega^{\mathsf{x}}_i) = \mathsf{w}^{\mathsf{x}}_i : I_{\mathcal{P}} \to I_{\mathcal{P}} : I &\mapsto \tfrac{I}{\dot{}P} \cup I_i \end{aligned}$$

For the mappings ω^{y}_i the resulting function is a bit more involved, and can be represented as

$$\begin{aligned} \omega^{\mathsf{y}}_i : \mathbb{R} \to \mathbb{R} : y &\mapsto \frac{y}{B^P} - \frac{f_{\mathcal{P}}(0)}{B^P} + f_{\mathcal{P}}(d^{\mathsf{x}}_i) \\ R^{-1}(\omega^{\mathsf{y}}_i) = \mathsf{w}^{\mathsf{y}}_i : I_{\mathcal{P}} \to I_{\mathcal{P}} : I &\mapsto \left(\tfrac{I}{\dot{}P} \setminus I^-_i\right) \cup I^+_i \end{aligned}$$

where $I^+_i = R^{-1}(f_{\mathcal{P}}(d^{\mathsf{x}}_i)) = T_{\mathcal{P}}(I_i)$ and $I^-_i = R^{-1}\left(\frac{f_{\mathcal{P}}(0)}{B^P}\right) = \frac{T_{\mathcal{P}}(\emptyset)}{\dot{}P}$.

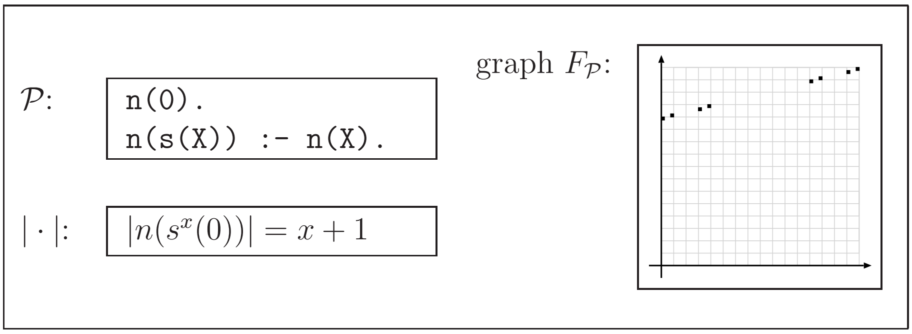

Figure 9: The natural numbers program and their embedded $T_{\mathcal{P}}$-operator.

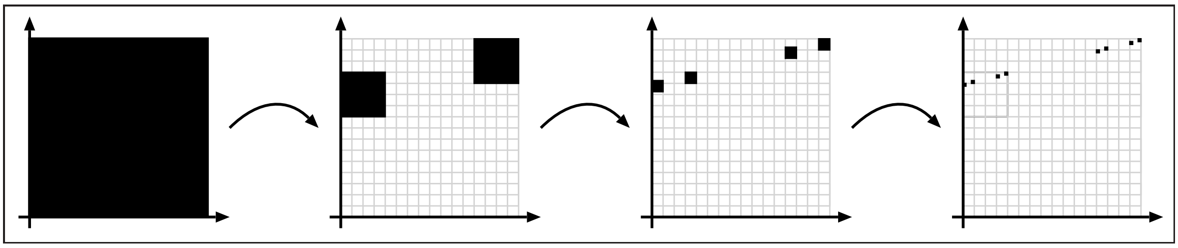

Figure 10: The first three iterations of the mappings.

Let us now explain the intuition behind the definitions of the mappings ω_i^{x} and ω_i^{y}. The choice of the ω_i^{x} is obvious for the same reasons as in Section 3.1 and in the proof of Theorem 3.4: It appears to be the most natural way to obtain D_{f} as projection of the resulting attractor to the x-axis, as required by Theorem 3.2. The corresponding mapping $\mathsf{w}_i^{\mathsf{x}}$ unmasks this as a right-shift with addition of a base point I_i. A first approximate candidate for $\mathsf{w}_i^{\mathsf{y}}(I)$ would therefore be $\frac{I}{P} \cup T_{\mathcal{P}}(I_i)$ — note that I in this case should be understood as being some image under $T_{\mathcal{P}}$. The occurrence of I_i^- is necessary as a correction in case of an overlap (i.e. a non-empty intersection) between $\frac{I}{P}$ and $T_{\mathcal{P}}(I_i)$. This would not be necessary, strictly speaking, for $\mathsf{w}_i^{\mathsf{y}}$, where such an overlap would have no effect since it is ignored by the set-union operation. When carried over to the reals, however, this correction becomes necessary in order to avoid the situation that the resulting number would not correspond to an interpretation.

Linear iterated function systems are constructed such that $\pi_{\mathsf{x}}(A) = D_{\mathsf{f}}$, which is one of the conditions imposed by Theorem 3.2. The other condition, $f_{\mathcal{P}}(\omega_i^{\mathsf{x}}(a)) = \omega_i^{\mathsf{y}}(a)$ for all a, will be shown on a case base in the following examples. We fix $B = 4$ for the examples, in order to have a concrete setting. This choice was also made in [30], and the reason for this was to guarantee that $f_{\mathcal{P}}$ is a contraction for acyclic programs with injective level mappings, as already mentioned.

Consider first the program from Figure 1. Figure 9 shows an associated graph with corresponding level mapping — we use the notation $s^x(0)$ to denote the term $s(\ldots(0)\ldots)$ in which the symbol s occurs x times. We now use Algorithm 3.7 for constructing an $\mathrm{IFS}_{\mathcal{P}}^{l}$ for the program. We choose periodicity $P = 1$ and obtain

$$\Omega = \left\{ \begin{pmatrix} \frac{1}{4} & 0 \\ 0 & \frac{1}{4} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{3}{16} \end{pmatrix}, \begin{pmatrix} \frac{1}{4} & 0 \\ 0 & \frac{1}{4} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{4} \\ \frac{1}{4} \end{pmatrix} \right\}.$$

The first three iterations of this $\mathrm{IFS}_{\mathcal{P}}^{l}$ are depicted in Figure 10.

$\omega_1^{\mathsf{x}}(x) = \frac{x}{4} + 0$	$\omega_1^{\mathsf{y}}(x) = \frac{x}{4} + \frac{3}{16}$
$\mathsf{w}_1^{\mathsf{x}}(I) = \overset{I}{\underset{1}{\rightarrow}}$	$\mathsf{w}_1^{\mathsf{y}}(I) = \left(\overset{I}{\underset{1}{\rightarrow}} \setminus \{n(s(0))\}\right) \cup \{n(0)\}$
$T_{\mathcal{P}}(\mathsf{w}_1^{x}(I)) = T_{\mathcal{P}}\left(\overset{I}{\underset{1}{\rightarrow}}\right) = \left(\overset{T_{\mathcal{P}}(I)}{\underset{1}{\rightarrow}} \setminus \{n(s(0))\}\right) \cup \{n(0)\} = \mathsf{w}_1^{\mathsf{y}}(T_{\mathcal{P}}(I))$	
$\omega_2^{\mathsf{x}}(x) = \frac{x}{4} + \frac{1}{4}$	$\omega_2^{\mathsf{y}}(x) = \frac{x}{4} + \frac{1}{4}$
$\mathsf{w}_2^{\mathsf{x}}(I) = \overset{I}{\underset{1}{\rightarrow}} \cup \{n(0)\}$	$\mathsf{w}_2^{\mathsf{y}}(I) = \overset{I}{\underset{1}{\rightarrow}} \cup \{n(0)\}$
$T_{\mathcal{P}}(\mathsf{w}_2^{x}(I)) = T_{\mathcal{P}}\left(\overset{I}{\underset{1}{\rightarrow}} \cup \{n(0)\}\right) = \overset{T_{\mathcal{P}}(I)}{\underset{1}{\rightarrow}} \cup \{n(0)\} = \mathsf{w}_2^{\mathsf{y}}(T_{\mathcal{P}}(I))$	

Table 1: $T_{\mathcal{P}}(\mathsf{w}_i^{\mathsf{x}}(I)) = \mathsf{w}_i^{\mathsf{y}}(T_{\mathcal{P}}(I))$ holds for the natural numbers program.

$\mathcal{P}$:

```
e(0).
e(s(X)) :- not e(X).
o(X) :- not e(X).
```

$|\cdot|$:

$|e(s^x(0))| = 2 \cdot x + 1$
$|o(s^x(0))| = 2 \cdot x + 2$

graph $F_{\mathcal{P}}$:

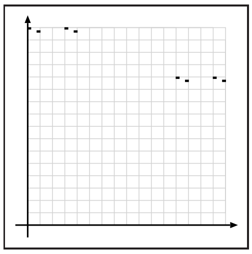

Figure 11: The even and odd numbers program.

In order to show that the resulting attractor coincides with $F_{\mathcal{P}}$, we need to verify the hypotheses of Theorem 3.2, i.e. in particular, we need to show that $f_{\mathcal{P}}(\omega_i^{\mathsf{x}}(a)) = \omega_i^{\mathsf{y}}(a)$ for all $a \in F_{\mathcal{P}}$. By the discussion following Proposition 3.9 it therefore suffices to show that $T_{\mathcal{P}}(\mathsf{w}_i^{\mathsf{x}}(I)) = \mathsf{w}_i^{\mathsf{y}}(T_{\mathcal{P}}(I))$ holds for all $I \in I_{\mathcal{P}}$. The necessary calculations are shown in Table 1, some details are straightforward and have been omitted.

As another example we discuss the program from Figure 11. We work with periodicity $P = 2$ and obtain the following $\mathrm{IFS}^l_{\mathcal{P}}$ by Algorithm 3.7.

$$\Omega = \left\{ \begin{pmatrix} \frac{1}{16} & 0 \\ 0 & \frac{1}{16} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{5}{16} \end{pmatrix}, \begin{pmatrix} \frac{1}{16} & 0 \\ 0 & \frac{1}{16} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{16} \\ \frac{5}{16} \end{pmatrix}, \right.$$
$$\left. \begin{pmatrix} \frac{1}{16} & 0 \\ 0 & \frac{1}{16} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{4} \\ \frac{15}{64} \end{pmatrix}, \begin{pmatrix} \frac{1}{16} & 0 \\ 0 & \frac{1}{16} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{5}{16} \\ \frac{15}{64} \end{pmatrix} \right\}$$

The first few iterations of the resulting $\mathrm{IFS}^l_{\mathcal{P}}$ are depicted in Figure 12. Verification of correctness is performed similarly as for the natural numbers program and details are given in Table 2.

4 Logic Programs by Fractal Interpolation

In Section 3 we have focused on the problem of exact representation of logic programs by iterated function systems. In this section we will provide a result for approximating logic programs by iterated function systems. Our approach is motivated by fractal interpolation

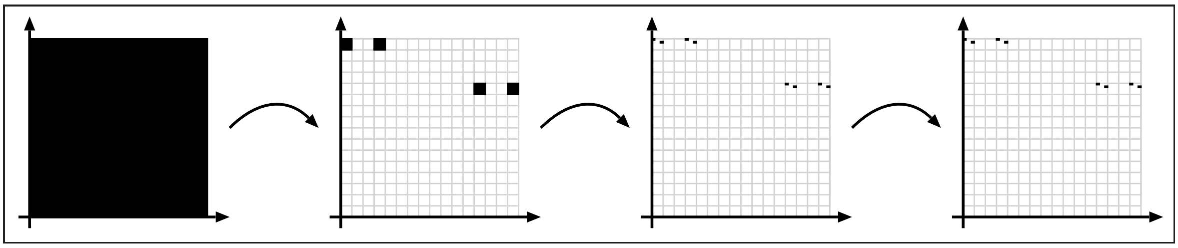

Figure 12: The first three iterations of the mappings.

$$\omega_1^{\mathsf{x}}(x) = \tfrac{x}{16} + 0 \qquad \omega_1^{\mathsf{y}}(x) = \tfrac{x}{16} + \tfrac{5}{16}$$

$$\mathsf{w}_1^{\mathsf{x}}(I) = \tfrac{I}{2} \qquad \mathsf{w}_1^{\mathsf{y}}(I) = \tfrac{I}{2} \cup \{e(0), o(0)\}$$

$$T_{\mathcal{P}}(\mathsf{w}_1^x(I)) = T_{\mathcal{P}}\left(\tfrac{I}{2}\right) = \tfrac{T_{\mathcal{P}}(I)}{2} \cup \{e(0), o(0)\} = \mathsf{w}_1^{\mathsf{y}}(T_{\mathcal{P}}(I))$$

$$\omega_2^{\mathsf{x}}(x) = \tfrac{x}{16} + \tfrac{1}{16} \qquad \omega_2^{\mathsf{y}}(x) = \tfrac{x}{16} + \tfrac{5}{16}$$

$$\mathsf{w}_2^{\mathsf{x}}(I) = \tfrac{I}{2} \cup \{o(0)\} \qquad \mathsf{w}_2^{\mathsf{y}}(I) = \tfrac{I}{2} \cup \{e(0), o(0)\}$$

$$T_{\mathcal{P}}(\mathsf{w}_2^x(I)) = T_{\mathcal{P}}\left(\tfrac{I}{2} \cup \{o(0)\}\right) = \tfrac{T_{\mathcal{P}}(I)}{2} \cup \{e(0), o(0)\} = \mathsf{w}_2^{\mathsf{y}}(T_{\mathcal{P}}(I))$$

$$\omega_3^{\mathsf{x}}(x) = \tfrac{x}{16} + \tfrac{1}{4} \qquad \omega_3^{\mathsf{y}}(x) = \tfrac{x}{16} + \tfrac{15}{64}$$

$$\mathsf{w}_3^{\mathsf{x}}(I) = \tfrac{I}{2} \cup \{e(0)\} \qquad \mathsf{w}_3^{\mathsf{y}}(I) = \left(\tfrac{I}{2} \setminus \{e(s(0))\}\right) \cup \{e(0)\}$$

$$T_{\mathcal{P}}(\mathsf{w}_3^x(I)) = T_{\mathcal{P}}\left(\tfrac{I}{2} \cup \{e(0)\}\right) = \left(\tfrac{T_{\mathcal{P}}(I)}{2} \setminus \{e(s(0))\}\right) \cup \{e(0)\} = \mathsf{w}_3^{\mathsf{y}}(T_{\mathcal{P}}(I))$$

$$\omega_4^{\mathsf{x}}(x) = \tfrac{x}{16} + \tfrac{5}{16} \qquad \omega_4^{\mathsf{y}}(x) = \tfrac{x}{16} + \tfrac{15}{64}$$

$$\mathsf{w}_4^{\mathsf{x}}(I) = \tfrac{I}{2} \cup \{e(0), o(0)\} \qquad \mathsf{w}_4^{\mathsf{y}}(I) = \left(\tfrac{I}{2} \setminus \{e(s(0))\}\right) \cup \{e(0)\}$$

$$T_{\mathcal{P}}(\mathsf{w}_4^x(I)) = T_{\mathcal{P}}\left(\tfrac{I}{2} \cup \{e(0), o(0)\}\right) = \left(\tfrac{T_{\mathcal{P}}(I)}{2} \setminus \{e(s(0))\}\right) \cup \{e(0)\} = \mathsf{w}_4^{\mathsf{y}}(T_{\mathcal{P}}(I))$$

Table 2: $T_{\mathcal{P}}(\mathsf{w}_i^{\mathsf{x}}(I)) = \mathsf{w}_i^{\mathsf{y}}(T_{\mathcal{P}}(I))$ holds for the even and odd numbers program.

as described in [3, Chapter 6], but our setting differs in that we reuse the linear iterated function systems introduced in Definition 3.6.

We will again assume the parameter $B > 2$ and some injective level mapping to be fixed. The parameter P is going to be reinterpreted as *accuracy*. Given a logic program $\mathcal{P}$, for which $f_{\mathcal{P}}$ is Lipschitz continuous, and given some accuracy P, we consider the associated iterated function system as given by Definition 3.6. It will be shown that the attractor of each of these systems is the graph of a continuous function defined on D_{f}, and that the sequence of attractors associated with an increasing sequence of accuracies converges to the graph of $f_{\mathcal{P}}$, with respect to the maximum metric on the space of continuous functions.

We begin by describing in detail the fractal interpolation systems which we will be using. Given a program $\mathcal{P}$ we need to extract a set of interpolation data which we can use for the interpolation process. This procedure — for each accuracy P — is described in Figure 13. Note that the data pairs $(R(X_i), R(Y_i))$ obtained in this way coincide with the values $(d_i^{\mathsf{x}}, f_{\mathcal{P}}(d_i^{\mathsf{x}}))$ used in Section 3.1.

4.2 Definition (IIFSP) Let $\{(d_i^{\mathsf{x}}, d_i^{\mathsf{y}}) \mid 1 \le i \le 2^P\}$ be a sequence of interpolation data

Algorithm 4.1 (Interpolation Data)
This algorithm computes a set of interpolation data for a given program $\mathcal{P}$.

1. Choose a natural number (the *accuracy*) $P > 0$.
2. Compute the set $D = \{A \mid |A| \leq P \text{ and } A \in B_{\mathcal{P}}\}$ and its powerset $\mathcal{D} = \mathcal{P}(D)$.
3. For any set $X_i \in \mathcal{D}$ compute $Y_i = T_{\mathcal{P}}(X_i)$ with respect to the program $\mathcal{P}$.
4. Return the sequence of pairs $(R(X_i), R(Y_i))$, with $R(X_i) < R(X_j)$ for all $i < j$.

Figure 13: Construction of Interpolation Data.

$\mathcal{P}$:

```
p(0).
p(s(X)) :- p(X).
p(X) :- not p(X).
```

$|\cdot|$:

$|p(s^x(0))| = x + 1$

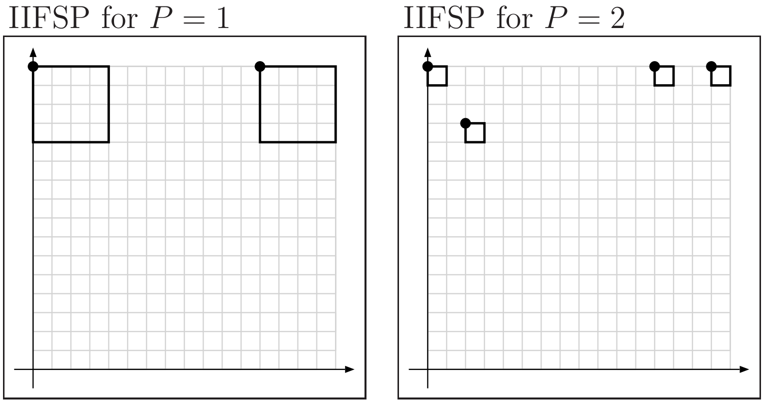

Figure 14: A logic program and two corresponding IIFSPs.

constructed via Algorithm 4.1 from the program $\mathcal{P}$ using accuracy P. Let $f_{\mathcal{P}}$ be the embedded $T_{\mathcal{P}}$-operator associated with the program $\mathcal{P}$ using the mapping R with base B. Then $((\mathbb{R}^2, d_2), \Omega)$ is called an *interpolating iterated function system (IIFSP)*, with $\Omega = \left\{\omega_i \mid 1 \leq i \leq 2^P\right\}$ and $\omega_i : \mathbb{R}^2 \to \mathbb{R}^2 : (x, y) \mapsto (\omega_i^{\mathsf{x}}(x), \omega_i^{\mathsf{y}}(y))$, where ω_i^{x} and ω_i^{y} are defined by

$$\begin{aligned} \omega_i^{\mathsf{x}}(x) &= \frac{1}{B^P} x + d_i^{\mathsf{x}}, \qquad \text{and} \\ \omega_i^{\mathsf{y}}(y) &= \frac{1}{B^P} y + f_{\mathcal{P}}(d_i^{\mathsf{x}}) - \frac{f_{\mathcal{P}}(0)}{B^P}. \end{aligned}$$

Figure 14 shows a logic program and schematically two corresponding interpolating iterated function systems for $B = 4$.

Each IIFSP constructed for the program $\mathcal{P}$ and accuracy P corresponds to a linear iterated function system constructed for the periodicity P as in Algorithm 3.7. Therefore it is obvious that the resulting mappings indeed constitute hyperbolic iterated function systems which satisfy $\pi_{\mathsf{x}}(A) = D_{\mathsf{f}}$ for their attractors A.

For the remainder of this section we denote by $\mathcal{F}$ the set of all continuous functions from D_{f} to $\mathbb{R}$, and by d_{f} the maximum metric $d_{\mathsf{f}}(f, g) = \max_{x \in D_{\mathsf{f}}}\{|f(x) - g(x)|\}$ on this set. Thus $(\mathcal{F}, d_{\mathsf{f}})$ is a complete metric space, and convergence with respect to it is uniform convergence.

4.3 Lemma The function $T_P : \mathcal{F} \to \mathcal{F}$ defined by $T_P(f)(x) = \omega_i^{\mathsf{y}} \circ f \circ \omega_i^{\mathsf{x}^{-1}}(x)$, where i is chosen appropriately depending on x, is a contraction on $(\mathcal{F}, d_{\mathsf{f}})$ with contractivity factor $\frac{1}{B^P}$.

Proof The function $T_P f$ can be characterized by cases depending on the input x, by setting

$$T_P f(x) = T_{P,i} f(x) \qquad \text{for } x \in \left[d_i^{\mathsf{x}}, d_i^{\mathsf{x}} + \frac{R_{\mathsf{m}}}{B^P}\right] \cap D_{\mathsf{f}}$$

with each $T_{P,i} f$ defined as

$$(T_{P,i} f)(x) = \omega_i^{\mathsf{y}}\left(f\left(\omega_i^{\mathsf{x}^{-1}}(x)\right)\right) = \frac{1}{B^P} \cdot f\left((x - d_i^{\mathsf{x}}) \cdot B^P\right) + f_{\mathcal{P}}(d_i^{\mathsf{x}}) - \frac{f_{\mathcal{P}}(0)}{B^P}.$$

In the sequel we will simply write $T_P f : D_{\mathsf{f}} \to \mathbb{R} : x \mapsto \omega_i^{\mathsf{y}}\left(f\left(\omega_i^{\mathsf{x}^{-1}}(x)\right)\right)$ since $T_P f$ is a well-defined function from D_{f} to $\mathbb{R}$.

To show that T_P maps $\mathcal{F}$ to itself, we need to show that $T_P f : D_{\mathsf{f}} \to \mathbb{R}$ is a continuous function for all $f \in \mathcal{F}$. The continuity of each $T_{P,i} f$ is obvious, since it is a composition of continuous functions. Since $d_i^{\mathsf{x}} + \frac{R_{\mathsf{m}}}{B^P} < d_{i+1}^{\mathsf{x}}$ for each $i < 2^P$ this observation suffices.

Contractivity of T_P follows immediately from the definition since

$$\begin{aligned} d_{\mathsf{f}}(T_P f, T_P g) &= \max\{|T_P f(x) - T_P g(x)| \mid x \in D_{\mathsf{f}}\} \\ &= \frac{1}{B^P} \cdot \max\left\{\left|f\left((x - d_i^{\mathsf{x}}) \cdot B^P\right) - g\left((x - d_i^{\mathsf{x}}) \cdot B^P\right)\right| \mid x \in D_{\mathsf{f}}\right\} \\ &\leq \frac{1}{B^P} \cdot d_{\mathsf{f}}(f, g), \end{aligned}$$

and we can conclude that T_P is a contraction with contractivity factor $\frac{1}{B^P}$. $\square$

4.4 Lemma Let $D = \{(d_i^{\mathsf{x}}, d_i^{\mathsf{y}})\}$ be a sequence of interpolation data and $((\mathbb{R}^2, d_2), \Omega)$ be an interpolating iterated function system with attractor A, as constructed in Definition 4.2 from the program $\mathcal{P}$ using accuracy P. Let $f_{\mathcal{P}}$ be the embedded $T_{\mathcal{P}}$-operator associated with the program $\mathcal{P}$ using the mapping R. Then there is a unique continuous function $f : D_{\mathsf{f}} \to \mathbb{R}$ with $T_P f = f$. Furthermore, f interpolates the data and its graph coincides with the attractor A.

Proof The proof is divided in two steps. First, we will show that the function f is uniquely determined and interpolates the data. Afterwards, we will show that the graph of this function coincides with the attractor A.

(i) From Lemma 4.3 we know that the contraction T_P maps $\mathcal{F}$ to itself. By the Banach contraction mapping theorem we can conclude that there is exactly one function f with $T_P f = f$. This function is continuous since it is an element of $\mathcal{F}$. To show that f interpolates the data we need to show that $f(d_i^{\mathsf{x}}) = d_i^{\mathsf{y}} = f_{\mathcal{P}}(d_i^{\mathsf{x}})$ for all $(d_i^{\mathsf{x}}, d_i^{\mathsf{y}}) \in D$. Since we know that $T_P f = f$ we obtain

$$\begin{aligned} f(d_i^{\mathsf{x}}) &= \frac{1}{B^P} \cdot f\left((d_i^{\mathsf{x}} - d_i^{\mathsf{x}}) \cdot B^P\right) + f_{\mathcal{P}}(d_i^{\mathsf{x}}) - \frac{f_{\mathcal{P}}(0)}{B^P} \\ &= f_{\mathcal{P}}(d_i^{\mathsf{x}}) + \frac{f(0)}{B^P} - \frac{f_{\mathcal{P}}(0)}{B^P}. \end{aligned}$$

As there is a d_i^{x} which is equal to 0 we get $f(0) - \frac{f(0)}{B^P} = f_{\mathcal{P}}(0) - \frac{f_{\mathcal{P}}(0)}{B^P}$, which gives us the equality $f(0) = f_{\mathcal{P}}(0)$ and hence $f(d_i^{\mathsf{x}}) = f_{\mathcal{P}}(d_i^{\mathsf{x}})$ holds for all d_i^{x}.

(ii) In order to show that the graph $F = \{(x, f(x)) \mid x \in D_{\mathsf{f}}\}$ of the function f coincides with the attractor, it suffices to show that $F = \Omega(F)$ — since there is only one fixed point of Ω, it then follows that $F = A$. So it suffices to show that **(ii.a)** $\Omega(F) \subseteq F$ and **(ii.b)** $F \subseteq \Omega(F)$. In order to prove **(ii.a)** we show that $\omega_i((x, f(x))) \in F$ for all $(x, f(x)) \in F$ and all $\omega_i \in \Omega$, i.e. $(\omega_i^{\mathsf{x}}(x), \omega_i^{\mathsf{y}}(f(x))) \in F$. This follows immediately from $f = T_P f = \omega_i^{\mathsf{y}} \circ f \circ {\omega_i^{\mathsf{x}}}^{-1}$, since this implies $f \circ \omega_i^{\mathsf{x}} = \omega_i^{\mathsf{y}} \circ f$ and hence $(\omega_i^{\mathsf{x}}(x), \omega_i^{\mathsf{y}}(f(x))) \in F$. Consequently, $\Omega(F) \subseteq F$. Since $\pi_{\mathsf{x}}(A) = D_{\mathsf{f}}$ it follows that for all $x \in D_{\mathsf{f}}$ there is an $x' \in D_{\mathsf{f}}$ and an $\omega_i \in \Omega$ such that $x = \omega_i^{\mathsf{x}}(x')$. From $(x', f(x')) \in F$ and $f \circ \omega_i^{\mathsf{x}} = \omega_i^{\mathsf{y}} \circ f$ we can conclude that $f(x) = \omega_i^{\mathsf{y}}(f(x'))$ and hence $\big(x, f(x)\big) = \big(\omega_i^{\mathsf{x}}(x'), \omega_i^{\mathsf{y}}(f(x'))\big)$. So **(ii.b)** holds which completes the proof. □

We call the function f from Lemma 4.4 a *fractal interpolation function for the program* $\mathcal{P}$ with respect to accuracy P: it is an interpolation function for a set of points which belong to the graph of the embedded $T_{\mathcal{P}}$-operator $f_{\mathcal{P}}$. Both $f_{\mathcal{P}}$ and the fractal interpolation function coincide at least on the given data points, the number of which depends on the chosen accuracy P. In the remainder of this section we will study the sequence of fractal interpolation functions obtained by increasing the accuracy. We show first that this sequence is a Cauchy sequence, and then that its limit converges to $f_{\mathcal{P}}$ for programs with Lipschitz continuous $f_{\mathcal{P}}$.

We next need to obtain upper and lower bounds on the values of fractal interpolation functions. Fixing an accuracy P, recall that the corresponding fractal interpolation function f is the unique fixed point of the function T_P, i.e. $f = T_P(f) = \omega_i^{\mathsf{y}} \circ f \circ {\omega_i}^{\mathsf{x}^{-1}}$. Since $\omega_i^{\mathsf{y}}(y) = \frac{y}{B^P} + f_{\mathcal{P}}(d_i^{\mathsf{x}}) - \frac{f_{\mathcal{P}}(0)}{B^P}$ it is easily verified that a lower bound for f is given by

$$f_{\mathsf{min}} = -\sum_{i=1}^{\infty} \frac{R_{\mathsf{m}}}{(B^P)^i} = -\frac{R_{\mathsf{m}}}{B^P - 1}.$$

Analogously, an upper bound f_{max} can be obtained as

$$f_{\mathsf{max}} = R_{\mathsf{m}} + \frac{R_{\mathsf{m}}}{B^P - 1}.$$

4.5 Lemma Let $\mathcal{P}$ be a program with Lipschitz continuous $f_{\mathcal{P}}$. For each accuracy i let f_i be the corresponding fractal interpolation function. Then the sequence $(f_i)_{i \in \mathbb{N}}$ is a Cauchy sequence in $(\mathcal{F}, d_{\mathsf{f}})$.

Proof The proof is divided into two steps. We first compute the distance between f_i and f_{i+1}, and then use this to show that the sequence is Cauchy.

(i) Let i be fixed. We compute the distance between the two fractal interpolation functions f_i and f_{i+1}. For convenience we use f for f_i and $\hat{f}$ for f_{i+1}. For both functions we know that $Tf = f$ and $\hat{T}\hat{f} = \hat{f}$ hold, where T and $\hat{T}$ denote the operators introduced in Lemma 4.3, constructed for the accuracies $P = i$ and $\hat{P} = i + 1$ respectively. We use the notation d_j^{x} for the interpolation values for f and $\hat{d}_k^{\mathsf{x}}$ for the interpolation values for $\hat{f}$.

From $Tf = f$ and $\hat{T}\hat{f} = \hat{f}$ we can conclude that

$$f(x) = \frac{f\left((x - d_j^{\mathsf{x}}) \cdot B^i\right)}{B^i} + f_{\mathcal{P}}(d_j^{\mathsf{x}}) - \frac{f_{\mathcal{P}}(0)}{B^i} \qquad \text{for } x \in \left[d_j^{\mathsf{x}}, d_j^{\mathsf{x}} + \frac{R_{\mathsf{m}}}{B^i}\right] \cap D_{\mathsf{f}},$$

$$\hat{f}(x) = \frac{\hat{f}\left(\left(x - \hat{d}_k^{\mathsf{x}}\right) \cdot B^{i+1}\right)}{B^{i+1}} + f_{\mathcal{P}}\left(\hat{d}_k^{\mathsf{x}}\right) - \frac{f_{\mathcal{P}}(0)}{B^{i+1}} \qquad \text{for } x \in \left[\hat{d}_k^{\mathsf{x}}, \hat{d}_k^{\mathsf{x}} + \frac{R_{\mathsf{m}}}{B^{i+1}}\right] \cap D_{\mathsf{f}}.$$

Therefore, we get for the distance $d_{\mathsf{f}}\left(f, \hat{f}\right)$:

$$\begin{aligned}
d_{\mathsf{f}}\left(f, \hat{f}\right) &= \max_x \Bigg\{ \Bigg| \left(\frac{f\left((x - d_j^{\mathsf{x}}) \cdot B^i\right)}{B^i} + f_{\mathcal{P}}\left(d_j^{\mathsf{x}}\right) - \frac{f_{\mathcal{P}}(0)}{B^i}\right) - \\
&\qquad\qquad \left(\frac{\hat{f}\left(\left(x - \hat{d}_k^{\mathsf{x}}\right) \cdot B^{i+1}\right)}{B^{i+1}} + f_{\mathcal{P}}\left(\hat{d}_k^{\mathsf{x}}\right) - \frac{f_{\mathcal{P}}(0)}{B^{i+1}}\right) \Bigg| \Bigg\} \\
&\le \max_x \Bigg\{ \left| \frac{B \cdot f\left((x - d_j^{\mathsf{x}}) \cdot B^i\right) - \hat{f}\left(\left(x - \hat{d}_k^{\mathsf{x}}\right) \cdot B^{i+1}\right)}{B^{i+1}} \right| + \\
&\qquad\qquad \left| f_{\mathcal{P}}\left(d_j^{\mathsf{x}}\right) - f_{\mathcal{P}}\left(\hat{d}_k^{\mathsf{x}}\right) \right| + \left| \frac{-B \cdot f_{\mathcal{P}}(0) + f_{\mathcal{P}}(0)}{B^{i+1}} \right| \Bigg\} \\
&\le \max_x \left\{ \frac{B \cdot \left(R_{\mathsf{m}} + \frac{R_{\mathsf{m}}}{B^i - 1}\right) + \frac{R_{\mathsf{m}}}{B^{i+1} - 1}}{B^{i+1}} + L \cdot \left| d_j^{\mathsf{x}} - \hat{d}_k^{\mathsf{x}} \right| + \frac{(B-1) \cdot R_{\mathsf{m}}}{B^{i+1}} \right\}
\end{aligned}$$

The last step uses the fact that $f_{\mathcal{P}}$ is continuous on (D_{f}, d_1) with some Lipschitz constant L, and the results concerning minima and maxima of f_i.

Since d_j^{x} and $\hat{d}_k^{\mathsf{x}}$ are chosen with respect to the same x we know that the distance between d_j^{x} and $\hat{d}_k^{\mathsf{x}}$ is bounded by $\frac{R_{\mathsf{m}}}{B^i}$. Hence

$$\begin{aligned}
d_{\mathsf{f}}\left(f, \hat{f}\right) &\le \frac{B \cdot \left(R_{\mathsf{m}} + \frac{R_{\mathsf{m}}}{B^i - 1}\right) + \frac{R_{\mathsf{m}}}{B^{i+1} - 1}}{B^{i+1}} + L \cdot \frac{R_{\mathsf{m}}}{B^i} + \frac{(B-1) \cdot R_{\mathsf{m}}}{B^{i+1}} \\
&\le \frac{B \cdot \left(R_{\mathsf{m}} + \frac{R_{\mathsf{m}}}{B^i - 1}\right) + \frac{R_{\mathsf{m}}}{B^{i+1} - 1} + L \cdot R_{\mathsf{m}} + (B-1) \cdot R_{\mathsf{m}}}{B^{i+1}} \\
&\le R_{\mathsf{m}} \cdot \frac{B \cdot \left(1 + \frac{1}{B^i - 1}\right) + \frac{1}{B^{i+1} - 1} + L + (B-1)}{B^{i+1}} \\
&< \frac{R_{\mathsf{m}}}{B^{i+1}} \cdot (4B + L) \le \frac{R_{\mathsf{m}}(4 + L)}{B^i}.
\end{aligned}$$

(ii) From part (i) we can conclude that for $j \le k$ we have

$$d_{\mathsf{f}}(f_j, f_k) < \sum_{i=j}^{k} \frac{R_{\mathsf{m}}(4 + L)}{B^i} = \frac{R_{\mathsf{m}}(4 + L)}{B - 1} \left(\frac{1}{B^j} - \frac{1}{B^k}\right).$$

So for fixed j the value of $d_{\mathsf{f}}(f_j, f_k)$ is bounded by $d_{\mathsf{f}}(f_j, f_k) \le \frac{R_{\mathsf{m}}(4+L)}{B-1} \cdot \frac{1}{B^j}$, and we obtain that $(f_i)_{i \in \mathbb{N}}$ is a Cauchy sequence, since for any $\epsilon > 0$ there is some n such that for all $j, k > n$ we have $d_{\mathsf{f}}(f_j, f_k) < \epsilon$. □

4.6 Theorem (Approximation Theorem) Let P be a program with Lipschitz continuous $f_{\mathcal{P}}$. Then the sequence $(f_i)_{i\in\mathbb{N}}$ of fractal interpolation functions with accuracies i converges uniformly to $f_{\mathcal{P}}$ in the complete metric space $(\mathcal{F}, d_{\mathsf{f}})$.

Proof First note that for each $x \in D_{\mathsf{f}}$ there is a sequence of interpolation data points $(d_i^{\mathsf{x}}, d_i^{\mathsf{y}})$ such that each $(d_i^{\mathsf{x}}, d_i^{\mathsf{y}})$ belongs to the interpolation data for accuracy i, each d_i^{x} is the offset of the appropriately chosen mapping ω_i^{x} for x, and $\lim_{i\to\infty} d_i^{\mathsf{x}} = x$.

From the continuity of f_i and the uniform convergence of f_i to some f by Lemma 4.5 we can conclude that the sequence $(f_i(d_i^{\mathsf{x}}))_{i\in\mathbb{N}}$ converges to $f(x)$. Knowing that the f_i are interpolation functions, hence $f_i(d_i^{\mathsf{x}}) = f_{\mathcal{P}}(d_i^{\mathsf{x}})$, we obtain that the sequence $(f_{\mathcal{P}}(d_i^{\mathsf{x}}))_{i\in\mathbb{N}})$ converges to $f(x)$. But $f_{\mathcal{P}}$ is continuous by assumption, so $\lim_i f_{\mathcal{P}}(d_i^{\mathsf{x}}) = f_{\mathcal{P}}(\lim_i d_i^{\mathsf{x}}) = f_{\mathcal{P}}(x)$ and hence $f(x) = f_{\mathcal{P}}(x)$ for all $x \in D_{\mathsf{f}}$. $\square$

Theorem 4.6 shows that we can approximate the graph of any logic program for which $f_{\mathcal{P}}$ is Lipschitz continuous arbitrarily well. Unfortunately, the necessary number of mappings grows exponentially with the accuracy. From $d_{\mathsf{f}}(f_j, f_k) < \frac{R_{\mathsf{m}}(4+L)}{B-1}\left(\frac{1}{B^j} - \frac{1}{B^k}\right)$ it follows that $d_{\mathsf{f}}(f_j, f_{\mathcal{P}}) \leq \frac{R_{\mathsf{m}}(4+L)}{B^j(B-1)}$, i.e. for any given $\varepsilon > 0$ we can construct an IIFSP, such that the corresponding fractal interpolation function f_P lies within an ε-neighbourhood of $f_{\mathcal{P}}$. This IFS needs to be constructed using accuracy P such that $\frac{R_{\mathsf{m}}(4+L)}{B^P(B-1)} < \varepsilon$, i.e.

$$P > \ln_B \frac{R_{\mathsf{m}}(4+L)}{\varepsilon(B-1)}.$$

5 Logic Programs as Recurrent RBF-Networks

We will now proceed to the task which motivated our investigations, namely the approximation of logic programs by artificial neural networks. Such networks consist of a number of simple computational units, which are connected in the sense that they can propagate simple information — usually in the form of real numbers — along these connections. We want to construct a network which computes an approximation of $f_{\mathcal{P}}(x)$ for a given x. To this end, we will employ the results of the previous sections. More precisely, we will show how the fractal interpolation systems from Section 4 can be encoded.

The basic idea underlying our encoding is to exploit the self-similarity of the fractal interpolation functions f, and the "recursive" nature of the corresponding iterated function systems. In order to obtain the function value $f(x)$ for some given $x \in D_{\mathsf{f}}$, we first need to find the correct mapping $\omega_i = (\omega_i^{\mathsf{x}}, \omega_i^{\mathsf{y}})$, i.e. the one for which $x \in \omega_i^{\mathsf{x}}(D_{\mathsf{f}})$, and compute $\omega_i^{\mathsf{y}}(y)$ (where initially $y = 0$). Then we zoom in on the image of $R_{\mathsf{m}} \times R_{\mathsf{m}}$ under ω_i and repeat the process.

For our implementation of this idea we use radial basis function networks (RBF-networks). These consist of simple units which perform "radial basis functions" as input-output-mappings. These are functions f for which the values $f(x)$ are distributed symmetrically around a center. Two examples and a very simple schematic RBF-network are shown in figure 15.

RBF-networks are known to be universal approximators, i.e. with networks as shown in Figure 15 it is possible to approximate any continuous function to any given accuracy, provided sufficiently many units are being used in the middle layer.

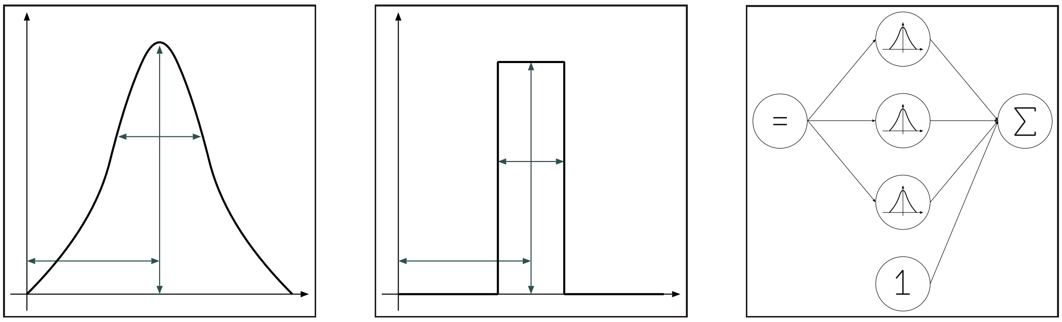

Figure 15: Two examples of radial basis functions and a simple RBF-network.

$$x' = \begin{cases} (x - \mu + \sigma) \cdot B^P & \text{if } |\mu - x| \leq \sigma \\ 0 & \text{otherwise} \end{cases}$$

$$y' = \begin{cases} s \cdot h & \text{if } |\mu - x| \leq \sigma \\ 0 & \text{otherwise} \end{cases}$$

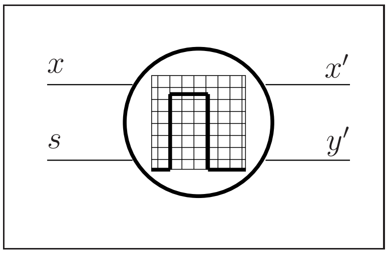

Figure 16: Dynamics and scheme of an $\mathrm{RBF}^{x,y}_{s,x}$-unit.

To simplify our exhibition and the construction of the network we introduce a new type of unit, which we call an $\mathrm{RBF}^{x,y}_{s,x}$-unit. It computes two distinct output-values x' and y'. Furthermore, it computes a parametrised radial basis function, where an additional scaling s is applied to y'. These units can be understood as abbreviations, since they can be converted into a network consisting of simple units, i.e. although we are using $\mathrm{RBF}^{x,y}_{s,x}$-units it is possible to encode the entire resulting network using standard RBF-units. Figure 16 shows the dynamics and a schematic plot of an $\mathrm{RBF}^{x,y}_{s,x}$-unit. The static parameters of each $\mathrm{RBF}^{x,y}_{s,x}$-unit are the center μ, the width σ and the height h.

Using $\mathrm{RBF}^{x,y}_{s,x}$-units we can construct the network as shown in Figure 18 using the algorithm shown in Figure 17. The example program for the construction is taken from Section 3.2, where the corresponding $\mathrm{IFS}^l_{\mathcal{P}}$ was already computed. The three initial inputs s_0, y_0 and x_0 need to be initialised with $s_0 = 1$, $y_0 = 0$, and $x_0 = x$. The network computes an approximation of $f_{\mathcal{P}}$ for a given input x. Each iteration through the network performs the following computations:

- The scaling factor s is multiplied with $\frac{1}{B^P}$.
- Each $\mathrm{RBF}^{x,y}_{s,x}$-unit computes the corresponding outputs x' and y', where for exactly one unit $x', y' \neq 0$. Since x_0 was initialised with $R(I)$, the output of the "active" $\mathrm{RBF}^{x,y}_{s,x}$-units after the first iteration is $x' = (x - \mu + \sigma) \cdot B^P$, i.e. we have $x' = (x - d^{\mathsf{x}}_i) \cdot B^P = R\big(R^{-1}(x) \backslash R^{-1}(d^{\mathsf{x}}_i)\big) \cdot B^P$. This is the "zooming into the interpretation" mentioned earlier, i.e. x' is a left-shifted version of x.
- The current y'-output of the "active" unit is added to the previous y.

The output y of the network converges to the value of the fractal interpolation function f defined by the IFS, which was used to construct the network. More precisely, we have $d_1(y, f(x)) = \left(\frac{1}{B^P}\right)^i$, where i denotes the number of iterations performed and P is the accuracy used for the construction. Furthermore, we know that $d_{\mathsf{f}}(f, f_{\mathcal{P}}) \leq \frac{R_{\mathsf{m}}(4+L)}{B^P(B-1)}$, which

Algorithm 5.1 (Construction of recurrent RBFN$_\mathcal{P}$)
Let $\mathcal{P}$ be a logic program and B the base of the embedding R.

1. Choose a periodicity $P \geq 1$.
2. Create an empty 3-layered RBF-network. Add three input units (s, x, y) to the first layer and three output units (s', x', y') to the third. The input units compute the identity function and the output units return a weighted sum of their inputs.
3. The hidden layer consists of 2^P RBF$^{x,y}_{s,x}$-units initialised as follows:
   a) Compute the IFS$^l_\mathcal{P}$ $((\mathbb{R}^2, d_2), \Omega)$ for $\mathcal{P}$ using the periodicity P, with $\Omega = \left\{(\omega^\mathsf{x}_i, \omega^\mathsf{y}_i) \mid 1 \leq i \leq 2^P\right\}$, $\omega^\mathsf{x}_i(x) = \frac{1}{B^P} \cdot x + d^\mathsf{x}_i$ and $\omega^\mathsf{y}_i(y) = \frac{1}{B^P} \cdot y + d^\mathsf{y}_i$, as described in Algorithm 3.7.
   b) For all i the RBF$^{x,y}_{s,x,i}$-unit is initialised with $\sigma_i = \frac{1}{2 \cdot B^P}$, $\mu_i = d^\mathsf{x}_i + \sigma_i$ and $h_i = d^\mathsf{y}_i$.
4. Connect the units as shown in Figure 18, where all weights are set to 1, but the connection from s to s' is set to B^{-P}.

Figure 17: Algorithm constructing RBF-network.

yields $d_1(y, f_\mathcal{P}(x)) \leq \frac{R_\mathsf{m}(4+L)}{B^P(B-1)} + \left(\frac{1}{B^P}\right)^i$. We conclude that we can approximate the single-step operator of any logic program for which the embedding is Lipschitz continuous up to any desired degree of accuracy.

6 Related Work

One of the key ideas on which our work on neural-symbolic integration is based, is to represent logic programs by representing their associated immediate consequence operators. This approach was put forward by Hölldobler and Kalinke [29], and reported also in [21], in order to encode propositional logic programs by feedforward neural networks with threshold activation functions. They also observe that these networks can be cast into a recurrent architecture in order to mimic the iterative behaviour of the operator.

Two major lines of investigation were spawned by this work. D'Avila Garcez, Broda, Gabbay, and Zaverucha [12, 14] extend the work by Hölldobler and Kalinke to cover networks with sigmoidal activation functions, and study machine learning and knowledge extraction aspects of the resulting frameworks.

The second line of investigation was initiated by Hölldobler, Kalinke, and Störr [30], who study first-order logic programs and how to approximate their single-step operators by feedforward neural networks. A general approximation theorem due to Funahashi [16] is of central importance for their approach, which is restricted to the study of acyclic programs with injective level mappings. They show that these programs can be approximated arbitrarily well by feedforward networks, but do not specify any means for actually constructing them.

Generalizations of this approach to programs with continuous single-step operators, and also to other semantic operators, are obtained by Hitzler and Seda [19, 22, 23], reported also in [21]. At this stage, topological and metric studies of declarative semantics, originally

```
e(0).
e(s(X)) :- not e(X).
o(X) :- not e(X).
```

$$|e(s^x(0))| = 2 \cdot x + 1$$
$$|o(s^x(0))| = 2 \cdot x + 2$$

$$\Omega = \left\{ \begin{pmatrix} \frac{1}{16} & 0 \\ 0 & \frac{1}{16} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{5}{16} \end{pmatrix}, \right.$$
$$\begin{pmatrix} \frac{1}{16} & 0 \\ 0 & \frac{1}{16} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{16} \\ \frac{5}{16} \end{pmatrix},$$
$$\begin{pmatrix} \frac{1}{16} & 0 \\ 0 & \frac{1}{16} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{1}{4} \\ \frac{15}{64} \end{pmatrix},$$
$$\left. \begin{pmatrix} \frac{1}{16} & 0 \\ 0 & \frac{1}{16} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \frac{5}{16} \\ \frac{15}{64} \end{pmatrix} \right\}$$

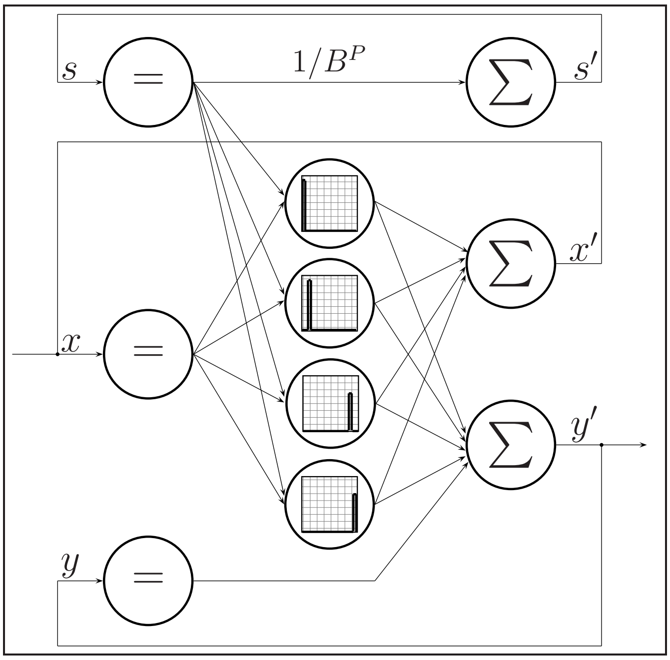

Figure 18: A logic program, level mapping, interpolating IFS for $P = 2$ and corresponding recurrent $\mathrm{RBF}^{x,y}_{s,x}$-network. The example program and its $\mathrm{IFS}^{l}_{\mathcal{P}}$ are taken from page 16.

developed for entirely different purposes [4, 5, 15, 18, 24, 39, 40, 42], come into play. From this perspective, our work is in the spirit of the general programme of research laid out by Blair et al. [9].

Work by Blair et al. on continualizations of discrete systems [8] relates very closely to the particular tool we have chosen for our approach, namely iterated function systems. In their paper, Blair et al. study covered programs and show, amongst other things, that their single-step operators can be obtained by means of attractors of affine hyperbolic finite automata, which in turn can be understood as iterated function systems. Their work also shows the intimate relationship between logic programming and dynamical systems related to self-similarity and chaos theory, which we have been able to put to use in this paper.

7 Conclusions and Further Work

We have presented results for exact and approximate representation of single-step operators associated with logic programs by iterated function systems, fractal interpolation systems, and recurrent radial basis function networks. Our results cover first-order logic programs with function symbols under the provision that the embedded associated single-step operator is Lipschitz continuous. We have given algorithms for constructing approximating iterated function systems and recurrent radial basis function networks for given logic programs.

As to the relation with the work by Blair et al. [8], we note that the exact relationship between the class of programs covered by their results, namely covered programs, and ours, namely those whose embedded single-step operator is Lipschitz continuous, remains to be determined and will require further research, as already mentioned. While the general observation in [8] that covered logic programs can be represented by iterated function systems breaks the ground for deep investigations into these matters, our results provide explicit approximations in the Euclidean plane, which can be converted to a standard neural network architecture in a straightforward way. The concrete results and constructions which we provide, however, come at the price of the stronger hypothesis of Lipschitz continuity required for our results. We believe that this requirement can be weakened, but different mathematical approaches than the one employed here may be needed in order to obtain satisfactory results.

There is also one caveat: If one would like to construct an approximating system or network which approximates a given logic program within some a priori given error bound, then we can only guarantee this if a Lipschitz constant L of the function $f_{\mathcal{P}}$ — which is the embedding of the single-step operator $T_{\mathcal{P}}$ in the reals — is not only existent but also *known*. This can be seen from the calculations of upper error bounds at the ends of Sections 4 and 5. We do not know of any general method for obtaining Lipschitz constants, and ways of doing this will be subject to further research. For certain well-behaved programs, Lipschitz constants are easily calculated. For acyclic programs with injective level mappings as covered in [30], for example, a Lipschitz constant is $\frac{1}{B-2}$, where $B > 2$ is the base used for the embedding R. In these cases our results yield exact algorithms for obtaining approximating networks given an a priori error bound.

Our results surpass those of [30] in at least two ways. Firstly, for the programs covered in [30], namely acyclic ones with injective level mappings, we are now able to give an algorithm for constructing approximating networks. Secondly, we show that a larger class of programs than covered in [30] can be approximated in principle, namely those with Lipschitz continuous embedded single-step operator, and furthermore, we have shown that for these we can provide explicit parameters for approximating recurrent neural networks, provided a suitable Lipschitz constant can be determined. This latter point is related to the results in [21, 22, 23], where a larger class of programs — those with continuous single-step operator — were treated, but without providing explicit constructions of approximating networks. So our conclusions are stronger, but so are our assumptions.

Let us also note that we use a different network architecture than in [21, 22, 23, 30], namely recurrent RBF-networks instead of three-layer feedforward networks with sigmoidal activation functions. Indeed, we believe that RBF-networks constitute a much more natural choice for representing logic programs at least under the general approach inspired by [29]. This is due to the intuition that points or interpretations which are "close" to each other (topologically or metrically speaking) are supposed to represent similar meaning. The specific shape of the activation functions in RBF-networks thus can be understood in such a way that a unit becomes active only for a cluster of values, i.e. interpretations, which have similar meaning. The binary nature of sigmoidal activation functions seems to be much more difficult to explain from an intuitive perspective. Certainly, our recurrent network can be unfolded to a feedforward architecture with several layers if this is desired, and on the mathematical level it should not make much of a difference which architecture is being used. The question of how to obtain algorithms for constructing approximating networks with sigmoidal activation functions, however, is probably rather hard, but may be solvable by first understanding Lipschitz constants of embedded single-step operators.

Investigating Lipschitz constants as mentioned provides a natural next step in our investigations. It has to be said, however, that it is not yet clear how our results can be used for designing useful hybrid systems. Nevertheless, certain questions are natural to be asked at this stage. Can we use our apporach for extracting symbolic knowledge from trained neural networks? Can network learning then be understood from a symbolic perspective by observing changes in the (extracted) symbolic knowledge during the learning process? Even in the finite (propositional) case research has not yet led to satisfactory answers to these questions, and the case of first-order logic which we address here is naturally much more difficult to work with, but should be investigated. Entirely new methods may have to be developed for this purpose, as argued by Hölldobler in [28].

References

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

[2] Sebastian Bader. From logic programs to iterated function systems. Master's thesis, Department of Computer Science, Dresden University of Technology, 2003.

[3] Michael Barnsley. *Fractals Everywhere*. Academic Press, San Diego, CA, USA, 1993.

[4] Aida Batarekh and V.S. Subrahmanian. The query topology in logic programming. In *Proceedings of the 1989 Symposium on Theoretical Aspects of Computer Science*, volume 349 of *Lecture Notes in Computer Science*, pages 375–387. Springer, Berlin, 1989.

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

[6] 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.

[7] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

[8] Howard A. Blair, Jagan Chidella, Fred Dushin, Audrey Ferry, and Polar Humenn. A continuum of discrete systems. *Annals of Mathematics and Artificial Intelligence*, 21(2–4):155–185, 1997.

[9] Howard A. Blair, Fred Dushin, David W. Jakel, Angel J. Rivera, and Metin Sezgin. Continuous models of computation for logic programs. In Krzysztof R. Apt, V. Wiktor Marek, Miroslav Truszczyński, and David S. Warren, editors, *The Logic Programming Paradigm: A 25-Year Perspective*, pages 231–255. Springer, Berlin, 1999.

[10] Anthony Browne and Ron Sun. Connectionist inference models. *Neural Networks*, 14(10):1331–1355, 2001.

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

[12] 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.

[13] 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.

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

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

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

[17] Hans W. Güsgen and Steffen Hölldobler. Connectionist inference systems. In Bertram Fronhöfer and Graham Wrightson, editors, *Parallelization in Inference Systems*, volume 590 of *Lecture Notes in Artificial Intelligence*, pages 82–120. Springer, Berlin, 1992.

[18] Roland Heinze, Pascal Hitzler, and Anthony K. Seda. Convergence classes and spaces of partial functions. In *Proceedings of the 2nd International Symposium on Domain Theory, ISDT'2001*, Semantic Structures in Computation. Kluwer Academic Publishers, 200x. To appear.

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

[20] Pascal Hitzler. Towards a systematic account of different logic programming semantics. In Andreas Günter, Rudolf Kruse, and Bernd Neumann, editors, *KI2003: Advances in Artificial Intelligence. Proceedings of the 26th Annual German Conference on Artificial Intelligence, KI2003, Hamburg, Germany, September 2003*, volume 2821 of *Lecture Notes in Artificial Intelligence*, pages 355–369. Springer, Berlin, 2003.

[21] Pascal Hitzler, Steffen Hölldobler, and Anthony K. Seda. Logic programs and connectionist networks. *Journal of Applied Logic*, 2004. In this volume.

[22] 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.

[23] Pascal Hitzler and Anthony K. Seda. Continuity of semantic operators in logic programming and their approximation by artificial neural networks. In Andreas Günter, Rudolf Kruse, and Bernd Neumann, editors, *KI2003: Advances in Artificial Intelligence. Proceedings of the 26th Annual German Conference on Artificial Intelligence,*

KI2003, Hamburg, Germany, September 2003, volume 2821 of *Lecture Notes in Artificial Intelligence*, pages 105–119. Springer, Berlin, 2003.

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

[25] Pascal Hitzler and Matthias Wendt. The well-founded semantics is a stratified Fitting semantics. In Matthias Jarke, Jana Koehler, and Gerhard Lakemeyer, editors, *Proceedings of the 25th Annual German Conference on Artificial Intelligence, KI2002, Aachen, Germany, September 2002*, volume 2479 of *Lecture Notes in Artificial Intelligence*, pages 205–221. Springer, Berlin, 2002.

[26] Pascal Hitzler and Matthias Wendt. A uniform approach to logic programming semantics. *Theory and Practice of Logic Programming*, 200x. To appear.

[27] Steffen Hölldobler. *Automated Inferencing and Connectionist Models.* Fakultät Informatik, Technische Hochschule Darmstadt, 1993. Habilitationsschrift.

[28] Steffen Hölldobler. Challenge problems for the integration of logic and connectionist systems. In François Bry, Ulrich Geske, and Dietmar Seipel, editors, *Proceedings 14. Workshop Logische Programmierung*, volume 90 of *GMD Report*, pages 161–171. GMD, 2000.

[29] 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.

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

[31] Vladimir Lifschitz. Answer set planning. In Danny De Schreye, editor, *Logic Programming. Proceedings of the 1999 International Conference on Logic Programming*, pages 23–37, Cambridge, Massachusetts, 1999. MIT Press.

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

[33] Michael J. Maher. Equivalences of logic programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 627–658. Morgan Kaufmann, Los Altos, CA, 1988.

[34] V. Wiktor Marek and Miroslav Truszczyński. Stable models and an alternative logic programming paradigm. In Krzysztof R. Apt, V. Wiktor Marek, Miroslav Truszczyński, and David S. Warren, editors, *The Logic Programming Paradigm: A 25-Year Persepective*, pages 375–398. Springer, Berlin, 1999.

[35] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[36] Stephen Muggleton and Luc de Raedt. Inductive logic programming: Theory and applications. *The Journal of Logic Programming*, 19–20:629–679, 1994.

[37] Gadi Pinkas. Propositional non-monotonic reasoning and inconsistency in symmetric neural networks. In John Mylopoulos and Raymond Reiter, editors, *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 525–530. Morgan Kaufmann, 1991.

[38] Jordan B. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1):77–105, 1990.

[39] Sibylla Prieß-Crampe and Paolo Ribenboim. Logic programming and ultrametric spaces. *Rendiconti di Mathematica*, VII:1–13, 2000.

[40] Sibylla Prieß-Crampe and Paolo Ribenboim. Ultrametric spaces and logic programming. *The Journal of Logic Programming*, 42:59–70, 2000.

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

[42] Anthony K. Seda and Máire Lane. On continuous models of computation: Towards computing the distance between (logic) programs. In *Proceedings of the Sixth International Workshop in Formal Methods (IWFM'03), Dublin City University, Dublin, Ireland, July, 2003*, Electronic Workshops in Computing (eWiC). British Computer Science, 2003. To appear.

[43] Lokenda Shastri. Advances in Shruti — A neurally motivated model of relational knowledge representation and rapid inference using temporal synchrony. *Applied Intelligence*, 11:78–108, 1999.

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

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