On Continuous Models of Computation: Towards Computing the Distance Between (Logic) Programs

Anthony Karel Seda
Department of Mathematics, University College Cork, Cork, Ireland
Email: a.seda@ucc.ie

Máire Lane
Department of Mathematics, University College Cork, Cork, Ireland
E-mail: mairelane@netscape.net

Abstract

We present a report on work in progress on certain aspects of a programme of research concerned with building formal, mathematical models both for aspects of the computational process and for features of programming languages. In this paper, considering work of Kozen showing that complete normed vector spaces (Banach spaces) and bounded linear operators provide a framework for the semantics of deterministic and probabilistic programs, we include logic programs within this framework. We thereby make it a framework in which it is possible to handle the semantics of all three types of program. Using these ideas, we advance a programme of research proposed by M. Bukatin and J.S. Scott concerned with defining and computing meaningful notions of metrics and generalized metrics measuring the distance between two programs, the terms metrics and generalized metrics being used here in the precise sense in which they are employed in mathematics. The long-term objective of this work is to use such metrics as tools in measuring correctness of programs.

Key Words: Formal methods, metrics, denotational semantics, deterministic programs, probabilistic programs, logic programs, Banach space, bounded linear operators.

1 Introduction

Roughly speaking, the term *software metric* as used in software engineering refers to some property of a piece of software, or of its specification, which can be measured. Such software metrics might include the number of lines of source code, the number of operators, the number of executable statements, the cyclometric complexity (McCabe) etc. They can be thought of as giving a rough measure of how far a program is from some ideal or, via similarity graphs, how far apart two programs are.

On the other hand, it is common practice in many branches of mathematics and related disciplines to introduce metrics, pseudometrics or generalized metrics, in the precise mathematical sense defined in Section 2, in order to measure the distance between any two members of a given class of objects under investigation. The intuition behind this is that two objects which are close together, that is, are a small distance apart should share many properties. Indeed, one might expect that two objects which are zero distance apart should always be identical and, conversely, that the distance from an object to itself should always be zero. Whether or not these expectations actually hold depends, of course, on the precise definition one adopts of "metric" and its generalizations, and we discuss this issue in Section 3.1.

In the context of computing science, the question of measuring the distance between two programs using metrics or generalized metrics has been addressed in [2, 4] in terms of domain theory. In fact, the work just cited may, in a certain sense, be viewed as an abstract formulation of the software development process. Indeed, granted for a moment that the goal of producing completely correct software is not feasible, it is argued in [2] that the provision of suitable optimal methods which produce close approximations to the ideal is an alternative to the absolute goal of formal methods. It is further argued in [2] that such methods depend on the powerful tools and methods of so called "continuous mathematics", and therefore that these tools should be made available in the context of program development environments (including that of domain theory).

Of course, the mathematical tools traditionally associated with computer science are logic and discrete mathematics, the latter including set theory, abstract algebra, combinatorics, graph theory, order theory, and the like, and these topics naturally reflect the discrete nature of computer science. On the other hand, there is a quite well-established usage of the methods of continuous mathematics, by which we mean topology, mathematical analysis, linear analysis etc.[1], in dealing with programming languages involving uncertainty and probabilistic elements in various ways, and in handling concurrency. For example, in the fundamental paper [6], Kozen defines a semantics for a simple programming language allowing assignment statements involving a random number generator. (Deterministic semantics is of course a special case of this obtained essentially by eliminating the random assignment.) In this semantics, each program denotes a bounded linear operator on a Banach lattice of measures, and we will give some details of this construction in Section 3.2. In [7] and its references, the reader will find some similar examples from the area of concurrency. Thus, the tools employed in these papers are sophisticated and include measure theory and functional analysis, and also the methods of linear algebra and the theory of operator algebras. Of considerable interest, however, is the fact that Kozen shows that Scott domains (and fixed-point semantics) can be embedded in his Banach-lattice framework and hence in the conventional structures found in mathematical analysis.

Consistent with the theme of the previous paragraph are a number of applications of mathematical analysis to the theory of logic programming semantics made by various writers, including the first-named author of this paper and his co-authors. In the main, such applications result from viewing certain semantic operators arising from logic programs as rather general dynamical systems, see [13] for example. (We support the view proposed by Prakash Panangaden in [11] that "... Indeed one can say that the semantics of programming languages is a branch of dynamical systems and should be studied as such.") Normally, these operators T are studied by means of order theory, and one is interested especially in their fixed points, if any exist. Our present point of view is different, and is to "linearize" such operators T by representing them as bounded linear operators on Banach spaces and hence determining their properties by studying their linearizations from the view point of linear analysis in a manner akin to that found in [6]. When this much is done, one then sees, by virtue of Kozen's embedding mentioned above, that Banach spaces and linear operators defined on them provide a common framework in which to define the denotational semantics of imperative programs, of probabilistic programs and of logic programs, and it is the first of our two main objectives in this paper to draw attention to this observation.

Moreover, the framework of Banach spaces and linear operators is highly conducive to the approximation methods proposed in [2], indeed more so in many ways than the framework of domain theory. In particular, once programs denote bounded linear operators, it is possible to consider the distance between two programs determined by the operator norm of the difference of the corresponding operators. This distance determines a pseudometric in general, but by a standard process one can

[1]Similarly, our usage of the term "continuous model of computation" in the title of this paper is informal and simply means a model of computation employing the tools of continuous mathematics.

pass to an associated metric. It is the second main objective of this paper to advance the programme of research proposed in [2] by reporting on initial investigations to found a theory of distance functions defined on pairs of programs in this way, although we focus here mainly on logic programs. Nevertheless, there are several issues which immediately arise whichever type of program one wishes to consider, as follows. The first is the question of how well the linearizations reflect properties of the programs in question. The second is the question of how meaningful the distance between two programs is, and in particular how meaningful it is in terms of program development when two programs are close together. The third is the question of actually computing the distance itself between two programs. These issues will be partially addressed here. As far as the second point is concerned, the linear operators in question are all related to the semantics of programs rather than to their syntax. In the case of logic programs, these operators are determined by semantic operators, as already noted. Thus, we note here straight away that two definite logic programs which are distance zero apart have the same (fixed-point) semantics and hence compute the same things, which is what one would expect, and this fact can be viewed as positive evidence that the methods proposed here have some significance.

The overall structure of the paper is as follows. First, in Section 2, we collect together the necessary background we need in domain theory and analysis. Then, in Section 3, we discuss the properties one would expect of functions measuring the distance between pairs of elements of various classes of programs, commencing with a brief summary of the results of [2, 4], moving then to a brief discussion of the results of [6] before commencing, in the latter part of Section 3, the presentation of our own results in the context of logic programs. This development is continued in Sections 4 and 5. In particular, we show in Section 5 that two definite programs which are distance zero apart are subsumption equivalent and hence compute the same things. We briefly discuss the issue of how well our linear operators represent programs, and close with a simple, but interesting, example showing that the metrics we define reflect properties of programs. Finally, in Section 6, we list our conclusions. Because of the technicalities involved and lack of space, we do not include details of all of the constructions and proofs.

Acknowledgement We thank two anonymous referees for their comments, which helped sharpen the focus and presentation of the paper.

2 Background Material

In this section, we collect together the basic facts we need in the sequel concerning semantics, domain theory, analysis etc. with the intention of making the paper as self contained as possible, but with no pretence at completeness. We begin with the following definition which formalizes the various notions of distance function that one needs to consider.

2.1 Definition Let X be a set and let $d : X \times X \to R^+$ be a mapping, where R^+ denotes the set of non-negative real numbers.

1. We call d a *metric* if it satisfies the following axioms:
    (a) For all $x, y \in X$, $x = y$ if and only if $d(x, y) = 0$.
    (b) For all $x, y \in X$, $d(x, y) = d(y, x)$.
    (c) For all $x, y, z \in X$, $d(x, z) \le d(x, y) + d(y, z)$.
2. We call d a *pseudometric* if it satisfies axioms (b) and (c) for a metric, and also the following axiom:

(a′) For all $x \in X$, $d(x,x) = 0$.

3. We call d a *partial metric* if it satisfies the following axioms:

   (a) For all $x, y \in X$, $x = y$ if and only if $d(x,x) = d(x,y) = d(y,y)$.

   (b) For all $x, y \in X$, $d(x,x) \leq d(x,y)$.

   (c) For all $x, y \in X$, $d(x,y) = d(y,x)$.

   (d) For all $x, y, z \in X$, $d(x,z) \leq d(x,y) + d(y,z) - d(y,y)$.

Thus, a pseudometric fails to be a metric just to the extent that $d(x,y) = 0$ does not imply that $x = y$. In general, if d is a pseudometric defined on a set X, it is a standard procedure, see [14], to define an equivalence relation $\sim$ on X by $x \sim y$ if and only if $d(x,y) = 0$. One then passes to the equivalence classes $[x]$ of this relation, and defines a distance function, still denoted by d, on the set of equivalence classes by $d([x],[y]) = d(x,y)$. It is routine to check that d is well-defined and is a metric.

It should be noted that a partial metric d can satisfy $d(x,x) \neq 0$. This property is slightly surprising, but in fact partial metrics were introduced by S.G. Matthews, see [10], in the context of Kahn's dataflow model, where they naturally occur, and they have found application in a number of other places in computing, one of which we will discuss below in Section 3.1. It can be argued that $d(x,x) \neq 0$ simply means that x is only partially defined.

As far as semantics is concerned, we will not be concerned here either with operational or axiomatic semantics of programming languages, but only with denotational semantics. Thus, by the term *semantics* we mean denotational semantics and therefore we are concerned with mappings from a syntactic space of programs (or program parse trees) to a space of meanings. This theory divides into two parts: one based on order and domain theory, the other based on more conventional mathematics, as we shall see.

The basic ingredients in order-based semantics are as follows. By the term *domain* we understand a directed complete partial order $(D, \sqsubseteq)$ equipped with the Scott topology, see [1]. Thus, $(D, \sqsubseteq)$ is a partially ordered set with bottom element, $\perp$, in which each directed subset M has a supremum, $\bigsqcup M$, in D. Furthermore, a subset U of D is, by definition, *Scott open* if it is upwards closed, that is, whenever $x \in U$ and $x \sqsubseteq y$ we have $y \in U$, and for any directed set M in D, if $\bigsqcup M \in U$, then $M \cap U \neq \emptyset$. If $(D, \sqsubseteq)$ and $(E, \sqsubseteq)$ are domains and $f : D \to E$ is a function, then f is called *Scott continuous* if it is continuous in the Scott topologies on D and E. In fact, this is equivalent to requiring that f is monotonic (whenever $x \sqsubseteq y$ we have $f(x) \sqsubseteq f(y)$) and for any directed set M, we have $f(\bigsqcup M) = \bigsqcup f(M)$. The usual thinking in semantics based on order is that all data types should be represented by domains, and all computable functions should be represented by Scott continuous functions between domains.

We assume that the reader has a basic familiarity with vector spaces. Most of the time we will be working over the real field of scalars, but most of what we say will apply equally well in the case of the complex field also. Let E be a vector space. We remind the reader that a *norm* on E is a mapping $\| \ \| : E \to R^+$ satisfying the properties: (1) $\|x\| = 0$ if and only if $x = 0$, (2) $\|ax\| = |a|\|x\|$ for all scalars a, and (3) $\|x+y\| \leq \|x\| + \|y\|$. It is an important fact that any norm $\| \ \|$ on E induces a metric d on E, where $d(x,y) = \|x - y\|$. As usual, we call a normed vector space *complete* or a *Banach space* if it is complete in this metric.

It will also be convenient to record some elementary facts concerning linear operators or linear transformations defined on Banach spaces. Thus, let $(E_1, \| \ \|_1)$ and $(E_2, \| \ \|_2)$ be (real or complex) Banach spaces, and let $L : E_1 \to E_2$ be a linear mapping. Then L is said to be *bounded* if there is a real number $K \geq 0$ with the property that $\|L(x)\|_2 \leq K\|x\|_1$ for all $x \in E_1$. When L is bounded, we define the *norm*, $\|L\|$, of L by

$$\|L\| = \inf\{K; K \geq 0 \text{ and } \|L(x)\|_2 \leq K\|x\|_1 \text{ for all } x \in E_1\}.$$

Of course, the value of $\|L\|$ depends on the choice of the norms on E_1 and E_2, and we indicate this fact when necessary by writing $\|L\|_{1,2}$. Furthermore, we will on occasions use, without further mention, the following well-known facts concerning the norm of an operator L.

2.2 Proposition Let $L : (E_1, \| \ \|_1) \to (E_2, \| \ \|_2)$ be a bounded linear operator. Then the following hold.

(1) L is continuous.

(2) $\|L\| = \sup\{\|L(x)\|_2; \|x\|_1 \leq 1\}$ and indeed $\|L\| = \sup\{\|L(x)\|_2; \|x\|_1 = 1\}$ if $E_1 \neq \{0\}$.

(3) $\|L(x)\|_2 \leq \|L\|\|x\|_1$ for all $x \in E_1$. ■

3 Distance Functions on Classes of Programs

It will be helpful to first summarize the results of [4] and [6].

3.1 Deterministic Programs

In [4], Bukatin and Scott consider the conventional semantic framework of a domain $\mathcal{P}$ of (parse trees of) programs, a domain $\mathcal{A}$ of meanings and a Scott continuous semantic function $[\![\ ]\!] : \mathcal{P} \to \mathcal{A}$ determined by the usual methods of denotational semantics. Furthermore, it is assumed that we have a domain D representing distances (usually, the intervals domain) and a Scott continuous function $\rho : \mathcal{A} \times \mathcal{A} \to D$, called a *generalized distance function.* Under certain natural topological conditions, ρ will reflect computational properties of $\mathcal{A}$ in which case $\rho([\![p_1]\!], [\![p_2]\!])$ can be thought of as yielding a computationally meaningful distance between programs p_1 and p_2, namely, the distance between their meanings. Finally, assume that there is an element 0 of D representing the ordinary numerical zero. One of the main conclusions of [4] then is that $\rho(a, a)$ must be non-zero for some values of a. Thus, it emerges that partial metrics and the associated relaxed metrics defined in [4] are the most appropriate distance functions to consider in the context of domain theory if one wants Scott continuity of ρ.

3.2 Probabilistic Programs

In [6], Kozen considers **while** programs which have simple assignment, composition, conditional tests, **while** loops and calls $x :=$ **random** to a random number generator. He gives two semantics for such programs: one operational in flavour and the other denotational in flavour, and it is the second which concerns us.

Suppose that the program variables are $x_1, \ldots, x_n$. Let (X, M) be a measurable space and let $\mathbf{B} = \mathbf{B}(X^n, M^{(n)})$ be the set of measures on the cartesian product measurable space $(X^n, M^{(n)})$. Thus, $\mathbf{B}$ consists of all linear combinations of all possible joint distributions of the program variables $x_1, \ldots, x_n$, where addition and scalar multiplication are defined by

$$(\mu + \nu)(B) = \mu(B) + \nu(B) \text{ for all } B \in M^{(n)},$$
$$(a\mu)(B) = a(\mu(B)) \text{ for all } B \in M^{(n)},\ a \in R.$$

Let $\mathbf{P}$ denote the cone of positive measures in $\mathbf{B}$, and let $\|\mu\|$ denote the total variation norm of μ given by $\|\mu\| = |\mu|(X^n)$, where $\|\mu\|$ denotes the total variation measure determined by μ, namely, the sum of the positive and negative parts of μ in the Hahn-Jordan decomposition of μ. Then $(\mathbf{B}, \mathbf{P}, \| \, \|)$ is a conditionally complete Banach lattice. Thus, $(\mathbf{B}, \| \, \|)$ is a Banach space; $\mathbf{P}$ orders $\mathbf{B}$ by $\mu \leq \nu$ if and only if $\nu - \mu \in \mathbf{P}$, each pair μ, ν has a least upper bound or join $\mu \vee \nu$, and each bounded subset of $\mathbf{B}$ has a least upper bound; and, finally, we have that $\||\mu|\| = \|\mu\|$, and whenever $0 \leq \mu \leq \nu$, we have $\|\mu\| \leq \|\nu\|$.

The measures of interest here are the probability measures, namely, the positive measures with norm 1, and the subprobability measures, namely, the positive measures with norm at most 1. Kozen shows that any program in the language in question maps, in a natural way, a probability distribution to a subprobability distribution, and that this mapping extends uniquely to a bounded linear operator $\mathbf{B} \to \mathbf{B}$; we do not, however, have space to give more details of this here other than to remark that the semantics of **while** loops is given in terms of least fixed points, as usual. Thus, ultimately, each program S denotes a bounded linear operator, and hence may be interpreted as an element of the Banach space $\mathbf{B}'$ of all bounded linear operators $\mathbf{B} \to \mathbf{B}$. Furthermore, $\mathbf{B}'$ is itself ordered by requiring $S \leq T$ if and only if $S(\mu) \leq T(\mu)$ for all $\mu \in \mathbf{P}$. Note that by eliminating the random assignment and restricting input distributions to point masses, one obtains deterministic semantics as a special case of probabilistic semantics. Using these facts, Kozen then further shows that one may embed the domain $Pfn(\omega \to \omega)$ of partial functions, endowed with the usual ordering $\sqsubseteq$ of graph inclusion, into $\mathbf{B}'$ for a suitable choice of $\mathbf{B}$, where ω denotes the set of natural numbers. Under this embedding, the totally undefined function (the bottom element) is mapped to 0, and $\sqsubseteq$ in $Pfn(\omega \to \omega)$ is mapped to $\leq$ in $\mathbf{B}'$, as one would require.

Thus, in a rather general sense, one sees that Banach spaces and bounded linear operators form a semantic framework for both probabilistic and deterministic programming languages. Furthermore, once each program denotes a bounded linear operator, one has a natural distance function d defined on classes of programs by $d(S, T) = \|S - T\|$, that is, the operator norm of the difference of S and T, where we are using the same symbols S, T etc. to denote both programs and their denotations which, in this case, are bounded linear operators.

Our remaining task is to carry out this sort of construction in the context of logic programs, and we do this next. Once this is done, we will have fulfilled our first objective of showing that Banach spaces and linear operators form a semantic framework for deterministic, probabilistic and logic programs. Then, by considering the distance function d just defined, we will have candidate distance functions to fulfill our second objective for each type of program we are discussing.

3.3 Logic Programs

There are various possible ways of associating Banach spaces and bounded linear operators with logic programs, but the one we pursue here uses the notion of *composition operator* defined below; other ways will be considered elsewhere. One obvious requirement in this process, of course, in any computing paradigm is that if the operators corresponding to programs P_1 and P_2 are equal, then P_1 and P_2 should be related in some sense. In other words, the representation in terms of operators should be faithful relative to some notion of equivalence on programs. We shall see later on that this is so in the case of logic programs.

Let X be a set, and let F denote either the real or complex scalar field. We let $\mathcal{F}(X)$ denote the set of all functions $f : X \to F$ defined on X and equipped with the usual vector space and algebra operations defined pointwise as follows:

$$\begin{aligned}(f_1+f_2)(x) &= f_1(x)+f_2(x) \text{ for all } x\in X,\\ (f_1f_2)(x) &= f_1(x)f_2(x) \text{ for all } x\in X, \text{ and}\\ (\alpha f)(x) &= \alpha f(x) \text{ for all } x\in X \text{ and all } \alpha\in F,\end{aligned}$$

where f_1, f_2 and f are elements of $\mathcal{F}(X)$.

3.1 Definition Let $T : X \to Y$ be a mapping. We define the *composition operator* $\mathcal{F}(T) : \mathcal{F}(Y) \to \mathcal{F}(X)$ induced by T by setting $\mathcal{F}(T)(g) = g \circ T$ for each $g \in \mathcal{F}(Y)$.

Thus, the situation in the previous definition can be pictured as follows

$$\begin{tikzcd} & F \\ X \arrow[ur, "\mathcal{F}(T)(g)"] \arrow[r, "T"] & Y \arrow[u, "g"'] \end{tikzcd}$$

3.2 Proposition For any mapping $T : X \to Y$, $\mathcal{F}(T)$ is a homomorphism $\mathcal{F}(T) : \mathcal{F}(Y) \to \mathcal{F}(X)$ of algebras over F, and $\mathcal{F}$ determines a contravariant functor from the category of sets and mappings to the category of algebras and algebra homomorphisms over F.

In the sequel, we will be concerned only with the case in which $X = Y$, that is, with functions T mapping X to itself and therefore with the corresponding composition operators $\mathcal{F}(T) : \mathcal{F}(X) \to \mathcal{F}(X)$. Nevertheless, it is useful to give the definition in general since it is this that makes clear the contravariance of $\mathcal{F}$.

In practice, the sets X we consider will be sets of partial functions or sets of valuations etc., and may be treated as sets, topological spaces (in the Scott topology, the Lawson topology etc.) or as measure spaces. Of particular importance is the case when X is a space of valuations equipped with the Cantor topology, and is thus a compact metric space. Furthermore, the scalar valued functions we consider defined on X will either be bounded or essentially bounded, in which case $\mathcal{F}(X)$ will be equipped with the uniform norm, or will be integrable functions, and then $\mathcal{F}(X)$ will be equipped with one of the L^p norms.

It will therefore be convenient next to summarize the properties we need of these two classes of normed spaces.

3.4 Bounded Functions and the Uniform Norm

Noting the categorical nature of $\mathcal{F}$, and in order to distinguish between the cases, it will be helpful to use the notation $B(X)$ for $\mathcal{F}(X)$ and $B(T)$ for $\mathcal{F}(T)$ in the case of bounded functions. Later on in this section, the notation $L(X)$ respectively $L(T)$ will be established in the case of integrable functions and the L^p norms. Thus, we introduce the following notation.

3.3 Notation Let $B(X)$ denote the set of all bounded real or complex valued functions defined on X. In both cases, we endow $B(X)$ with the uniform or supremum norm, $\| \; \|_\infty$, defined by $\|f\|_\infty = \sup\{|f(x)|\,; x \in X\}$.

3.4 Proposition For any set X, the pair $(B(X), \| \; \|_\infty)$ is a Banach space.

3.5 Proposition For any mapping $T : X \to X$, $B(T)$ is a bounded linear operator whose norm $\|B(T)\|_\infty = \|B(T)\|_{\infty,\infty}$ is equal to 1.

3.5 Integrable Functions and the L^p Norms

Suppose that $(X, \mathcal{B}, \mu)$ is a measure space. For a real number p satisfying $1 \leq p < \infty$, we define $L^p(X, \mathcal{B}, \mu)$ to be the set of all measurable real valued functions defined on X for which $\int_X |f|^p d\mu$ is finite. If $(X, \mathcal{B}, \mu)$ is understood, we write $L^p(X, \mu)$, $L^p(X)$ or just L^p for $L^p(X, \mathcal{B}, \mu)$. For an element $f \in L^p(X)$, we define $\|f\|_p$ by

$$\|f\|_p = \left\{ \int_X |f|^p d\mu \right\}^{\frac{1}{p}}.$$

In case $p = \infty$, we define $L^\infty(X, \mathcal{B}, \mu)$ to be the set of all essentially bounded real valued measurable functions defined on X. Once again, we write $L^\infty(X)$ or L^∞ instead of $L^\infty(X, \mathcal{B}, \mu)$ when $(X, \mathcal{B}, \mu)$ is understood. In this case, we define the norm $\|f\|_\infty$ of an element $f \in L^\infty(X)$ by

$$\|f\|_\infty = \text{ess sup}(f),$$

where ess sup denotes the essential supremum of f and is defined in the usual way, namely, $\text{ess sup}(f) = \inf\{M; \mu\{x \in X; f(x) > M\} = 0\}$. Note that if f is actually bounded, then the supremum and the essential supremum coincide and so there is no conflict of notation here. Note also that, for all values of p, $1 \leq p \leq \infty$, we follow the common practice of identifying two functions in $L^p(X)$ which are equal μ-almost everywhere.

Now suppose that $(X, \mathcal{B}, \mu)$ is a measure space, and that $T : X \to X$ is a mapping on X which is measurable with respect to $\mathcal{B}$, that is, $T^{-1}(A) \in \mathcal{B}$ for each $A \in \mathcal{B}$. We want to consider the possibility of defining the composition operator $L(T) : L^p(X) \to L^p(X)$ by setting $L(T)(g) = g \circ T$, as before, but there are a number of technical obstacles to overcome. First, although $g \circ T$ will be measurable for each function $g \in L^p(X)$, it need not be the case that $|g \circ T|^p$ is μ-integrable for a given $g \in L^p(X)$, that is, $\int_X |g \circ T|^p d\mu$ need not be finite. Second, even if $|g \circ T|^p$ is μ-integrable for each $g \in L^p(X)$, it need not be the case that $L(T)$ is a bounded operator.

There are some important situations, which we consider, when both of the obstacles just mentioned can be overcome. The first of these, which we discuss in a number of simple examples later, is to take the measure consisting of unit masses placed at each point of a finite set. In this case, integration just amounts to summation, and the operator norms of interest coincide with well-known matrix norms. The second important case is when X is a compact metric space and T is continuous. In this case, by the classical theory of Krylov and Bougliabov, T has invariant Borel probability measures. Thus, there are measures μ defined on the Borel sets of X such that $\mu(X) = 1$, and $\mu(T^{-1}(A)) = \mu(A)$ for each Borel set $A \in \mathcal{B}$. Indeed, this latter condition is equivalent to the condition that $\int_X (g \circ T)\, d\mu = \int_X g\, d\mu$ for each measurable function g. In fact, such invariant measures μ are the limit points of the sequences of Cesaro sums $\frac{1}{n}\sum_{i=0}^{n-1} T_*^i \nu$ in the weak topology on $P(X)$ determined by the functions $\nu \mapsto \int_X f\, d\nu$, where f is a continuous real valued function. Here, $P(X)$ denotes the space of probability Borel measures on X, and $T_*\nu$ denotes the Borel probability measure on X defined by $(T_*\nu)(A) = \nu(T^{-1}(A))$ for each $A \in \mathcal{B}$, where $\nu \in P(X)$. Finally, if μ is an invariant measure for T, then because of the identity $\int_X (g \circ T)\, d\mu = \int_X g\, d\mu$ mentioned above, we have $\|L(T)\|_{p,p} = \|L(T)\|_p = \sup\{\|L(T)(g)\|_p; \|g\|_p \leq 1\} = \sup\left\{ \left\{\int_X |g \circ T|^p d\mu\right\}^{\frac{1}{p}} ; \left\{\int_X |g|^p\, d\mu\right\}^{\frac{1}{p}} \leq 1 \right\} = 1$.

4 Metrics Determined by Composition Operators

Let $T_1, T_2 : X \to X$ be mappings on X. As already suggested, we want to use the expressions $d(T_1, T_2) = \|B(T_1) - B(T_2)\|_{\infty,\infty}$ (denoted simply by $\|B(T_1) - B(T_2)\|_\infty$) and $d(T_1, T_2) = \|L(T_1) - L(T_2)\|_{p,p}$ (denoted simply by $\|L(T_1) - L(T_2)\|_p$) to determine metrics on the collection of all mappings on X.

4.1 Proposition Consider the distance functions d defined below on the collection of all mappings from X to X.

(1) Let X be a set and let $T_1, T_2 : X \to X$ be mappings. We define $d(T_1, T_2)$ to be $\|B(T_1) - B(T_2)\|_\infty$. Then d is a metric.

(2) Let X be a compact metric space, let $T_1, T_2 : X \to X$ be continuous, let μ be a Borel measure on X and suppose that $L(T_1), L(T_2) : L^p(X) \to L^p(X)$ are bounded linear operators. We define the distance function d by $d(T_1, T_2) = \|L(T_1) - L(T_2)\|_p$. Then d is a pseudometric, and is a metric if we identify those functions T_1 and T_2 which are μ almost everywhere equal.

5 Pseudometrics and Metrics for Logic Programs

5.1 Generalities

Let $\mathcal{L}$ be a first order language. We will suppose that all logic programs P we discuss have underlying language $\mathcal{L}$, although, of course, a given program P need not employ all the symbols of $\mathcal{L}$. We refer the reader to [8] for background concepts and our notation concerning logic programs. In fact, it will be convenient to treat each logic program P as the set ground(P) of all ground instances over B_P of each clause in P, where B_P denotes the Herbrand base of all ground instances of atoms occurring in P. (This is common practice and we will follow it without further mention.)

Each logic program has associated with it a number of semantic operators which capture many of its properties, and the one we work with here mainly is the usual two-valued immediate consequence operator $T_P : I_P \to I_P$ which we define below, where I_P, or more precisely $I_{\mathcal{L}}$, denotes the set of all two-valued interpretations for $\mathcal{L}$. We endow I_P with the Cantor topology in which I_P is homeomorphic to the usual Cantor set in R, and hence is a compact metric space, see [12]. It turns out that T_P is sometimes continuous in this topology, but not always, see [12] again, but is always Borel measurable, see [5].

We are thus in a position to consider the distances, $d(P_1, P_2)$, between two logic programs P_1 and P_2 as given by Proposition 4.1 by taking T_1 and T_2 to be T_{P_1} and T_{P_2} respectively.

5.1 Definition Let P_1 and P_2 be logic programs with underlying language $\mathcal{L}$. We define the distances $d_\infty(P_1, P_2)$ and $d_p(P_1, P_2)$ between them by setting $d_\infty(P_1, P_2) = \|B(T_{P_1}) - B(T_{P_2})\|_\infty$ and $d_p(P_1, P_2) = \|L(T_{P_1}) - L(T_{P_2})\|_p$.

The distance functions just defined are not actually metrics since $d(P_1, P_2) = 0$ does not necessarily mean that $P_1 = P_2$. What it does mean is that $T_{P_1} = T_{P_2}$. Indeed, it is readily checked that d_∞ and d_p satisfy all the axioms for a metric other than the property just mentioned, and hence are pseudometrics. We therefore suppose that the procedure for obtaining a metric from a pseudometric discussed in Section 2 has been carried out, and refer loosely to the metrics d_∞ and d_p on the set of all programs whose underlying language is $\mathcal{L}$, and denote them collectively just by d.

In fact, for definite programs, the equivalence relation just described above coincides with Maher's notion of subsumption equivalence, see [9], since two definite programs are subsumption equivalent if and only if $T_{P_1} = T_{P_2}$. Hence, for definite programs the identity $d(P_1, P_2) = 0$ has a well-established meaning within logic programming, and in fact implies that the two-valued fixed-point semantics for P_1 and P_2 coincide. More precisely, it implies that the least fixed points of T_{P_1} and T_{P_2} coincide, and hence that the least Herbrand models for P_1 and P_2 coincide and in turn this means that P_1 and P_2 compute the same things. Indeed, we expand on this point a little next to clarify the situation.

Suppose that P is a normal logic program, and let $A \leftarrow A_1, \dots A_n, \neg B_1, \dots, \neg B_m$ denote a clause C in P. Thus, A is the head, head(C), of C and $A_1, \dots A_n, \neg B_1, \dots, \neg B_m$, or $A_1 \wedge \cdots \wedge A_n \wedge \neg B_1 \wedge \dots \wedge \neg B_m$, denotes its body, body$(C)$. We let pos$(C)$ denote the set $\{A_1, \dots, A_n\}$ of positive or unnegated atoms in the body of C and let neg(C) denote the set $\{B_1, \dots, B_m\}$ of negated atoms in the body of C. We recall that in classical two-valued logic, interpretations for (the underlying language $\mathcal{L}$ of) a logic program P are identified with subsets I of B_P, and hence the set I_P of all such interpretations can be identified with the power set of B_P. In these terms, we define the immediate consequence operator $T_P : I_P \to I_P$ determined by a program P by setting $T_P(I)$ to be the set of $A \in B_P$ for which there is a (ground instance of) a clause C of the form $A \leftarrow$ body(C) satisfying $I \models$ body(C). Of course, $I \models$ body(C), that is, body(C) is true in I precisely when pos$(C) \subseteq I$ and neg$(C) \subseteq I^c$, where I^c denotes the complement of I in B_P.

In [9], Maher defined the notion of subsumption equivalence for definite programs containing variable symbols (recall that a *definite* program is one in which neg(C) is empty for each clause C). As far as sets of ground clauses are concerned, his definition amounts to the following. Suppose that C and C' are two clauses with the same head A, say. We say that C' *subsumes* C if pos$(C') \subseteq$ pos(C). One then says that a program P_1 is *subsumed* by a program P_2 if each clause of P_1 is subsumed by a clause of P_2, and that P_1 and P_2 are *subsumption equivalent* if each program subsumes the other. A main result concerning these notions, as already mentioned, is that P_1 and P_2 are subsumption equivalent if and only if $T_{P_1} = T_{P_2}$.

We want to work more generally with normal logic programs. An obvious extension of the definition of subsumption to this case is as follows. If C and C' are two clauses over $\mathcal{L}$ with the same head, then we say that C' *subsumes* C if pos$(C') \subseteq$ pos(C), and neg$(C') \subseteq$ neg(C). The remaining definitions are extended in the obvious way, and one has the following result.

5.2 Proposition If P_1 and P_2 are subsumption equivalent, then $T_{P_1} = T_{P_2}$.

Proof. Let I be an arbitrary interpretation for $\mathcal{L}$. Suppose that $T_{P_1}(I) \neq \emptyset$ and that $A \in T_{P_1}(I)$. Then there is a clause C in P_1 with head$(C) = A$ and such that $I \models$ body(C). By hypothesis, there is a clause C' in P_2 such that C' subsumes C. But then head$(C') = A$, and we immediately have that $I \models$ body(C'). Therefore, $A \in T_{P_2}(I)$ and we have $T_{P_1}(I) \subseteq T_{P_2}(I)$.

The reverse inclusion is obtained similarly. Moreover, the same argument shows that $T_{P_1}(I) = \emptyset$ if and only if $T_{P_2}(I) = \emptyset$. It follows that $T_{P_1}(I) = T_{P_2}(I)$ for all I and hence that $T_{P_1} = T_{P_2}$, as stated. ■

Consider the following example; it shows that the converse of the previous proposition fails.

5.3 Example Let P_1 be the following program

$$\begin{aligned} p(a) &\leftarrow p(a) \\ p(a) &\leftarrow \neg p(a) \\ p(b) &\leftarrow p(a) \end{aligned}$$

and let P_2 be as follows

$$\begin{aligned} p(a) &\leftarrow p(b) \\ p(a) &\leftarrow \neg p(b) \\ p(b) &\leftarrow p(a) \end{aligned}$$

The following claims are easily checked: $T_{P_1} = T_{P_2}$; T_{P_1} and T_{P_2} are not constant; no clause in either program is redundant in the sense that removing any clause from either program changes the corresponding immediate consequence operator; no atom in any clause is redundant in the same sense (hence, these programs are in some sense irreducible); the first (and second) clause in P_1 is not subsumed by any clause in P_2; the first (and second) clause in P_2 is not subsumed by any clause in P_1. Furthermore, there is no obvious modification of the definition of subsumption just using simple containment of the sets pos and neg which overcomes the previous observation.

Finally, consider the following program P

$$\begin{aligned} p(a) &\leftarrow \\ p(b) &\leftarrow p(a) \end{aligned}$$

We see that $T_P = T_{P_1} = T_{P_2}$ and that P is definite. Again, no clause and no atom in P is redundant. This time, however, P subsumes both P_1 and P_2, but, conversely, neither P_1 nor P_2 subsumes P. $\blacksquare$

We note in passing the well-known fact that if P is definite, then T_P is monotonic and even continuous in the order-theoretic sense, but that the programs used in the previous example show that the converse of this is false.

As already noted, the converse of the previous proposition does hold for definite programs, and for the sake of completeness we prove this fact next in our present terms, that is, not using variable symbols nor theorems concerning addition of constants to $\mathcal{L}$, see [9].

5.4 Proposition Suppose that P_1 and P_2 are definite programs. If $T_{P_1} = T_{P_2}$, then P_1 and P_2 are subsumption equivalent.

Proof. Let C be any clause in P_1. Suppose that $\text{head}(C) = A$ and let I be the (possibly empty) set $\text{pos}(C)$. Then we immediately have that $A \in T_{P_1}(I)$. Hence, $A \in T_{P_2}(I)$ and therefore there is a clause C' in P_2 whose head is A and is such that $I \models \text{body}(C')$. But this latter statement simply says that $\text{pos}(C') \subseteq \text{pos}(C)$ and hence C' subsumes C.

By the symmetry of this argument, we obtain that P_1 and P_2 are subsumption equivalent, as required. $\blacksquare$

5.2 Further Generalities

One of the virtues of introducing the operators $B(T)$ and $L(T)$ is that they can be used to study properties of T and hence to study, ultimately, the underlying programs. This point is quite general, of course, provided the representation of programs in terms of operators is sufficiently faithful. In this context, we have the following result.

5.5 Proposition Let P_1 and P_2 be normal logic programs. Then $T_{P_1} = T_{P_2}$ iff $B(T_{P_1}) = B(T_{P_2})$ iff $L(T_{P_1}) = L(T_{P_2})$.

We further illustrate the points just made by showing next how periodic and fixed points of a transformation $T : X \to X$ are simply related to the corresponding entities for $B(T)$ and $L(T)$. This observation then applies to any of the semantic operators T_P, Φ_P and Ψ_P. In fact, we just work with $B(T)$ since the same facts for $L(T)$ can be established in the same way.

Suppose that $T : X \to X$ and form the operator $B(T) : B(X) \to B(X)$ as usual, where $B(T)(g) = g \circ T$:

$$\begin{tikzcd} & F \\ X \arrow[ur, "B(T)(g)"] \arrow[r, "T"] & X \arrow[u, "g"'] \end{tikzcd}$$

Consider the following calculation, where $g \in B(X)$ is arbitrary.
$B(T)^2(g) = B(T)(B(T)(g)) = B(T)(g \circ T) = B(T)(h) = h \circ T$, where $h = g \circ T$. Thus, $B(T)^2(g) = h \circ T = (g \circ T) \circ T = g \circ T^2$ by associativity of composition of functions. Again, $B(T)^3(g) = B(T)(B(T)^2(g)) = B(T)(g \circ T^2) = (g \circ T^2) \circ T = g \circ T^3$. Thus we have, by induction, the following identity

$$B(T)^n(g) = g \circ T^n$$

for all $n \in \mathbb{N}$ which shows that we can transform iterates of T into iterates of $B(T)$, and vice-versa. For example, if T is periodic with period n, that is, $T^n = Id_X$ for some $n \in \mathbb{N}$, then it is clear that $B(T)^n = Id_{B(X)}$ and so $B(T)$ is periodic with period n. Conversely, if $B(T)$ is periodic with period n, the same identity shows that T is periodic with period n.

Next, we relate fixed points of T to fixed points of $B(T)$, as follows.

5.6 Proposition Let $x \in X$. Then x is a fixed point of T if and only if we have the identity

$$B(T)(g)(x) = g(x)$$

for all $g \in B(X)$.

Proof. If $T(x) = x$, then immediately we have that $B(T)(g)(x) = g(T(x)) = g(x)$ for all $g \in B(X)$.

Conversely, if $T(x) \neq x$, choose $g \in B(x)$ such that $g(T(X)) \neq g(x)$. Then we obtain $B(T)(g)(x) = g(T(x)) \neq g(x)$, and the result follows. ■

5.3 A Simple Example

We close by considering two simple example programs illustrating the theory. Despite their simplicity, they reveal two interesting points.

To make the examples here genuinely simple, we suppose that no function symbols are present in our language and therefore that the sets I_P are finite, $I_P = \{I_1, I_2, \ldots, I_n\}$, say, where n depends on the number of constant and predicate symbols present in P. In this case, $\mathcal{F}(I_P)$ is naturally equivalent to R^n under the identification of $f : I_P \to R$ with the n-tuple $(f(I_1), f(I_2), \ldots, f(I_n))$. So, $\mathcal{F}(T_P)$ is then an $n \times n$ real matrix $A = (a_{ij})$ obtained in the usual way as the matrix of a linear operator. We want to consider the L^p norms in this case, $1 \leq p \leq \infty$, and also the supremum norm, $\| \; \|_\infty$. These norms are all equivalent due to the fact that R^n is finite dimensional, but their actual numerical values are of interest; we want to consider the corresponding operator norms, $\|\mathcal{F}(T_P)\|_p$ and $\|\mathcal{F}(T_P)\|_\infty$, determined by the norms on R^n. We consider here just the norm $\| \; \|_1$, where we take the counting measure on I_P. Thus, $\|\mathbf{x}\|_1$ is in fact the sum $\sum_{i=1}^{n} |x_i|$ for $\mathbf{x} = (x_1, x_2, ..., x_n) \in \mathbf{R}^n$. In this case, $\|\mathcal{F}(T_P)\|_1$ coincides with the maximum column sum norm, $\max_j \sum\limits_{i=1}^{n} |a_{ij}|$.

5.7 Example For example, take P_1 and P_2 to be, respectively, the following programs

$$q(a) \leftarrow \qquad\qquad q(a) \leftarrow$$
$$q(b) \leftarrow q(c) \qquad\qquad q(b) \leftarrow \neg q(c)$$

The underlying language $\mathcal{L}$ for both these programs contains the three constant symbols a, b, c and the unary predicate symbol q. Thus, in both cases the Herbrand base $B_{\mathcal{L}}$ is $[q(a), q(b), q(c)]$, which we write as a list since the order matters for our present purposes. Hence, there are eight distinct subsets of $B_{\mathcal{L}}$, which we list as follows (again, the order matters):
$I_{\mathcal{L}} = [[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1], [1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]]$.

Our convention here relates to the representation above of $B_{\mathcal{L}}$ as a list and, for example, the set denoted by the list $[0, 1, 1]$ is the set $\{q(b), q(c)\}$ ($q(a)$ is not contained in the set because 0 is the first element of the list, and $q(b)$ and $q(c)$ are elements because 1 is in the second and third position of the list $[0, 1, 1]$, and so on).

Thus, $\mathcal{F}(I_P)$ is equivalent to R^8. Indeed, a function $f \in \mathcal{F}(I_P)$ maps each of the sets in $I_{\mathcal{L}}$ to a number in R, and so each f can be represented as an 8-tuple of real numbers $(x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$, where $f([0, 0, 0]) = x_1$, $f([0, 0, 1]) = x_2$, etc. Conversely, an 8-tuple of real numbers determines an element of $\mathcal{F}(I_P)$ in the obvious way. Bearing in mind the definition of T_{P_1} and T_{P_2}, we now consider the effect of $\mathcal{F}(T_{P_1})$ and $\mathcal{F}(T_{P_2})$ on the standard basis $\{\mathbf{e}_i; i = 1, \ldots, 8\}$ for R^8, where $\mathbf{e}_i$ is the vector consisting of zeros in all places but the ith, which contains 1, and we regard the standard basis as an element of $\mathcal{F}(I_P)$. Then the jth column of the required matrices is obtained from the coordinate vector of $\mathcal{F}(T_{P_k})(\mathbf{e}_j)$, $k = 1, 2$, relative to the standard basis. For example, $(\mathbf{e}_5 \circ T_{P_1})([0, 0, 0]) = \mathbf{e}_5([1, 0, 0]) = 1$, $(\mathbf{e}_5 \circ T_{P_1})([0, 0, 1]) = \mathbf{e}_5([1, 1, 0]) = 0$, $(\mathbf{e}_5 \circ T_{P_1})([0, 1, 0]) = \mathbf{e}_5([1, 0, 0]) = 1$ etc. Thus, the fifth column of the matrix representing T_{P_1} is the fifth column of the first matrix below, and we proceed similarly to find the other columns and thereby obtain the following matrices.

$$\mathcal{F}(T_{P_1}) = \begin{pmatrix} 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1&0 \end{pmatrix} \qquad \mathcal{F}(T_{P_2}) = \begin{pmatrix} 0&0&0&0&0&0&1&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&1&0&0&0 \end{pmatrix}$$

Therefore, we have

$$\mathcal{F}(T_{P_1}) - \mathcal{F}(T_{P_2}) = \begin{pmatrix} 0&0&0&0&1&0&-1&0 \\ 0&0&0&0&-1&0&1&0 \\ 0&0&0&0&1&0&-1&0 \\ 0&0&0&0&-1&0&1&0 \\ 0&0&0&0&1&0&-1&0 \\ 0&0&0&0&-1&0&1&0 \\ 0&0&0&0&1&0&-1&0 \\ 0&0&0&0&-1&0&1&0 \end{pmatrix}$$

Hence, $d(P_1, P_2) = \|\mathcal{F}(T_{P_1}) - \mathcal{F}(T_{P_2})\| = \max_j \sum_{i=1}^{8} \mid a_{ij} \mid = 8$.

We note that this value for d is the maximum possible here, and this is consistent with the expected effect (which in a sense is also maximal) of replacing $q(c)$ in the first program by $\neg q(c)$ in the second. Thus, the metric d correctly reflects the large effect that the introduction of negation has, and this is the first of the two points we want to make about the example. The second is a general fact and is also illustrated by the matrix representations above, and it is this. Consider the listing of the elements of the power set of B_P and the matrix representation of T_P for a program P. Then a "1" occurring on the main diagonal in the ith row and ith column of the matrix corresponds exactly to the ith element in the listing being a fixed point of T_P.

5.8 Example Considering the programs defined in Example 5.3, we see that $d(P_1, P_2) = 0$, $d(P_1, P) = 0$ and $d(P_2, P) = 0$.

6 Conclusions

We have made two contributions. In the first, we have shown that logic programming semantics can be studied within the framework of Banach spaces and linear operators. Since Kozen [6] showed how to do this for deterministic and probabilistic programs, see also [7], it can now be seen that this framework is rather general in that it provides a semantic framework for all three of the computing paradigms just mentioned. In this process, the denotation of a program is a bounded linear operator.

The view is often expressed that the full objective of formal methods (complete proof of correctness of software) is not feasible. On the other hand, software metrics provide only a rough measure of how different two programs are or how far a program is from some ideal. The framework described in the previous paragraph provides a means of defining metrics, based on semantics, with the potential to formalize the process of comparing two programs, and this is our second contribution. In realizing this programme of research, initiated in [4], a number of criteria have to be met and a number of obstacles have to be overcome if software metrics are to be formalized; some of these criteria are as follows.

1. The operator denoting a program must faithfully reflect properties of the program, especially its semantics.

2. The distance between two programs must be meaningful in terms of program development.

3. It must be possible/feasible to calculate the distance between two programs.

The initial results presented here, at least for logic programs, give some positive support for this programme. In fact, the proposal envisaged in [2, 4] is even more ambitious than that described here in that it proposes the use of optimal approximation methods, based on techniques of mathematical analysis, to find "a reasonable practically acceptable approximation to the ideal solution". Whether or not any of this is really feasible remains to be seen.

Finally, we note that, aside from the issues discussed above, there are several purely mathematical questions raised by our work, which lack of space prevents us from mentioning. However, one example is the following: the process of embedding a domain inside a Banach lattice is going in the opposite direction to the known problem of representing classical spaces (in this case a Banach space) as the maximal point space of some domain in the Scott topology. It should prove to be of interest to investigate this and many related questions.

References

[1] Abramsky, S. and Jung, A. (1994) Domain Theory. In Abramsky, S., Gabbay, D. and Maibaum, T.S.E. (eds.), *Handbook of Logic in Computer Science Volume 3.* Oxford University Press, Oxford.

[2] Bukatin, M.A. (1996) Continuous Functions as Models for Programs: Mathematics and Applications. Technical Report, Department of Computer Science, Brandeis University, Waltham, Mass.

[3] Bukatin, M.A. (2002) *Mathematics of Domains.* PhD Thesis, Department of Computer Science, Brandeis University, Waltham, Mass.

[4] Bukatin, M.A. and Scott, J.S. (1997) Towards Computing Distances between Programs via Scott Domains. In Adian, S. and Nerode, A. (eds.), *Logical Foundations of Computer Science*, Lecture Notes in Computer Science, 1234, pp. 33–43, Springer-Verlag, Berlin.

[5] Hitzler, P. and Seda, A.K. (2000) A Note on the Relationships between Logic Programs and Neural Networks. In Gibson, P. and Sinclair, D. (eds.), *Proceedings of the 4th Irish Workshop on Formal Methods (IWFM'00)*, Maynooth, July. Electronic Workshops in Computing (eWiC), pp. 1–9, British Computer Society, Swindon, U.K.

[6] Kozen, D. (1981) Semantics of Probabilistic Programs. *J. Computer and Systems Science*, **22**, 328–350.

[7] Di Pierro, A. and Wiklicky, H. (2001) Linear Structures for Concurrency in Probabilistic Programming Languages. In Hurley, T., Mac an Airchinnigh, M., Schellekens, M. and Seda, A.K. (eds.), *Proceedings of MFCSIT2000*, Cork, July, 2000. Electronic Notes in Theoretical Computer Science, Vol. 40, pp. 1–44, Elsevier, Amsterdam.

[8] Lloyd, J.W. (1988) *Foundations of Logic Programming.* 2nd Edition, Springer-Verlag, Berlin.

[9] Maher, M. (1988) Equivalences of Logic Programs. In Minker, J. (ed.), *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann Publishers Inc., Los Altos.

[10] Matthews, S.G. (1994) Partial Metric Topology. In Andima, S. et al. (eds), *Proceedings of the Eighth Summer Conference on General Topology and its Applications.* Annals of the New York Academy of Science, Vol. 728, pp. 183–197, New York.

[11] Panangaden, P. (1997) Stochastic Techniques in Concurrency. Notes from Course on Stochastic Techniques in Concurrency, Aarhus, Fall 1996 and EATCS Summer School, Udine, 1997. BRICS, Department of Computer Science, Aarhus University, Aarhus, Denmark.

[12] Seda, A.K. (1995) Topology and the Semantics of Logic Programs. *Fundamenta Informaticae*, **24** (4), 359–386.

[13] Seda, A.K. and Hitzler, P. (1997) Topology and Iterates in Computational Logic. *Proceedings of the 12th Summer Conference on Topology and its Applications: Special Session on Topology in Computer Science*, Ontario, August 1997. Topology Proceedings Vol. 22, pp. 427–469, Summer 1997.

[14] Willard, S. (1970) *General Topology.* Addison-Wesley, Reading MA.